Reliability assessment of phased-mission systems with AltaRica 3.0

Michel Batteux, Tatiana Prosvirnova, Antoine Rauzy, Liu Yang

Reliability assessment of phased-mission systems with AltaRica 3.0

Michel Batteux
IRT SystemX
Palaiseau, France
Email: michel.batteux@irt-systemx.fr

Tatiana Prosvirnova
LGI, CentraleSupélec, Université Paris-Saclay
Gif-sur-Yvette, France
Email: tatiana.prosvirnova@centralesupelec.fr

Antoine Rauzy, Liu Yang
MTP, NTNU
Trondheim, Norway
Email: antoine.rauzy@ntnu.no, liu.yang@ntnu.no

***Abstract*—In this article, we show how phased-mission systems can be described and assessed by means of the AltaRica 3.0 technology. AltaRica 3.0 is a formal, object-oriented modeling language dedicated to probabilitic risk/safety analyses. We illustrate here its power by introducing a modeling pattern making it possible to represent phased-mission systems in an elegant way. This modeling pattern fits with the CESAMES framework for systems engineering. We show that AltaRica 3.0 models for phased-mission systems can be assessed efficiently by means of the stepwise simulator and the stochastic simulator we developed for the language.**

***Index Terms*—phased-mission systems, AltaRica 3.0, modeling patterns**

I. Introduction

It is often the case that the mission of complex systems can be decomposed into successive phases. Such systems are called phased-mission systems (PMS). Typically, the flight of an aircraft involves take-off, ascent, level-flight, descent, and landing phases. During each of these phases, the aircraft has to accomplish a specific task and is subject to different stresses as well as different dependability requirements, hence justifying different reliability assessment models. Phases are however not independent: if a certain capacity is lost during a phase, it has good chance to remain lost for the subsequent phases. Reference [16] presents a state of the art on different techniques and assessment methods for PMS.

In this article, we show how phased-mission systems can be described and assessed by means of the AltaRica 3.0 technology. AltaRica 3.0 is a formal, object-oriented modeling language dedicated to probabilitic risk/safety analyses [13]. AltaRica 3.0 can be seen as a combination of two parts:

- Its mathematical framework, based on guarded transition systems (GTS) [7], [15];
- Its set of structuring constructs, based on the system structure modeling language (S2ML) [6].

GTS are stochastic state automata. They have been designed to increase as much as possible the expressive power of the language without increasing the computational cost of the assessment algorithms. S2ML gathers and unifies structuring constructs stemmed from object-oriented programming [2], and prototype-oriented programming [12]. The combination of GTS and S2ML results in a powerful, versatile language which exploits assessment algorithms in an optimum way.

One of the big advantages of high level modeling is the possibility to reuse modeling components within models and between models. In this way, the modeling process is made both more efficient and less error prone. In languages such as Modelica [9], reuse is achieved via the design of libraries of "on-the-shelf" modeling components. This approach is also feasible in probabilistic risk and safety analyses, but to a much lesser extent. The reason is that these analyses represent systems at a high level of abstraction. Modeling components, except for very basic ones, tend thus to be specific to each system.

In AltaRica 3.0, reuse is mostly achieved by the design of modeling patterns, i.e. examples of models representing remarkable features of the system under study. Once identified, patterns can be duplicated and adjusted for specific needs [4]. Patterns are not only a mean to organize and to document models, but also and more fundamentally a way to reason about systems under study.

In this article, we introduce a new modeling pattern dedicated to the phased-mission systems. We show that this modeling pattern fits into the CESAMES framework for systems architecting [10]. We illustrate the proposed approach by applying this pattern to model a satellite communication system. We show how phased-mission systems can be assessed using the AltaRica Wizard modeling and simulation environment.

The remainder of this article is organized as follows. Section II presents a phased-mission system used as an illustrative example throughout this article. Section III gives an overview of the key concepts of the AltaRica 3.0 modeling language. Section IV describes the modeling pattern for phased-mission systems and applies it to the case study. Section V discusses why this pattern fits into the CESAMES framework. Section VI gives some numerical results. Finally, section VII concludes the article.

II. Case study

As an illustrative example, we shall consider a phased-mission system borrowed from [11], which represents the communication mission between a given satellite Sat and the ground stations F1 and F2, as shown Fig. 1.

Telecommunications between the satellite Sat and the ground stations rely on different equipment at different periods of time. Generally there are two channels available

for Sat to communicate with the stations: the first channel uses geostationary satellites SatRelay1 and SatRelay2 as the relay (and then SatRelay1 or SatRelay2 retransmit images to the stations); the second channel allows the satellite Sat to communicate directly with the stations when they are visible to each other. The communication channel can be considered as a subsystem which may contain antennas, batteries, transmitters, and receivers as shown in the reliability block diagrams Fig. 1.

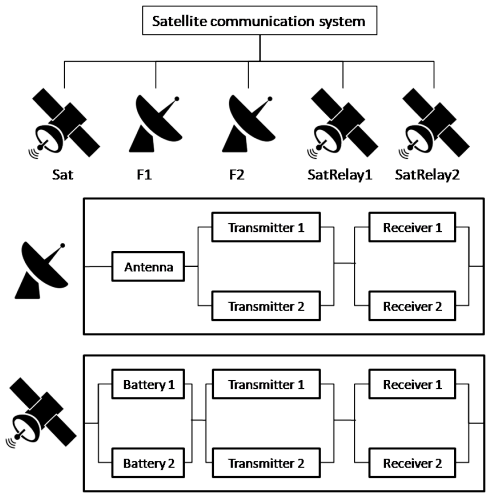

Fig. 1. Satellite communication system.

The satellite Sat orbits the Earth for 300 laps, each orbital lap contains four phases. The subsystems used in each phase are represented by the reliability block diagrams shown Fig. 2. The durations of the different phases are given Table I. The satellite communication system is thus a PMS made of 28 independent components and operating through 1200 successive phases.

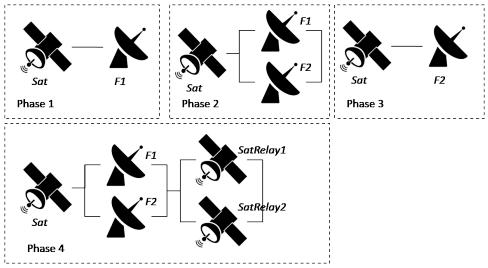

Fig. 2. Phases of the satellite communication system.

All components may fail in operation even if they are not in use during the considered phase. The radar (ground station) components can be repaired while the satellite components cannot. Failure and repair rates are shown in Table I.

Our objective is to assess the reliability of this phased-mission system for a 3600 hours mission.

III. AltaRica 3.0

A. Guarded Transition Systems

Guarded transition systems, introduced in [15], are at the core of AltaRica 3.0. Formally, a guarded transition system is

TABLE I
Parameters

Parameters	Values
Failure rates	$10^{-5}h^{-1}$
Repair rates for radars	$0.025h^{-1}$
Phase duration	$D1 = D3 = 2h$, $D2 = 1h$, $D4 = 7h$

a quintuple $\langle V, E, T, A, \iota \rangle$, where:

- V is a set of variables. Each variable v of V has a type, i.e. can take its value in a certain set of constants (Booleans, integers, reals or set of symbolic constants), called its domain and denoted by $\text{dom}(v)$. V is actually the disjoint union of two subsets S and F, where S is the subset of state variables and F is the subset of flow variables.
- E is a set of events. Each event e of E can be associated with a non-decreasing invertible function $delay_e$ from $[0, 1]$ to $\mathbb{R}^+ \cup \{+\infty\}$. The inverse $delay_e^{-1}$ of this function is a cumulative probability distribution.
- T is a set of transitions. A transition t is a triple $\langle e, g, a \rangle$, denoted by $g \xrightarrow{e} a$, where e is an event of E, g is a Boolean condition on variables of V called the guard of the transition, and a is an instruction, called the action of the transition, that changes the value of (some of) the state variables.
- A is an instruction, called the assertion, that calculates the values of flow variables from the values of state variables.
- Finally, ι is a function that gives the initial value of state variables and the default value of flow variables.

As an illustration, we shall consider a non-repairable component in Fig. 3.

```
1 class NonRepairableComponent
2   Boolean vsWorking (init = true);
3   event failure (delay = exponential(pLambda));
4   parameter Real pLambda = 1.0e-5;
5   transition
6     failure: vsWorking -> vsWorking := false;
7 end
```

Fig. 3. AltaRica 3.0 code representing a non-repairable component.

This modeling component is reused in many different models and often many times within a model. For this reason, it is declared as a class in the code of Fig. 3. A class is an on-the-shelf modeling component. It can be instantiated as many times as necessary into models. The state of a component is represented by a Boolean variable named `vsWorking` (declared line 2). Initially, the component is working, so the attribute `init` of the variable `vsWorking` is set to true. This attribute indicates also that `vsWorking` is a state variable. The component has a failure transition (declared line 6) that goes from the state working to the state failed. In the code of Fig. 3, the event `failure` (declared line 3) is associated with a delay obeying the inverse of a negative exponential distribution of parameter `pLambda`. This parameter is declared at

line 4. AltaRica 3.0 provides several built-in distributions, like Dirac, exponential, Weibull as well as empirical distributions (given as a list of points).

AltaRica 3.0 comes with a standard library that declares a number of classes such as `NonRepairableComponent` to represent various types of components.

B. Composition

The class `NonRepairableComponent` involves no flow variable and therefore no assertion. Flow variables are mainly a mean to connect components. The composition of two (or more) guarded transition systems is actually a guarded transition system. Formally, let $M_1 : \langle V_1, E_1, T_1, A_1, \iota_1 \rangle$ and $M_2 : \langle V_2, E_2, T_2, A_2, \iota_2 \rangle$ be two guarded transition systems. Then $M_1 \otimes M_2$ is simply the guarded transition system $\langle V, E, T, A, \iota \rangle$ such that $V = V_1 \cup V_2$, $E = E_1 \cup E_2$, $T = T_1 \cup T_2$, $A = A_2 \circ A_1$ and $\iota = \iota_2 \circ \iota_1$.

Larger models can thus be obtained by composing smaller models.

Consider the satellite subsystem of our illustrative example. It is composed of two batteries, two transmitters and two receivers. All the components may fail in operation and cannot be repaired. They can be represented as instances of the class `NonRepairableComponent`. The model of the whole satellite subsystem is given Fig. 4.

```
1  class NonRepairableInOutComponent
2    extends NonRepairableComponent;
3    Boolean vfInput, vfOutput(reset = false);
4    assertion
5      vfOutput := vsWorking and vfInput;
6  end
7
8  class SatelliteSubSystem
9    NonRepairableInOutComponent Battery1;
10   NonRepairableInOutComponent Battery2;
11   NonRepairableInOutComponent Transmitter1;
12   NonRepairableInOutComponent Transmitter2;
13   NonRepairableInOutComponent Receiver1;
14   NonRepairableInOutComponent Receiver2;
15   Boolean vfOutput( reset = false );
16
17   assertion
18     Battery1.vfInput := true;
19     Battery2.vfInput := true;
20     Transmitter1.vfInput := Battery1.vfOutput
21       or Battery2.vfOutput;
22     Transmitter2.vfInput := Battery1.vfOutput
23       or Battery2.vfOutput;
24     Receiver1.vfInput := Transmitter1.vfOutput
25       or Transmitter2.vfOutput;
26     Receiver2.vfInput := Transmitter1.vfOutput
27       or Transmitter2.vfOutput;
28     vfOutput := Receiver1.vfOutput
29       or Receiver2.vfOutput;
30 end
```

Fig. 4. AltaRica 3.0 code implementing the satellite subsystem.

This code declares first the class `NonRepairableInOutComponent` (lines 1-6). This class inherits from the class `NonRepairableComponent` (via the `extends` clause, line 2). This means that a `NonRepairableInOutComponent` is a `NonRepairableComponent` with additional properties. In this case, Boolean flow variables `vfInput` and `vfOutput` are declared (line 3). Their default values are set to false via the attribute `reset`. This attribute indicates also that `vfInput` and `vfOutput` are flow variables. Finally, the assertion line 4 tells how the value of the flow variable `vfOutput` is calculated from the value of the state variable `vsWroking` and the flow variable `vfInput`.

The satellite subsystem itself is encoded as a class as it shall be reused several times in the model (lines 8-30). The class `SatelliteSubSystem` declares as many instances of `NonRepairableInOutComponent` as there are components (lines 9-14) and a Boolean flow variable `vfOutput` (line 15). The assertion consists then simply in connecting together the inputs and outputs of the components (lines 18-29). Variables declared inside a component (prototype or instance of class) are accessed via the dot notation: `Battery1.vfInput` denotes the variable `vfInput` of the component `Battery1`.

Note that as in the code Fig. 3, the values of the parameters `pLambda` can be redefined at instantiation, if necessary. It is even possible to change the distributions themselves at instantiation. This makes it possible to design generic classes for components.

To represent the radar subsystem from our illustrative example, we need first to define a class `RepairableInOutComponent` which inherits from the class `NonRepairableComponent` and defines a new event `repair` and the associated transition; second we shall use this class to define the class `RadarSubSystem` in the same way as the class `SatelliteSubSystem` given Fig. 4.

C. Aggregation

Composition describes a "is-part-of" relation. This relation assumes that a component cannot belong to two different parent components. There are cases however where the same component is used in several places or to contribute to different functions of the system. This kind of "uses" relations can be described by means of aggregation.

As an illustration, consider again our satellite communication system. This system is made of five sub-systems; these sub-systems are further decomposed, and so on. This structural breakdown is correct, but does not represent the whole system. The system description is completed by the reliability block diagrams Fig. 2 for each phase of the system. These reliability block diagrams encode more functional than physical groupings. Physical components are shared by different reliability block diagrams. Therefore, we need to be able to describe hierarchical breakdowns with branches sharing components. This is the very purpose of the notion of aggregation. It is implemented in AltaRica 3.0 by the "embeds-as" clause.

The AltaRica 3.0 code describing the 1st phase of the satellite communication system is shown Fig. 5

```
1  block SatelliteCommunicationSystem
2    /* System breakdown structure */
3    SatelliteSubSystem Sat, SatRelay1, SatRelay2;
4    RadarSubSystem F1, F2;
5    ...
6    /* Subsystem used during the 1st phase */
7    block Phase1
8      embeds main.Sat as Sat;
9      embeds main.F1 as F1;
10     Boolean vfWorking (reset = false);
11     assertion
12       vfWorking := F1.vfOutput and Sat.vfOutput;
13   end
14   ...
15 end
```

Fig. 5. AltaRica 3.0 code describing phase 1.

The satellite communication system itself is encoded as a prototype, i.e. a modeling component with a unique occurrence (lines 1-15). There is a little chance for this modeling component to be reused somewhere. It is specific to the system under study, conversely to the components `NonRepairableInOutComponent` or `RepairableInOutComponent` that are reused several times in this and other models. Prototypes are introduced by the keyword `block`.

The block `SatelliteCommunicationSystem` is composed of five subsystems defined as the instantiation of the previously defined classes `SatelliteSubSystem` and `RadarSubSystem` (lines 3-4).

The block `Phase1` aggregates (via the clause `embeds`) the components used during the 1st phase (lines 7-13). It also defines a Boolean variable `vfWorking` (a flow variable) which is true when the system is working and false otherwise (line 10). The assertion (lines 11-12) defines the value of the flow variable `vfWorking` according to the outputs of all the components.

IV. Architectural pattern for phased-mission systems

Fig. 6 gives a snapshot of the architectural modeling pattern we designed for phased-mission systems. This pattern consists in organizing the model in 4 parts:

1) The system breakdown structure (the hierarchy of components together with their behavior);
2) A unit that monitors the changes of phases, called "Phase controller";
3) The set of modeling units representing each phase of the system. Each phase unit aggregates the components used during this phase and declares interactions between them.
4) The global assertion, called "Multiplexer". This assertion describes whether the system is able to perform its mission in the current phase (decided by the phase controller). Performance indicators (e.g. system reliability) are calculated thanks to this assertion.

Connections between the 4 parts are described by two types of relations:

- "Flow propagation" by means of assertions, represented by plain arrows Fig. 6.
- "Aggregation" relation, represented by dashed arrows Fig. 6;

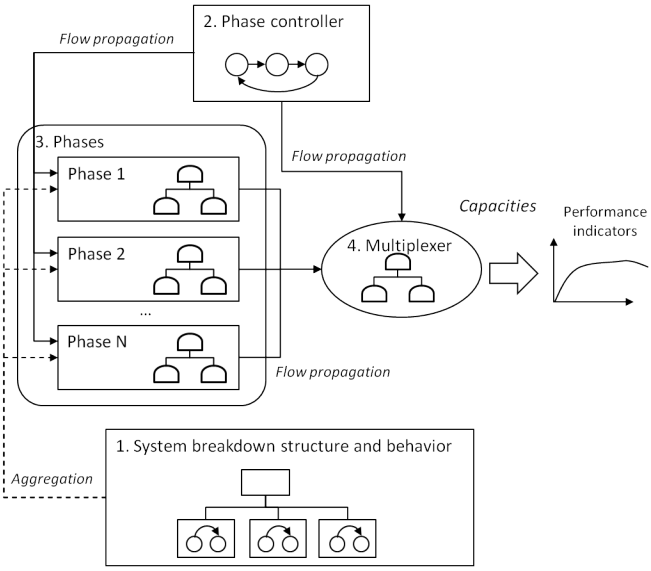

Fig. 6. Architectural pattern for phased-mission systems

As an illustration consider the satellite communication system described Section II. The structure of the AltaRica 3.0 model representing this system is given Fig. 7.

Inside the block `SatelliteCommunicationSystem`, the breakdown structure of the whole system is described lines 3-4) by instantiating previously defined classes `SatelliteSubSystem` and `RadarSubSystem`. This part corresponds to the block 1 in Fig. 6.

Second, the unit representing changes of phases is defined by the block `PhaseController` (lines 7-9). This block is specific to this system, that is why it is defined as a prototype. Its behavior is detailed Fig. 9. Changes of phases are represented by an automaton graphically represented Fig. 8. The phase is represented by a state variable `vsPhase` which is an Integer and can take 4 values. Changes of states are represented by the transitions which are labeled by deterministic events. Each event is thus associated with a deterministic delay defined by the parameters `Dj`. Their values are taken from Table I.

Note that it is possible to use an enumeration to define different phases. For example,

```
domain PhaseDomain {PHASE1, PHASE2, PHASE3, PHASE4}
```

This block corresponds to the block 2, "Phase Controller", Fig. 6.

Third, the architecture of the subsystem used during each phase is defined by the blocks `Phase1`, `Phase2`, `Phase3` and `Phase4` (lines 10-34). Consider for instance the block `Phase4`. This block aggregates (via the clause `embeds`) the

```
1  block SatelliteCommunicationSystem
2    /* System breakdown structure */
3    SatelliteSubSystem Sat, SatRelay1, SatRelay2;
4    RadarSubSystem F1, F2;
5    Boolean vfWorking (reset = false);
6    /* Unit representing the changes of phases */
7    block PhaseController
8      // behavior of the block PhaseController
9    end
10   /* Subsystem used during the 1st phase */
11   block Phase1
12     // definition of the block Phase1
13   end
14   /* Subsystem used during the 2nd phase */
15   block Phase2
16     // definition of the block Phase2
17   end
18   /* Subsystem used during the 3rd phase */
19   block Phase3
20     // definition of the block Phase3
21   end
22   /* Subsystem used during the 4th phase */
23   block Phase 4
24     embeds main.F1 as F1;
25     embeds main.F2 as F2;
26     embeds main.Sat as Sat;
27     embeds main.SatRelay1 as SatRelay1;
28     embeds main.SatRelay2 as SatRelay2;
29     Boolean vfWorking (reset = false);
30     assertion
31       vfWorking := (F1.vfOutput or F2.vfOutput)
32         and Sat.vfOutput
33         and (SatRelay1.vfOutput
34           or SatRelay2.vfOutput);
35   end
36   assertion // the global assertion
37     vfWorking := switch {
38       case (Phases.vsPhase==1): Phase1.vfWorking
39       case (Phases.vsPhase==2): Phase2.vfWorking
40       case (Phases.vsPhase==3): Phase3.vfWorking
41       default: Phase4.vfWorking
42     };
43 end
```

Fig. 7. Global view of the satellite communication system model.

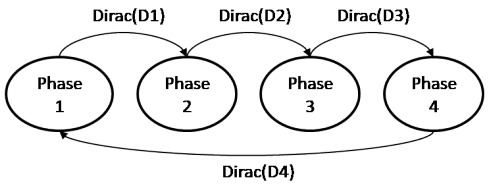

Fig. 8. Automaton defining the changes of phases.

components used during the 4th phase (lines 24-28). It defines a Boolean variable `vfWorking` (a flow variable) which is true when the system is working and false otherwise (line 29). The assertion (lines 30-33) defines the value of the flow variable `vfWorking` according to the outputs of all the components.

The AltaRica 3.0 code of the block `Phase1` is given Section III-C. The code of the blocks `Phase2` and `Phase3` is similar to the other blocks. It implements the reliability block diagrams given Fig. 1. This part corresponds to the block 3, "Phases", Fig. 6.

```
1  block PhaseController
2    Integer vsPhase (init = 1);
3    parameter Real D1 = 2.0; // D1==D3
4    parameter Real D2 = 1.0;
5    parameter Real D3 = 2.0;
6    parameter Real D4 = 7.0;
7    event evChangePhase1_2(delay = Dirac(D1));
8    event evChangePhase2_3(delay = Dirac(D2));
9    event evChangePhase3_4(delay = Dirac(D3));
10   event evChangePhase4_1(delay = Dirac(D4));
11   transition
12     evChangePhase1_2: vsPhase==1 -> vsPhase := 2;
13     evChangePhase2_3: vsPhase==2 -> vsPhase := 3;
14     evChangePhase3_4: vsPhase==3 -> vsPhase := 4;
15     evChangePhase4_1: vsPhase==4 -> vsPhase := 1;
16 end
```

Fig. 9. AltaRica 3.0 code for the block PhaseController.

Finally, the global assertion (lines 36-42) defines the value of the flow variable `vfWorking` (declared line 5) according to the phase (state variable `Phases.vsPhase`) and the status of the corresponding phase unit (flow variable `PhaseJ.vfWorking`). This part corresponds to the block 4, "Multiplexer", Fig. 6.

Note that subsystems `Sat`, `F1`, `F2`, `SatRelay1` and `SatRelay2` are shared between the blocks `Phase1`, `Phase2`, `Phase3` and `Phase4`, which define the system configuration used in each phase. It is achieved thanks to the aggregation relation (`embeds` clause). The aggregation relation allows to define phased mission systems in an elegant way.

It is also possible to define conditional connections between components depending on the phase. Again consider our illustrative example. First, we declare a flow variable `vfPhase` inside of each phase unit (Fig. 10, line 4). Second, we connect it to the state variable `Phases.vsPhase` in the global assertion (Fig. 10, lines 15-16). Finally, we make the local assertion depend on the variable `vfPhase` (Fig. 10, lines 9-13).

```
1  block SatelliteCommunicationSystem
2    ...
3    block Phase1
4      Integer vfPhase (reset = 1);
5      embeds main.Sat as Sat;
6      embeds main.F1 as F1;
7      Boolean vfWorking (reset = false);
8      assertion
9        if (vfPhase == 1) then {
10         Sat.vfInput := true;
11         F1.vfInput := Sat.vfOutput;
12         vfWorking := F1.vfOutput;
13         }
14   end
15   assertion
16     Phase1.vfPhase := Phases.vsPhase;
17     ...
18 end
```

Fig. 10. AltaRica 3.0 code implementing conditional connections.

V. From system architectures to safety analyses

System architecture is an emerging discipline. It provides a conceptual framework making it possible to merge in a coherent way all of the point of views on a system, and to reason about the system in an accurate way relying on approach by levels of abstraction. System architects apply methodologies that involve the design of models. These methodologies are often called architecture frameworks [1].

The CESAMES method for systems architecting [10] considers three different abstraction levels of a system: the operational level, the functional level and the physical level (see Fig. 11).

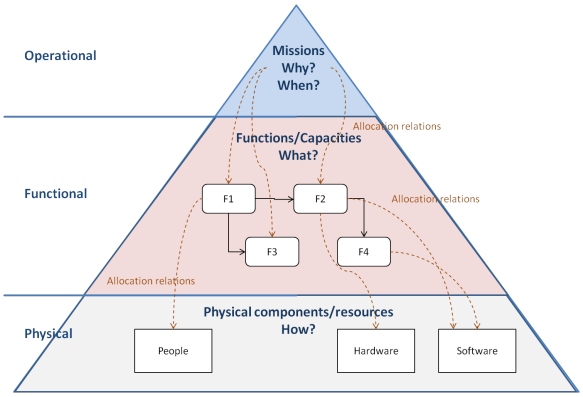

Fig. 11. CESAM systems architecture pyramid.

The operational level is the analysis of the environment of the system. It considers the system (more or less) as a black box and models the interactions of the system with the external systems. The result of the operational analysis is thus a description of the missions of the system, i.e. of the services it provides to its users.

The functional level is an abstract analysis of the inside of the system. It considers the system as a white box and models abstract functions/capacities of the system.

Finally, the physical level is a concrete analysis of the inside of the system. It considers also the system as a white box and models the concrete components of the system, in terms of hardware, sofware and human elements. The physical level describes thus the concrete resources the system involves.

Models produced at the three different levels are strongly connected. The operational level is connected with the two other levels because missions are naturally implemented by functions and by components. The functional level is connected with the physical level because each (abstract) function must be concretely allocated to, or implemented by, some set of physical components. In the reverse way, physical components implement functions which are required by missions.

The CESAMES method for systems architecting proposes five types of SysML diagrams for each abstraction level. They are summarized Table II.

Safety analyses need to gather in the same model operational, functional and physical aspects of the system under study. Typically, the top event of a fault tree represents the loss of a function/capacity, i.e. the incapacity to accomplish a mission, and the basic events represent the failures of physical components.

TABLE II
CESAMES systems architecture diagrams.

Operational	Functional	Physical
Needs Architecture	Requirement Architecture	Requirement Architecture
Lifecycle	Functional mode	Configuration
Use case	Decomposition/ Interaction	Decomposition/ Interaction
Scenario	Scenario	Scenario
Flow	Flow	Flow

Nevertheless, the architectural pattern in the previous section can be seen as an implementation of the CESAMES method for systems architecting for the specific case of phased-mission systems, as illustrated Fig. 12.

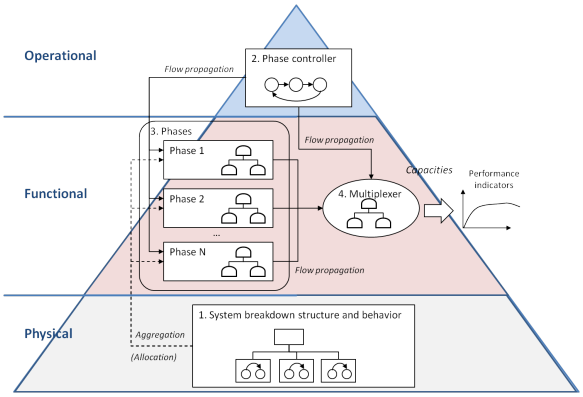

Fig. 12. The architectural pattern for phased-mission systems view from the perspective of the CESAMES method for systems architecting.

The block “Phase Controller” (block 2), which describes the different phases of the system, belongs to the operational level and corresponds to the lifecycle diagram of the CESAMES method.

The block “System breakdown structure and behavior” (block 1) belongs to the physical abstraction level and corresponds to the physical breakdown structure, which represents the hierarchy of physical components.

The blocks “Phases”(block 3) and “Multiplexer”(block 4) are parts of the functional abstraction level. They correspond to functional interaction diagrams and represent functions (or capacities) provided by the system. Performance indicators (e.g. system reliability) are calculated from the values of the capacities.

Links between different abstraction levels (or different diagrams) are represented by two different means:

- The links between functional and physical levels are carried out by the “aggregation” relation. Phases (or functions) aggregate physical components/resources and define interactions needed to produce the capacity.

- The links between operational and functional levels are implemented by "Flow propagation", also called connections.

VI. Experiments

An integrated modeling environment for AltaRica 3.0 (AltaRica Wizard) is currently under development as joint effort of the OpenAltaRica team at IRT SystemX (Paris, France) and the Norwegian University of Science and Technology. Industrial partners (Airbus, Safran and Thales) support this project. A versatile set of assessment tools has been developed, which includes:

- A stepwise simulator,
- A Fault Tree compiler [14],
- A Markov chain generator [8],
- A stochastic simulator,
- A sequence generator.

AltaRica Wizard [5] already integrates three calculation engines: a stepwise simulator, a Fault Tree compiler and a stochastic simulator. Phased mission systems can be assessed with the stepwise simulator in order to validate the model and with the stochastic simulator in order to calculate different types of indicators, for example its reliability or mean down time.

First a step by step simulator has been used to play scenarios and to validate the model. Fig. 13 shows an example of the stepwise simulation of the satellite communication system. In the tab "Tree View" the simulator displays the current system configuration: the value of state and flow variables, the enabled transitions and the value of observers. It is possible to fire transitions by double clicking on them in the tab "Tree View". The simulation history is shown in the tab "Sequences". Users can go back to the initial configuration using the "Restart" button and backtrack using the button "Backtrack".

In Fig. 13, in the initial configuration the system is in the phase 1 and it is working (the value of the observer `oFailed` is false). Then the antenna of the station `F1` used in the phase 1 fails (the transition labeled by the event `F1.Antenna.evFailure` is fired). The system is failed (the value of the observer `oFailed` becomes true). Then the system goes in the phase 2 (the transition labeled by the event `PhaseController.evChangePhase1_2` is fired). The system is working (the value of the observer `oFailed` becomes false) because in the phase 2 both stations `F1` and `F2` are used in parallel. And so on.

Second, to calculate the reliability of the satellite communication system, the stochastic simulator has been used [3]. It is a versatile tool to assess performance indicators of discrete event systems [17]. The principle of stochastic, or Monte-Carlo, simulation is quite simple. It consists in drawing at random a sample of executions of the model, to observe a number of quantities during these executions, and to make statistics on these observations. Quantities to observe depend indeed on the performance indicators one wants to obtain. In AltaRica 3.0, they are defined in two steps. First, one declares observers in the model. These observers can be either symbolic (Boolean or symbolic constants) or numerical (integers or real numbers). In both cases, their values evolve through an execution. Second, one declares indicators, which are elements separated from the AltaRica 3.0 model (they are defined into a separate file). An indicator defines what to measure about the evolution of an observer, i.e. on which quantity to make statistics. It can be for instance:

- the first date at which an observer takes a specified value;
- the number of times an observer takes a specified value during an execution;
- the average value of an observer during an execution; etc.

Each execution runs from time 0 to a given mission time T. Statistics are indeed made on the values of indicators at time T. It may be interesting to make statistics on their values, at least for some of them, at intermediate times $0 \leq t_1 < t_2 < \ldots < t_k < T$. For our purpose the mission time is $T = 3600h$. We define the following Boolean observer

```
observer Boolean oFailed = not vfWorking;
```

and the associated indicator

```
first-occurrence-date(oFailed, true)
```

defining, for a run, the first date at which the observer oFailed took the value true over the time period $[0, 3600]$.

The stochastic simulator have been parametrized to perform 10^5 histories and to compute the reliability for different mission times. The results are given Fig. 14.

We have also performed a sensitivity analysis. In order to determine the most critical components we have calculated the system mean down time considering each component being failed. The results are presented Fig. 15. These results are not surprising in view of the architecture of the system under study. The most critical components are the components of the main satellite `SatR` as they cannot be repaired and are used in each phase. The antennas of the stations `F1` and `F2` are also critical as they are not redundant. There are different ways to improve the overall system availability:

- Use redundant antennas which is costly and sure.
- Improve the antennas' reliability and decrease their mean time to repair.

VII. Conclusion

In this article, we proposed an architectural pattern for phased-mission systems. We showed how to implement this pattern in AltaRica 3.0. We applied the proposed modeling pattern to a satellite communication system and calculated its reliability and mean down time using the stochastic simulator developed for the language.

The systematic exploration and design of modeling patterns is at the core of our research projects. It is actually of primary importance, in order to make the modeling process efficient, to reuse as much as possible modeling components within models and between models.

An another important issue is the synchronization of models developed by the different engineering disciplines. With that respect, the connection of systems engineering modeling

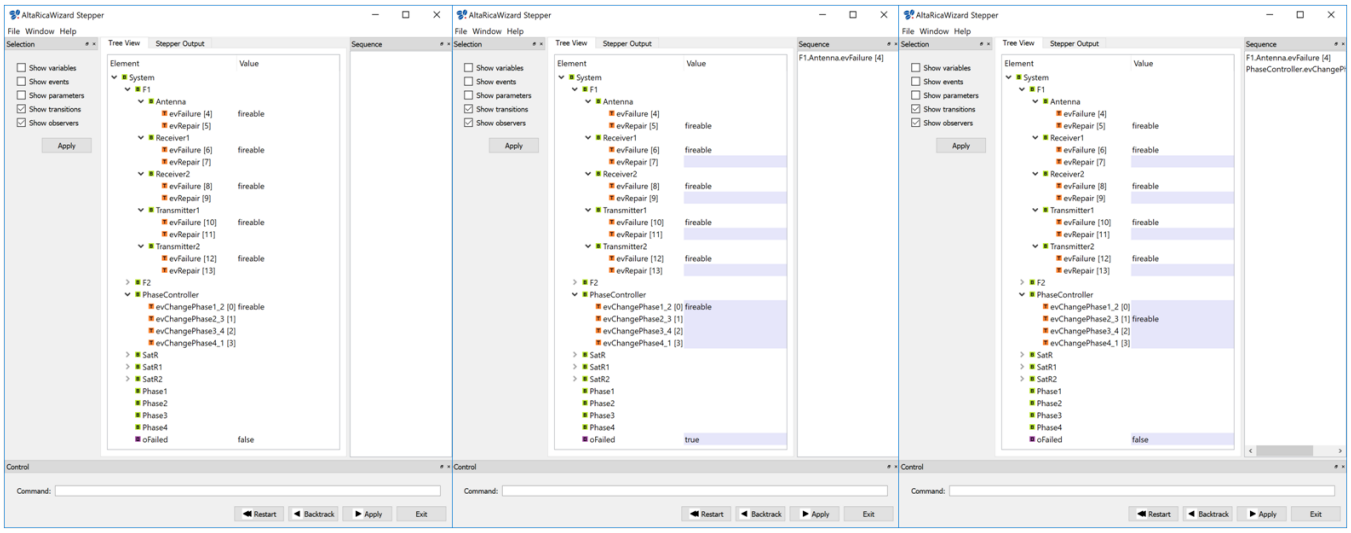

Fig. 13. Stepwise simulation of the satellite communication system

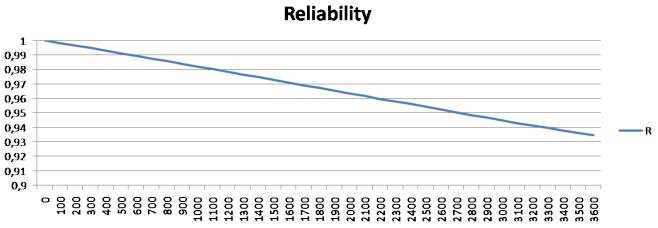

Fig. 14. Reliability of the satellite communication system

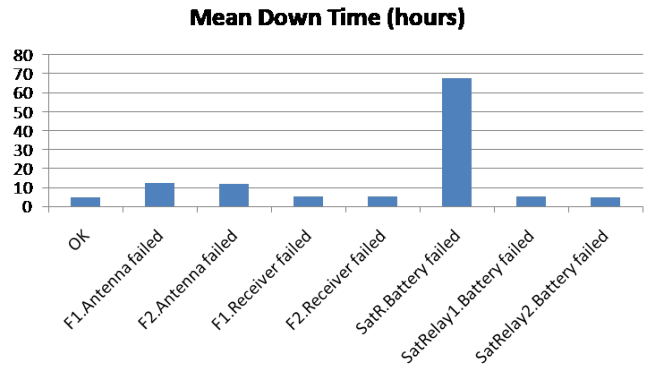

Fig. 15. Sensitivity analysis of the satellite communication system

frameworks with architectural patterns for models used in risk/safety analyses seems promising.

REFERENCES

[1] Systems and software engineering – architecture description, 2011-11-24.

[2] Mauricio Abadi and Luca Cardelli. *A Theory of Objects*. Springer-Verlag, New-York, USA, 1998.

[3] B. Aupetit, M. Batteux, A. Rauzy, and J.-M. Roussel. Towards a defintion of an evaluation kit for stochastic simulators. In *LambdaMu 20*, Saint-Malo, France, 2016. in press.

[4] M. Batteux, T. Prosvirnova, and A.Rauzy. Altarica 3.0 in 10 modeling patterns. *International Journal of Critical Computer-Based Systems (IJCCBS)*, 2018.

[5] M. Batteux, T. Prosvirnova, and A.Rauzy. Altarica wizard: an integrated modeling and simulation environment for altarica 3.0. In *LambdaMu 21*, Reims, France, 2018. in press.

[6] M. Batteux, T. Prosvirnova, and A.Rauzy. S2ml for structuring models. In *4th IEEE International Symposium on Systems Engineering, ISSE 2018*, Rome, Italy, 2018. in press.

[7] Michel Batteux, Tatiana Prosvirnova, and Antoine Rauzy. Altarica 3.0 assertions: the why and the wherefore. *Journal of Risk and Reliability*, September 2017.

[8] Pierre-Antoine Brameret, Antoine Rauzy, and Jean-Marc Roussel. Automated generation of partial markov chain from high level descriptions. *Reliability Engineering and System Safety*, 139:179–187, July 2015.

[9] Peter Fritzson. *Principles of ObjectOriented Modeling and Simulation with Modelica 3.3: A CyberPhysical Approach*. Wiley-IEEE Press, Hoboken, NJ 07030-5774, USA, 2015.

[10] Daniel Krob. *CESAM: CESAMES Systems Architecting Method: A Pocket Guide*. CESAMES, http://www.cesames.net, January 2017.

[11] Ji-Min Lu, Xiao-Yue Wu, Yiliu Liu, and Mary Ann Lundteigen. Reliability analysis of large phased-mission systems with repairable components based on success-state sampling. *Reliability Engineering & System Safety*, 142:123–133, 2015.

[12] James Noble, Antero Taivalsaari, and Ivan Moore. *Prototype-Based Programming: Concepts, Languages and Applications*. Springer-Verlag, Berlin and Heidelberg, Germany, 1999.

[13] Tatiana Prosvirnova, Michel Batteux, Pierre-Antoine Brameret, Abraham Cherfi, Thomas Friedlhuber, Jean-Marc Roussel, and Antoine Rauzy. The altarica 3.0 project for model-based safety assessment. In *Proceedings of 4th IFAC Workshop on Dependable Control of Discrete Systems, DCDS'2013*, pages 127–132, York, Great Britain, September 2013. International Federation of Automatic Control.

[14] Tatiana Prosvirnova and Antoine Rauzy. Automated generation of minimal cutsets from altarica 3.0 models. *International Journal of Critical Computer-Based Systems*, 6(1):50–79, 2015.

[15] Antoine Rauzy. Guarded transition systems: a new states/events formalism for reliability studies. *Journal of Risk and Reliability*, 222(4):495–505, 2008.

[16] L. Xing and S.V. Amari. Reliability of phased-mission systems. In *Misra K.B. (eds) Handbook of Performability Engineering*, London, UK, 2008. Springer.

[17] Enrico Zio. *The Monte Carlo Simulation Method for System Reliability and Risk Analysis*. Springer Series in Reliability Engineering. Springer London, London, England, 2013.