The New Open-PSA Format: a Model-Based Approach

Michel Batteux, Tatiana Prosvirnova, Antoine Rauzy

Institut pour la Maîtrise des Risques
Sûreté de Fonctionnement - Management - Cindyniques

The New Open-PSA Format: a Model-Based Approach

Michel Batteux
Institut de Recherche Technologique SystemX
Palaiseau, France
michel.batteux@irt-systemx.fr

Tatiana Prosvirnova
ONERA/DTIS
Université de Toulouse
Toulouse, France
tatiana.prosvirnova@onera.fr

Antoine Rauzy
Department of Mechanical and Industrial Engineering
Norwegian University of Science and Technology
Trondheim, Norway
antoine.rauzy@ntnu.no

***Abstract*— This communication presents the underlying principles of the new version of Open-PSA model exchange format, which relies on the S2ML+X paradigm. S2ML stands for System Structure Modeling Language. It unifies structuring constructs coming from object-oriented and prototype-oriented programming languages. The new version of Open-PSA format, based on S2ML for its structural part, improves very significantly the previous one. Both its theoretical foundations and examples of use are provided.**

Keywords—Probabilistic risk and safety analyses, model exchange formats, model-based systems engineering

I. INTRODUCTION

In 2008, the description of the Open-PSA model exchange format was published by Steven "Woody" Epstein and Antoine Rauzy [1]. The design of this format resulted from a joint effort of a small group of researchers, tools developers and expert safety analysts (mostly coming from the nuclear domain). The primary goal was to improve the quality of probabilistic risk and safety assessment models by proposing an XML format making it possible to exchange seamlessly fault trees and event trees from one tool to the other. This, in turn, makes it possible to cross check results, to facilitate peer reviews and more generally to allow new ideas to be tested. Technically, the project has been a success. The format has been validated [2]. Several tools adopted it, e.g. [3], and new ideas were proposed, see e.g. [4].

However, it is time to revisit the format, for two categories of reasons. First, XML has the drawbacks of its advantages: it is easy to parse by computer programs, but definitely unreadable by humans. It is thus of importance to propose an analyst a friendly version of the format so to facilitate its adoption. Second and more importantly, very significant progresses have been made recently in the design of behavioral modeling languages, notably via the introduction of the paradigm S2ML+X [5]. This communication presents the underlying principles of the new version of the Open-PSA format, which relies on this paradigm, and is now based on both textual and XML forms.

Any behavioral modeling language results from the combination of an underlying mathematical framework and a set of constructs to structure models. The choice of the underlying mathematical framework fully depends on which aspect of the behavior of the system under study one wants to capture. In the case of probabilistic risk and safety analyses, a good comprise consists for instance in using systems of stochastic Boolean equations. Fault trees and reliability block diagrams are typically interpreted as such systems.

The choice of the set of structuring constructs is to a very large extent independent of the one of the underlying mathematical framework. The new version of Open-PSA format relies on S2ML (System Structure Modeling Language), which is such a set of constructs that gathers in a unified and coherent way ideas stemmed from object- and prototype-oriented programming [6]. This opens new perspective to probabilistic risk and safety analyses. By implementing the model-as-script principle, the new version of the format makes easier the design, the debug and the maintenance of models. Even more importantly, it provides the technical infrastructure to better capitalize knowledge from project to project. It is worth to notice that the AltaRica 3.0 modeling language [7] already relies on S2ML.

The remainder of this article is organized as follows. Section II presents a case study used to illustrate different concepts of the new Open-PSA format. Section III introduces the paradigm "S2ML+Boolean equations" and describes the new syntax of the Open-PSA format. Section IV gives an overview of different architectural views of the case study. Section V presents the model of the case study using fault trees and section VI reliability block diagrams. Finally, section VII concludes this article.

II. CASE STUDY: OVERFLOW PROTECTION SYSTEM

Consider a safety instrumented system given Fig. 1**Erreur ! Source du renvoi introuvable.**. The goal of this system is to protect the tank from an overflow.

The system is composed of two safety barriers:

- The primary prevention barrier, which contains a sensor LS1, a controller C and a shutdown valve SDV1;

- The backup safety barrier made of a sensor LS2, a controller C, a shutdown valve SDV2 and a discharge valve DV.

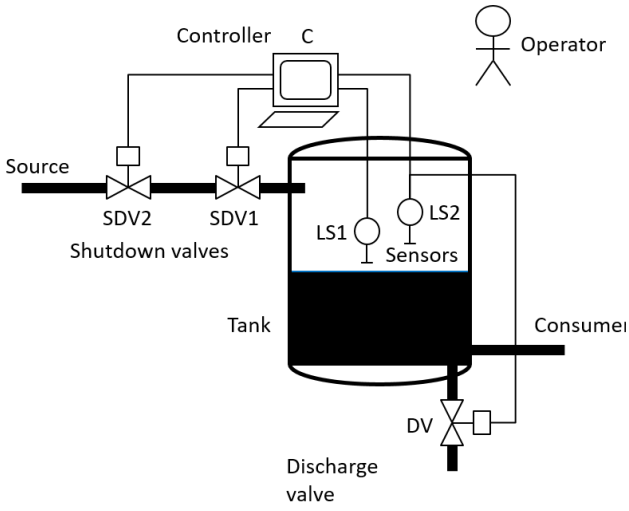

Fig. 1. Overflow protection system

When LS1 detects that the liquid level is too high, it sends the information to the controller that orders to shut down the valve SDV1. If the primary safety barrier is not operational, the backup one is used: the sensor LS2 detects the overflow of the tank, sends the information to the controller, which orders to shut down the valve SDV2 and to open the discharge valve DV. Note that the failure of the controller causes the loss of the primary and the backup safety barriers.

In the following, consider that failures of components obey exponential distribution with a failure rate $\lambda = 0.0001$.

The feared event is the overflow of the Tank, which occurs when there is an overflow and the overflow protection system is failed.

We use this case study throughout this article to illustrate different concepts of the new Open-PSA format.

III. S2ML + Boolean Equations

A. *System Structure Modeling Language (S2ML) in a glance*

The new version of Open-PSA format is based on S2ML. S2ML stands for System Structure Modeling Language [6]. It unifies concepts to structure models coming from object-oriented and prototype-oriented programming languages.

The basic structural construct in S2ML is a block, also called a prototype. A block is a container for variables, parameters and all the other modeling artifacts. The simplest structuring relation is the composition. A block may be composed of several other blocks. Classical safety analysis formalisms, such as fault trees and reliability block diagrams, use only blocks and composition for structuring models.

In order to be able to reuse blocks, structured programming languages introduce the notions of class and instantiation of classes. A class is a reusable "on-the-shelf" block, which is stored in a library and can be reused everywhere in the model via instantiation.

In some cases, it is necessary to modify or to extend a modeling unit (a class or a block) without instantiation. It can be achieved via inheritance relation introduced in object-oriented programming languages. If a modeling unit A inherits from a modeling unit B, then A contains all the characteristics of B and adds some new characteristics.

There are cases where the same component is used in several places or to contribute to different functions of the system. In other words, a modeling unit is shared between several other modeling units. This kind of "uses" relation between modeling units is called aggregation.

In object-oriented programming languages, the reuse of modeling units is done by means of instantiation of classes. In modeling languages using only blocks (called prototype-oriented languages), the reuse of blocks is also possible. It is achieved via the notion of cloning. If a block A is a clone of a block B, then the block A has exactly the same characteristics as the block B.

To summarize, there are the following constructs to organize and structure models:

- Two types of modeling units: block and class;
- Three structural relations: composition, inheritance and aggregation; and
- Two mechanisms making possible to reuse modeling elements: prototype/cloning and class/instantiation.

These constructs originate from programming languages (see TABLE I.).

TABLE I. Structuring constructs

Structuring paradigm	Structural constructs	Formalisms
Block diagrams	Blocks + composition	Fault Trees, Reliability Block Diagrams
Structured programming	Classes + composition	
Object-oriented programming	Classes + composition + inheritance	
Prototype-oriented and object-oriented programming	Blocks + Classes + composition + inheritance + aggregation + cloning	S2ML, AltaRica 3.0, new Open-PSA format

In the sequel, we show how these concepts are introduced in the new version of Open-PSA format. Their textual form will be presented; nevertheless their dual XML form also exists, as for the previous version of Open-PSA.

B. *Basic components of Open-PSA models*

Basic components of Open-PSA models are:

- Blocks that contain declarations of other objects of the model;
- Declarations of **states**, representing basic events;
- Declarations of **flows**, representing intermediate events;
- Declarations of **sources**, representing house-events;

- Finally declarations of **parameters** of probability distributions.

1) Blocks

Blocks are basic containers of the Open-PSA modeling language. They are prototypes. Blocks contain declarations of parameters, states, flows and sources (and other modeling elements described later).

A block declaration always starts with the keyword "**block**", followed by the name of the block. It finishes with the keyword "**end**". See, for example, Fig. 2.

Note that, within a block, all modeling elements must have a different name, even though they are of different types (for example, a state and a parameter). Elements can be declared in any order.

2) States and parameters

States are Boolean variables. They play the role of basic events of fault trees and are associated with a probability distribution. States are declared one at a time, even though two states have the same probability distribution. The declaration of states starts with a keyword "**state**", followed by the name of the state. Then comes the sign "=", followed by the probability distribution. The declaration ends with ";".

Probability distributions associated with states (basic events) may contain parameters. Parameters are real-valued variables. Declarations of parameters are similar to those of states, except they start with the keyword "**parameter**".

```
1  block Valve
2   state failed = 0.001 ;
3   state stuck = exponential lambda ;
4
5   parameter lambda = 0.0001 ;
6   // …
7  end
```

Fig. 2. Example of a block declaration.

In the example given Fig. 2, the block "**Valve**" contains the declarations of two basic events and a parameter. The basic event "**failed**" (defined line 2) has a constant probability distribution equal to 0.001. The basic event "**stuck**" (defined line 3) obeys the exponential probability distribution with a parameter "lambda" defined line 5. The parameter "lambda" equals to 0.0001.

Note that a parameter can be used in several probability distributions and therefore is shared by several state variables.

3) Probability distributions

Probability distributions are defined by stochastic expressions. Stochastic expressions are arithmetic expressions involving special operators (built-ins) to represent the most popular probability distributions. They may depend on parameters, themselves defined by stochastic expressions.

The following stochastic expressions are available:

- Floating point numbers;
- Parameters;
- Special distributions (exponential, Weibull, periodic).

4) Flows

Flows are Boolean variables. They play the role of intermediate events of fault trees and are associated with Boolean formulae. Like states and parameters, flows are declared one at a time. The declaration of flows starts with the keyword "**flow**", followed by the name of the flow. Then comes the sign "=" followed by the Boolean formula (its definition). The declaration ends with ";".

A flow depends on the variables that occur in its definition.

The following Boolean formulae are considered:

- References to another variable;
- Constants **true** and **false**;
- Operators (for example, **or**, **and**, **atleast**, **not**, etc.).

C. Structuring constructs

As we have seen earlier, classical safety analysis formalisms, such as fault trees and reliability block diagrams only use blocks and composition as structural constructs. S2ML provides more structural constructs. We show how they can be used in the context of the new Open-PSA format.

1) Composition

A block is a container for flows, connections and other blocks. Each block is a prototype: it has a unique occurrence in the model.

```
1   block SIS
2    block InflowPipe
3     /* body of the block InflowPipe */
4    end
5    block Tank
6     /* body of the block Tank */
7    end
8    block ControlRoom
9     /* body of the block ControlRoom */
10   end
11   block OutflowPipe
12    /* body of the block OutflowPipe */
13   end
14   block DischargePipe
15    /* body of the block DischargePipe
16  */
17   end
18  end
```

Fig. 3. Example of composition.

In the example given Fig. 3 the block "**SIS**" (representing the Safety Instrumented System described in section II) is composed of blocks "**InflowPipe**", "**Tank**", **ControlRoom**", "**OutflowPipe**" and "**DischargePipe**".

2) Cloning

A system may contain similar components, for example sensors or valves. A first way to avoid duplicating the description of a block consists in cloning an already existing block.

In the example given Fig. 4 the block "**Tank**" contains two blocks representing the sensors "**LS1**" and "**LS2**". The block "**LS1**" is declared lines 2-5. It contains a state variable (basic event) "**failed**" with a constant probability distribution and a flow (intermediate event) "**out**" equal to "**failed**". The block "**LS2**" is a clone of the block "**LS1**" (declared lines 6-7). It

has exactly the same structure as "**LS1**", except for the basic event probability, which is different (it is declared line 7).

```
1  block Tank
2   block LS1
3    state failed = 0.0001;
4    flow out = failed;
5   end
6   clones LS1 as LS2
7    state failed = 0.002;
8   end
9  end
```

Fig. 4. Example of cloning.

3) Classes and instances

A second way to avoid duplicating the description of a block consists in declaring a model of the duplicated block in a separate modeling entity, called class, and then in instantiating this class in the model.

In the example given Fig. 5, the class "**Sensor**" is defined lines 1-4. It contains a state variable (basic event) "**failed**" with a probability equal to 0.0001 and a flow (intermediate event) "**out**" equal to "**failed**". Inside the block "**Tank**" the class "**Sensor**" is instantiated twice to create "**LS1**" and "**LS2**" (see lines 7-8).

```
1  class Sensor
2    state failed = 0.0001;
3    flow out = failed;
4  end
5
6  block Tank
7    Sensor LS1;
8    Sensor LS2;
9  end
```

Fig. 5. Example of instantiation of classes.

4) Inheritance

Aside the composition, the object-oriented paradigm provides also an inheritance mechanism. A class or a block can inherit the content of another class (or another block in the same modeling entity).

In the example given Fig. 6, the class "**BasicComponent**" is defined lines 1-3. It contains only a state variable (basic event) "**failed**". The class "**Valve**" inherits from the class "**BasicComponent**" (declaration line 6). It means that the class "**Valve**" contains all the elements of the class "**BasicComponent**" (the state variable "**failed**" with its probability) and adds new elements: two flows (intermediate events) "**in**" and "**out**" with their definitions (see lines 7-8).

```
1  class BasicComponent
2    state failed = 0.0001;
3  end
4
5  class Valve
6    extends BasicComponent;
7    flow in = false;
8    flow out = in and (not failed);
9  end
```

Fig. 6. Example of inheritance.

5) Aggregation

To represent the fact that the same component is used in several places or to contribute to different functions of the system, the aggregation is used. It is introduced by the keyword "**embeds**", followed by a path to the element to aggregate, followed by the keyword "as" and the name (alias) of the element. An example is given in section IV.D.

IV. Architecture views

System architecture is an emerging discipline that 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. For example, the CESAMES method for systems architecting [8] considers three different abstraction levels of a system: the operational level, the functional level and the physical level (see Fig. 7).

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.

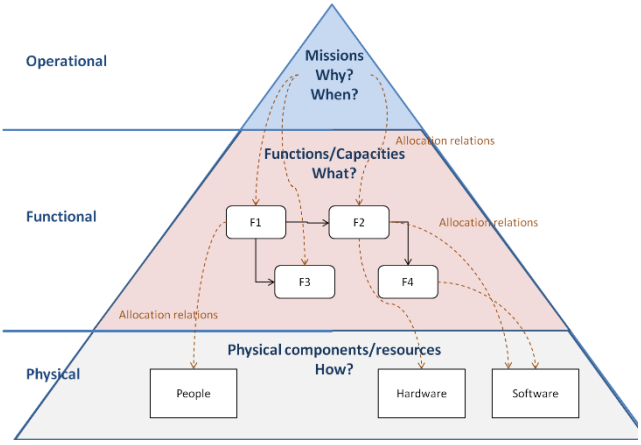

Fig. 7. CESAM system architecture pattern.

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.

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.

In the following, we use the CESAMES architecture framework to create safety models of the case study presented in section II with the new Open-PSA format.

A. Functional architecture

Fig. 8 shows the functional architecture of the overflow protection system presented in Section II. Functions are decomposed into sub-functions. These sub-functions are themselves decomposed into sub-sub-functions and so on until the suitable granularity is reached. The result is a tree-like structure. However, some nodes may be shared by several branches, for example to represent support functions like power supply.

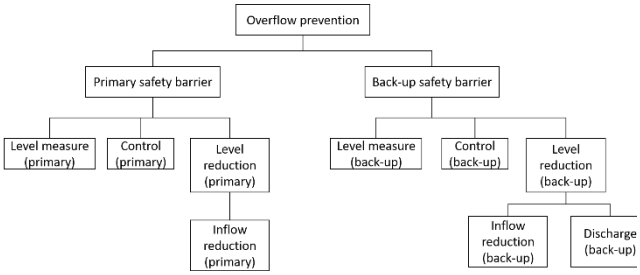

Fig. 8. Functional architecture of the overflow protection system.

B. Physical architecture

Fig. 9 shows a physical architecture of the overflow protection system. The system is decomposed into sub-systems. These sub-systems are themselves decomposed into sub-sub-systems and so on until the suitable granularity is reached.

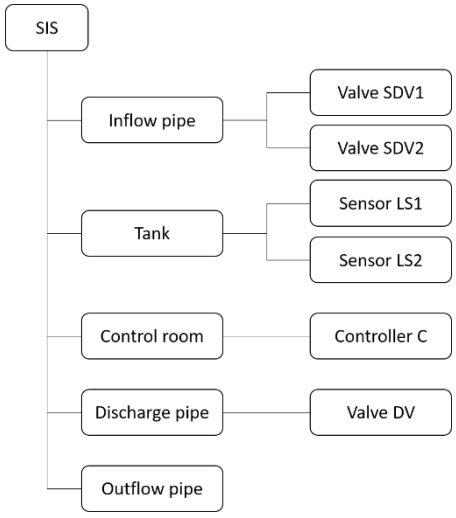

Fig. 9. Overflow protection system physical architecture**.**

Note that several physical decompositions of the system may be possible, depending on the point of view. In this article, we are interested in the safety analysis of the studied system.

C. Allocation of functions to components

TABLE II. shows the allocation of functions to physical components for the overflow protection system.

TABLE II. OVERFLOW PROTECTION SYSTEM: ALLOCATION OF FUNCTIONS TO PHYSICAL COMPONENTS

Functions			**Components**
Primary safety barrier	Level measure		Sensor LS1
	Control		Controller C
	Level reduction	Inflow reduction	Valve SDV1
Back-up safety barrier	Level measure		Sensor LS2
	Control		Controller C
	Level reduction	Inflow reduction	Valve SDV2
		Discharge	Valve DV

D. Open-PSA model of the overflow protection system

First, we represent the physical architecture of the system under study. Its safety model is given Fig. 10. The block "**SIS**" (Safety Instrumented System) is composed of 5 blocks "**InflowPipe**", "**Tank**", "**ControlRoom**", "**OutflowPipe**" and "**DischargePipe**". The block "**InflowPipe**" is composed of two instances of the class "**Valve**", defined Fig. 6. The block "**Tank**" contains two instances of the class "**Sensor**" defined Fig. 5. The block "**ControlRoom**" contains a block "**Controller**" defined lines 9-14. The block "**DischargePipe**" contains an instance of the class "**Valve**" named "**DV**".

Note that the model given Fig. 10 has the same decomposition as the physical architecture depicted Fig. 9.

```
1  block SIS
2   block InflowPipe
3    Valve SDV1, SDV2;
4   end
5   block Tank
6    Sensor LS1, LS2;
7   end
8   block ControlRoom
9    block Controller
10     state failed = exponential lambda;
11     parameter lambda = 0.0001;
12     flow in = false;
13     flow out = in and (not failed);
14   end
15  end
16  block OutflowPipe
17  end
18  block DischargePipe
19   Valve DV;
20  end
21 end
```

Fig. 10. Safety model: physical architecture view.

Second, a safety model of the functional architecture is defined. Fig. 11 represents the functional view of the safety model of the overflow protection system. The block "**OverflowPrevention**" contains two blocks "**PrimarySafetyBarrier**" and "**BackupSafetyBarrier**" and a flow (intermediate event) "**lost**", which represents the loss of the capacity to prevent the overflow. The value of "**lost**" is true when both "**PrimarySafetyBarrier**" and "**BackupSafetyBarrier**" are lost.

```
1  block OverflowPrevention
2   block PrimarySafetyBarrier
3    /* … */
4    flow lost = …;
5   end
6   block BackupSafetyBarrier
7    /*… */
8    flow lost = …;
9   end
10  flow lost = PrimarySafetyBarrier.lost
11           or BackupSafetyBarrier.lost;
12 end
```

Fig. 11. Safety model: functional architecture view.

The link between functional and physical views of the overflow protection system is shown Fig. 12. The block "**OverflowProtectionSystem**" contains two blocks: "**SIS**" given Fig. 10, and "**OverflowPrevention**", representing the functional view of the system. As explained earlier, this function is decomposed in two functions. The model of the block "**PrimarySafetyBarrier**" is given. The model of the block "**BackupSafetyBarrier**" is quite similar.

```
1  block OverflowProtectionSystem
2   block SIS
3   /* body of the block SIS,
4      see Fig. 10 */
5   end
6
7   block OverflowPrevention
8    block PrimarySafetyBarrier
9
10    block LevelMeasure
11     embeds main.SIS.Tank.LS1 as S;
12     flow lost = S.failed;
13    end
14
15    block Control
16     embeds main.SIS.ControlRoom.C
17           as C;
18     flow lost = C.failed;
19    end
20
21    block LevelReduction
22     embeds main.SIS.InFlowPipe.SDV1
23            as A;
24     flow lost = A.failed;
25    end
26
27    flow lost = S.lost or C.lost
28                      or A.lost;
29   end
30   /* The remainder of the block
31      OverflowPrevention */
32  end
33 end
```

Fig. 12. Safety model: example of allocation between functions and physical components.

The block "**PrimarySafetyBarrier**" is composed of three functions: "**LevelMeasure**", "**Control**" and "**LevelReduction**". Each function contains a flow (intermediate event) named "**lost**". Moreover, each function aggregates components allocated to this function (see TABLE II.). For example, the block "**LevelMeasure**" aggregates the class instance "**LS1**" (sensor), renames it and uses it to calculate the value of its flow "**lost**". The function is lost when its allocated physical component is failed. The function may be allocated to several physical components, in this case the calculation of "**lost**" may be more complex.

In addition, note that the component "**Controller**" is used by two functions: "**PrimarySafetyBarrier.Control**" and "**BackupSafetyBarrier.Control**".

The safety model of the overflow protection system presented in this section combines two architecture views: the functional and the physical. Thanks to advanced structural constructs, it can be easily represented with the new version of Open-PSA format.

V. Fault Tree Model

The overflow protection system presented in Section II, can be modeled by a fault tree, depicted Fig. 13.

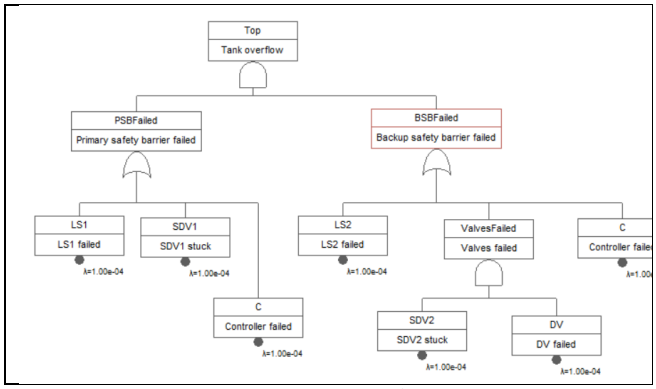

Fig. 13. Fault tree representing the overflow protection system.

This fault tree, represented within the new Open-PSA format, is given Fig. 14.

```
1  block OverflowProtectionSystemFT
2  /* Declarations of Basic events */
3   state LS1Failed = exponential lambda;
4   state LS2Failed = exponential lambda;
5   state ControllerFailed =
6                     exponential lambda;
7   state SDV2Failed =
8                     exponential lambda;
9   state SDV1Failed =
10                    exponential lambda;
11  state DVFailed =
12                    exponential lambda;
13 /* Declarations of parameters */
14  parameter lambda = 0.0001;
15 /* Declarations of intermediate
16    events */
17  flow Top = PSBFailed and BSBFailed;
18  flow PSBFailed = LS1Failed
19                  or SDV1Failed;
20  flow BSBFailed = LS2Failed
21           or ValvesFailed or CFailed;
22  flow ValvesFailed = SDV2Failed
23                    and DVFailed;
24 end
```

Fig. 14. Fault tree representing the overflow protection system in Open-PSA format.

Lines 3-12 define basic events of the model (introduced by the keyword "**state**"). They are associated with exponential

probability distribution with the failure rate defined by a parameter lambda, defined line 14. The parameter lambda is shared by all the basic events.

The top event of the fault tree is defined line 17. It is introduced by the keyword "**flow**", followed by its name "**Top**". It is defined as an "and" gate between two intermediate events "**PSBFailed**", defined line 18, and "**BSBFailed**", defined line 20.

VI. Reliability Block Diagram Model

The overflow protection system described in Section II can be also represented by a reliability block diagram depicted Fig. 15. In this reliability block diagram:

- There are a source flow S and a target flow T.
- Each physical component of the system is represented by a block (for example LS1, LS2, SDV1).
- The primary and backup safety barriers are connected in parallel.
- Components of each safety barrier are connected in series, except the valves SDV2 and DV, that are connected in parallel.
- The controller is connected in series with the safety barriers as it is shared by both of them.

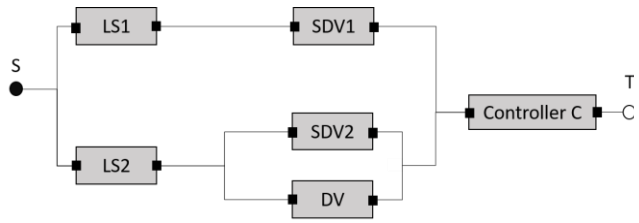

Fig. 15. Reliability Block Diagram representing the overflow protection system.

This reliability block diagram can be represented using the new Open-PSA format in the following way. First, let's define a class to represent each basic block of the diagram (see Fig. 16).

```
1  class BasicBlock
2   state failed = exponential lambda ;
3   parameter lambda = 0.0001 ;
4
5   flow in = false ;
6   flow out = (not failed) and in ;
7  end
```

Fig. 16. Basic Block of RBD in new Open PSA format.

In this class we define a state variable "failed" (line 2), which is associated with an exponential probability distribution with a parameter lambda defined line 3. Two flows are also defined "in" and "out" (lines 5-6).

Then we use this class to define the reliability block diagram, given Fig. 17. In this reliability block diagram, first, all the blocks are defined (instances of the class "**BasicBlock**", see line 3). Second, flows (intermediate events) are connected together, which redefines their definitions.

```
1  block OverflowProtectionSystemRBD
2  /* Declaration of blocks */
3  BasicBlock LS1, LS2, SDV1, SDV2,
4             DV, C;
5  /* Declaration of connections
6     between blocks */
7   flow S = true;
8   LS1.in = S;
9   LS2.in = S;
10  SDV1.in = LS1.out;
11  SDV2.in = LS2.out;
12  DV.in = LS2.out;
13  C.in = SDV1.out
14         or (SDV2.out or DV.out);
15  flow T = C.out;
16 end
```

Fig. 17. RBD of the overflow protection system in new Open-PSA format.

VII. Conclusion

This communication presents the underlying principles of the new version of Open-PSA model exchange format, which relies on the S2ML+X paradigm. This new version improves very significantly the previous one. A case study is used to illustrate different concepts of the new Open-PSA format. The new Open-PSA format can be used to represent classical safety assessment formalisms, such as Fault Trees and Reliability Block Diagrams, and also to create models close to system functional and physical architecture. Both textual and XML forms of this new version of the Open-PSA format are available.

References

[1] Steven Epstein, Olivier Nusbaumer, Antoine Rauzy and Donald Wakefield. A Modest Proposal: A Standard PSA Model Representation Format. Proceedings of the conference Nuclear Energy for New Europe, 2007. I. Jencic and M. Lenosek Ed.. INIS. ISBN 978-961-6207-28-7. Portoroz, Slovenia. 2007.

[2] Steven Epstein, Mark Reinhart and Antoine Rauzy. Validation Project for the Open-PSA Model Exchange using RiskSpectrum and CAFTA. Proceeding of the PSAM'10 Conference. B.P. Hallbert Ed..ISBN 9781622765782. Seatle, USA. June, 2010. (also in E. Fadier ed., Actes du congrès LambdaMu'10)..

[3] Emmanuel Clément, Thierry Thomas and Antoine Rauzy. Arbre Analyste : un outil d'arbres de défaillances respectant le standard Open-PSA et utilisant le moteur XFTA. Actes du congrès Lambda-Mu 19 (actes électroniques). Institut pour la Maîtrise des Risques. ISBN 978-2-35147-037-4. Dijon, France. October, 2014.

[4] Mohamed Hibti, Thomas Friedlhuber and Antoine Rauzy. Overview of The Open PSA Platform. Proceedings of International Joint Conference PSAM'11/ESREL'12. Reino Virolainen Ed.. ISBN 978-162276436-5. pp. 2798–2807. June, 2012.

[5] Antoine Rauzy and Cecilia Haskins. Foundations for Model-Based Systems Engineering and Model-Based Safety Assessment. Journal of Systems Engineering. Wiley Online Library. 2018. doi:10.1002/sys.21469.

[6] Michel Batteux, Tatiana Prosvirnova and Antoine Rauzy. From Models of Structures to Structures of Models. IEEE International Symposium on Systems Engineering (ISSE 2018). IEEE. Roma, Italy. October, 2018. doi:10.1109/SysEng.2018.8544424. Best paper award.

[7] Michel Batteux, Tatiana Prosvirnova and Antoine Rauzy. Altarica 3.0 in 10 modeling patterns. International Journal of Critical Computer-Based Systems (IJCCBS), 2018.

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