A Meta-model for Integrating Safety Concerns into Systems Engineering Processes

Pierre-Yves Piriou, Jean-Marc Faure, Gilles Deleuze

A Meta-model for Integrating Safety Concerns into Systems Engineering Processes

Pierre-Yves Piriou, Jean-Marc Faure
LURPA, ENS Cachan
63 avenue du Président Wilson
94235 Cachan cedex, France
Email: {pierre-yves.piriou; jean-marc.faure}
@lurpa.ens-cachan.fr

Gilles Deleuze
EDF R&D
1 avenue du Général de Gaulle
92140 Clamart, France
Email: gilles.deleuze@edf.fr

***Abstract*—In order to overcome the increasing difficulty of critical systems development, integrating the safety concerns into Systems Engineering processes seems to be the relevant solution. This paper proposes a meta-model to perform this integration by considering phased mission systems composed of repairable components. This kind of system requires in particular that several redundancy policies be defined. The benefits of this contribution are illustrated on a small example from the domain of electric power production.**

I. Introduction

The System Engineering (SE) approach offers relevant solutions for formalizing and comprehending complex systems. But it usually focuses on the normal operation of the system, whereas for critical systems, safety matters too. Efforts are already produced in order to link functional and dysfunctional analyzes, but a wide gap persists between SE and Safety Analysis. Moreover, to comply with the continuous improvement of the safety-critical systems, more and more sophisticated dysfunctional studies need to be achieved and SE processes have to be applied. Then as to bridge this gap, notions must be defined or refined in the SE processes. Taking advantages of these notions besides requires to build models combining both the functional and dysfunctional aspects. Moreover, safety/dependability studies assume very often that the system which is analyzed is single phased. This is no more true for numerous critical systems of different industrial domains, like aerospace, chemical processes, communication networks, transportation, power production and distribution , etc). This explains why this paper considers phased mission systems with repairable components. As depicted on the figure 1, the main purpose of this study is to propose a meta-model that allows safety analyzes for that kind of system be integrated into a SE process.

The second section of this paper discuss related works. The construction of the meta-model is addressed in the third section. In this paper, every meta-model is represented using UML class diagrams and OCL constraints. The meta-model proposed is instantiated in section IV on a small, but with several mission phases example: a part of water level control of the steam generator of a nuclear plant. Finally, conclusions and outlooks are presented.

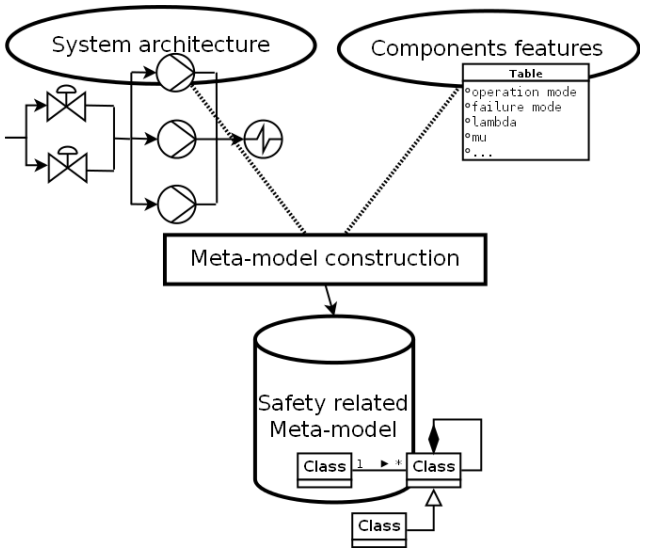

Fig. 1. Contribution of the paper

II. Related works

This section focuses on previous works that proposed solutions to integrate safety concerns in SE processes.

In the field of SE processes, the best practices are mattered by a wide theoretical and technical documentation. The meta-model built in this paper is an extension of a SE knowledge meta-model defined in [1]. This meta-model is designed to be an aid for making models compliantly with the SE processes suggested by the International Council on Systems Engineering (INCOSE). Due to space limitations, it is not possible to show completely this meta-model but a part is depicted by the figure 2.

In their paper [2], R. Guillerm and al. describe a method for declining safety requirements of complex systems. The refinement of the requirement notion for treating the safety ones is a necessary step for achieving the safety integration. In our approach, we assume that this issue is already solved.

SOPHIA [3] is a UML profile for integrated some safety notions in SE processes. This language enables to perform risk analyzes and define automatically some safety attributes (for instance the Safety Integrity Level). Once again, our approach

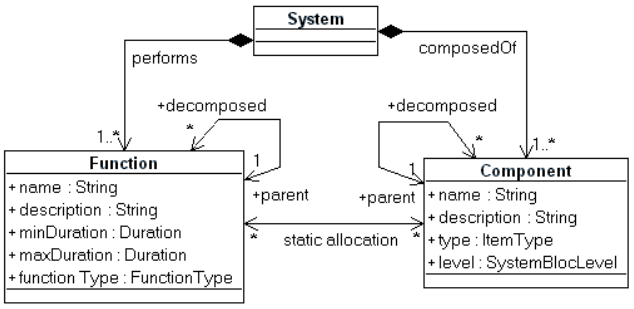

Fig. 2. Part of the meta-model defined in [1]

considers this issue already solved.

The Dysfunctional behavior Data Base defined by David, P. and al. in [4] allows to model relevantly the dysfunctional behavior through a clever refinement of the failure mode notion.

The main benefits of these four contributions are summarized in Table I. Nevertheless, none of them considers redundancy policies or phased mission systems. The aim of this paper is to fill these gaps.

TABLE I
COMPARATIVE STUDY

	a)	b)	c)	d)	e)	f)
Pfister [1]	✔	-	-	-	-	-
Guillerm [2]	✔	✔	✔	-	-	-
Cancila [3]	✔	-	✔	-	-	-
David [4]	✔	✔	-	✔	-	-
Our approach	✔	-	-	✔	✔	✔

- a) To take into account the SE processes.
- b) To refine the requirement notion for safety ones.
- c) To define notions in order to make risk analysis.
- d) To allow a realistic failure/repair scenario modeling.
- e) To consider phased mission systems.
- f) To define a redundancy policy.

III. META-MODEL CONSTRUCTION

To address the problem of safety studies integration into Systems Engineering process, this paper proposes to extend the meta-model defined in [1]. The templates designed according this meta-model are compliant with SE processes. The following meta-model adds to the first the semantics required to perform safety studies on phased mission system. It is completed by a list of modeling constraints and definitions. Those adds are expressed in natural language and in OCL (Object Constraint Language [5]).

A. Assumptions and definitions

According to [6], safety is the aptitude of an entity to satisfy one or several aimed functions in given conditions. A safety study is based on observation of the function dysfunctional status. Moreover, the components of a system produce effects on the functions. Those depend on the conditions and mainly their state. Then in order to determine if a function is satisfied or not, it is necessary to know whether the components are able to perform the function in their current state. Let us assume that the achievement of a function is quantifiable, i.e. it is possible to express that a function is achieved at x%, and that a component in a given state performs a function according to an achievement rate.

The redundancy of components is widely used to enhance the safety of a system. In that case, the designer has to define a redundancy policy in order to manage on-line the allocation of the functions to its components. Then a redundancy policy is defined for a function and a set of components, and it consists in determining the conditions for which each component has to perform the function.

The following class descriptions are voluntarily minimal for being as generalist as possible. Each class may be refined for containing semantics more operational.

For more convenience, the instances of the following classes are designated by their attribute $name$.

B. Definition of a mission phase

A phased mission system is characterized by several phases, where the system structure, failure and recovery processes, or success criteria can change with each phase ([7] and [8]). Then the components and functions are not used similarly across the mission phases. The phase determines how the components should be used to perform the functions that have to be achieved in this phase.

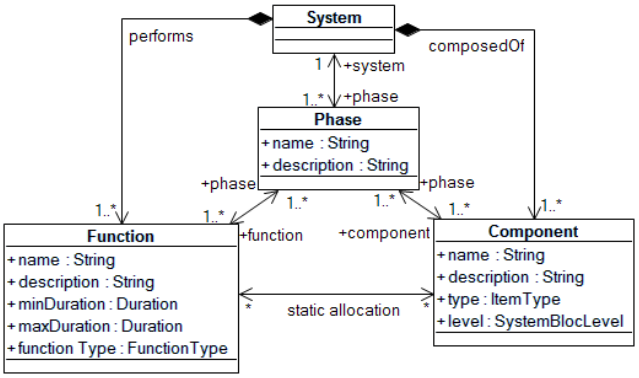

Fig. 3. Step 1: definition of the system phases

The figure 3 depicts this notion of mission phase which impacts the entire modeling.

C. Definition of a component state

Each component can run and fail according to several operation and failure modes designated by character strings. Those modes represent respectively the functional and dysfunctional properties of the component. For instance, at least one of each mode is always defined by assuming that each component can be disabled and can be non-faulty (constraint 1).

Constraint 1: Every component must have at least one operation mode called OFF and one failure mode called OK.

context Component **inv**:
self.operation mode −>one (om | om.name = 'OFF')
and self.failure mode −>one(fm | fm.name = 'OK')

A component state is a pair built with one operation mode and one failure mode. As depicted on figure 4, the possible states of a component are defined by instantiating its failure modes and operation modes.

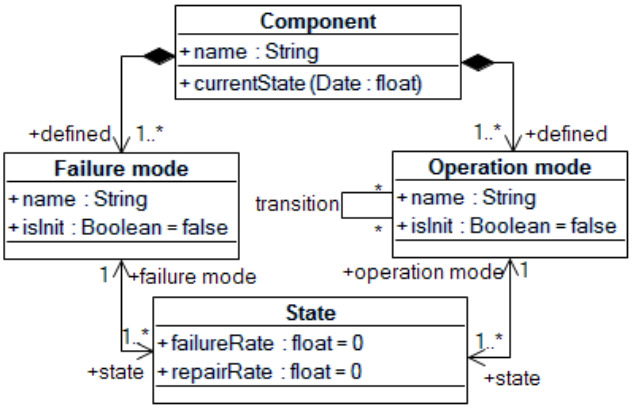

Fig. 4. Step 2: definition of the components states

The deterministic behavior of a component results in updating its current operation mode by means of a transition to be determined. But for specifying its stochastic behavior, the designer has to describe its possible failure/repair scenarios. The notion of behavior leads to the one of initial state. The constraint 2 guarantees the uniqueness of the initial state.

Constraint 2: Every component must have a unique initial state.

context Component **inv**:
self.operation mode −>one(om: Operation mode | om.isInit = True) and self.failure mode −>one(fm: Failure mode | fm.isInit = True)

The failure behavior of a component (into the set of its failure modes) depends on its behavior into the set of its operation modes. Considering this fact, the figure 4 shows that the proposed meta-model allows a free definition of failure/repair scenarios. Such scenarios are expressed as in the first statement.

Statement 1: Let C be a component, OK be its "non-faulty" failure mode, and (om,fm) be a state of C where the attributes $failureRate$ and $repairRate$ are not null:

- If C is in the state (om,OK), then it can fail according to the failure mode fm. That means the transition from (om,OK) to (om,fm) is possible with the failure rate specified.
- If C is in the state (om,fm), then it can be repaired. That means the transition from (om,fm) to (om,OK) is possible with the repair rate specified.

The components are considered in the horizontal part of the bathtub curve. Then the failure rate does not depend on time but is defined by om and fm. In the same way, if a component can be repaired in its current state, the repair rate is not null and depends only on the current state.

If one of those attributes is null, the corresponding transition cannot be fired. For example, if a failed component is repairable only when it is disabled (state (OFF,fm)), then this state among those constructed with fm, is the only one from which the attribute $repairRate$ is not null.

D. Definition of the effect of a component on a function

A function is statically allocated to some components which impact its achievement. Those effects depend on the component current state. An instance of effect is built for each couple (state, function), from which every term is linked to the considered component (see the constraint 3).

Constraint 3: Every possible state of a component leads to a unique effect on each function statically allocated to this component.

context Component **inv**:
self.function −>forAll(f: Function |
self.operation mode.state −>forAll(s: State |
s.effect −>including(f.effect) −>size() = 1)))

As depicted on figure 5, the effect attributes are $achievementRate$ and $isUnacceptable$. The first allows to determine the function achievement rate that the component performs when it is in the considered state. The second allows to determine the forbidden states by taking into account the safety (the constraint 4 fixes particular non-forbidden states).

Constraint 4: A disabled component (operation mode OFF) or non-failed (failure mode OK), does not have an unacceptable effect on a function.

context Effect **inv**:
self.state.operation mode.name = 'OFF'
or self.state.failure mode.name = 'OK'
implies self.isUnacceptable = False

The attribute $goal$ of the class $Function$ is the achievement threshold that the components have to reach. Then at a given time, a function is satisfied if and only if the sum of the achievement rates of the components, for which the function is dynamically allocated to them, in their current state, is greater than $goal$. The definition 1), makes explicit this test.

Definition 1: This definition gives the result of the operation $isSatisfy()$.

context Function::isSatisfy(Date):boolean **body**:
self.effect −>select(e:Effect |
e.state.operation mode.component.currentState(Date) =
e.state and self.allocation −>includes(
e.state.operation mode.component)).achievementRate
−>sum() ≥ self.goal

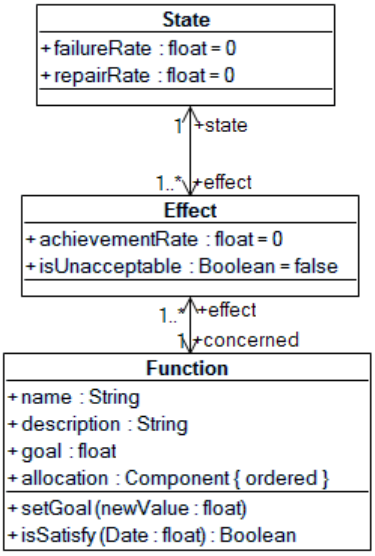

Fig. 5. Step 3: definition of the effects

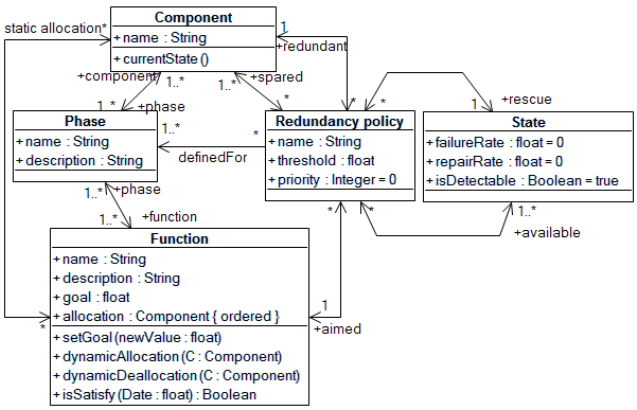

Fig. 6. Step 4: definition of the redundancy policies

E. Definition of a redundancy policy

The redundancy policies describe how to update the functions allocation to the components considering the system phase and the failure/repair events. The methods $dynamicAllocation$ and $dynamicDeallocation$ are used for updating the attribute $allocation$ of the class $Function$. If a component is able to perform a function in some conditions, then the function can be statically allocated to the component. If those conditions are verified, then the function is allocated to it dynamically.

Let us assume that a redundancy policy can ever be expressed by the following statement:

Statement 2: If the set of component E_c does not perform fittingly the function F during the phase P, and if the component C is available (i.e. its current state is in the set E_s), then C has to be powered on the state S for participating in the achievement of F.

Each component of the set E_c performs the function F according to an achievement rate. The set E_c "performs fittingly the function F" if the sum of those rates in their current state is greater than a threshold t to be determined.

In comparison to the meta-model (see the class *Redundancy policy* on figure 6), the $spared$ components correspond to the set E_c, the $aimed$ function is F, the phase for which the policy is defined is P, the $redundant$ component is C, the $available$ states correspond to E_s, the $rescue$ state is s, and the attribute $threshold$ is t. A redundancy policy is not directly linked to the failures, because only the achievement of the function is important. Nevertheless, the knowledge of the components current states is required to determine if they "perform fittingly the function F". The designer can define a priority rule for avoiding the redundancy policies competition.

Furthermore, a redundancy policy must respect the two following constraints (5 and 6).

Constraint 5: The states linked to a redundancy policy (available and rescue) must be linked to the redundant component of this redundancy policy.

context Redundancy policy **inv**:
self.redundant.operation mode.state $->$includesAll (self.available $->$including(self.rescue))

Constraint 6: A redundant component must be able to achieve fittingly the aimed function.

context Redundancy policy **inv**:
self.rescue.effect $->$includes(e:Effect | e.function = self.aimed).achievementRate $\geq$ self.threshold

F. Synthesis and limitation of modeling

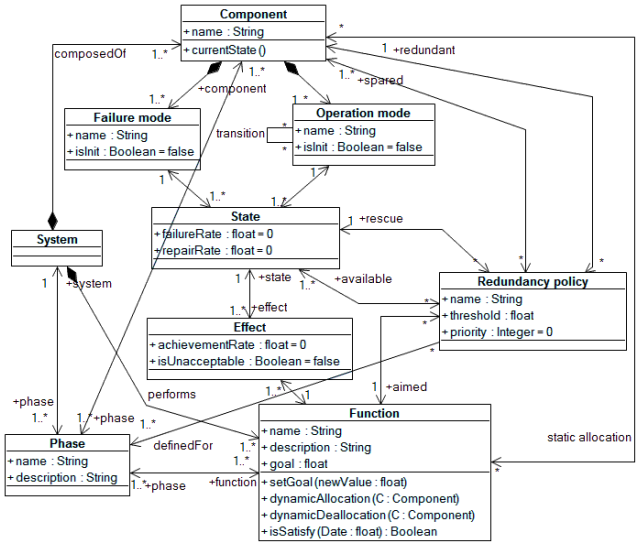

Fig. 7. Complete Meta-model for the integration of safety analyzes into SE processes

At this step, some limitations of the power of expression

can be identified. Indeed, the proposed semantics do not allow to easily express the following notions: failure mode degradation, failure propagation and common cause failure. The link called *transition* (on the figure 7) is designed to model the deterministic change of the operation mode of a component without modification of its failure mode. But nothing enables to give a better description of this transition (guarding, delaying, ...). Then this meta-model is relevant only if it is used in addition to a dynamical formal model to design the control, the preventive maintenance policy, etc...

IV. Example

In this part the meta-model is instantiated in order to model a simple sub-system of two pumps. It is a part of a water level control system of the steam generator in a nuclear power plant [9]. This system is used in [10] for modeling dynamic reliability.

The considered sub-system is made up of two feeding turbo pumps (TPA1,TPA2) which have to perform a single function F: "To supply enough water at the water flow regulation valve sub-system". The pumps may failed and be repaired, then for increasing the dependability of the function, its allocation at the two pumps must be dynamically managed by redundancy policies to be determined.

For a complete system, the meta-model proposed in part III cannot be graphically instantiated without the aid of a modeling software with an encapsulated view opportunity such as arKItect© [11]. For this simple example, the meta-model is instantiated by means of three instance diagrams completed by tables.

A. Introducing the system phases

The global system operates according to three phases (Table II). The function F is mandatory in all phases but only one pump is necessary for the first and third phase.

TABLE II
PHASES DESCRIPTION

id	role	description
$P1$	To increase the power	A single pump is able to perform fittingly the function.
$P2$	To produce energy	The two pumps have to run together to perform efficiently the function.
$P3$	To decrease the power	A single pump is able to perform fittingly the function.

B. Defining the components states with failure/repair attributes

As shown in figure 8, each pump may be powered on three operation modes: OFF (as every component), RUN and $OVERSPEED$. The initial mode of $TPA1$ is RUN and the one of $TPA2$ is OFF. The pumps are considered non-failed at the initial state, then the initial failure mode is OK. Each pump may fail according two failure modes called $LEAK$ and $RUPTURE$.

The table III gives the value of the attributes ($failureRate$, $repairRate$) of the $State$ class instances. Let us remark that

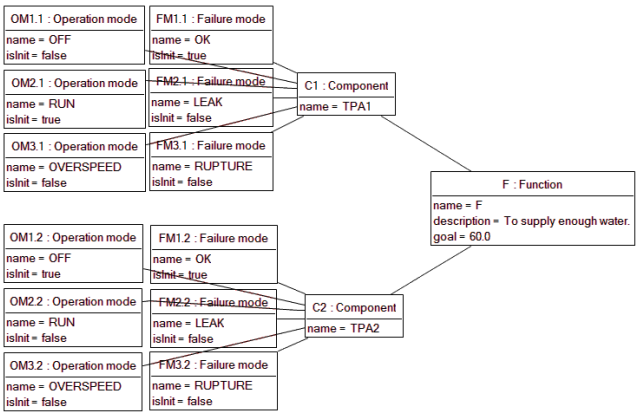

Fig. 8. First partial instance diagram from the considered sub-system

a leak can be repaired when the pump is in the operation mode RUN. The pumps are identical, then a unique table is sufficient.

TABLE III
FAILURES' FEATURES

	$LEAK$	$RUPTURE$
OFF	$(0, 0.2)$	$(0, 0.1)$
RUN	$(0.01, 0.1)$	$(0.001, 0)$
$OVERSPEED$	$(0.05, 0)$	$(0.002, 0)$

C. Describing the effects on the function

The first step consists in quantifying the achievement of the function. Addressing this task requires more specific knowledge about the system phases. During the first phase, the system increases its power until it reaches 60% of the nominal power. A single feed turbo pump is able to supply enough water for meeting this global goal. During the second phase, the system produces energy. For an efficient production, the system should work at 100% of its nominal power. Fulfilling this production expectation requires in particular two running pumps or a single over-speeding pump. But if the pumps do not succeed in that task, the system decreases its power until 60%, without changing its phase, and wait the reparation of the pumps. In that case, the production continues and the function is considered satisfied. During the third phase, the system decreases its power until return to 0% of the nominal power. As during the first phase, a single feed turbo pump is able to supply enough water for meeting this global goal. Next the achievement rates will be expressed as the rates of nominal power.

The attribute called $goal$ is the achievement rate for performing fittingly F. In this example, the goal does not depend on any parameters, then its value is constantly 60.0. Indeed, the function is satisfied if the sum of the achievement rates of the components, for which the function is dynamically allocated to them, is greater than 60.0, without consideration of the system phase.

The table IV gives the value of the achievement rate of F performed by a pump according to its state. Once again, a unique table is enough due to the uniqueness of F and the pump kind.

TABLE IV
ACHIEVEMENT RATE OF F PERFORMED BY A PUMP

	OK	$rupture$	$leak$
OFF	0	0	0
run	60	0	50
$overspeed$	100	0	80

Let us remark that, during the second phase, a pump in the state (run,$leak$) performs fittingly the function despite the failure.

D. Determining the redundancy policies

The three following statements enable to define three instances of the class *Redundancy policy*.

Statement 3: R1.1: If the set of component $\{TPA1\}$ does not perform fittingly the function F during the phase $P1$ or $P3$, and if the component $TPA2$ is available (i.e. its current state is in the set $\{OFF - OK\}$), then $TPA2$ has to be powered on the state $RUN - OK$ for participating in the achievement of F.

Statement 4: R1.2: If the set of component $\{TPA2\}$ does not perform fittingly the function F during the phase $P1$ or $P3$, and if the component $TPA1$ is available (i.e. its current state is in the set $\{OFF - OK\}$), then $TPA1$ has to be powered on the state $RUN - OK$ for participating in the achievement of F.

Statement 5: R2.1: If the set of component $\{TPA1\}$ does not perform fittingly the function F during the phase $P2$, and if the component $TPA2$ is available (i.e. its current state is in the set $\{RUN - OK\}$), then $TPA2$ has to be powered on the state $OVERSPEED - OK$ for participating in the achievement of F.

Statement 6: R2.2: If the set of component $\{TPA2\}$ does not perform fittingly the function F during the phase $P2$, and if the component $TPA1$ is available (i.e. its current state is in the set $\{RUN - OK\}$), then $TPA1$ has to be powered on the state $OVERSPEED - OK$ for participating in the achievement of F.

The value of the attribute $threshold$ is 60.0 for the first two policies, and 50.0 for the last two policies. Let us remark that, during the second phase, if the pumps fail with the failure mode $LEAK$, according to the tables IV and III, the function is fittingly achieve anyway and this failure mode can be repaired in the operation mode RUN. That explains that the leak failures don't trigger the redundancy policies 2.1 and 2.2.

Those instances of the class *Redundancy policy* are shown by the two instance diagrams on the figures 9 and 10. Each redundancy policy is linked to the function F.

E. Outcome

Hence this example shows that a template made by instantiating the meta-model proposed in this paper permits to model useful interaction. Indeed, a designer can easily models the coupling between the failure modes and the operation modes. Moreover, it is possible to define a policy redundancy observing a function for determining whether its allocation must be updated and how it must be. The system may perform several mission phases and a policy redundancy is defined for one of them.

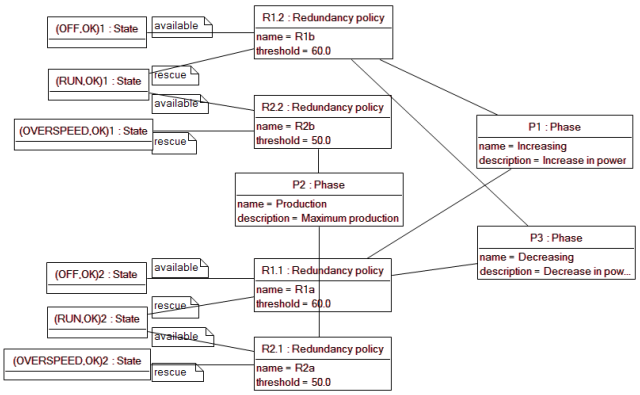

Fig. 9. Second partial instance diagram from the considered sub-system

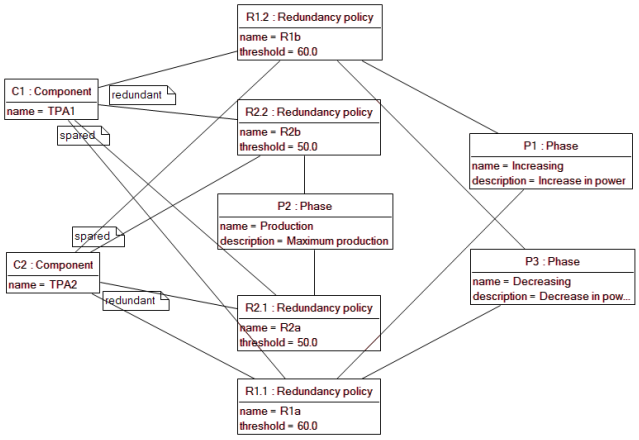

Fig. 10. Third partial instance diagram from the considered sub-system

V. CONCLUSIONS AND OUTLOOKS

The meta-model defined by Pfister and al. in [1] extended by the one proposed in this paper formalizes a framework for integrating the safety analysis into SE processes. Indeed, after making the casual processes (phased mission specification, functional architecture, physical architecture, allocation...), it is possible to describe the functional and dysfunctional features of the components. And then, the designer can model realistic failure/repair scenarios. He also can define redundancy policies for updating the dynamical allocation of functions caused by dysfunctional events, taking into account the different phases of the system. The treated example shows that this meta-model allows to integrate useful informations in the model. This meta-model was fully implemented into the modeling software tool arKItect© [11]. By way of proof of

concept, some safety calculus was successfully applied on the model of the example presented in this paper with the aid of the analysis software tool PyCATSHOO [12].

Currently we are working on an algorithm for automating the construction of a formal dynamic model from an instance of the meta-model. The utilization of this dynamic model for performing formal methods or simulations is also an ongoing work. Then the full method could be implemented with the support of a SE platform. Extending this approach for modeling dynamic reliability in order to cover a greater field in safety concerns is another way to be investigated.

REFERENCES

[1] F. Pfister, V. Chapurlat, M. Huchard, C. Nebut, and J.-L. Wippler, "A proposed meta-model for formalizing systems engineering knowledge, based on functional architectural patterns," *Systems Engineering*, vol. 15, pp. 321–332, Autumn 2012.

[2] R. Guillerm, N. Sadou, and H. Demmou, "Combining FMECA and Fault Trees for declining safety requirements of complex systems," in *ESREL 2011*, C. . G. Soares, Ed., Troyes (France), september 2011, p. 12871293.

[3] D. Cancila, F. Terrier, F. Belmonte, H. Dubois, H. Espinoza, S. Gérard, and A. Cuccuru, "Sophia : a modeling language for model-based safety engineering," in *MoDELS ACES-MB*, Denver, Colorado, USA, October, 6th 2009, pp. 11–25.

[4] P. David, V. Idasiak, and F. Kratz, "Reliability study of complex physical systems using sysml," *International Journal in Reliability Engineering and System Safety*, vol. 95, no. 4, pp. 431 – 450, 2010.

[5] OMG, *Uml 2.0 OCL specification*, Object Management Group, 2003.

[6] A. Villemeur, *Reliability, Availability, Maintainability and Safety Assessment, Methods and Techniques*. Wiley, 1992.

[7] G.-R. Burdick, J.-B. Fussell, D.-M. Rasmuson, and J.-R. Wilson, "Phased mission analysis : A review of new developments and an application," *IEEE Transactions on Reliability*, vol. R-26, pp. 43–49, April 1977.

[8] L. Meshkat, L. Xing, S. Donohue, and O. S.K., "An overview of the phase-modular fault tree approach to phased mission system analysis," in *Proceedings of the International Conference on Space Mission Challenges for Information Technology*, Pasadena, CA, USA, July 2003, p. 10.

[9] M. Kothare, B. Mettler, M. Morari, P. Bendotti, and C.-M. Falinower, "Level control in the steam generator of a nuclear power plant," in *Decision and Control, 1996, Proceedings of the 35th IEEE (10 pages)*, vol. 4, Kobe, Hyogo, Japan, December 11th-13th 1996, pp. 4851–4856.

[10] H. Zhang, B. de Saport, F. Dufoura, and G. Deleuze, "Dynamic reliability: Towards efficient simulation of the availability of a feedwater control system," in *NPIC-HMIT 2012*, San Diego, USA, July 22-26 2012.

[11] H. Aboutaleb, M. Bouali, M. Adedjouma, and E. Suomalainen, "An integrated approach to implement system engineering and safety engineering processes : Sasha project," in *ERTS2012 (6 pages)*, Toulouse, France, February 2nd 2012.

[12] H. Chraibi, "Dynamic reliability and assessment with PyCATSHOO: Application to a test case." in *PSAM (10 pages)*, Tokyo, Japan, April, 14th-18th 2013.