A Meta-model to Support the Integration of Dependability Concerns into Systems Engineering Processes: an Example from Power Production

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

A Meta-model to Support the Integration of Dependability Concerns into Systems Engineering Processes: an Example from Power Production

Pierre-Yves Piriou, Jean-Marc Faure, *Member, IEEE,* and Gilles Deleuze

***Abstract*—Systems engineering (SE) is a very promising approach to facilitate the development of complex systems. This explains why several SE processes have been already proposed. However, these proposals focus mainly on systems with faultless components. Integration of dependability concerns into SE processes must be supported by a suitable organization of the data which are dealt with during the system life-cycle. A meta-model which defines the concepts used during this cycle as well as the relations between these concepts is a way to rigorously describe this organization.**

This article proposes such a meta-model developed for power production systems. These systems are phased mission systems composed of repairable and multi-state components; moreover, several redundancy policies shall be defined for each phase. This proposal is illustrated on a small example from a power plant. Last, the merit of this contribution to support the integration of dependability concerns is shown by proposing a method to build systematically, from the instance diagrams derived from the proposed meta-model, the Markov Chains which represent the dysfunctional dynamic behavior of a system.

***Index Terms*—System Engineering, Dependability, Redundancy Policy, Phased Mission System, UML Class Diagram, Markov Chain.**

I. Introduction

In the current socio-economical context, where costs, delays and dependability are crucial concerns while promising technological solutions frequently appear, engineering of critical systems is a complex issue that must be thought in the framework of systems engineering [1] and [2]. This approach permits in particular to avoid that incorrect industrial practices, which focus on counting or managing failures instead of preventing them, as detailed in [3], are introduced during the development of the system. The numerous activities of the systems engineering process (requirements analysis, functional analysis, risk management, dependability analysis, verification and validation, etc.) must be supported nevertheless by a suitable organization of the data they produce or consume. An efficient solution to describe this organization is to develop a meta-model, in the form of a UML/SysML class diagram for instance. A meta-model defines the concepts which are used during the engineering process as well as the relations between these concepts, then ensures data consistency, and facilitates the automatic construction of some models from previously defined data. Such a model has been already proposed in [4] to support the systems engineering (SE) processes defined by the International Council on Systems Engineering (INCOSE). Unfortunately, this proposal considers only the normal, faultless, operation of the system and cannot be used as it is for critical systems.

Several worthwhile recent results may be considered to remove this limitation, however. A method to integrate two classical fault forecasting methods: Failure Mode Effects and Criticality Analysis (FMECA) and Fault Tree Analysis (FTA) in SE processes, is proposed in [5]. An UML profile, termed SOPHIA, for integrating risk analysis in these processes is described in [6] while [7] presents a framework for hazard analysis of systems of systems software. Last, the dysfunctional behavior database defined in [8] allows the dysfunctional behaviors be considered through a relevant refinement of the failure mode concept, for physical systems with non-repairable components.

Nevertheless, these valuable results are not fully appropriate to deal with the class of systems which are considered in this work, which focuses on engineering of power production systems [9], for the following reasons:

- First of all, since a power plant is built for several decades, its components must be *repairable*. This constraint is not commonly taken into account in most of dependability analysis where only non-repairable components are considered. The rare authors who discussed this issue ([10] and [11]) do not integrate their work into a SE process.
- Dependability analysis assumes very often that the objective of the system is fixed, which is no more true for power plants (and for numerous other critical systems: airplanes, chemical processes, etc.) which are *phased mission* systems [12].
- Each component can be activated with several operation modes and can fail according to several failure modes whatever its current operation mode. As the state of a component must describe both its operation and failure modes, the components of such systems are *multi-state*

P.-Y. Piriou and J.-M. Faure are with Ecole Normale Superieure of Cachan, France (e-mail: {pierre-yves.piriou; jean-marc.faure}@lurpa.ens-cachan.fr)

G. Deleuze is with Electricite de France R&D, Clamart, France (e-mail: gilles.deleuze@edf.fr)

and not merely binary components with one faultless state and one faulty state [13].

- This last feature implies that several *redundancy policies* are possible according to the current mission phase. A faulty component may be replaced by another component which was previously inactive in a given phase and by an already active component which changes its operation mode (speed increase for instance) in another phase. This non-usual kind of redundancy policy may be seen as a resilience strategy [14].

As depicted on Figure 1, the aim of this paper is to propose a meta-model that contains all the relevant concepts for the class of systems described above and that shall be connected to a meta-model to support SE processes. The result will permit to support processes where functional and dysfunctional analysis will cooperate in a seamless manner to assess dependability attributes, like reliability or unavailability.

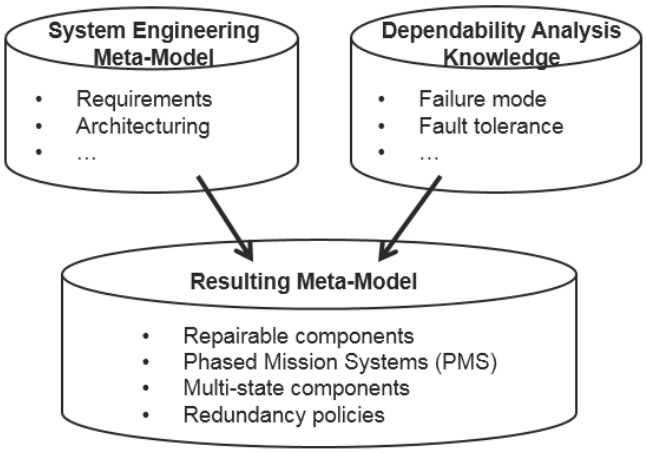

Fig. 1. Contribution of the article

The construction of this meta-model is addressed in the next section. The merit of this proposal is illustrated by instantiating, in the third section, the meta-model for a small phased mission system: a part of the water supply system of the steam generator of a power plant, and by building from this instantiated model, in the fourth section, a dysfunctional model, in the form of Markov Chains, for unavailability assessment. Finally, concluding remarks and some outlooks are drawn up.

II. Meta-model Construction

To address the problem of dependability analysis integration into Systems Engineering processes, this article proposes to extend the SE knowledge meta-model defined in [4]. This meta-model has been designed to be an aid for building models that comply with the SE processes suggested by the International Council on Systems Engineering (INCOSE). It includes several classes (Context, Need, Requirement, FunctionalArchitecture, PhysicalArchitecture, Interface, etc.) and relations that can be instantiated to describe the main features of a specific system with faultless components. Due to space limitations, it is not possible to show completely this meta-model but a part of it is depicted by Figure 2.

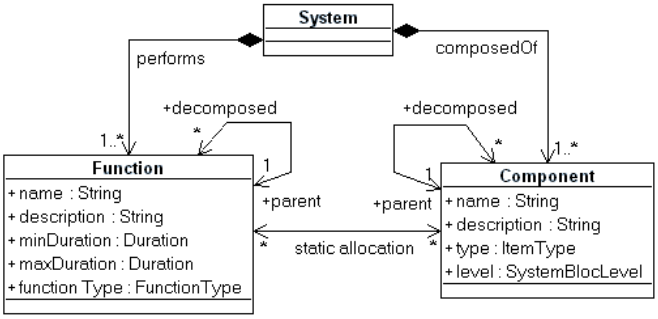

Fig. 2. Part of the meta-model defined in [4]

It expresses that a system is composed of an organized set of components and performs an organized set of functions. Functions are performed by components to which they are allocated.

The meta-model proposed in this paper is aiming at extending this result by adding the semantics required to perform dependability analysis on phased mission systems with repairable multi-state components. It is completed by a list of modeling constraints and definitions. The meta-model is represented by using UML class diagrams [15] and the modeling constraints and definitions are expressed in natural language and in OCL (Object Constraint Language [16]).

A. Modeling Phased Mission Systems

As shown at Figure 3, a phased mission system is characterized by several phases. The system structure, failure and recovery modes, or success criteria can change from one phase to another one ([12] and [17]). Components and functions are not used similarly during the different phases. Characterizing a phase consists in instantiating the links *Phase-Component* and *Phase-Function*. Indeed, these links permit to specify respectively which components must be used and which functions must be performed for each phase.

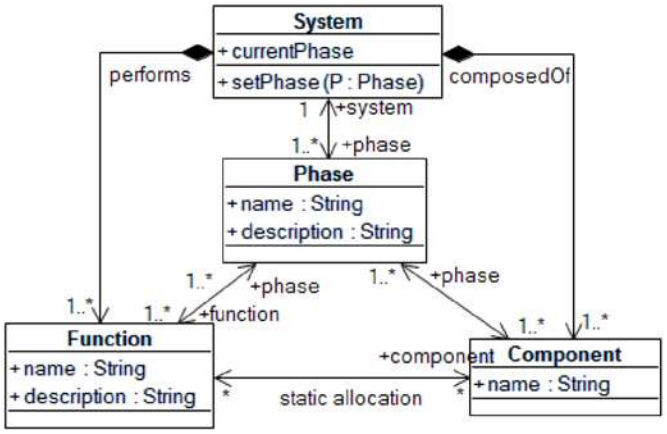

Fig. 3. Step 1: definition of the system phases

To simplify the representation, some attributes and links of

the initial meta-model (Figure 2) are removed in Figure 3; they must be considered nevertheless in the global model. The current phase can be updated by the method *setPhase*.

Definition 1: Method *setPhase*.

context System::setPhase(P: Phase)
pre: self.Phase −>includes(P)
post: self.currentPhase = P

B. Modeling the Component States

Each component can be activated and fail according to several operation and failure modes. These modes represent respectively the functional and dysfunctional properties of the component. Moreover, at least one operation mode: inactive (noted *OFF*), and one failure mode: faultless (noted *OK*) must exist (Constraint 1).

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

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

Therefore, 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 and operation modes.

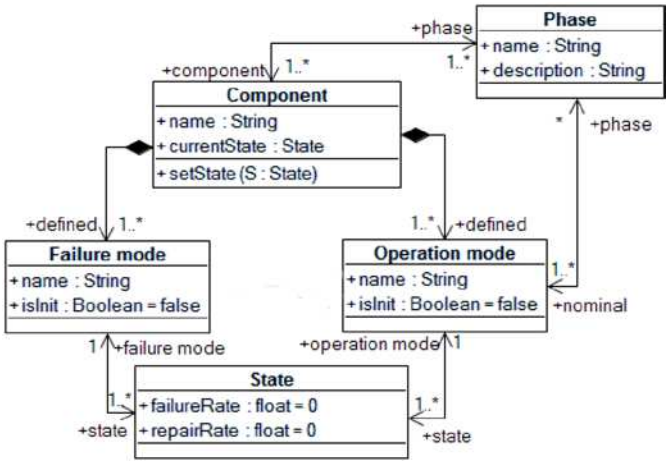

Fig. 4. Step 2: definition of the component states

An initial state of the component must be also defined as stated by the Constraint 2 which guarantees its uniqueness.

Constraint 2: Every component must have one 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 link between the classes *Phase* and *Operation mode* specify the nominal operation mode of a component for the considered mission phase. The existence and uniqueness of the nominal operation mode of each component involved in a given phase is ensured by the Constraint 3.

Constraint 3: A mission phase must define the nominal operation mode for each component which is involved in this phase.

context Phase **inv**:
self.Component.Operation mode
−>one(om: Operation mode | om = self.nominal)

Moreover, the class diagram represented at Figure 4 permits to model the stochastic evolutions of the component in the form of transitions from a state (om_i, fm_i) to a state (om_i, fm_j). These transitions are provoked by failure and repair events and are defined as follows (Statement 1).

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

- If *C* is in the faultless state (*om*,*OK*), then it can fail according to the failure mode *fm* and the transition from (*om*,*OK*) to (*om*,*fm*) occurs with the specified failure rate.
- If *C* is in the faulty state (*om*,*fm*), then it can be repaired and the transition from (*om*,*fm*) to (*om*,*OK*) occurs with the specified repair rate.

The current state of a component can be updated by the method *setState* defined below.

Definition 2: Method *setState*.

context Component::setState(P: Phase)
pre: self.Operation mode.State −>includes(S)
post: self.currentState = S

C. Modeling the Effects of Component States on Function Achievement

This subsection introduces the new class *Effect* and enhances the definition of the class *Function* by adding new attributes and methods (Figure 5). The aim of the new class is to relate the states of the components that are allocated to a function to the complete or partial achievement of this function. To meet this objective, a new attribute *allocation* must be added to the list of attributes of the class *Function*. The value of this attribute is the set of components which are allocated to the function.

The class *Effect* models the contribution of a component in a particular state to the achievement of a function to which it is allocated. Since this contribution depends on the active state, an instance of the class *Effect* must be defined for every

couple (state, function) as stated by Constraint 4.

Constraint 4: Every state of a component contributes in a unique manner to the achievement of each function to which this component is allocated.

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 attributes of the class *Effect* are *achievementRate* and *isUnacceptable*. The first one quantifies the contribution of the component, in the considered state, to the achievement of the function; in a system where two identical pumps operate in parallel to fill in a tank the *achievementRate* for a faultless pump is equal to 50% for a faultless pump and 0% for a failed pump, for instance. The second attribute points out the states which correspond to unsafe conditions; the values of this attribute for the different states are defined when instantiating the meta-model but is always False for the states which satisfy Constraint 5.

Constraint 5: A disabled (operation mode *OFF*) or faultless (failure mode *OK*) component has no unacceptable effect.

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

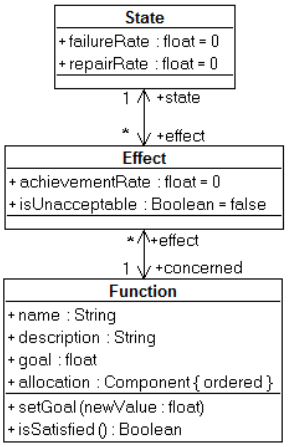

Fig. 5. Step 3: definition of the effects

The new attribute *goal* of the class *Function* is a threshold on the sum of the achievement rates of the components allocated to a function, i.e. a function will be declared correctly achieved if and only if the current states of the components which are allocated to it provide an overall achievement rate greater than this threshold. This attribute may be updated when the mission phase changes by the method *setGoal* given below.

Definition 3: Method *setGoal*.

context Function::setGoal(newValue: float)
pre: newValue ≥ 0.0
post: self.goal = newValue

Last, the method *isSatisfied* checks whether the sum of the achievement rates of the components allocated to a function, in their current states, is greater than *goal*.

Definition 4: Method *isSatisfied*().

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

D. *Introducing Redundancy Policies*

Using redundant components to continue to perform a function despite of the failure of other components that were allocated to this function is a well-known and widespread strategy to increase dependability [18]. This solution implies that the set of components allocated to a function changes during operation and that redundancy policies which specify these changes are defined. Therefore, a new class *Redundancy policy* as well as new relations must be introduced in the meta-model and two methods added to the class *Function* (Figure 6). These methods *dynamicAllocation* and *dynamicDeallocation* update the *allocation* attribute of the class *Function* and are defined below.

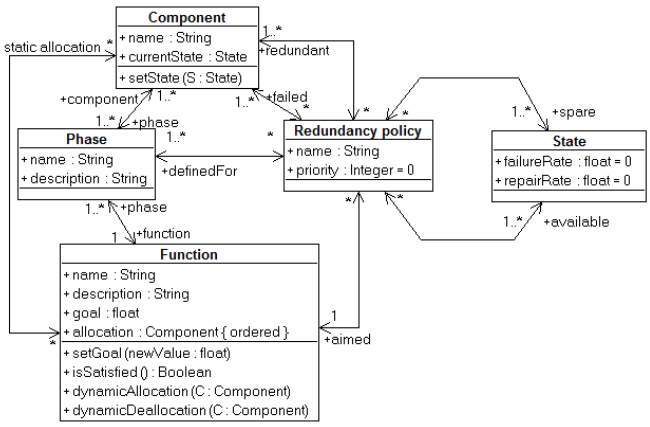

Fig. 6. Step 4: definition of the redundancy policies

Definition 5: Method *dynamicAllocation*.

context Function::dynamicAllocation(C: Component)
pre: self.static allocation −>includes(C)
post: self.allocation = self.allocation@pre −>including(C)

Definition 6: Method *dynamicDeallocation*.

context Function::dynamicDeallocation(C: Component)
pre: self.static allocation −>includes(C)
post:self.allocation = self.allocation@pre −>excluding(C)

The new class *Redundancy policy* and the new relations where this class is involved permit to specify how the faulty components are replaced by redundant components as follows.

Statement 2: If the *aimed* function is no more satisfied during a given phase, the *failed* components are deallocated whereas the *redundant* components whose current state is declared *available* (neither failed nor already used for another function), are allocated to the *aimed* function and their current state is set to *spare*.

It must be noted that, when several redundancy policies are possible after a component failure, priorities over these policies must be defined. Furthermore, the following two constraints are to be satisfied to ensure consistency of the model.

Constraint 6: The states whose roles are *available* or *spare* for a redundancy policy must be linked to the *redundant* components of this redundancy policy.

context Redundancy policy **inv**:
self.redundant.operation mode.state −>includesAll
(self.available −>union(self.spare))

Constraint 7: Exactly one spare state must be defined for each redundant component.

context Redundancy policy **inv**:
self.redundant −>forAll(c: Component |
c.Operation mode.State −>one(s: State | self.spare
−>includes(s)))

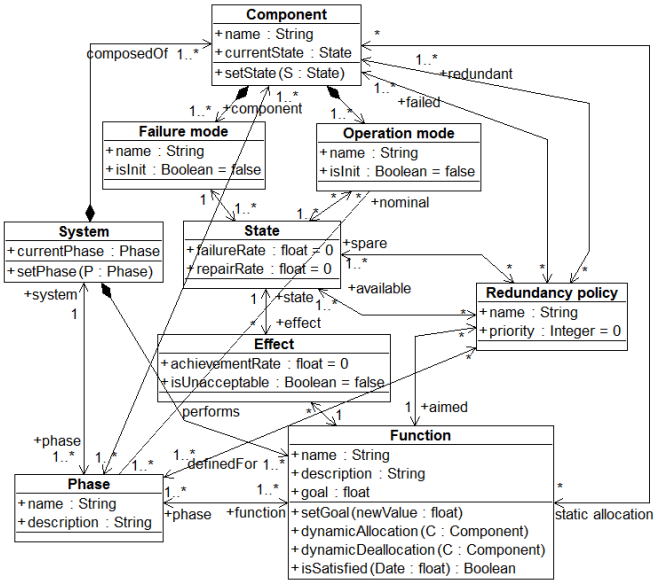

Fig. 7. Complete meta-model to support the integration of dependability analysis into SE processes

Figure 7 shows the complete meta-model built step by step in this section. This model fits the domain needs since it is possible to construct, from this class diagram, object diagrams for phased mission critical systems with repairable multi-state components and several redundancy policies, as it will be illustrated in the next section.

III. Building Object Diagrams from the Meta-model

The example considered in this section comes from a power plant: it is a part of the water level control system of the steam generator (Figure 8). This system has been previously described in [19] and [20], in particular to illustrate contributions in dynamic reliability assessment in the latter reference.

Within this system, only the sub-system composed of the two feeding turbo pumps *FTP*1 and *FTP*2, darkened part of Figure 8, will be considered in what follows, for space reasons. This sub-system has to perform only one function F: To supply enough water to the steam generator. The pumps *FTP*1 and *FTP*2 may fail and be repaired. To increase dependability of the function, the operation mode of each pump must be managed dynamically according to redundancy policies which will be described later.

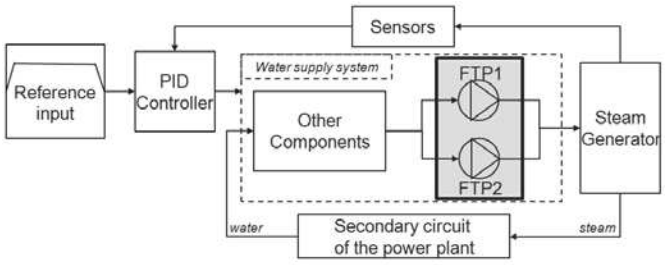

Fig. 8. Diagram of the water level control system and its environment (the considered sub-system is darkened)

Three instance diagrams completed by tables will be necessary to model this system. It must be underlined that the instance diagrams for a larger system must be constructed with the aid of a dedicated software tool, like for instance arKItect© [21].

A. Defining the Mission Phases

The main mission of the plant is to produce electric power and is decomposed in three phases (Table I). Function *F* is mandatory in all phases but only one pump is necessary to perform this function during first and third phases.

TABLE I
PHASES DESCRIPTION

id	role	description
P1	To increase the power from zero to the nominal value	A single pump is able to perform correctly the function.
P2	To produce the nominal power	The two pumps have to run together to perform correctly the function.
P3	To decrease the power from the nominal value to zero	A single pump is able to perform correctly the function.

B. Defining the Component States

According to the first instance diagram (Figure 9), each pump has three operation modes: OFF (as every component), *Run* and *Overspeed*[1]. It will be assumed that FTP1 is the main pump and FTP2 a spare pump for the phases P1 and P3; hence, the initial modes of FTP1 and FTP2 are respectively *Run* and OFF. The pumps are considered faultless at the initial state; therefore the initial failure mode is OK for both and each pump may fail according two failure modes called *Leak* and *Rupture*.

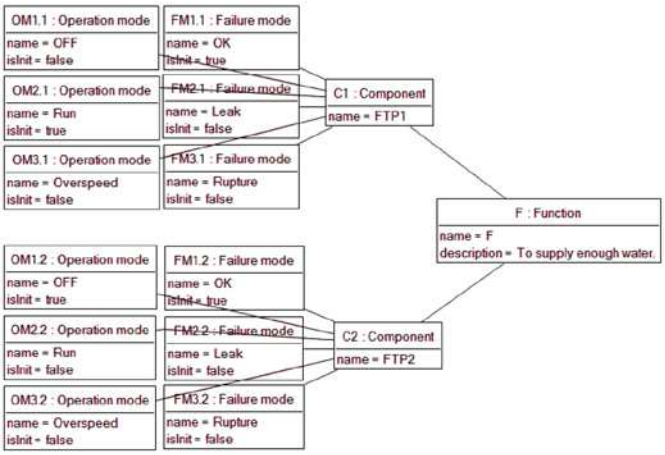

Fig. 9. First partial instance diagram for the considered sub-system

Table II contains the values of the attributes (*failureRate*, *repairRate*) of the State class instances[2]. In this study, the probability rates are assumed to be constant for a couple (*operation mode, failure mode)*.

TABLE II
FAILURE/REPAIR RATES

Operation \ Failure	*OK*	*Leak*	*Rupture*
OFF	*OFF-OK* Not relevant	*OFF-Leak* (0, 0.2)	*OFF-Rupt* (0, 0.1)
Run	*Run-OK* Not relevant	*Run-Leak* (0.01, 0.1)	*Run-Rupt* (0.001, 0)
Overspeed	*Over-OK* Not relevant	*Over-Leak* (0.05, 0)	*Over-Rupt* (0.002, 0)

The first value (e.g. 0.01 for the state *Run-Leak*) is the failure rate of the transition that leads to this state and the second one (0.1 for the same state), the repair rate to leave this state. These attributes are not relevant for the states of the first column which are faultless. It must be noted that a pump can fail only if it is active (with a higher probability in the operation mode *Overspeed* than in *Run*)[3]. Moreover, the *Leak* and the *Rupture* can be repaired if the pump is disabled whereas only the *Leak* can be repaired if the pump is in the operation mode *Run* (even if the repair time is longer than in *OFF*), and no repair is possible if the pump is in the operation mode *Overspeed*. As the pumps are assumed identical, only one table is necessary.

C. Describing the Effects of the Component States

The contribution of one pump to the function depends on its state. When the pump is disabled (operation mode *OFF*), this contribution is obviously equal to zero. This is also the case when the *Rupture* failure has occurred. The analysis is not so simple for the remaining four states: *Run-OK*, *Run-Leak*, *Overspeed-Ok* and *Overspeed-Leak*.

The attribute *goal* of function *F* must be first defined. For phased mission systems this goal is usual equal to 100% for the most demanding phase, *P*2 in this study, and to smaller values for the other phases. Expert knowledge is mandatory to set these values; it will be assumed hereafter that the goal of *F* during the phases *P*1 and *P*3 is equal to 60%, i.e. that 60% of the maximal water flow is sufficient during these two phases.

Once the goal set for each phase, it is possible to define the values of the attribute *achievementRate* for every state of a pump (Table III). According to this table, when the operation mode is *Run*, function *F* is achieved at 60% when the pump is faultless and 50% when it is leaking. Hence, only one faultless pump is necessary to meet the goal of *F* during the phase *P*1 and *P*3 whereas two running pumps, faultless or leaking, are mandatory for the second phase. Last, a faultless over-speeded pump is sufficient to meet the goal, whatever the phase. This table will permit to define the redundancy policies in the next section.

TABLE III
ACHIEVEMENT RATES FOR THE DIFFERENT STATES OF A PUMP

	OK	*Leak*	*Rupture*
OFF	0	0	0
Run	60	50	0
Overspeed	100	80	0

Moreover it is assumed that the states built over the failure mode *Rupture* are forbidden by taking into account safety. Then, the attribute *isUnacceptable* of each instance of the class *Effect* linked to these states is *True*.

D. Describing the effects of the component states

Three redundancy policies are defined to improve the dependability of the system. Hence, the class *Redundancy policy* is instantiated three times:

[1] *Overspeed* means that the speed of the pump is not the nominal value but a higher speed that fits the physical limitations of this component however.

[2] These values are rounded average values obtained from several tens of power plants during about forty years.

[3] The state *OFF-Leak* (respectively *OFF-Rupture*) is then not reachable from the state *OFF-OK* but from the state *Run-Leak* (respectively *Run-Rupture*).

- *R*1*a* is defined for the phases *P*1 and *P*3, and consists in replacing the pump *FTP*1 by *FTP*2, if function *F* is not satisfied and *FTP*2 is available.
- *R*1*b* is defined for the phases *P*1 and *P*3, and consists in replacing the pump *FTP*2 by *FTP*1, if function *F* is not satisfied and *FTP*2 is available.
- *R*2 is defined for the phase *P*2, and consists in forcing the operation mode of the faultless pump to *Overspeed* if function *F* is not satisfied because the other pump has failed.

It can be noted that, during the second phase, if the two pumps fail with the failure mode *Leak*, according to the table III, the function is correctly achieved anyway and this failure mode can be repaired in the operation mode *Run*. Then the *Leak* failure mode does not trigger the redundancy policy *R*2.

Fig. 10. Second partial instance diagram for the considered sub-system

The instance diagram that represents graphically these policies and their relations to other class instances has been split in Figures 10 and 11, for clarity reasons.

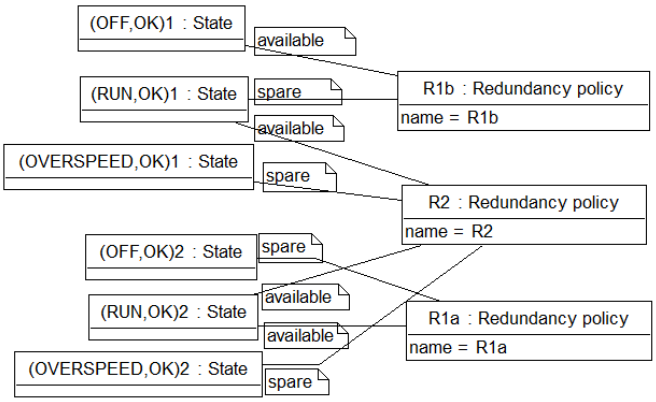

Fig. 11. Third partial instance diagram for the considered sub-system

The attribute priority is not specified because there is no concurrency between these redundancy policies. It can be noted that for the redundancy policy *R*2 the two pumps are both defined *failed* and *redundant*, because they could play the two roles: either *FTP*1 has failed and *FTP*2 is redundant, or *FTP*2 has failed and *FTP*1 is redundant.

The system features during each phase are summarized in table IV.

TABLE IV
SUMMARY OF THE FEATURES OF THE INSTANCES OF THE CLASS *Phase*

Phase	*Nominal OM*		*Function goal*	*Redundancy policies*
	FTP1	*FTP2*		
P1	*Run*	*OFF*	60.0	*R1a, R1b*
P2	*Run*	*Run*	100.0	*R2*
P3	*Run*	*OFF*	60.0	*R1a, R1b*

IV. USING THE META-MODEL FOR DEPENDABILITY ANALYSIS

It has been claimed in the introduction that one benefit of a meta-model is to ease the automatic construction of some specific models from previously defined data. This claim will be illustrated in this section where a method to construct systematically the continuous Markov Chains (MC) that represent the behavior of a phased mission system which includes redundant components will be proposed. Continuous MCs are indeed common and relevant models to assess dependability attributes, like (un)availability; systematic construction of such models from instance diagrams is surely a positive consequence of the integration of dependability concerns into a meta-model for SE processes. This construction method is described and exemplified, on the basis of the previous example, in the first sub-section, while analytic assessment of unavailability, from these chains, is dealt with in the second sub-section. It must be underlined that, when dealing with larger systems, both steps must be automated. Automatic construction of the MC will be based on the systematic procedure described below. The reader interested in automatic assessment of unavailability from large MC models will find more details on this issue in [22].

A. Systematic Construction of MCs from Instance Diagrams

Modeling the evolutions of a phased mission system with repairable components by MCs is a classical approach in the field of dependability analysis ([23], [24]). One MC must be constructed for each phase and a transition matrix specifies how the probabilities are distributed over the states when the active phase changes.

From the instance diagrams presented in section III, it is possible to build systematically the MCs depicted in Figures 12 and 13; the MC of Figure 12 describes the system behavior for both phases 1 and 3 whereas this behavior for the second phase is given at Figure 13. These MCs have been built systematically, to avoid errors that may appear happen when no method is employed, according to the following generic procedure:

1) The initial state of the chain is defined from the initial states of the components of the system, for the considered phase.

2) For each state of the chain, an outgoing transition is added for each possible failure or repair event.
3) If this transition does not lead to a state which has been already defined, a new state is introduced by taking into account the redundancy policies.
4) The system is unavailable if the sum of the achievement rates of the components in this state is smaller than the goal of the function.

This procedure stops when it is no longer possible to add any transition or state.

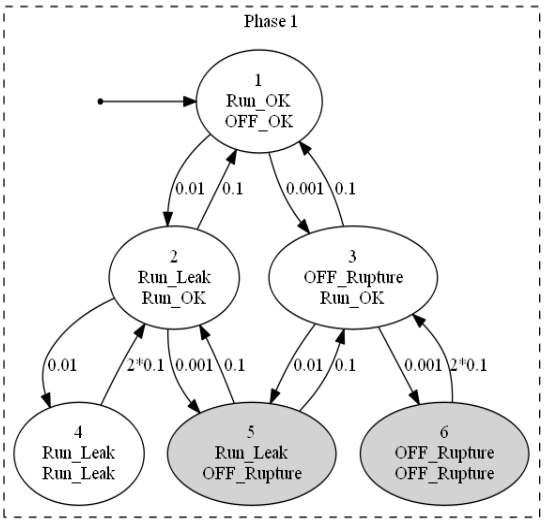

Fig. 12. MC for the phases 1 and 3

The MC of Figure 12 has been built by using the above-defined procedure:

1) The initial states of the pumps are respectively *Run-OK* and *OFF-OK* for the phases 1 and 3 (table IV). Then, the initial state of the chain gathers these two states.
2) Two transitions are starting from the initial state of the chain to represent the two possible failure modes *Leak* and *Rupture*. The failure rates are given by the first term of the cells *Run-Leak* and *Run-Rupt* in the table II.
3) The states 2 and 3 are introduced to represent a system where one failure has occurred in one component and the other component has been activated according to the appropriate redundancy policy. As the failure mode *Rupture* is unacceptable, the component for which this failure occurred is forced to the operation mode *OFF* (state 3).
2) From these new states, the failed pump can be repaired with repair rates given by the second term of the cells *Run-Leak* and *OFF-Rupt* in Table II, what leads to the initial state, or the running pump can fail with failure rates given by the first term of the cells *Run-Leak* and *Run-Rupt* in the table II.
3) The states 4, 5 and 6 are then introduced to model a system where failures have occurred in the two components.
2) From these states, the failed pumps can be repaired, what leads to the already known states 2 or 3. It shall be noted that the probability rate associated to the transition from the state 4 to the state 2 (respectively 6 to 3) is two times the repair rate defined in the cell *Run-Leak* (respectively *OFF-Rupt*) because the two pumps are in the same state; therefore the probability to have a pump repaired at a given time, assuming that they can be repaired simultaneously, is twice better.
4) As the goal of the function in these phases is equal to 60.0, the states 5 and 6 correspond to an unavailable system because the sum of the achievement rates of the two pumps is indeed equal to 50.0 for the state 5 and 0.0 for the state 6, according to the table III.

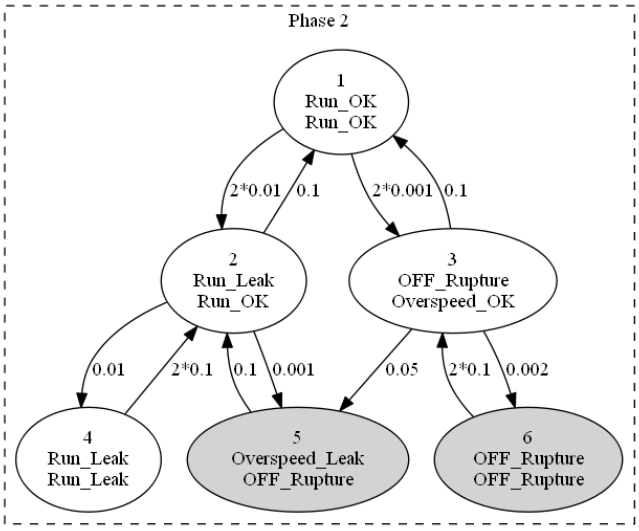

Fig. 13. MC for the phase 2

The MC for the second phase (Figure 13) is built in the same fashion. Two faultless pumps are running in the state 1 and the transition from this state to the state 3 models the redundancy policy *R2*. The states 2 and 4 are identical to those of Figure 12 but it shall be underlined that no redundancy policy is involved to define these states in the second phase; the transitions between the states 2 and 4 are merely provoked by failure or repair events. The state 5 differs from the corresponding state in Figure 12 because the active pump is over-speeded; there is no transition from this state to the state 3 because the *Leak* failure cannot be repaired when the operation mode is *Overspeed*. Last, some transition rates are different because either two pumps are in the same operation mode in the source state of the transition (transition from 1 to 2, 1 to 3, 4 to 2, 6 to 3) or because the rate defined in the table II is different. The darkened states (5 and 6) correspond to an unavailable system; the sum of the achievement rates is smaller than the goal.

The model of the whole behavior of the system for the three phases is given at Figure 14. This model is composed of the three previous MCs as well as three transition matrices $\varphi_{i \to j}$ that allow the computation of the probability p_l^j to be in a state l of the MC for the phase j at the date t_{ij}, where t_{ij} is the date of a phase change from the phase i to the phase j, from the probabilities p_k^i to be in a state k of the MC for the phase i

at the date t_{ij}. In our case, these matrices are identity matrices, i.e.

$$\forall k \in [\![0,6]\!] \begin{cases} p_k^1(t_{12}) = p_k^2(t_{12}) \\ p_k^2(t_{23}) = p_k^3(t_{23}) \\ p_k^3(t_{31}) = p_k^1(t_{31}) \end{cases} \quad (1)$$

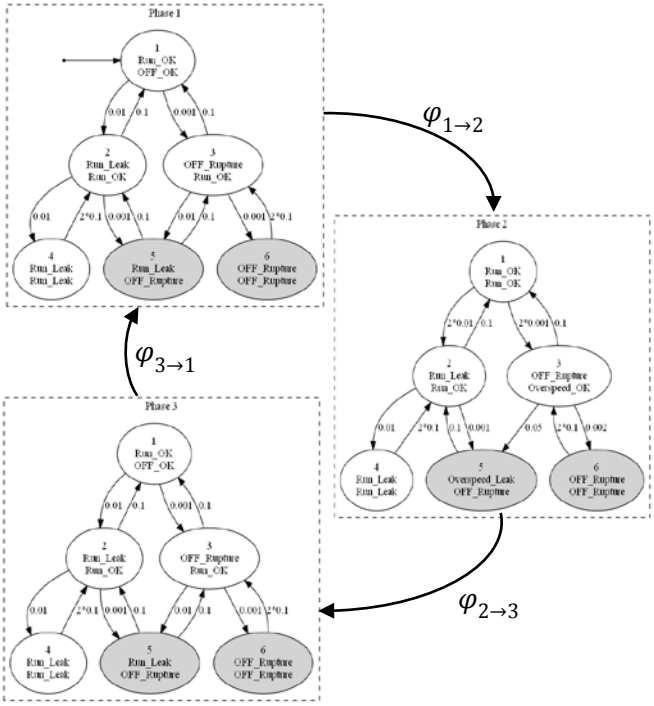

Fig. 14. MCs for a mission

B. Unavailability Assessment

Once this model obtained, it is possible to compute the unavailability of the system by analytic calculus or Monte-Carlo simulation. The first technique was selected for this work because it can be easily applied to small-sized systems and provides generic solutions (more details on this issue may be found in [22]).

For the phases 1 and 3, the transition matrix of the MC is (cf. Figure 12):

$$M_{13} = \begin{bmatrix} -0.011 & 0.01 & 0.001 & 0 & 0 & 0 \\ 0.1 & -0.111 & 0 & 0.01 & 0.001 & 0 \\ 0.1 & 0 & -0.111 & 0 & 0.01 & 0.001 \\ 0 & 0.2 & 0 & -0.2 & 0 & 0 \\ 0 & 0.1 & 0.1 & 0 & -0.2 & 0 \\ 0 & 0 & 0.2 & 0 & 0 & -0.2 \end{bmatrix}$$

Let $P_{13}(t) = [p_1^{13}(t), p_2^{13}(t), p_3^{13}(t), p_4^{13}(t), p_5^{13}(t), p_6^{13}(t)]$, be the probability to be in each state of the chain at the date t, P_{13} is solution of the system of differential equations:

$$\frac{dP_{13}(t)}{dt} = P_{13}(t).M_{13}$$

It can be expressed as:

$$P_{13}(t) = P_{13}(0).e^{M_{13}t}$$

From this result, the asymptotic unavailability for each one of the phases 1 and 3 can be computed from the asymptotic probabilities to be in the states 5 and 6 of the chain:

$$\bar{A}_{13}(\infty) = p_5^{13}(\infty) + p_6^{13}(\infty) = 9.41 \times 10^{-4}$$

The asymptotic unavailability for the phase 2 can be calculated in a similar manner: $\bar{A}_2(\infty) = 7.156 \times 10^{-3}$.

The result obtained for a sequence of twelve identical missions where the first phase lasts one day, the second 28 days and the third also 1 day is given at Figure 15. The average unavailability is equal to 0.623827%, which corresponds to approximately 2 days and 6 hours in absolute value.

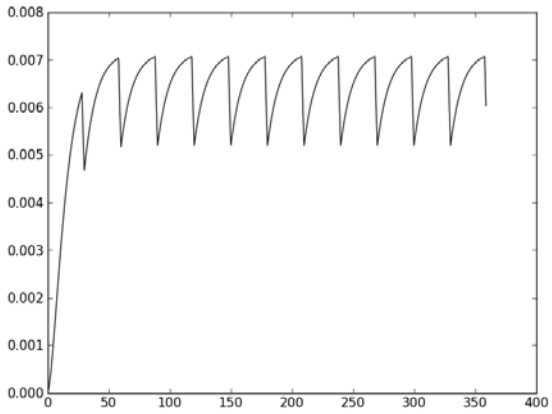

Fig. 15. Unavailability for twelve consecutive missions

V. Conclusions and Outlooks

This paper has proposed a meta-model based on [4] to support the integration of dependability analysis into system engineering processes. This meta-model has been defined for a wide class of systems: phased mission systems with multi-state and repairable components and several redundancy policies; it can be easily specialized to deal with more restricted classes of systems, like, for instance, systems with binary and non-repairable components or systems whose mission comprises only one phase.

The benefit of this proposal has been shown on a small example of a critical system coming from a power plant. A systematic procedure to construct MCs which are classical models for availability analysis, from object diagrams derived from the meta-model, has been defined and exemplified.

On-going works are aiming to automate the construction of object diagrams and MCs to be able to deal with full-scale industrial cases. Automatic construction of MCs will take into account uncertainties on the failure and repair rates to better assess the impact of these uncertainties on the final results (probabilities to be in a given state, (un)availability).

Introduction of new classes, attributes and relations in the

meta-model, to integrate the new concepts recently presented in the domain of dynamic dependability analysis ([11], [25]), is also under investigation. This will allow, in particular, to integrate faulty behaviors due to software or hardware components of the Instrumentation and Control system, into the dependability analysis.

Finally, validation of this meta-model on case studies larger than the small example which has been presented in this paper will permit to really assess its usefulness, completeness, and flexibility and to estimate correctness and scalability of the analysis methods of the derived models.

REFERENCES

[1] N.-G. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety,* J. Moses, Ed. MIT Press, Cambridge (Massachusetts), 2011, pp. 3–72; 307-348.

[2] Y.-Y. Haimes, *Risk Modeling, Assessment, and Management*, P. Sage Ed., WILEY, Hoboken (New Jersey), 2009

[3] R.-W.-A. Barnard, "What is wrong with Reliability Engineering", INCOSE, 2008

[4] 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.

[5] 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, pp. 1287–1293.

[6] 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.

[7] J. Michael, M.-T. Shing, K. Cruickshank, and P. Redmond, "Hazard analysis and validation metrics framework for system of systems software safety," *Systems Journal, IEEE*, vol. 4, no. 2, pp. 186–197, 2010.

[8] 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.

[9] R. Billinton and N. Allan, *Reliability Evaluation of Power Systems*, Springer, 1996

[10] D.-C. Raiteri, G. Franceschinis, M. Iacono and V. Vittorini, "Repairable fault tree for the automatic evaluation of repair policies", International Conference on Dependable Systems and Networks, Florence (Italy) July 2004, pp.659-668

[11] P.-Y. Chaux, J.-M. Roussel, J.-J. Lesage, G. Deleuze, M. Bouissou, « Towards a unified definition of Minimal Cut Sequences », in *DCDS 2013,* York (UK), Paper n°1, 6 pages, September 2013

[12] 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.

[13] G. Levitin, A. Lisnianski, H. Ben-Haim and D. Elmakis, "Redundancy Optimization for Series-Parallel Multi-State Systems", in *IEEE Transactions on Reliability*, vol. 47, No. 2, pp 165-172, June 1998

[14] Y.-Y. Haimes, "On the Definition of Resilience in Systems", in *Risk Analysis,* vol. 29, No. 4, pp 498-501, 2009

[15] OMG, *Uml 2.0 Infrastructure specification*, Object Management Group, 2005.

[16] ——, Uml *2.0 OCL specification*, Object Management Group, 2003.

[17] L. Meshkat, L. Xing, S.-K. Donohue, and Y. Ou, "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.

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

[19] 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.

[20] 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.

[21] H. Aboutaleb, M. Bouali, M. Adedjouma, and E. Suomalainen, "An integrated approach to implement system engineering and safety engineering processes: Sasha project," in *ERTS'2012*, Toulouse, France, February 2nd 2012.

[22] W.-J. Stewart, *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, 1994.

[23] M. Alam, U.-M. Al-Sagaaf, and M. Ubaud, "Quantitative reliability evaluation of repairable phased-mission systems using Markov approach," *IEEE Transactions on Reliability*, vol. R-35, no. 5, pp. 498–503, December 1986.

[24] K.-E. Murphy, C.-M. Carter and A.-W. Malerich, "Reliability analysis of phased-mission system: A correct approach," in *RAMS,* 2007.

[25] A. Rauzy, "Sequence algebra, sequence decision diagrams and dynamic fault trees," *Reliability Engineering and System Safety*, vol. 96, no. 7, pp. 785–792, 2011.

Pierre-Yves Piriou received an Engineering Degree (Mechatronic and Complex Systems) from the Institut Superieur de Mecanique de Paris and a Master Degree in Complex Systems Engineering from the ENS Cachan in 2012. He is a PhD student at the Ecole Normale Superieure de Cachan. His research concerns the assessment of dependability attributes of I&C architectures.

Jean-Marc Faure received the Ph.D. degree from the Ecole Centrale de Paris in 1991. He is currently Professor of Automatic Control and Automation Engineering at the Institut Superieur de Mecanique de Paris and researcher at Ecole Normale Superieure de Cachan, France. His research fields are modeling, synthesis and analysis of Discrete Event Systems (DES) with special focus on formal verification and test methods to improve dependability of critical systems. J.-M. Faure is member of the IEEE and Associate Editor of the Journal T-ASE since 2012. He is chair of the steering committee of the IFAC workshops series "Dependable Control of Discrete Systems". He has served in many committees of IFAC and IEEE conferences.

Gilles Deleuze received an Engineering Degree (Naval & Nuclear Engineering) from ENSTA (Ecole Nationale Superieure de Techniques Avancees) in 1986 and a Technology Management Degree from Paris IX University in 1987. He is project leader and senior researcher in the R&D labs of Electricite de France, Industrial Risks Management Department. He currently works on the implementation of risk assessment frameworks the improvement of the modeling of I&C systems in probabilistic safety assessment and co-development of Dependability and System Engineering. Previously, he worked for THALES in the field of electronic components and systems dependability for space, avionics and military equipment.