# OPTIMISATION D'UNE STRATÉGIE DE MAINTIEN EN CONDITION OPERATIONNELLE D'UNE FLOTTE D'ÉQUIPEMENTS PAR L'UTILISATION D'UNE MODÉLISATION DYNAMIQUE

Emmanuel Clement, Michel Batteux, Pierre Laize

# OPTIMISATION D'UNE STRATEGIE DE MAINTIEN EN CONDITION OPERATIONNELLE D'UNE FLOTTE D'EQUIPEMENTS PAR L'UTILISATION D'UNE MODELISATION DYNAMIQUE

# OPERATIONAL CONDITION STRATEGY OPTIMIZATION OF EQUIPMENTS FLEET BY USE OF DYNAMIC MODELISATION

Emmanuel CLEMENT et Pierre LAIZE.
Thales Defense Mission Systems
10 Avenue 1ère Dfl
29200 Brest

Michel BATTEUX
IRT SystemX
8, avenue de la Vauve
CS 90070 Palaiseau

## Résumé

Nous présentons une méthode permettant de considérer l'ensemble d'une chaîne complexe de maintien en condition opérationnelle, dans l'estimation de la disponibilité prévisionnelle, par l'intermédiaire d'une modélisation avec AltaRica 3.0 (Prosvirnova et al.,2013). Cette modélisation permettra d'optimiser les coûts, tout en démontrant un niveau de disponibilité prévisionnelle important.

## Summary

We are presenting an approach which aims to take into account all operational condition strategy parameters as a whole to estimate the availability forecast by the use of a dynamic modeling in AltaRica 3.0. This simulation will provide an operational condition strategy which will be cost effective and will provide high availabity.

## Introduction

### 1 Contexte

La complexité croissante des systèmes entraîne, notamment en sûreté de fonctionnement, la nécessité de progresser dans les méthodes d'analyse en développant l'ingénierie dirigées par les modèles (MBSA) (Boiteau *et al.*, 2006) (Riera et al.,2012). Le MBSA (Model-Based Safety Assessment) permet, par la réalisation de modèles plus haut niveau, c'est-à-dire proches des architectures fonctionnelles et physiques des systèmes, d'assurer le partage et la cohérence des hypothèses, des données d'entrée et des résultats des analyses, entre les différents acteurs de la Sûreté de fonctionnement. Cela conduit à une cohérence, à une validation et à une capitalisation des modèles et des analyses. C'est dans ce cadre qu'intervient cette communication.

### 2 Objectif

L'objectif de cette publication est de proposer une méthode permettant de considérer l'ensemble d'une chaîne complexe de maintien en condition opérationnelle dans l'estimation de la disponibilité prévisionnelle par l'intermédiaire d'une modélisation avec AltaRica 3.0 (Prosvirnova et al.,2013). Cette modélisation permettra d'optimiser les coûts tout en démontrant un niveau de disponibilité prévisionnelle important.

### 4 Problématique

Thales Defense Mission Systems conçoit et réalise des systèmes critiques qui sont déployés sur des ensembles de plateformes. Ces systèmes présentent les caractéristiques suivantes :

- Niveau de disponibilité élevé ;
- Répartition géographique mondiale ;
- Notion de « flotte » ;
- Délai d'approvisionnement parfois important ;
- Personnel de maintenance limité en nombre.

Thales Defense Mission Systems doit assurer un niveau de disponibilité important exigé par ses clients tout en proposant des coûts de maintien en condition opérationnelle maîtrisés.

### 4 Approche

Nous proposons une méthode permettant de prendre en compte le Maintien en Condition Opérationnelle (MCO) dans un modèle MBSA dans le but d'optimiser la chaîne MCO tout en démontrant l'atteinte d'un niveau de disponibilité prévisionnelle. Cette modélisation permet de tenir compte des contraintes opérationnelles de nos clients parmi lesquelles :

- L'organisation et la quantité de stocks ;
- Les délais d'approvisionnement ;
- La disponibilité des équipes de maintenance ;
- La disponibilité des moyens de maintenance :
- La performance du diagnostic.

Cette démarche s'inscrit dans une démarche plus globale d'organisation des analyses FMDTS (Fiabilité, Maintenabilité, Disponibilité, Testabilité et Sécurité) autour d'un modèle MBSA unique, validé et partagé.

Afin de bien illustrer le sujet, un cas concret sera étudié tout au long de la communication. Une méthode heuristique d'identification des équipements candidats aux stocks sera présentée. Les patterns de modélisation de la chaîne de MCO seront explicités. Les paramètres de la simulation stochastique et les résultats de l'analyse seront décrits.

Notre publication est articulée de la façon suivante :

1. Présentation du cas industriel ;
2. Présentation d'OpenAltaRica et d'AltaRica 3.0 ;
3. Description de la modélisation ;
4. Présentation des résultats ;
5. Conclusion

# Cas industriel

## 1. Contexte

Les coûts globaux de possession ainsi que la disponibilité opérationnelle des équipements sont deux performances hautement sensibles pour nos clients et peuvent faire la différence sur des marchés où la concurrence est importante. Il est donc nécessaire, pour un industriel, de maîtriser ces deux aspects.

Le coût du maintien en condition opérationnelle d'un système contribue de façon importante au coût global de possession. Or, aujourd'hui, les systèmes sont de plus en plus sophistiqués et nécessitent des moyens de maintenance de plus en plus complexes ainsi que du personnel hautement qualifié.

Pour maîtriser les coûts tout en justifiant un niveau de disponibilité opérationnelle élevé, il apparaît nécessaire de dimensionner la stratégie de maintien en condition opérationnelle au plus juste. Pour cela, plusieurs leviers sont possibles :

- Dimensionnement des stocks ;
- Délais d'approvisionnement ;
- Performance des moyens de diagnostic ;
- Disponibilité des équipes et des moyens de maintenance ;
- Accessibilité des éléments interchangeables.

Or, évaluer une disponibilité prévisionnelle en prenant en compte l'ensemble de ces leviers devient assez vite compliqué.

Les outils et méthodes de modélisation sont aujourd'hui très performants et deviennent accessibles aux ingénieurs de la Sûreté de fonctionnement et du Soutien.

Nous vous proposons donc une méthode et des patterns de modélisation en AltaRica 3.0, sous la plateforme outillée OpenAltaRica, permettant de quantifier une disponibilité prévisionnelle en :

- Dimensionnant les différents stocks ;
- Tenant compte des délais d'approvisionnement ;
- Tenant compte de la disponibilité et de la quantité des équipes de maintenance ;
- Tenant compte de la disponibilité et de la quantité des moyens de maintenance ;
- Tenant compte des performances de localisation des pannes.

## 2. Présentation du cas applicatif

Afin d'illustrer la méthode, nous vous proposons un cas applicatif. Il s'agit d'un système de mission aéroporté constitué de 130 éléments remplaçables en ligne.

Le système est déployé sur plusieurs avions répartis dans le monde par binôme. Pour assurer le maintien en condition opérationnelle, trois niveaux de stocks sont prévus :

1. Un stock industriel ;
2. Un stock « Base » unique pour toute la flotte;
3. Un stock « Mission » pour chaque binôme d'avions.

Par ailleurs, une équipe de maintenance est disponible pour chaque binôme d'avions.

Le schéma ci-après représente les binômes d'avions ainsi que les différents stocks associés.

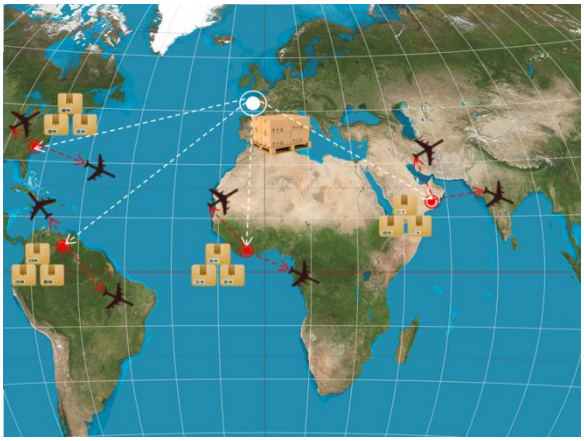


**Figure 1** -Description de la configuration des stocks

## OpenAltaRica et AltaRica 3.0

### 1. AltaRica 3.0

AltaRica 3.0 (Prosvirnova *et al.*, 2013) est, comme son nom l'indique, la troisième version du langage de modélisation AltaRica . Il améliore la version précédente, AltaRica Data-Flow (Boiteau *et al.*, 2006) suivant deux points fondamentaux. La sémantique d'exécution basée les Systèmes de Transitions Gardées (GTS) et le paradigme de structuration basé sur S2ML.

En premier lieu sa sémantique d'exécution est basée sur les Systèmes de Transitions Gardées, GTS pour Guarded Transitions Systems (Rauzy, 2008). Les GTS généralisent les formalismes classiques d'analyse du risque (arbres de défaillance, chaînes de Markov, réseaux de Pétri stochastiques, etc.), sans augmenter la complexité des calculs d'indicateurs du risque. L'exécution des modèles AltaRica 3.0 est similaire aux autres formalismes à événements discrets dans le sens où chaque fois qu'une transition est tirable, elle est alors planifiée et sera alors potentiellement tirée après un certain délai (Cassandras et al., 2008). Ces délais peuvent être déterministes ou stochastiques ; et dans ce dernier cas, AltaRica 3.0 fournit les délais usuels (du type exponentiel, Weibull, etc.) et des délais empiriques.

De plus, le paradigme de structuration du langage AltaRica 3.0 est basé sur S2ML, pour System Structure Modeling Language (Batteux et *al.* 2015). S2ML unifie les deux paradigmes (de structuration) dominants des langages de modélisation, i.e. l'orienté objet et l'orienté prototype. Il permet de modéliser suivant deux approches combinables : (i) l'approche dite 'top-down', avec une vision au niveau système permettant la réutilisation de schémas de modélisation, et donc utilisant le paradigme orienté prototype ; (ii) l'approche dite 'bottom-up', avec une vision au niveau composants permettant la réutilisation de composants, définis en bibliothèques, et donc utilisant le paradigme orienté objet.

Le pouvoir d'expression du langage AltaRica 3.0 est seulement une partie de la solution d'analyse du risque de systèmes complexes. L'autre partie concerne les outils d'évaluation associés, notamment l'efficacité pratique des algorithmes d'évaluation. En effet, l'évaluation pour la sûreté ou la fiabilité est un problème dit de complexité #P-hard (Valiant 1979). Cela signifie que cette barrière de complexité n'est pas un problème de technologie, mais un problème théorique. Il est donc nécessaire de concevoir des outils d'évaluation implémentant efficacement les algorithmes de traitement. Le langage AltaRica 3.0 vient de ce fait avec un ensemble d'outils permettant de calculer différents indicateurs. La figure ci-après représente l'ensemble des outils d'évaluation associés au langage AltaRica 3.0. Cette figure indique aussi les outils graphiques permettant de concevoir et éditer des modèles AltaRica 3.0, mais aussi de les simuler graphiquement. Différents outils permettent d'évaluer les modèles AltaRica 3.0. Le simulateur pas-à-pas est l'outil permettant de réaliser des expériences virtuelles sur les modèles (vérification manuelle de modèles par exemple). Les deux outils de compilation vers les arbres de défaillance et vers les chaînes de Markov (pas encore disponible) permettent de générer, à partir de modèles AltaRica 3.0, des modèles vers ces formalismes cibles. L'utilisation d'outils d'évaluation associés à ces formalismes est ensuite possible. Le générateur de séquences permet d'obtenir l'ensemble des séquences d'évènements amenant à un événement redouté considéré. Enfin, le simulateur stochastique permet d'obtenir des statistiques sur des exécutions aléatoires de modèles AltaRica 3.0. L'ensemble de ces outils est distribué au sein de la plateforme OpenAltaRica issue du projet du même nom.

### 2. OpenAltaRica

Le projet OpenAltaRica est un projet réalisé au sein de l'Institut de Recherche Technologique (IRT) SystemX et en partenariat avec Thales, Apsys, Safran et AltaRica Association. D'une durée de 5 ans et ayant démarré fin 2014, ce projet vise l'implémentation de la plateforme de référence pour le langage AltaRica 3.0. Cette plateforme d'expérimentation, disponible sur le site du projet (www.openAltaRica.fr), intègre déjà l'ensemble des outils développés. Elle sera complétée par les outils en cours de développement.

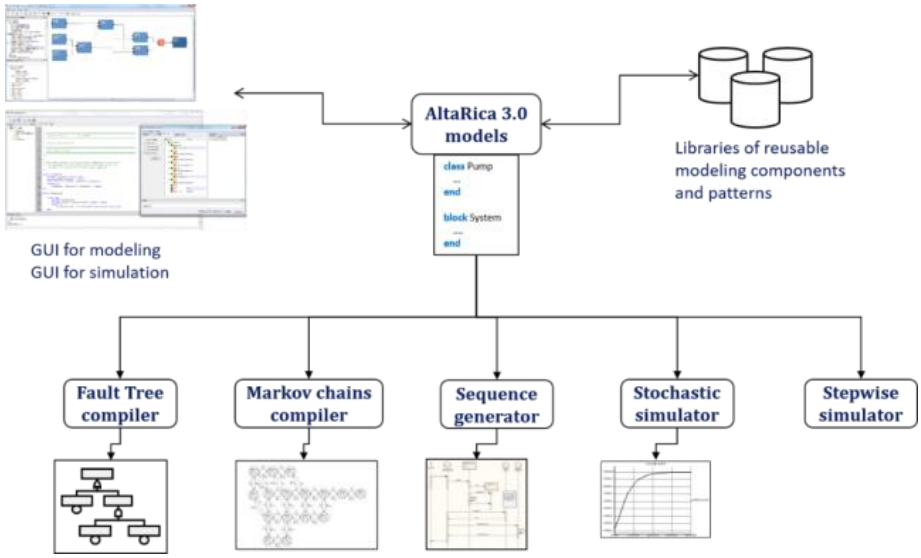



**Figure 2**. Description des outils d'évaluation associés au langage AltaRica 3.0

## Description de la modélisation

### 1. Modélisation des éléments remplaçables en ligne

#### 1.1. Principe général

La figure 3 représente le plus bas niveau de la modélisation retenu. C'est-à-dire l'unité remplaçable en ligne (URL). Il est modélisé par un bloc avec une entrée et une sortie.


**Figure 3** - Représentation d'un élément réparable

Son comportement, sans l'intégration des contraintes présentées dans les paragraphes suivants, est décrit au moyen du graphe d'état présenté dans la figure 4. La valeur de la sortie « OUT » de l'URL (figure 3) est égale à la valeur de son entrée « IN » s'il se trouve dans l'état «WORKING», sinon, sa sortie « OUT » vaut « FAUX ».

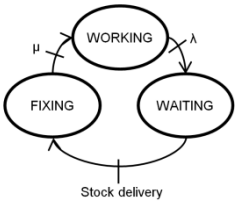



**Figure 5** – Comportement d'un élément réparable

Le code AltaRica 3.0 correspondant est montré ci-dessous.

```
class RepairableComponent
  STATES_LRU  STATES (init = WORKING);
  Boolean inflow, outflow ( reset = false);
  parameter Real lambda = Lambda_defaut;
  parameter Real mu = Mu_defaut;
  event failure (delay = exponential( lambda ));
  event repair (delay = exponential( mu ));
  event delivery(delay = Dirac( 0 ), hidden = true ) ;
  transition
    failure : STATES == WORKING -> STATES := WAITING;
    delivery : STATES == WAITING -> STATES := FIXING;
    repair : STATES == FIXING -> STATES := WORKING;

  assertion
    outflow := if STATES == WORKING then inflow else false;
end
```

#### 1.2. Modélisation des contraintes

La prise en compte d'un nombre limité d'opérateurs de maintenance ou bien de la disponibilité limitée d'un banc de tests est réalisée par l'intermédiaire de contraintes.

Une contrainte est ajoutée en deux étapes. Premièrement, nous créons un bloc à notre modèle comprenant les états de la contrainte. Puis, nous modifions la classe « RepairableComponent » afin de la synchroniser avec la contrainte.

Le graphe d'états d'une contrainte représentant la disponibilité d'une équipe de maintenance est affichée ci-après.

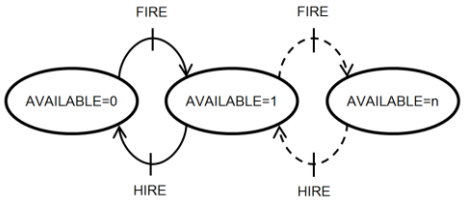



**Figure 6** - Comportement d'une contrainte

Le code AltaRica 3.0 correspondant à la contrainte est présenté ci-après.

```
class QuantityConstraint
    Integer AVAILABLE (init = 1000);
end
```

Il est nécessaire de modifier la classe «RepairableComponent » afin de la lier avec la contrainte. Pour cela, nous lui ajoutons un état supplémentaire.

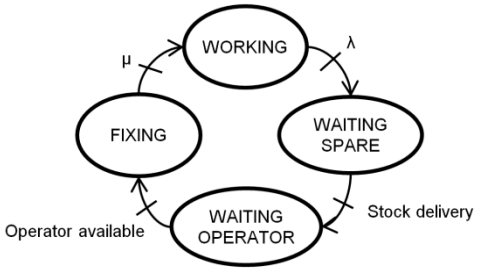



**Figure 4** - Comportement d'un élément avec contraintes

Le code AltaRica 3.0 de la classe « RepairableComponent » devient :

```
class RepairableComponent
  STATES_LRU  STATES (init = WORKING);
  Boolean inflow, outflow (reset = false);
  parameter Real lambda = Lambda_defaut;
  parameter Real mu = Mu_defaut;
  parameter Real phi = Phi_defaut;
  event failure (delay = exponential(lambda));
  event operatorAvailable (delay = Dirac(0));
  event delivery(delay = Dirac(0), hidden = true) ;
  event repair (delay = exponential(mu));

  transition
    failure : STATES == WORKING -> STATES := WAITING_SPARE;
    delivery : STATES == WAITING_SPARE -> STATES := WAITING_OP;
    operatorAvailable : STATES == WAITING_OP and
                        main.OP.AVAILABLE > 0 -> { STATES := FIXING;
                        main.OP.AVAILABLE := main.OP.AVAILABLE - 1;}
    repair : STATES == FIXING -> {  STATES := WORKING;
                        main.OP.AVAILABLE := main.OP.AVAILABLE + 1;}

  assertion
    outflow := if STATES == WORKING then inflow else false;
end
```

#### 1.2. Modélisation des performances de diagnostic

En fonction des technologies, diagnostiquée une panne peut être une tâche longue et délicate nécessitant d'une part des opérateurs très qualifiés et d'autre part des moyens de tests lourds.

Le temps et les ressources nécessaires pour mener les tâches de diagnostic peuvent donc devenir non négligeables dans l'estimation de la disponibilité prévisionnelle et des coûts.

Ainsi, nous proposons l'ajout de deux états supplémentaires à la classe « RepairableComponent ». Un premier état symbolise l'attente de la disponibilité du moyen de tests. Celui-ci sera « synchronisé » avec une contrainte telle que décrite au paragraphe précédent, modélisant la quantité de moyens de tests disponibles.
Puis, un deuxième état représente la durée d'identification de l'élément en panne. Celle-ci suivra une loi exponentielle de paramètre « phi ».

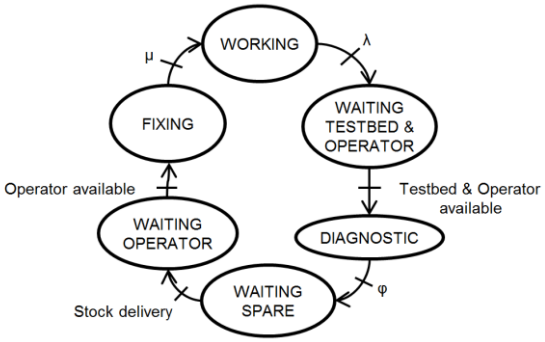



**Figure 7** - Comportement d'un élément avec diagnostic

Le code AltaRica 3.0 correspondant au graphe ci-dessus est le suivant.

```
class RepairableComponent
  STATES_LRU  STATES (init = WORKING);
  Boolean inflow, outflow (reset = false);
  parameter Real lambda = Lambda_defaut;
  parameter Real mu = Mu_defaut;
  parameter Real phi = Phi_defaut;
  event failure (delay = exponential(lambda));
  event operatorAvailable (delay = Dirac(0));
  event operatorTestbedAvailable (delay = Dirac(0));
  event localisation (delay = exponential(phi));
  event delivery(delay = Dirac(0), hidden = true) ;
  event repair (delay = exponential(mu));

  transition
    failure : STATES == WORKING -> STATES := WAITING_OP_TESTBED;
    operatorTestbedAvailable : STATES == WAITING_OP_TESTBED and
                               main.OP.AVAILABLE > 0 and
                               main.TEST.AVAILABLE > 0 -> {   STATES := DIAGNOSTIC;
                                                              main.OP.AVAILABLE := main.OP.AVAILABLE - 1;
                                                              main.TEST.AVAILABLE := main.TEST.AVAILABLE - 1;}
    localisation : STATES == DIAGNOSTIC -> {    STATES := WAITING SPARE;
                                                main.OP.AVAILABLE := main.OP.AVAILABLE + 1;
                                                main.TEST.AVAILABLE := main.TEST.AVAILABLE + 1; }
    delivery : STATES == WAITING_SPARE -> STATES := WAITING_OP;
    operatorAvailable : STATES == WAITING_OP and main.OP.AVAILABLE > 0 -> { STATES := FIXING;
                                                                            main.OP.AVAILABLE := main.OP.AVAILABLE - 1;}
    repair : STATES == FIXING -> { STATES := WORKING; main.OP.AVAILABLE := main.OP.AVAILABLE + 1;}

  assertion
    outflow := if STATES == WORKING then inflow else false;
end
```

### 1.2. Modélisation des performances d'accessibilité

Suivant les technologies et les types de porteurs, les délais d'accessibilité, lors de l'échange d'un élément défectueux, peuvent être important.

Il est donc intéressant de prendre en compte ce paramètre lors des simulations. Pour cela, nous avons simplement additionné le temps de réparation « MTTR » avec le temps d'accessibilité qui diffère d'un porteur à l'autre.

## 2. Modélisation des stocks

### 2.1. Principe général

La gestion des stocks de pièces de rechange est un élément important pour l'atteinte d'un niveau de disponibilité élevé mais est aussi une des causes d'un coût global de possession important. Il est donc nécessaire de le dimensionner au plus juste.

Le principe que nous avons retenu, est de définir une classe modélisant chacun des stocks puis de synchroniser l'évènement « Stock delivery » de la classe « RepairableComponant » vue précédemment, à la condition qu'au moins une pièce soit disponible dans au moins un stock.

Nous avons fait le choix d'utiliser des synchronisations dans le sens AltaRica 3.0 afin de pouvoir spécifier un délai d'approvisionnement différent en fonction du lieu du stock.

Dans notre cas applicatif, nous avons trois sources d'approvisionnement :

1. « Mission » avec un délai d'approvisionnement nul;
2. « Base » avec un délai d'approvisionnement supérieur à celui du stock « Mission »;
3. « Industriel » avec un délai d'approvisionnement supérieur à celui du stock « Base ».

Les graphes d'états des stocks « Mission » et « Base » sont identiques et présentés ci-dessous.

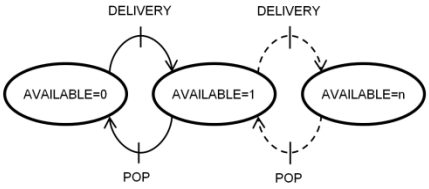



**Figure 8** - Graphe d'état des stocks

Le code AltaRica 3.0 correspondant à la modélisation des stocks est celui-ci.

```
class Stock
    Integer COUNT (init = 1);
    Integer MAX (init = 1);
    event delivery (delay = Dirac(0.0), hidden = true) ;
    event pop (delay = Dirac(0.0), hidden = true) ;

    transition
        delivery : COUNT < MAX -> COUNT := COUNT + 1;
        pop : COUNT > 0 -> COUNT := COUNT - 1;
end
```

Créer une synchronisation en AltaRica 3.0 est une tâche simple. Elle nécessite, dans notre cas, deux actions. Premièrement, il est nécessaire de s'assurer que les évènements qui seront synchronisés soient bien déclarés avec le paramètre « hidden = true » afin qu'ils ne soient pas exécutés directement par le simulateur stochastique (confer code AltaRica des classes « Stock » et « RepairableComponent »). La deuxième étape est la déclaration de l'évènement déclencheur de la synchronisation ainsi que la déclaration de la synchronisation en elle-même.

Le code AltaRica 3.0 suivant représente la déclaration d'un élément réparable, d'un stock associé ainsi que de la synchronisation.

```
block LRU
  Boolean inflow, outflow (reset = false);
  //Stock and Component initialisation
  Stock StockBase(COUNT.init = 2 , MAX.init = 2);
  RepairableComponent Composent (lambda= 1e-6, mu= 0.125);
  //Synchronization trigger (delay = 24h)
  event Stock_delibery (delay = Dirac (24));
  transition
    Stock_delivery : !Component.delivery &! StockBase.pop;
  assertion
    Component.inflow := inflow;
    outflow := Component.outflow
end
```

### 2.2. Gestion des quantités d'éléments identiques

Il est courant qu'un composant soit présent en quantité dans un même système industriel. Deux solutions sont possibles pour tenir compte de cela. Premièrement, il est envisageable d'intégrer la quantité directement dans le paramètre « lambda » du composant en multipliant le taux de défaillance par la quantité d'éléments identiques. La deuxième solution est d'instancier autant de fois la classe « RepairableComponant » que nécessaire en la synchronisant au même stock. Même si la deuxième solution augmente considérablement la taille du modèle, nous la privilégions à la première solution. Car, dans le cas de la première solution, si un des composants est en panne et tant que celui-ci ne sera pas réparé, les autres composants ne pourront tomber en panne. Dit autrement, les composants ne sont plus indépendants. Si la quantité de composants identiques est importante et si les délais d'approvisionnement et de réparation sont importants, la simulation donnera un résultat optimiste.

## 3. Dimensionnement des stocks

### 3.1. Principe général

Maîtriser les coûts de maintien en condition opérationnelle tout en s'engageant sur une exigence de disponibilité, passe par la mise en place de stocks ainsi que par leur juste dimensionnement. Sous-dimensionner un stock, entrainera une baisse forte de la disponibilité et, à contrario, un surdimensionnement entrainera des coûts prohibitifs. Il s'agit d'un sujet très largement discuté dans la littérature. Nous nous limiterons ici à proposer seulement une méthode permettant, par l'utilisation de simulations « Monte-Carlo » sur le modèle présenté dans les paragraphes précédents, d'obtenir une solution.

### 3.2 Algorithme

La méthode que nous avons appliquée est une méthode dite « Brute Force ». La méthode « Brute Force » est une approche complète des problèmes. La solution est trouvée en analysant tous les cas possibles jusqu'à la découverte de la solution.

Le principe est de démarrer avec tous les stocks vides. Pour chacun des éléments remplaçables et pour chaque stock, nous incrémentons d'un rechange. Nous calculons ensuite la nouvelle disponibilité obtenue puis nous supprimons la rechange du stock. Lorsque tous les cas sont réalisés, nous retenons l'élément remplaçable dont le rechange, ajouté à un des stocks, nous a permis d'obtenir la meilleure disponibilité. Ce rechange est mis définitivement en stock et nous reprenons depuis le début. Nous réalisons cette boucle jusqu'à ce que nous obtenions la disponibilité souhaitée.

Le logigramme suivant décrit graphiquement l'algorithme employé.

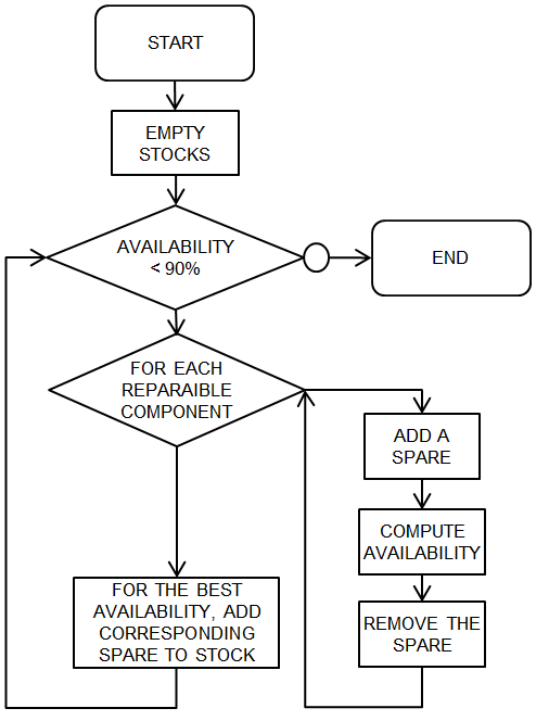



**Figure 9** - Algorithme de dimensionnement des stocks

### 3.3 Résultats

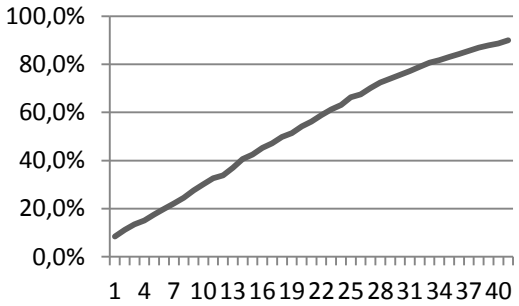
Les analyses ont été réalisées par le simulateur stochastique d'OpenAltaRica. Le temps de mission retenu est 10ans et le nombre d'histoires simulées est de $10^5$.

Une simulation de $10^5$ histoires prend environ 1mn sur un ordinateur de bureau.

Le tableau ci-après présente les enchaînements des simulations en retenant à chaque fois la configuration « stock + rechange » apportant le meilleur gain de disponibilité.

| ITERATION | COMPOSANT RETENU | STOCK RETENU | DISPO |
|---|---|---|---|
| 1 | C1 | MISSION | 8,4% |
| 2 | C2 | MISSION | 11,3% |
| 3 | C2 | BASE | 13,5% |
| 4 | C2 | MISSION | 15,0% |
| 5 | C3 | MISSION | 17,6% |
| 6 | C4 | MISSION | 19,8% |
| 7 | C5 | BASE | 22,1% |
| 8 | C1 | BASE | 24,5% |
| 9 | C4 | BASE | 27,6% |
| 10 | C6 | BASE | 30,1% |
| 11 | C7 | MISSION | 32,6% |
| 12 | C4 | MISSION | 33,8% |
| 13 | C8 | MISSION | 37,0% |
| 14 | C9 | MISSION | 40,6% |
| 15 | C10 | BASE | 42,6% |
| 16 | C11 | MISSION | 45,3% |
| 17 | C12 | MISSION | 47,1% |
| 18 | C13 | MISSION | 49,9% |
| 19 | C14 | BASE | 51,4% |
| 20 | C2 | MISSION | 54,2% |
| 21 | C15 | MISSION | 56,2% |
| 22 | C16 | MISSION | 58,8% |
| 23 | C1 | MISSION | 61,3% |
| 24 | C5 | MISSION | 63,2% |
| 25 | C17 | MISSION | 66,3% |
| 26 | C4 | MISSION | 67,5% |
| 27 | C18 | MISSION | 70,2% |
| 28 | C19 | MISSION | 72,4% |
| 29 | C2 | MISSION | 74,0% |
| 30 | C20 | MISSION | 75,6% |
| 31 | C10 | MISSION | 77,2% |
| 32 | C21 | MISSION | 79,0% |
| 33 | C22 | MISSION | 80,7% |
| 34 | C23 | BASE | 81,8% |
| 35 | C8 | BASE | 83,1% |
| 36 | C6 | MISSION | 84,3% |
| 37 | C4 | MISSION | 85,6% |
| 38 | C2 | BASE | 86,9% |
| 39 | C12 | BASE | 87,9% |
| 40 | C24 | MISSION | 88,7% |
| 41 | C25 | MISSION | 90,0% |

L'évolution de la disponibilité en fonction du nombre d'itérations est présentée dans le graphique ci-après.




La configuration complète des stocks pour obtenir une disponibilité moyenne à 90% sur 10 ans est la suivante :

| COMPOSANT | STOCKS | | |
|---|---|---|---|
| | MISSION | BASE | TOTAL |
| C1 | 2 | 1 | 3 |
| C10 | 1 | 1 | 2 |
| C11 | 1 | 0 | 1 |
| C12 | 1 | 1 | 2 |
| C13 | 1 | 0 | 1 |
| C14 | 0 | 1 | 1 |
| C15 | 1 | 0 | 1 |
| C16 | 1 | 0 | 1 |
| C17 | 1 | 0 | 1 |
| C18 | 1 | 0 | 1 |
| C19 | 1 | 0 | 1 |
| C2 | 4 | 2 | 6 |
| C20 | 1 | 0 | 1 |
| C21 | 1 | 0 | 1 |
| C22 | 1 | 0 | 1 |
| C23 | 0 | 1 | 1 |
| C24 | 1 | 0 | 1 |
| C25 | 1 | 0 | 1 |
| C3 | 1 | 0 | 1 |
| C4 | 4 | 1 | 5 |
| C5 | 1 | 1 | 2 |
| C6 | 1 | 1 | 2 |
| C7 | 1 | 0 | 1 |
| C8 | 1 | 1 | 2 |
| C9 | 1 | 0 | 1 |
| | | TOTAL: | 41 |

## Conclusion

La méthode que nous vous présentons permet d'optimiser la chaîne du maintien en condition opérationnelle au travers de l'ingénierie dirigée par les modèles, dans le but d'assurer une démonstration du niveau de disponibilité prévisionnelle. Cette méthode permet ainsi de prendre en compte un nombre plus important de contraintes opérationnelles afin d'obtenir une valeur de la disponibilité plus représentative des attentes du client.

Les outils de modélisation et de simulation AltaRica sont de plus en plus accessibles pour les ingénieurs de la Sûreté de fonctionnement et du Soutien. Les performances des simulateurs stochastiques autorisent la réalisation de modèles plus complexes et l'utilisation de méthodes non optimisée telle que la méthode « Brute Force ». Par ailleurs, les évolutions apportées par la version 3 d'AltaRica telles que la gestion des classes et l'héritage, apportent la possibilité de structurer et d'organiser les modèles pour les rendre plus simples à valider, à capitaliser et à réutiliser.

## 8 Références

**(Boiteau *et al.*, 2006)**
M. BOITEAU, Y. DUTUIT, A RAUZY, J-P SIGNORET, 2006 The AltaRica Data-Flow language in use : Assessment of Production Availability of a MultiStates System, Reliability Engineering and System Safety, 91:747-755.

**(Prosvirnova *et al.*, 2013)**
T. PROSVIRNOVA, M. BATTEUX, P-A BRAMERET, A. CHERFI, T. FRIEDLHUBER, J-M ROUSSEL, A. RAUZY, 2013, The AltaRica 3.0 Project for Model-Based Safety Assessment, in Proceedings of 4th IFAC Workshop on Dependable Control of Discrete Systems, DCDS 2013, York (Great Britain).

**(Riera et al.,2012)**
D. RIERA, F. MILCENT, J. PARISOT, E. CLEMENT, 2012, Modélisation dynamique en Sûreté de Fonctionnement : une avancée pour l'analyse des systèmes complexes, Actes du Congrès Lambda-Mu 18.

**(Rauzy, 2008)**
RAUZY A. (2008). Guarded transition systems: a new states/events formalism for reliability studies

**(Batteux et *al.* 2015)**
BATTEUX, M.; PROSVIRNOVA,T.; RAUZY,A. (2015). System Structure Modeling Language (S2ML).

**(Valiant 1979)**
VALIANT, L. (1979). The Complexity of Computing the Permanent

**(Cassandras et al., 2008)**
CASSANDRAS, C.; LAFORTUNE, S. (2008), Introduction to Discrete Event Systems, 2nd Edition, Springer - Verlag.