# Processus d'identification de contraintes de sécurité innocuité vérifiables en ligne pour des systèmes autonomes critiques

Amina Mekki-Mokhtar

▶ **To cite this version:**

Amina Mekki-Mokhtar. Processus d'identification de contraintes de sécurité innocuité vérifiables en ligne pour des systèmes autonomes critiques. Autre [cs.OH]. Université Paul Sabatier - Toulouse III, 2012. Français. NNT : . tel-00800859

**HAL Id: tel-00800859**

**https://theses.hal.science/tel-00800859**

Submitted on 14 Mar 2013

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# THÈSE

**En vue de l'obtention du**

## DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

**Délivré par:**
Université Toulouse 3 Paul Sabatier (UT3 Paul Sabatier)

**Discipline ou spécialité:**
Informatique et Robotique

---

**Présentée et soutenue par:**
Amina MEKKI MOKHTAR

**le :** 12 décembre 2012

**Titre:**
Processus d'identification de propriétés de sécurité-innocuité vérifiables en ligne pour des systèmes autonomes critiques

---

**JURY**

| | | |
|---|---|---|
| *Rapporteurs :* | Françoise SIMONOT-LION | Professeur des Universités |
| | Jean-Marc THIRIET | Professeur des Universités |
| *Examinateurs :* | Guy JUANOLE | Professeur des Universités Émérite |
| | David ANDREU | Maître de Conférences |
| *Invité :* | Jean-Paul BLANQUART | Ingénieur Astrium |
| *Directeurs de Thèse :* | David POWELL | Directeur de Recherche |
| | Jérémie GUIOCHET | Maître de Conférences |

---

**École doctorale:**
Systèmes (EDSYS)
**Unité de recherche:**
LAAS-CNRS
**Directeur(s) de Thèse:**
David POWELL et Jérémie GUIOCHET
**Rapporteurs:**
Françoise SIMONOT-LION et Jean-Marc THIRIET

# Avant-propos

Les travaux de recherche présentés dans ce manuscrit ont été réalisés au Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS). Je remercie les directeurs successifs du LAAS-CNRS, Messieurs Raja Chatila, Jean-Louis Sanchez et Jean Arlat, de m'avoir accueillie dans ce laboratoire. Je remercie également Madame Karama Kanoun de m'avoir reçu avec autant de bienveillance au sein de l'équipe Tolérance aux fautes et Sûreté de Fonctionnement informatique (TSF). Je remercie aussi Monsieur Guy Juanole, Professeur Emérite à l'Université Paul Sabatier d'avoir présidé mon jury de thèse, ainsi que :

- Madame Françoise Simonot-Lion, Professeur des Universités et Directrice du laboratoire Lorrain de Recherche en Informatique et ses Applications (LORIA),
- Monsieur Jean-Marc Thiriet, Professeur des Universités et Directeur du laboratoire Grenoble Images Parole Signal Automatique (gipsa-lab),
- Monsieur David Andreu, Maître de Conférence à l'Université de Montpellier 2,
- Monsieur Jean-Paul Blanquart, Ingénieur d'études à Astrium, Toulouse,

pour avoir participé à ce jury. Je remercie en particulier Madame Françoise Simonot-Lion et Monsieur Jean-Marc Thiriet qui ont accepté la charge d'être rapporteurs.

Un grand merci à mes encadrants de thèse : David Powell et Jérémie Guiochet. Merci à David de m'avoir appris à être rigoureuse et critique par rapport au travail fourni. Je remercie également Jérémie pour son dynamisme, sa disponibilité et son aide plus que précieuse. Je remercie particulièrement Jean-Paul, qui a également suivi mes travaux, pour ses remarques toujours très pertinentes, ses conseils et aussi pour son humour légendaire.

Merci aux membres du groupe TSF et plus particulièrement les membres des bureaux A10 et A11 pour l'ambiance de travail très conviviale.

Je remercie bien évidemment mes parents qui ont toujours cru en moi et qui ont fait en sorte que je sois là où je suis aujourd'hui. Une pensée à ma sœur, à mes neveux, aux amis de Toulouse et de partout dans le monde, qui ont fait que ces dernières années soient inoubliables.

# Table des matières

# Introduction

Les progrès en matière d'autonomie des systèmes robotiques ont fait que ces derniers sont de plus en plus utilisés dans des domaines comme la robotique de service, l'exploration de zones dangereuses (zones irradiées par exemple), etc. Ceci explique l'engouement de la communauté scientifique dont le but est de développer des systèmes autonomes performants opérant dans des environnements inconnus et dynamiques.

Ces progrès en matière d'autonomie sont principalement dus aux avancées, lors de ces dernières décennies, de l'intelligence artificielle et, plus précisément, des mécanismes décisionnels. Ces mécanismes ont les particularités d'être relativement complexes, et de faire appel à des heuristiques pour améliorer leur performance, ce qui rend leur vérification difficile.

La complexité des systèmes autonomes et l'environnement incertain dans lequel ils opèrent font que les méthodes traditionnelles de sûreté de fonctionnement doivent être complétées par des méthodes appropriées. La vérification en ligne par moniteur de sécurité apparaît comme une solution complémentaire attractive permettant d'offrir un comportement sûr malgré la présence de fautes et d'incertitudes. Elle permet d'assurer la sécurité-innocuité du système autonome en enclenchant des actions de recouvrement lorsque le système se trouve dans une situation dangereuse. Un moniteur de sécurité permet de vérifier que, lors de l'exécution, un ensemble de propriétés de sécurité sont respectées. L'efficacité d'un tel dispositif repose principalement sur les propriétés de sécurité à vérifier.

La problématique de nos travaux de recherche s'articule autour de la spécification de ces propriétés de sécurité. Etant donné que ces propriétés visent à assurer la sécurité-innocuité de systèmes autonomes critiques, il est essentiel de définir un processus clair et concis pour leur spécification. Les travaux abordant la vérification par moniteur de sécurité se focalisent majoritairement sur l'implémentation de propriétés déjà définies ainsi que sur des aspects architecturaux du moniteur. Par conséquent, nous avons focalisé nos recherches sur le processus d'élicitation et de formalisation des propriétés à vérifier.

Le premier chapitre de ce mémoire présente les systèmes autonomes ainsi que les concepts et la terminologie de la sûreté de fonctionnement. Il aborde quelques méthodes de sûreté de fonctionnement existantes qui ont pour objectif d'accroître la confiance que l'on peut avoir en des systèmes autonomes. Il introduit un état de l'art relatif à des systèmes autonomes implémentant des dispositifs de sécurité et discute des choix de mise en œuvre de ces derniers.

Le deuxième chapitre présente notre méthodologie de génération de contraintes de

sécurité à partir d'une analyse de risque du système fonctionnel. Pour cela, il commence par aborder les concepts relatifs au risque et à la gestion du risque, ainsi que des méthodes d'analyse de risque existantes. Il explique par la suite, la motivation quant à l'utilisation de la méthode que nous avons retenue : HazOp/UML. Pour finir, il expose en détail les étapes permettant de générer les contraintes de sécurité.

Le troisième chapitre commence par présenter notre cadre conceptuel. Il introduit le concept d'*état d'alerte* au sein duquel une action de sécurité devra être enclenchée afin de remettre le système dans un état sûr. Il expose le processus permettant la spécification des états d'alerte à partir des contraintes de sécurité précédemment identifiées. Dans le cas où de tels états ne peuvent pas être définis, il propose une alternative afin d'empêcher que le système n'atteigne un état catastrophique. Enfin, on propose une méthode de détection d'actions qui peuvent être enclenchées simultanément dans l'objectif d'éviter toute incohérence lors de l'exécution.

Le quatrième chapitre présente un cas d'étude qui a permis de valider l'approche proposée. Le chapitre applique la méthodologie proposée à un robot d'aide à la déambulation, en s'appuyant sur des analyses de risque antérieures. Pour finir, il dresse le bilan de l'application de la méthode ainsi que des entraves rencontrées.

Enfin, nous concluons ce manuscrit en retraçant les points essentiels, en faisant le bilan des apports de nos travaux de recherche et en exposant quelques perspectives identifiées pour la suite de ces travaux.

# Chapitre 1

# Dispositifs de surveillance des systèmes autonomes

## Introduction

Grâce aux progrès des mécanismes décisionnels, les systèmes réactifs sont de plus en plus autonomes. Cette autonomie élargit leur champ d'application : robotique de service d'aide à la personne, exploration spatiale, missions de sauvetage dans des conditions extrêmes, etc. Ces applications ont la particularité d'être critiques ; une défaillance pourrait mener à des pertes humaines ou financières. Les méthodologies de la sûreté de fonctionnement permettent d'avoir une confiance justifiée dans de tels systèmes.

Une des techniques de la sûreté de fonctionnement est l'utilisation de dispositifs qui surveillent les systèmes autonomes et qui vérifient que l'exécution ne viole pas un certain nombre de propriétés de sécurité. Dans ce chapitre, nous présentons tout d'abord les systèmes autonomes ainsi que les menaces auxquelles ils sont exposés. Ensuite, nous abordons la terminologie et les concepts relatifs à la sûreté de fonctionnement. Nous présentons les principales techniques qui sont utilisées pour augmenter la sécurité des systèmes autonomes. Parmi ces techniques, nous présenterons des exemples de systèmes de surveillance en ligne utilisés dans le domaine des systèmes autonomes.

## 1.1 L'autonomie

Cette section a pour objectif d'introduire les *systèmes autonomes* et de présenter des exemples d'architectures décisionnelles.

### 1.1.1 Définitions

Il existe dans la littérature plusieurs définitions de l'autonomie ; nous en avons retenu deux qui nous paraissaient les plus appropriées à l'autonomie d'un système robotique :

1. « un système autonome est capable de raisonner et de prendre des décisions afin d'atteindre des objectifs en se basant sur ses connaissances et sa perception de l'environnement fluctuant dans lequel il évolue » [BBGP07].

2. « l'autonomie est la capacité d'un système à percevoir, analyser, communiquer, planifier, établir des décisions et agir afin d'atteindre des objectifs assignés par un opérateur humain ; cette autonomie est mesurée par les aptitudes du système selon divers facteurs incluant la complexité de la mission, la difficulté de l'environnement et les interactions désirées ou non désirées avec différentes catégories d'êtres humains (opérateurs, co-équipiers, passants) » [Hua04].

Ces deux définitions mettent en avant les notions de perception, de prise de décision, et d'action dans un environnement dynamique. Un système autonome comporte typiquement :

- un planificateur qui ordonnance les tâches à effectuer pour atteindre un objectif donné,
- un composant d'exécution qui raffine les plans haut-niveau en actions élémentaires à exécuter,
- des capteurs et des actionneurs qui permettent de percevoir l'environnement et d'agir.

Un système autonome peut aussi inclure un mécanisme d'apprentissage afin d'adapter son comportement aux situations rencontrées.

En résumé, un système autonome possède des fonctionnalités délibératives basées sur les mécanismes décisionnels afin de prendre les bonnes décisions. Ces décisions ne peuvent pas être calculées en amont vu l'environnement complexe et incertain dans lequel le système autonome opère ; ces décisions doivent être prises en temps-réel lors de la phase opérationnelle. Les systèmes automatiques se basent, quand à eux, sur la théorie du contrôle afin d'atteindre des objectifs assignés par l'opérateur. Cette différence est due à la complexité des tâches à effectuer et à l'environnement dynamique. La limite entre systèmes autonomes et systèmes automatiques n'est pas franche : entre un système réactif totalement automatique et un système délibératif totalement autonome, il y a un large spectre de systèmes à prendre en compte.

Dans la suite du mémoire, nous considérerons qu'un système est autonome lorsqu'il requiert un processus décisionnel explorant un espace d'états afin d'effectuer sa mission. Nous nous focalisons sur cet aspect des systèmes autonomes afin de faire face aux dangers engendrés par cette fonctionnalité décisionnelle. Trois critères caractérisent les mécanismes décisionnels [LCI+04] : la *solvabilité*, ou la capacité à produire des résultats corrects ; la *complétude*, qui indique que si une conclusion correcte existe, elle sera inéluctablement inférée ; et la *calculabilité*, qui indique si le mécanisme décisionnel peut résoudre un problème en temps et en espace polynomiaux [1].

1. C'est-à-dire, si le temps et l'espace mémoire nécessaires au mécanisme pour trouver une solution sont du même ordre qu'une fonction polynomiale dépendant de la taille du problème posé.

### 1.1.2 Illustrations

Dans cette section, nous allons présenter deux modèles d'autonomie : le premier est le *modèle domino*, un modèle haut-niveau d'un processus de raisonnement. Le second est l'architecture à trois-niveaux pour le contrôle des systèmes autonomes.

#### Le "modèle domino"

Ce modèle a été proposé dans [FD00] pour modéliser une entité autonome de prescription de traitements médicaux (voir Figure 1.1). Il modélise dans un diagramme un processus décisionnel générique ; les nœuds représentent des bases de connaissances, les transitions représentent les procédures de déduction qui effectuent un pas en avant dans le processus.

Ce modèle est générique dans le sens où il peut être appliqué dans différentes situations. Les bases de connaissances et les procédures d'inférence ne sont pas présentées en détail ; elles doivent être implémentées d'une manière adéquate dans le contexte choisi. Le modèle commence par une base de connaissances *Connaissances de la situation* qui, dans un système de prescriptions, pourrait représenter les symptômes, le traitement médical actuel, etc. Ces informations doivent mener aux objectifs via la transition *Définition du problème*. La base de connaissances *Objectifs de résolution* contient les objectifs de la prise de décision : diagnostique, traitement, prescription etc. Une fois les objectifs identifiés, un ensemble de solutions est proposé : *Solutions candidates*. Ces propositions sont basées sur l'historique médical du patient ainsi que sur les données médicales. A partir de cette base de propositions, un ensemble des décisions possibles est constitué : *Décisions*, accompagnées de l'argumentation qui mène à chaque décision. Ces décisions potentielles sont présentées au médecin sous forme d'une liste de recommandations accompagnées de justifications. Elles peuvent, soit enrichir la base de données *Connaissances de la situation*, soit engendrer un plan (par exemple : traitement par chimiothérapie, chirurgie, etc.). Pour finir, ce plan est décomposé en actions élémentaires : *Actions* dont les résultats viennent s'ajouter à la base de données *Connaissances de la situation*.

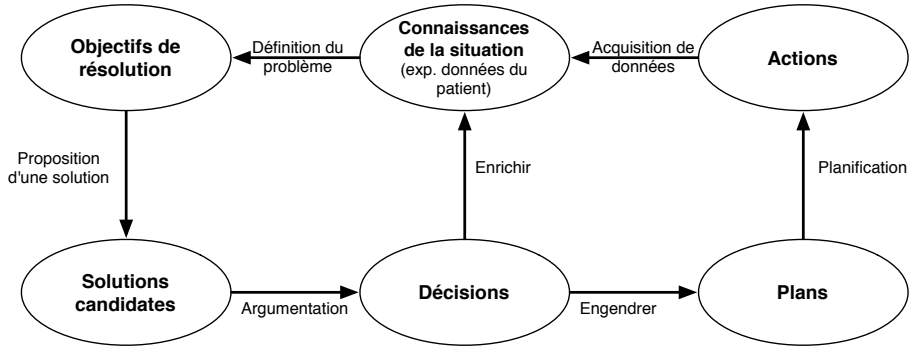


FIGURE 1.1 – Le "modèle domino", adapté de [FD00]

Malgré l'aspect générique du *domino model*, il peut être associé partiellement à un logiciel décisionnel d'un système autonome. Par exemple, *Proposition d'une solution,*

*Argumentation* peuvent représenter la recherche d'une solution dans un espace d'états ; *Planification* peut représenter l'adoption d'un plan spécifique ; *Acquisition de données* peut représenter la perception de l'environnement à partir de capteurs, etc.

### L'architecture trois niveaux

L'architecture trois niveaux a été proposé dans les années '90 pour structurer le contrôle d'un robot mobile en trois composants : un mécanisme de commande réactive, un mécanisme d'exécution de plans, et un mécanisme pour effectuer des calculs délibératifs longs [Gat97]. L'architecture LAAS est un exemple typique de cette approche. Il s'agit d'une architecture hiérarchique dotée de couches de différents niveaux d'abstraction [ACF+98] (voir Figure 1.2) :

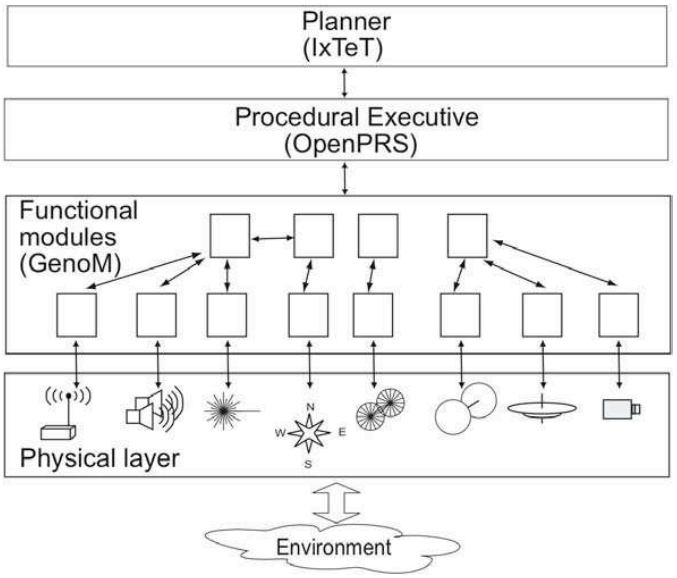


FIGURE 1.2 – L'architecture robotique du LAAS en trois couches

– Une couche *décisionnelle* incluant un planificateur IxTeT-eXeC. Afin de produire un plan pour atteindre des objectifs, IxTeT-eXeC s'appuie sur un solveur du problème de satisfaction de contrainte (CSP) en utilisant un modèle. Ce modèle est un ensemble de contraintes qui décrivent la façon dont le robot peut atteindre ses objectifs : contraintes de positionnement, contraintes d'acquisition, contraintes d'actions, contraintes d'ordre, etc. La recherche d'une solution a été accélérée par l'utilisation d'heuristiques. Les heuristiques permettent de toujours obtenir une solution si elle existe, néanmoins elles n'offrent aucune garantie quant au temps nécessaire pour la trouver.

– Une couche *exécution* : OpenPRS est en charge de la supervision de l'exécution du plan. Il effectue des requêtes haut-niveau afin d'obtenir un ensemble d'actions de bas-niveau à exécuter par la couche fonctionnelle.

– Une couche *fonctionnelle* qui gère les fonctionnalités de bas-niveau. Elle se compose d'un ensemble de modules communicants où chaque module offre un ensemble de services en réponse à des requêtes et émet des rapports à la couche supérieure.

### 1.1.3 Fragilité des systèmes autonomes

L'autonomie soulève des problématiques de sûreté de fonctionnement dues à l'environnement dynamique et inconnu dans lequel le système opère. L'autonomie induit aussi des logiciels de contrôle complexes qui sont difficiles à valider vu l'explosion du nombre d'états. Pour ces mêmes raisons, il est difficile d'évaluer la complétude des cas de tests effectués. De plus, l'utilisation des heuristiques afin d'accélérer la recherche de solution induit une complexité supplémentaire ; le système décisionnel devient imprévisible, ce qui rend le test et la vérification encore plus difficiles.

La couche décisionnelle est particulièrement vulnérable car elle produit et raffine les commandes ; ce qui pourrait être particulièrement dangereux.

En s'inspirant des travaux présentés dans [ALRL04] et [PTF02], Baudin et *al.* ont proposé, dans [BBGP07], de classifier les menaces que peuvent rencontrer les systèmes autonomes comme suit :

1. Les menaces endogènes, provenant du système autonome lui même :
   – Fautes de développement : introduites lors de la modélisation du système ou lors de l'implémentation.
   – Fautes physiques se produisant pendant la phase opérationnelle et affectant les ressources physiques (capteurs, actionneurs, processeurs, sources d'énergie).
2. Les menaces exogènes, provenant de l'environnement physique, logique et humain du système autonome :
   – Fautes externes comme les interférences, les attaques malveillantes et les fautes de coopération inter-systèmes (dont les fautes d'interaction avec l'humain).
   – Mauvaise perception de l'environnement (manque d'observabilité, capteurs imparfaits, etc.).
   – Les aléas de l'environnement (événements incontrôlables, circonstances imprévues, etc.).

L'environnement dynamique est pris en compte par les mécanismes décisionnels en utilisant des techniques de l'intelligence artificielle comme les systèmes experts, la planification à l'aide de solveurs, les réseaux Bayésiens, les réseaux de neurones artificiels, le raisonnement basé modèle etc. Ces mécanismes prennent des décisions selon leur perception de l'environnement, leur perception de leur état interne, et selon les moyens disponibles pour atteindre leurs objectifs. Ces processus décisionnels comportent des connaissances spécifiques au domaine et des mécanismes d'inférence sur ces connaissances. Ces techniques conduisent à considérer plusieurs classes de fautes spécifiques [PTF02] :

1. Fautes propres au mécanisme décisionnel :
   – une base de connaissances erronée, incomplète ou inconsistante,

- une inférence non-valide : la base de connaissances est correcte mais l'inférence produite peut être fausse à cause d'une procédure d'inférence utilisée d'une façon incorrecte,
- les imprévus : la base de connaissances peut être correcte, mais le raisonnement qui y est appliqué peut ne pas aboutir lorsqu'il est confronté à une situation inhabituelle, non prévue par le concepteur,
- spécificité des critères de décision : les critères de décision intégrés dans le système peuvent ne pas être universellement acceptables, c'est-à-dire, ils pourraient avoir des effets secondaires indésirables dans certaines situations.

2. Fautes d'interaction entre le mécanisme décisionnel et l'humain :
   - incompatibilité ontologique : la base de connaissances peut être correcte, mais une discordance se produit entre le sens du terme tel qu'il est utilisé par le système et le sens que l'utilisateur lui attribue,
   - un *optimisme* déplacé dans les capacités du système : l'utilisateur place une trop grande confiance dans les décisions prises par le mécanisme décisionnel ; par exemple les résultats donnés par le mécanisme décisionnel sont très précis, conférant une illusion d'exactitude, mais faux,
   - une *incrédulité* dans les capacités du système : l'utilisateur n'a pas confiance dans les décisions du système, par exemple si le processus de raisonnement du système lui paraît incompréhensible.

## 1.2 Principes généraux de la sûreté de fonctionnement informatique

Avižienis, Laprie *et al.* ont défini, dans [ALRL04, LAB+96], la sûreté de fonctionnement d'un système comme étant « son aptitude à délivrer un service de confiance justifiée ». Cette définition souligne la nécessité que la confiance qu'on a en un système puisse être justifiée. Comme présenté dans la Figure 1.3 [LAB+96], la sûreté de fonctionnement est spécifiée selon ses *attributs*, ses *entraves* et les *moyens* de l'assurer.

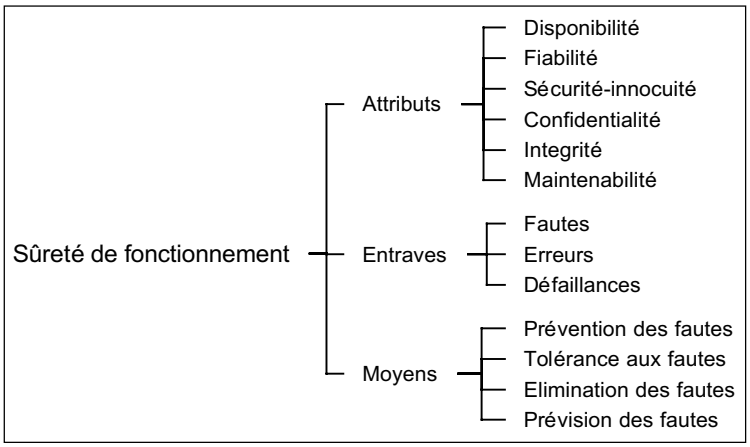


FIGURE 1.3 – L'arbre de la sûreté de fonctionnement, [LAB+96]

### 1.2.1 Les attributs

Selon l'application à laquelle le système est destiné, l'accent peut être mis sur différentes facettes de la sûreté de fonctionnement. La sûreté de fonctionnement peut être considérée selon différentes propriétés complémentaires, qui définissent ses *attributs* :

- la *disponibilité* : est le fait qu'un système soit prêt à l'utilisation,
- la *fiabilité* : est la continuité du service,
- la *sécurité-innocuité* ou (*safety*) : est l'absence de conséquences catastrophiques pour l'environnement (en particulier humain),
- l'*intégrité* : est l'absence d'altération inappropriée de l'information,
- la *maintenabilité* : est l'aptitude aux réparations et aux évolutions.

Un attribut supplémentaire est nécessaire lorsqu'on parle de la sécurité-immunité (ou *security*) : la *confidentialité* qui est définie comme étant « l'absence de divulgations non-autorisées de l'information ». La sécurité-immunité est un composé de confidentialité, d'intégrité et de disponibilité.

Avižienis, Laprie, *et al.* [ALRL04] ont donné une définition complémentaire à la sûreté de fonctionnement qui est : « l'aptitude à éviter des défaillances du service plus fréquentes ou plus graves que ce qui est acceptable ». On retrouve cet aspect relatif de la sûreté de fonctionnement, qui part du principe que le "zéro risque" n'existe pas, dans les normes telles que l'ISO 61508 [IEC10]. Cette dernière inclut cette notion en définissant ce que nous appelons ici "sécurité-innocuïté" (que nous appellerons simplement "sécurité" dans la suite de notre manuscrit) comme étant « l'absence de risques **inacceptables** ». Elle définit le risque acceptable comme étant « un risque accepté dans un contexte donné basé sur des valeurs courantes de notre société ». Le risque acceptable est donc le résultat d'un équilibre entre l'idéal de la sécurité absolue et des facteurs comme les bénéfices pour l'humain, les règles et les conventions de la société concernée [IG99, GMTB07]. Ces notions seront abordées en détail dans le Chapitre 2, Section 2.1.

### 1.2.2 Les entraves

Les entraves de la sûreté de fonctionnement consistent en la *faute*, l'*erreur* et la *défaillance*. Une défaillance survient lorsque le comportement du système dévie de la spécification. Une défaillance survient aussi lorsque le comportement du système satisfait sa spécification mais est considéré comme inacceptable par les utilisateurs du système (révélant une faute de spécification). Une défaillance se produit lorsque, par propagation, elle affecte le service délivré par le système, donc lorsqu'elle "franchit" l'interface système-utilisateur.

Par propagation, plusieurs erreurs peuvent être générées avant qu'une défaillance ne survienne. Par propagation, une erreur crée de nouvelles erreurs. Une erreur peut être *latente* (tant qu'elle n'a pas été reconnue comme telle) ou *détectée* (par une algorithme ou un mécanisme de détection). Du point de vue de la sécurité, ce sont les défaillances systèmes qui seront observées, elles mêmes résultant de fautes telles que décrites section 1.1.3.

### 1.2.3 Les moyens de la sûreté de fonctionnement

Afin d'aboutir à un système aussi sûr que possible, on peut avoir recours à une combinaison de méthodes [ALRL04] qui sont :

- **la prévention de fautes** : comment empêcher l'occurrence ou l'introduction de fautes,
- **la tolérance aux fautes** : comment fournir un service à même de remplir la fonction du système en dépit des fautes,
- **l'élimination des fautes** : comment réduire la présence (nombre, sévérité) des fautes,
- **la prévision des fautes** : comment estimer la présence, les taux futurs, et les possibles conséquences des fautes.

Comme présenté dans la Figure 1.4, l'activation d'une faute engendre une erreur, qui, si rien n'est fait, peut conduire à une défaillance. La notion de tolérance aux fautes ne se limite pas aux mécanismes mis en place pour assurer une continuité de service (fiabilité) ; elle s'étend à tout mécanisme servant à endiguer la propagation d'erreurs dans le but d'éviter ou diminuer la gravité des défaillances (sécurité-innocuité).

La prévention de fautes et l'élimination des fautes peuvent être regroupées en une classe de méthodes d'*évitement* des fautes car elles visent la réalisation de systèmes exempts de fautes. La tolérance aux fautes et la prévision de fautes font partie de la classe de méthodes d'*acceptation* des fautes car elles prennent en compte le fait que tout système réel est sujet à des fautes, qu'il s'agisse de fautes matérielles, de fautes de développement résiduelles ou des fautes d'interaction, etc.

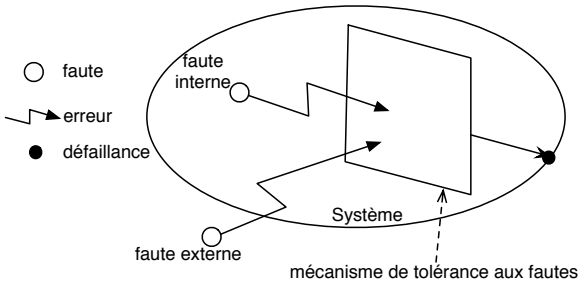


FIGURE 1.4 – Propagation des erreurs et mécanisme de tolérance aux fautes

## 1.3 La sécurité des systèmes autonomes

Dans cette section, nous présentons des exemples des différentes méthodes de sûreté de fonctionnement dans le contexte des systèmes autonomes. Nous distinguons, d'une part, les méthodes qui s'appliquent hors-ligne, avant le déploiement du système et, d'autre part, les méthodes embarquées dans le système et qui s'appliquent en-ligne pendant son utilisation.

### 1.3.1 Méthodes de sécurité hors-ligne

Nous focalisons sur les méthodes d'*élimination* de fautes, et en particulier la *vérification basée modèle*, la *preuve de théorème* et le *test*.

#### 1.3.1.1 La vérification basée modèle

La vérification basée modèle (ou le *model checking*) est une méthode de vérification automatique qui explore l'ensemble des états du système et détermine si, étant donné un modèle $\mathcal{M}$ du système et une propriété $\phi$, toutes les exécutions du modèle $\mathcal{M}$ satisfont $\phi$. La vérification basée modèle est une méthode applicable principalement à des systèmes à états finis pour lesquelles toutes les exécutions possibles du système peuvent être énumérées exhaustivement [CGP99]. Si l'ensemble des propriétés vérifiées sont satisfaites, l'outil de vérification de modèle informe l'utilisateur que la vérification a réussi. Dans le cas contraire, un contre-exemple d'une exécution qui ne satisfait pas une des propriétés est donné. Le contre-exemple est utile pour remonter jusqu'aux causes du problème. Le nombre d'états, le temps et l'espace mémoire nécessaire pour les explorer croît rapidement avec la complexité des systèmes vérifiés, ce que l'on appelle l'*explosion des états* (ou l'*explosion combinatoire*). Le problème est particulièrement exacerbé avec les systèmes autonomes, dont la complexité est notoire. L'explosion combinatoire a fait l'objet de recherches actives au sein de la communauté scientifique. Le problème a été partiellement résolu dans les années '80 à la suite de la découverte des OBDD *(Ordered Binary Decision Diagrams)*, qui permettent d'obtenir une représentation compactée des états du système. La vérification basée modèle traditionnelle combinée à l'utilisation des OBDD a donné naissance à la *vérification basée modèle symbolique* qui permet de vérifier des systèmes extrêmement grands en un temps et un espace mémoire acceptables [CGP99].

La vérification basée modèle a l'avantage de pouvoir être utilisée dès les premières phases de spécification du système, vu qu'elle ne nécessite qu'un modèle du système. Ceci a une contrepartie : les vérifications qui montrent que le système répond aux propriétés énoncées ne sont valables que pour *le* modèle du système utilisé, qui ne correspond pas forcément à l'implémentation réelle du système. Plus le modèle utilisé lors de la vérification est proche du système réel plus la vérification est pertinente.

Les propriétés vérifiées dans la vérification basée modèle peuvent être des :

- propriétés de *sûreté* : qui stipulent que quelque chose ne doit pas se produire (par exemple : pas de *deadlock*),
- propriété de *vivacité* : qui stipule que quelque chose doit nécessairement se produire (par exemple : chaque client doit avoir accès à une ressource partagée).

#### 1.3.1.2 La preuve de théorème

La preuve de théorème (ou *theorem proving*) est une autre méthode qui permet d'effectuer des vérifications sur des modèles formels du système. Elle permet de démontrer que le modèle du système obéit à un ensemble de propriétés de la même façon que la preuve en mathématique démontre l'exactitude d'un théorème. Elle consiste à

énoncer des propositions et à les démontrer dans un système de déduction de la logique mathématique, en particulier dans le calcul des prédicats. La preuve de théorèmes est une méthode difficile à manier car elle requiert une grande expertise technique et une connaissance pointue de la spécification du système [MP05]. Cette approche tend à être automatisée et assistée par ordinateur en utilisant les *démonstrateurs de théorème*. L'inconvénient majeur de cette approche est le fait que si un expert n'arrive pas à finir une preuve en utilisant un démonstrateur de preuves, l'outil ne donne aucune information quant à la faisabilité de la preuve ou quant à la nécessité d'informations supplémentaires pour compléter la preuve [LS09].

La preuve de théorème, par rapport à la vérification basée modèle, a l'avantage d'être indépendante de la taille de l'espace des états, et peut donc s'appliquer sur des systèmes complexes. L'explosion du nombre d'états n'étant plus un problème, la vérification peut être effectuée sur le modèle entier. La majorité des démonstrateurs de preuves sont très expressifs. Certaines propriétés, qui ne peuvent pas être spécifiées en utilisant les *vérificateurs de modèles* (par exemple : comparer les propriétés de deux états qui ne sont pas temporairement liés), peuvent être aisément spécifiées dans la majeure partie des langages des démonstrateurs de théorèmes [LS09].

Néanmoins, comme cela est le cas pour la vérification basée modèle, la validité de cette méthode repose sur l'hypothèse que le modèle du système et de son environnement reflète parfaitement le système et l'environnement réels. Or, ce n'est généralement pas le cas, surtout dans le contexte des systèmes autonomes complexes.

#### 1.3.1.3 Le test

Le test est défini par la norme IEEE 829-2008 [IEE08] comme étant : « une activité dans laquelle un système ou un composant est exécuté sous des conditions spécifiques ; les résultats sont observés ou enregistrés ; et l'évaluation de certains aspects du système ou du composant est effectuée ». Un test est un ensemble de *cas à tester*. Un cas à tester est défini, selon la même norme, par : des données d'entrées, des conditions d'exécution et les résultats attendus pour un objectif donné (l'objectif peut être, par exemple, que le programme respecte une propriété spécifique). Différents types de test permettent de détecter différents types d'erreurs. Le test peut être classé selon le niveau du cycle de développement du logiciel dans lequel il est effectué : test composant (unitaire), test d'intégration, test système (ou test fonctionnel) et test d'acceptation. Il peut être classé aussi selon sa caractéristique : test de performance (validation que les performances spécifiées du système sont respectées), test fonctionnel (vérification que les fonctionnalités répondent aux attentes), test de vulnérabilité (vérification de la sécurité du logiciel) et test de robustesse (validation de l'aptitude du système à délivrer un service correct dans la durée, ou bien en présence d'entrées invalides ou de conditions environnementales stressantes) [Chu11].

Dans le cadre des systèmes autonomes, des tests en simulation intensifs ont été effectués pour le projet DS1 sur l'architecture RAX [2] [BGR+96, FS99]. Six bancs de test ont été réalisés tout au long du développement, ciblant différentes parties du

2. Architecture hiérarchisée développée par le *NASA Ames Research Center* dans le cadre du projet *Deep Space One* [MNPW98]

système ou le système dans son ensemble ; en tout, près de 600 tests ont été effectués. Les auteurs de [BGR+96] soulignent l'importance de ces tests intensifs, et remarque des difficultés posées au test par les systèmes autonomes, notamment le problème de l'oracle (c'est-à-dire, la façon dont le système doit réagir face à un jeu de test donné).

### 1.3.2 Méthodes de sécurité en-ligne

Etant données les limitations intrinsèques des méthodes de vérification basées modèles, de preuve de théorème et de test, il nous semble indispensable, surtout pour les systèmes critiques, de les compléter par des méthodes de vérification en ligne pour assurer la sécurité.

Nous classons dans cette catégorie de techniques, toute approche visant à détecter un comportement incorrect d'un système et éventuellement à réagir avant qu'un événement indésirable ne survienne. Nous pouvons citer, par exemple, toutes les méthodes classiques de détection d'erreurs utilisées en tolérance aux fautes (voir Section 2.1.1.2 de [LAB+96]), ainsi que les techniques de diagnostic utilisées de façon pro-active pour révéler l'existence de fautes, et éventuellement les isoler (par exemple, l'approche FDIR) [3] [PRDS07].

Cependant, nous allons nous focaliser sur deux techniques de vérification en-ligne plus spécifiquement ciblées sur des comportements incorrects dus à des fautes résiduelles de développement ou sur des comportements dangereux vis-à-vis de la sécurité.

#### 1.3.2.1 La vérification à l'exécution

Au début des année 2000, le terme "vérification à l'exécution" (ou *runtime verification*) a été introduit dans le contexte de recherches à la frontière entre la vérification formelle (basée modèle ou par preuve de théorèmes) et le test [4]. Cette communauté s'est intéressée à la vérification qu'une exécution respecte des propriétés spécifiées formellement, issues ou non de la spécification du système. Il s'agit en fait d'une généralisation de la notion d'assertions exécutables [Sai77, RBQ96] à des propriétés le plus souvent dynamiques, relatives par exemple, aux événements dans une trace d'exécution.

L'IEEE définit la vérification à l'exécution comme « une discipline de l'informatique qui pour objectif l'étude, le développement et l'application de techniques de vérification qui permettent de vérifier si l'exécution du système répond ou viole des propriétés prédéfinies » [IEE05]. De nombreux travaux ont étudié la pertinence d'avoir recours à cette technique dans le contexte des systèmes autonomes critiques [LS09, GP10, PNW11].

Les principales différences entre la vérification basée modèle et la vérification en ligne sont :

– la dimension de l'espace d'états à vérifier : la vérification basée modèle vérifie si

3. FDIR (Fault diagnosis, isolation and reconfiguration) est un terme utilisé pour désigner toute approche de tolérance aux fautes distinguant les trois étapes : détermination des causes des erreurs (les fautes), isolation de celles-ci, puis reconfiguration du système de telle façon que les composants non-défaillants puissent délivrer un service acceptable (même s'il est dégradé).

4. http ://runtime-verification.org

*toutes les exécutions possibles* d'un système satisfont une propriété donnée alors que la vérification à l'exécution vérifie une trace d'exécution finie,
– la représentation du système à vérifier : la vérification basée modèle s'appuie, comme son nom l'indique, sur un *modèle* du système alors que la vérification à l'exécution vérifie l'exécution du *système réel*.

#### 1.3.2.2 Les dispositifs de surveillance pour la sécurité

Lorsque les propriétés à vérifier ciblent plus spécifiquement la sécurité, au sens de l'évitement de défaillances catastrophiques, la vérification en ligne s'effectue au moyen d'un dispositif de sécurité.

Les dispositifs de surveillance apparaissent dans la littérature sous différentes appellations : "safety manager" [PS00], "autonomous safety system" [RRAA04], "checker" [IP02], "guardian agent" [Fox01], "safety bag" [Kle91], ou bien "moniteur de sécurité" [IEC10]. Les auteurs de ces travaux présentent différents dispositifs de surveillance que nous pouvons distinguer selon deux critères : indépendance *physique* ou *matérielle* du dispositif de surveillance par rapport au système surveillé et l'indépendance *logicielle* du dispositif de surveillance par rapport au système surveillé.

1. Indépendance *physique* : la norme IEC 61508 [IEC10] distingue « le moniteur et le système surveillé présents sur la même plate-forme avec une certaine garantie d'indépendance entre les deux » et « le moniteur et le système surveillé sur des plates-formes indépendantes ». Dans ce dernier cas, le dispositif de surveillance est appelé *moniteur diversifié*[5] dont le but est « de garantir une protection contre les fautes résiduelles de spécification et de mise en œuvre du logiciel qui pourraient porter atteinte à sécurité du système ». Schroeder [Sch95] parle de *moniteur intrusif* lorsque le dispositif de surveillance et le système surveillé partagent des ressources communes (CPU, dispositifs d'entrée/sortie, chaînes de communication etc.). A l'opposé, le dispositif de surveillance est appelé *moniteur non-intrusif* lorsqu'il utilise un équipement qui lui est dédié. L'auteur définit un *moniteur de sécurité* comme étant : « un observateur externe voué à être permanent et qui surveille une application opérationnelle ».

2. Indépendance *logicielle* : Goodloe et Pike [GP10] définissent un *moniteur de sécurité* ainsi : « le moniteur observe le comportement d'un système et détecte s'il est cohérent avec une spécification donnée ». Ils parlent de *moniteur "in-line"* lorsque le moniteur est inséré dans le code (annotations, assertions etc.) et de *moniteur "out-line"* lorsque le moniteur est exécuté dans un processus indépendant du système surveillé. Ils abordent aussi la notion de recouvrement, qui n'est pas présente dans les précédentes définitions : « les moniteurs (*in-line* et *out-line*) vérifient le respect de la spécification lors de l'exécution et peuvent remettre le système dans un état acceptable lorsque ce dernier se met à dévier de sa spécification ».

5. Dans la première version de l'IEC 61508 (1998), un tel moniteur était appelé *safety bag*, terme utilisé dans de précédents travaux sur les *moniteurs diversifiés* dans le contexte du contrôle ferroviaire [Kle91]

Nous nous intéressons particulièrement aux dispositifs de surveillance *indépendants* physiquement car ils permettent de se prémunir des fautes résiduelles de spécification et de développement qui ont résisté aux méthodes de vérification hors-ligne. De plus, ils permettent de se protéger des défaillances de mode commun (une défaillance du système cible n'aura pas d'impact sur le dispositif de surveillance) et de la propagation des erreurs (une donnée erronée utilisée par le système cible n'affectera pas le traitement effectué par le dispositif de surveillance).

On peut distinguer deux sortes de dispositifs de surveillance selon le type d'action qu'ils peuvent enclencher vis-à-vis du système surveillé :

1. Action de blocage : le dispositif de surveillance observe que le système est dans état *sûr* mais que l'exécution de telle ou telle tâche le mettrait dans un état catastrophique, il enclenche une action qui consiste à *bloquer* cette (ou ces) tâche(s). Un tel dispositif fonctionne comme un *interverrouillage*.

   Leveson [Lev91] définit l'interverrouillage comme un mécanisme qui permet de s'assurer qu'une séquence d'opérations se produit dans le bon ordre. L'auteur a donné des exemples d'interverrouillages qui permettent de s'assurer qu'un événement A ne se produira pas :

   – par inadvertance : par exemple obliger l'occurrence d'un événement B avant que l'événement A ne puisse se produire,
   – tant qu'une condition C est vérifiée : par exemple une porte d'accès est placée en amont d'un équipement à haute tension de telle sorte que, tant que la porte est ouverte, l'équipement est mis hors tension,
   – avant un événement D : par exemple la cuve ne se remplit que lorsque la soupape d'aération est ouverte.

   Un interverrouillage peut faire partie intégrale du système fonctionnel, en ayant par exemple recours à des techniques de synthèse de contrôleurs [RW89, ACMR03, PI04a, BDSG+10], ou bien il peut être externe au système en utilisant des moyens électriques, électroniques ou mécaniques. Les interverrouillages externes au système permettent une protection de bout en bout, ce qui permet d'assurer la sécurité du système contre des fautes internes au système. Néanmoins, implémenter un tel mécanisme peut être coûteux, notamment lorsque cela nécessite une modification du système physique.

2. Action de mise en sécurité : le dispositif de surveillance observe que le système est dans un état *dangereux* mais non catastrophique, il doit alors enclencher une action qui consiste à *remettre* le système dans un état sûr. Nous réservons le terme *moniteur de sécurité* à cette sorte de dispositif de surveillance.

Nous concrétisons cette distinction par les définitions suivantes :

**Définition 1.** *Un interverrouillage de sécurité est un mécanisme qui a pour objectif d'empêcher que le système atteigne un état catastrophique en inhibant les événements qui peuvent mener à un tel état à partir de l'état actuel du système.*

**Définition 2.** *Un moniteur de sécurité est un mécanisme qui a pour objectif d'empêcher que le système atteigne un état catastrophique, en détectant les états non-sûrs et en enclenchant les actions appropriées afin de remettre le système surveillé dans un état sûr.*

## 1.4 Exemples de dispositifs de sécurité et analyse

Dans cette partie, nous allons présenter plusieurs dispositifs de sécurité pour systèmes autonomes. Les mécanismes présentés effectuent la surveillance et les actions de mise en état sûr de différentes façons et à différents niveaux de l'architecture. Nous nous intéresserons également à la méthode de détermination des contraintes, propriétés ou règles de sécurité vérifiées par les dispositifs étudiés. Nous porterons notre attention sur leur expression ainsi que sur leur vérification en ligne.

### 1.4.1 Ranger

La NASA et le département de la défense américain ont mis au point un système robotique qui permet de réparer, approvisionner en carburant et renouveler les stations spatiales (voir Figure 1.5). Les recherches menées à l'université du Maryland ont permis de mettre au point Ranger [RRAA04, GP06, BBGP07] : un robot télécommandé équipé de deux bras permettant d'opérer avec précision près d'engins spatiaux fragiles tout en offrant un retour vidéo aux opérateurs.

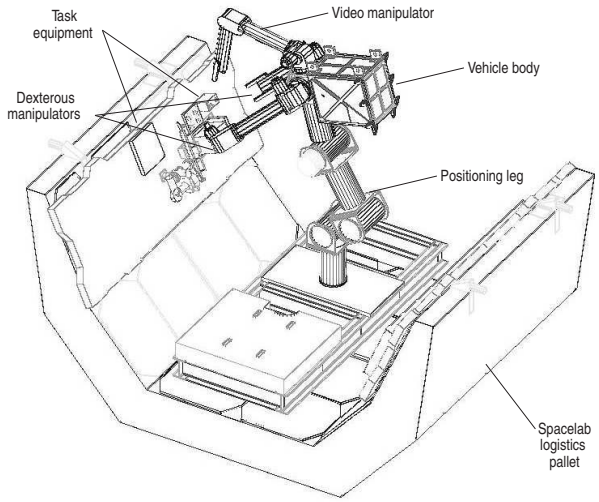


FIGURE 1.5 – Vue globale du Ranger

Comme tout système robotique, Ranger doit faire face aux défaillances qui peuvent se produire tout en préservant les humains et les équipements à proximité. L'analyse de risque a mis en évidence les situations catastrophiques suivantes :

– Les bras endommagent la navette empêchant son retour sur terre.

– Perdre ou larguer un objet qui peut endommager la navette.
– Exercer une force excessive sur un objet et le casser.

Les techniques traditionnelles de tolérance aux fautes ne permettaient pas d'éviter ce genre de défaillances, l'université du Maryland a mis au point un système de sécurité autonome pour Ranger.

La figure 1.6 représente l'architecture du contrôleur de Ranger. Il se compose de plusieurs processeurs. Les deux *DMU* (Data Management Unit) communiquent avec la station de contrôle et toutes les *LPUs* (Local Processing Unit) et *PMUs* (Power Management Unit). Le *Main DMU* commande les bras ; il effectue également les vérifications de la sécurité. De son côté, le *Monitor DMU* n'effectue que les vérifications liées à la sécurité. Cette vérification porte sur :

– la position des éléments du robot (calcul des distances minimales d'approche des bras du robot ; déterminées par un composant dédié : Analyse de l'Energie de l'Impact),

– les vitesses de déplacement maximales des articulations,

– l'état des outils de préhension (éviter la perte d'un objet par inadvertance),

– les forces exercées sur les objets.

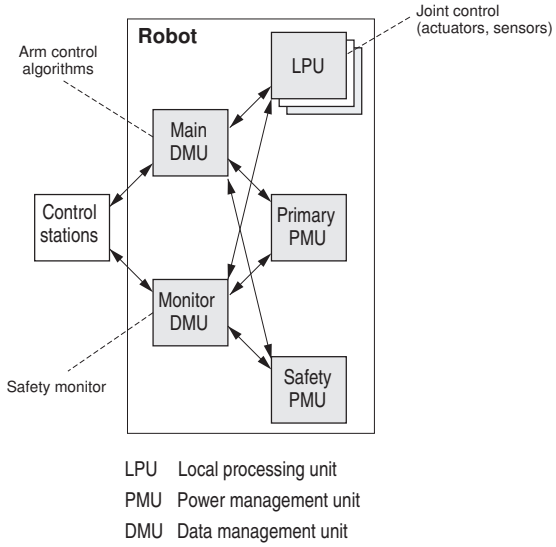


FIGURE 1.6 – Architecture du contrôleur de Ranger

Chaque LPU et PMU reçoit des commandes des deux DMU. En cas de désaccord entre les deux DMU, chaque LPU et PMU sélectionne la commande la plus sûre (technique appelée par les auteurs : "l'accord passif"). Les auteurs ont classifié les trois commandes qui peuvent être envoyées au LPU, du plus sûr au moins sûr : la commande *safe* est plus sûre que la commande *halt*, qui est elle-même plus sûre que la commande

*running*. Lorsque la LPU reçoit deux commandes de changement d'états différentes, par exemple *halt* et *running*, elle sélectionne la première et rejette la seconde.

Le code de vérification est répliqué sur les deux DMU excepté celui en charge de la vérification de la position qui a été développée de deux manières différentes. Chaque DMU utilise des données de position différentes issues de trois encodeurs de position de différentes technologies. Les auteurs considèrent alors que des défaillances de mode commun ne sont pas crédibles.

Les quatre vérifications citées ci-dessus sont uniquement réalisées sur la base de données de capteurs et non sur des commande internes. Ainsi, il n'existe pas de système de contrôle permettant de filtrer ou de bloquer des commandes envoyées au LPU même si celles-ci peuvent entraîner des situations dangereuses.

De notre point de vue, les vérifications cités ci-dessus peuvent être assimilées à des contraintes de sécurité qui doivent être respectées. Ces contraintes peuvent être exprimées de la manière suivante :
- Il ne faut pas que le robot intervienne en dehors du périmètre de sécurité.
- Il ne faut pas que la vitesse au niveau des articulations dépasse $x$ m/s.
- Il ne faut pas larguer un équipement dans l'espace.
- Il ne faut pas que la force appliquée sur les bras dépasse $y$ newton.

### 1.4.2 Elektra

Elektra ou *LockTrac Electronic Interlocking System* est un système d'aiguillage pour le transport ferroviaire développé par Alcatel Autriche [Kle91]. Ce système de sécurité a pour but d'éviter toute défaillance qui puisse engendrer une catastrophe comme une collision de trains par exemple. Les exigences de sécurité proviennent principalement d'experts dans le domaine du ferroviaire et de normes comme l'*Austrian Federal Railway standard*.

La figure 1.7 représente l'architecture du *safety bag*[6] d'Elektra. L'architecture consiste en deux canaux : un premier qui effectue les traitements fonctionnels et un second qui assure la sécurité en vérifiant les commandes produites par le précédent canal. Si la commande n'obéit pas à certaines règles de sécurité, le safety bag n'approuvera pas l'exécution de la commande.

Le safety bag d'Elektra consiste en un système expert, c'est-à-dire, d'une base de connaissances qui contient les règles de sécurité, ainsi que l'état courant du système ; et un moteur d'inférence qui choisit quelle règle est applicable puis décide de transmettre ou bloquer la requête. Un vote matériel est réalisé en cas de désaccord des deux canaux de calcul et le système est mis dans un état sûr. D'autre part, Elektra s'appuie sur VOTRICS [The86], qui assure une redondance matérielle pour chacun des deux calculateurs ainsi que des canaux de communication afin d'accroître la disponibilité du système.

Dans les premières versions du système sans logiciel (uniquement des relais), des analyses du système électronique de type AMDEC[7] ont été effectuées pour détermi-

6. A notre connaissance, cette expression fut utilisée pour la première fois au sein de ces travaux.
7. AMDEC : Analyse des Modes de Défaillances, de leurs Effets et de leur Criticité

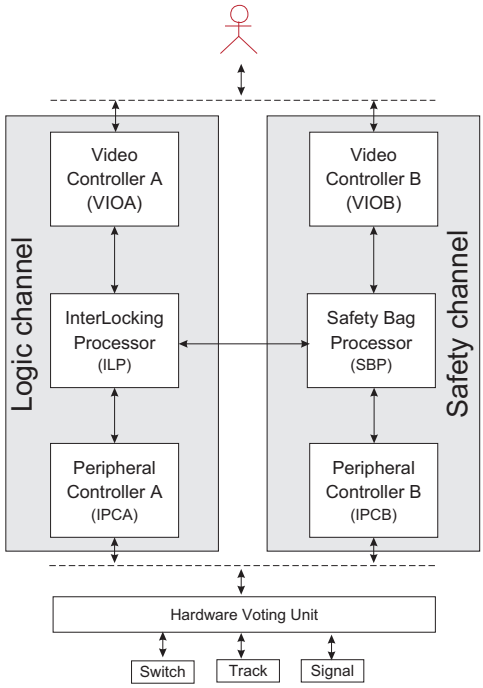


FIGURE 1.7 – Modèle basique de l'architecture d'Elektra

ner les règles de sécurité à assurer. Mais à partir du moment où les concepteurs ont introduit du logiciel, il semble que de telles analyses n'ont pas été réalisées [Erb89]. Les règles ont été formalisées avec l'aide des normes, des experts et de l'expérience dans le ferroviaire des développeurs.

Les règles de sécurité sont exprimées à un haut niveau d'abstraction en utilisant PAMELA, un langage de programmation logique dédié au développement de systèmes experts de contrôle en temps réel. Les articles ne présentent pas d'exemples de règles dans ce langage mais seulement une traduction en anglais (voir figure 1.8) de ces règles, exprimées sous forme de *IF (condition) THEN (action).*

### 1.4.3 R2C (Request & Resource Checker)

R2C [IP02, PI04b, PI04a] est une implémentation de la couche de contrôle de l'exécution de l'architecture LAAS (vue en Section 1.1.2) pour les systèmes autonomes (voir Figure 1.9).

Cette couche est le lien entre la couche décisionnelle et la couche fonctionnelle. La modularité de la couche fonctionnelle fait qu'un module n'a pas de connaissance sur les états internes des autres modules ce qui peut engendrer des conflits. Cette couche va, par conséquent, agir comme un moniteur qui surveille l'état du système en temps réel. Il doit en premier lieu *observer* le comportement du système et capturer tous les événements menant à un état indésirable et dans un second lieu *commander le système* en interdisant ou en autorisant l'exécution d'une commande.

IF there is a request to throw over a switch
AND the switch is either in position full left or full right
AND the switch is not locked or interlocked
THEN commit the request
IF there is a request to interlock a switch
AND the switch is in correct position
THEN commit the request
IF there is a request to perform the check to establish the route
AND all tracks and switches in the route are free
AND the start signal is not locked
AND the status of the route is locked
THEN commit the request
[...]
If a request could not be committed by one of these rules it is rejected
by a general rule with low priority

FIGURE 1.8 – Exemple de règles de sécurité dans Elektra.

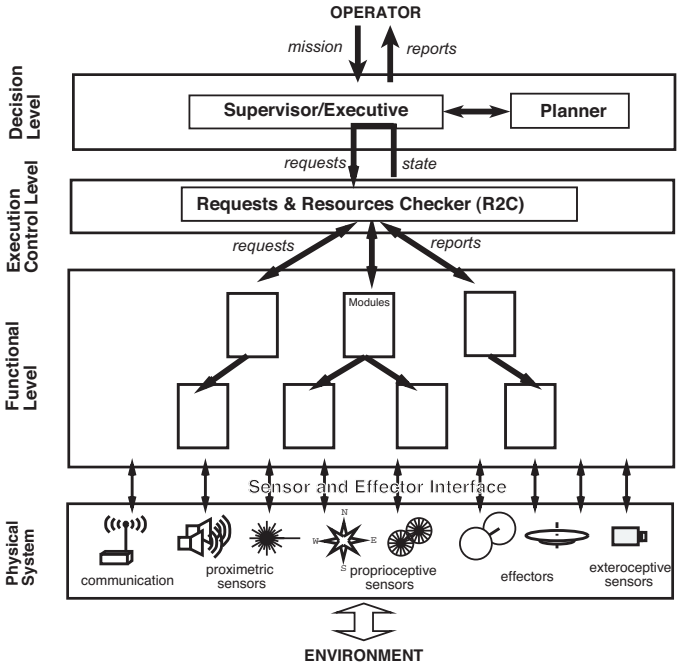


FIGURE 1.9 – Implémentation de R2C au sein de l'architecture 3 niveaux du LAAS

R2C ou le Request & Resource Checker, présenté Figure 1.10, effectue les tâches suivantes :

- capture tous les événements pouvant modifier l'état du système,
- met à jour la base de données des états du système,
- vérifie la validité des requêtes grâce au *Model Checker*. Si elles ne sont pas valides, il décide des actions à entreprendre pour garder le système dans un état sûr,
- exécute les actions choisies et envoie des rapports aux deux couches voisines.

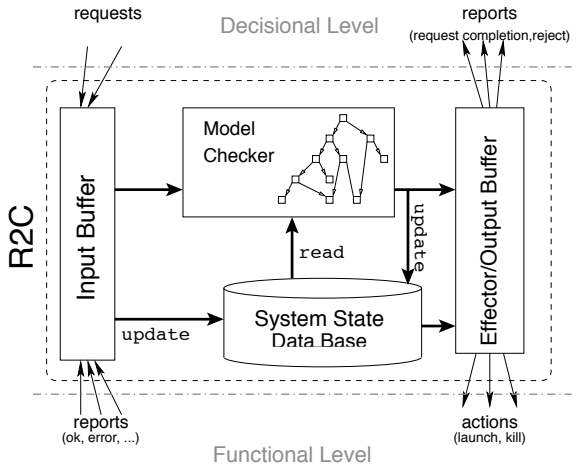


FIGURE 1.10 – The Request & Resource Checker

Le Model Checker est le composant le plus important de R2C. Il doit vérifier des règles de sécurité exprimées sous forme d'assertions logiques. Les auteurs des travaux de R2C ont réalisé un travail important pour exprimer les règles de sécurité dans un langage de haut niveau. Il se présente sous la forme d'un graphe acyclique dirigé (ou DAG) en utilisant un compilateur développé au LAAS : ExoGen. Chaque nœud de l'arbre représente un test d'une variable d'état et se présente sous la forme d'un prédicat (contrainte). Ces prédicats décrivent le contexte qui doit être vérifié pour permettre l'exécution d'un service. La figure 1.11 illustre le langage ExoGen sur la spécification d'un service (appelé *goto*) dans un module de planification d'itinéraire (module appelé *PathPlanning*).

```
request PathPlanning_goto(?goal) {
  fail {
  maintain:
    past(Localization_set(?mode) with ?mode==STEREO) &&
      ( !past(Stereo_start()) || [Stereo_start()<Stereo_stop()] );

    past(Localization_set(?mode) with ?mode==GPS) && GPS_status in [0,0.1];
  }
}
```

FIGURE 1.11 – Exemple de règle en ExoGen.

Une règle de sécurité en ExoGen exprime une contrainte de sécurité ainsi que l'action de sécurité enclenchée si cette contrainte est violée. Si on prend comme exemple

la règle présentée dans la figure 1.11, elle exprime le fait que l'exécution du service "PathPlanning _goto" ne doit pas avoir lieu si :

- le mode de localisation est *STEREO* et la *STEREO* n'a pas été enclenchée ou bien elle a été enclenchée mais arrêtée, ou,
- le mode de localisation est *GPS* et le niveau de réception de ce dernier est faible (moins de 10%).

La conjonction de ces deux conditions définit la condition de déclenchement de l'action de recouvrement "annuler l'exécution".

Dans le graphe acyclique dirigé généré par ExoGen, chaque nœud représente le test d'une variable d'état (chaque nœud est une sorte de *if-then-else*) et s'écrit sous la forme d'un prédicat de forme spécifique. Le résultat consiste en un graphe de type OCRD [8]. Il représente la formule générale qui exprime toutes les transitions valides, c'est-à-dire tous les prédicats vérifiés. ExoGen extrait alors de cette formule les prédicats qui vont permettre de maintenir cette formule à vrai en prenant en compte les événements contrôlables et les prédicats incontrôlables (représentent l'état courant du système et les événements passés ou à venir). Le contrôle du système se fait à l'aide des prédicats *contrôlables*, ces prédicats représentent les actions qu'entreprend R2C pour maintenir la formule générale à vrai ; c'est-à-dire maintenir le système dans un état sûr. Le graphe OCRD obtenu à partir de l'exemple précédent est présenté dans la figure 1.12. Les prédicats incontrôlables sont vérifiés en premier.

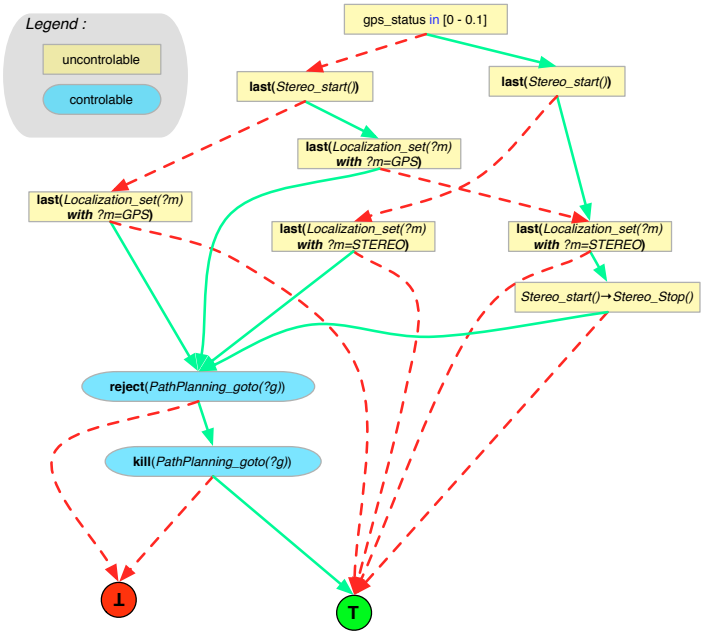


FIGURE 1.12 – Le graphe OCRD obtenu à partir de l'exemple.

Dans cet exemple le service *goto* ne peut pas être exécuté dans les deux cas déjà décrits plus haut. L'annulation de l'exécution "PathPlanning _goto" peut se décliner

8. OCRD : Ordered Constrained Rule Diagram.

soit en le rejet d'une requête dont l'exécution n'a pas encore commencé (*reject*), soit en l'avortement d'une exécution en cours (*kill*).

### 1.4.4 MARAE

MARAE ou "Méthode et Architecture Robustes pour l'Autonomie dans l'Espace" était un projet commun au LAAS-CNRS, Vérimag [9] et EADS Astrium [10] et partiellement financé par la Fondation Nationale de Recherche en Aéronautique et l'Espace (FNRAE).

Le projet visait, entre autre, à mettre en œuvre des mécanismes de sécurité pour un robot autonome, implémentés au sein de son logiciel de contrôle. Ce mécanisme peut être vu comme une évolution de R2C, visant à assurer le respect d'un ensemble de contraintes de sécurité qui spécifient que des comportements dangereux ou incohérents ne se produiront pas. Le mécanisme est implémenté dans la couche fonctionnelle de l'architecture LAAS, car celle-ci interagit directement avec les équipements du robot. Les clients de la couche fonctionnelle peuvent émettre des requêtes, initialiser des modules, mettre à jour des structures de données internes ou initier ou arrêter différentes activités. Le projet s'appuyait, d'une part, sur l'environnement GenoM du LAAS [FHC97] pour le développement de couches fonctionnelles modulaires et, d'autre part, sur la méthodologie BIP de Vérimag [BBS06] pour la modélisation de programmes temps-réel hétérogènes.

#### L'environnement GenoM

L'environnement GenoM (Generator of Modules) a été développé pour faciliter la modélisation de modules temps-réel réutilisables ainsi que leur intégration à l'architecture à trois couches du LAAS. Chaque module fournit une fonctionnalité particulière du robot. Un module est responsable d'une ressource physique (capteur, actionneur) ou d'une ressource logique (une donnée). Il contient les algorithmes et les fonctionnalités nécessaires ainsi que des mécanismes de traitement d'erreurs et de fautes (par exemple détection d'erreur, interruption de procédure, etc.) afin de contrôler la ressource et assurer l'intégrité. Les fonctions complexes (comme la navigation par exemple) sont assurées grâce à la coopération de différents modules.

Chaque module offre une interface de service composée de plusieurs requêtes qui peuvent être initiées par les utilisateurs ou par les couches de plus haut niveau (paramétrer, initier ou arrêter), afin de commander les services offerts par le module.

#### Le cadre conceptuel BIP

BIP est un cadre conceptuel utilisé pour modéliser, vérifier et coder des programmes temps-réel hétérogènes. BIP tient son nom de *Behavior*, *Interaction* et *Priority*, les

9. http ://www-verimag.imag.fr
10. http ://www.astrium.eads.net

trois principes fondamentaux du cadre conceptuel. Les principales caractéristiques de BIP sont :

- BIP gère les méthodes de spécification basées modèle où les programmes parallèles sont obtenus par la superposition de trois couches. La couche inférieure décrit les comportements. La couche intermédiaire inclut un ensemble de connecteurs décrivant les interactions entre les transitions des comportements. La couche supérieure est un ensemble de règles de priorité décrivant les politiques d'ordonnancement pour les interactions.

- BIP utilise un opérateur de composition paramétré sur les composants. Le produit de deux composants consiste en la composition des couches correspondantes séparément. Des paramètres sont utilisés pour définir de nouvelles interactions ainsi que de nouvelles règles de priorité entre les programmes parallèles. Un tel opérateur de composition permet une construction incrémentale, c'est-à-dire, l'obtention d'un programme parallèle par compositions successives d'autres programmes.

- BIP offre un mécanisme pour structurer des interactions permettant une synchronisation forte (rendez-vous) ou faible (diffusion). Une exécution synchrone résulte de la combinaison des propriétés des trois couches.

Dans le projet MARAE, BIP a été utilisé pour reconstruire une couche fonctionnelle précédemment développée en GenoM, tout en intégrant des mécanismes similaires à ceux de R2C afin d'assurer la sécurité du système. A la différence de R2C, les règles de sécurité et leur mise en œuvre ne sont plus regroupées dans une couche intermédiaire entre la couche décisionnelle et la couche fonctionnelle. Dans MARAE, les règles sont exprimées à l'aide de connecteurs BIP, qui régissent la synchronisation entre les modèles représentant les modules GenoM. Les propriétés ainsi définies sont alors mises en œuvre par le code produit automatiquement à partir des modèles et des connecteurs BIP. La couche fonctionnelle ainsi produite intègre à la fois le code fonctionnel des modules GenoM originels, et un code de synchronisation mettant en vigueur les propriétés spécifiées dans les connecteurs.

A titre d'exemple, les connecteurs BIP montrés sur la Figure 1.13 vérifient la précondition d'une activité A d'un module M telle que A ne peut démarrer que s'il y a eu au moins une exécution complète d'un service B et au moins une exécution complète d'un service C.

### 1.4.5 DLR Co-Worker

Dans le contexte de coopération entre un humain et un robot, une plate-forme de démonstration appelée *DLR Co-Worker* a été mise en place au sein de l'institut de robotique de l'agence aérospatiale allemande : DLR [11], voir Figure 1.14 [HSF+09].

Cette coopération homme/robot induit une préoccupation particulière vis-à-vis

11. http ://www.dlr.de/dlr/en

```
connector allow_A_if_B_and_C_is_set(M.do_A, M.status_B, M.status_C)
define [M.do_A, M.status_B, M.status_C]
on M.do_A, M.status_B, M.status_C
provided M.status_B.done ∧ M.status_C.done
do {}

connector reject_A_if_B_and_C_is_not_set(M.do_A, M.status_B, M.status_C)
define [M.do_A, M.status_B, M.status_C]
on M.do_A, M.status_B, M.status_C
provided ¬M.status_B.done ∨ ¬M.status_C.done
do { M.do_A.rep ← B-OR-C-NOT-SET }
```

FIGURE 1.13 – Exemples de connecteurs BIP

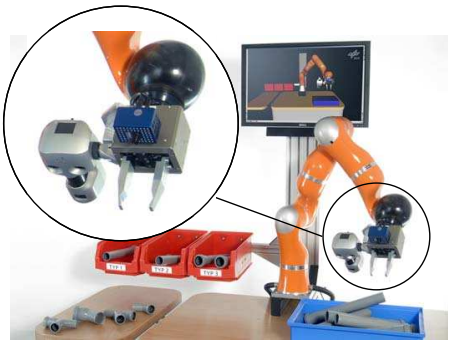

FIGURE 1.14 – Le DLR Co-Worker.

de la sécurité. Dans ces travaux, on part du principe que toutes les collisions entre l'humain et le robot ne sont pas fatales si le robot possède des stratégies de réactions appropriées.

Afin d'assurer la sécurité dans ce contexte, un soin particulier est donné à la détermination de l'endroit où se trouve l'homme par rapport au robot. Pour cela, quatre variables booléennes observables ont été définies :

- **oP** (out of Perception) : l'homme est en dehors du champ de perception du robot et par conséquent ne peut pas être pris en compte.
- **iP** (in Perception) : le robot perçoit l'homme et sa présence est prise en compte.
- **iHF** (in Humain-Friendly zone) : l'homme est près du robot mais sans aucune interaction physique.
- **iCM** (in Collaborative Mode) : l'homme interagit physiquement avec le robot.

Le changement de position de l'homme provoque le passage d'un mode fonctionnel à un autre. Quatre modes ont été définis :

- **Autonomous task execution** : mode autonome en l'absence de l'homme.
- **Humain-friendly behavior** : mode autonome en présence de l'homme.
- **Co-Worker behavior** : coopération avec l'homme.
- **Fault reaction behavior** : mise en état sûr.

Les travaux se focalisent sur la détection de collisions et les réactions appropriées à déclencher. Le but est de réagir différemment à chaque collision, selon la gravité de cette collision.

Le système se compose d'un ensemble de tâches. Une certaine tolérance aux fautes est assurée au niveau de chaque tâche en activant un ensemble de tâches de sécurité (voir figure 1.15).

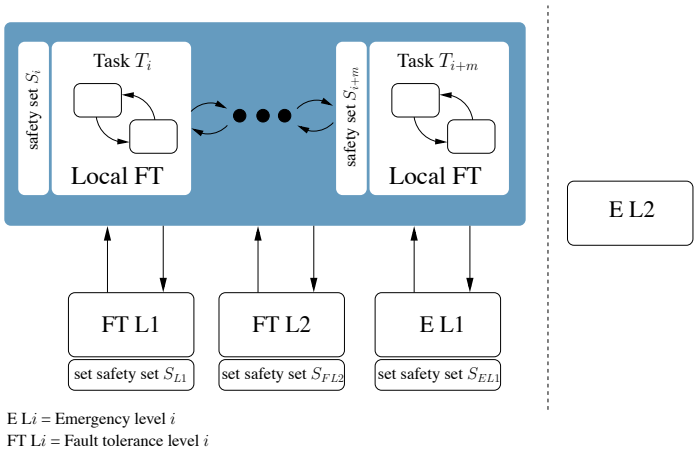


FIGURE 1.15 – Le mécanisme de sécurité du DLR Co-Worker.

Lorsqu'une collision est détectée, son niveau de gravité est évalué. Selon cette gravité, une des couches de sécurité est activée, déclenchant les tâches de sécurité associées à cette couche. En effet, dans l'architecture de sécurité du DLR Co-Worker présentée dans la figure 1.16, chaque couche correspond à un niveau de tolérance aux fautes différent. Chaque niveau doit s'acquitter à son tour d'un ensemble de tâches de sécurité en cas d'échec de la couche locale. On remarquera que le niveau le plus élevé correspond à l'Emergency Layer 2 qui agit directement sur les circuits électroniques en déclenchant les freins.

Notons que les auteurs n'ont pas précisé la façon de mesurer la gravité d'une collision ni comment définir l'ensemble des tâches de sécurité à exécuter au niveau des tâches fonctionnelles du robot et celles correspondant aux différents niveaux de tolérance aux fautes.

Les auteurs ne définissent pas explicitement ni les règles, ni les contraintes de sécurité. Néanmoins, nous pouvons conclure que les combinaisons des variables observables (citées ci-dessus) permettent de déclencher des changements de mode, avec des actions et protections de sécurité correspondantes. En effet, selon le mode courant du système, le changement de position de l'homme provoque la valuation de ces variables. Si une variable change de valeur, le système passe à un mode fonctionnel différent qui possède des tâches de sécurité spécifiques. La figure 1.17 montre les modes fonctionnels du DLR Co-Worker. Par exemple, si le mode de fonctionnement courant du Co-Worker est *Autonomous mode* et si la condition : $iP \wedge iHF$ devient vraie, cela provoque le passage du mode de fonctionnement courant du robot de *Autonomous mode* à *Human-friendly mode*.

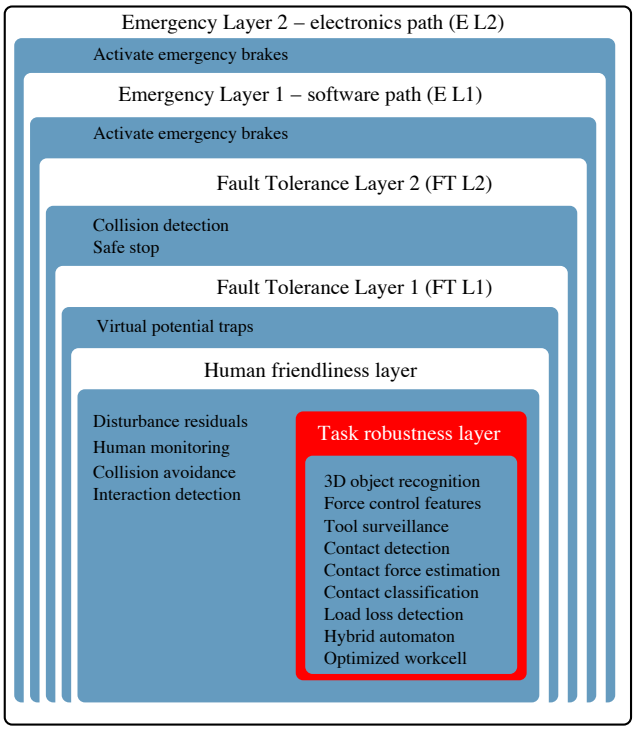


FIGURE 1.16 – L'architecture de sécurité du DLR Co-Worker.

On peut interpréter les transitions de la Figure 1.17 conduisant au mode *Fault Reaction* comme correspondant à des violations de contraintes de sécurité. Il en va de même pour les transitions de la Figure 1.15 conduisant au lancement des couches de tolérance (voir Figure 1.16).

### 1.4.6 Discussion

La Table 1.1 résume les caractéristiques des travaux présentés dans la précédente section, selon plusieurs fonctionnalités : l'*observation* et la *réaction*, qui peuvent être effectuées de différentes manières et l'*indépendance* du dispositif de surveillance qu'elle soit matérielle ou logicielle.

Le DLR Co-Worker observe des données de bas niveau telles que la position de l'homme par rapport au robot, la force exercée sur les articulations lors d'une collision etc. D'autres observent des données de plus haut niveau. En effet, R2C et MARAE observent les requêtes : le premier observe celles échangées entre la couche décisionnelle et fonctionnelle ; le second observe celles échangées au sein même de la couche fonctionnelle. D'autres comme Ranger et Elektra observent les deux types de données (haut niveau et de bas niveau) pour effectuer leurs vérifications. Le premier observe les commandes de contrôle des axes envoyées par le Main DMU et le Monitor DMU au LPU et PMU en se basant sur les variables environnementales observables telles que la position des axes, l'état des outils de préhension, etc. Le second observe et vérifie les requêtes générées par la chaîne de commande en utilisant les données de

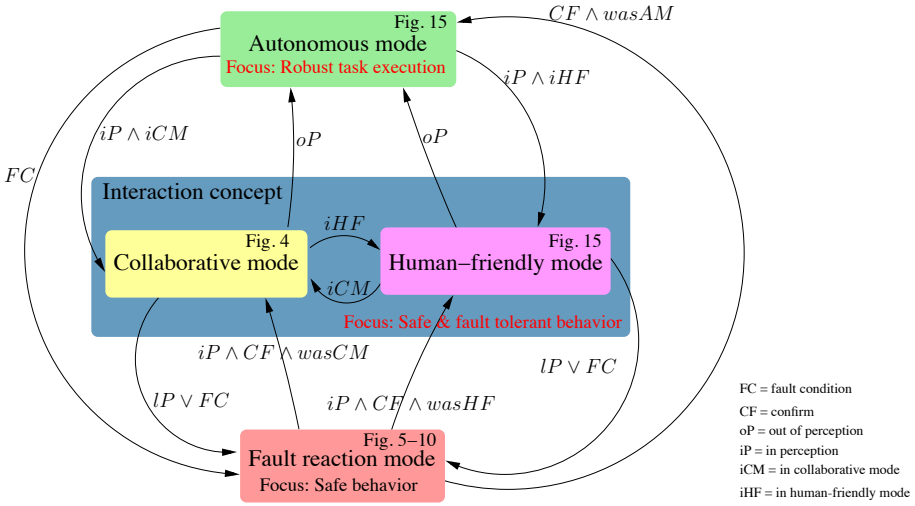


FIGURE 1.17 – Les modes fonctionnels du DLR Co-Worker.

l'environnement fournies par les capteurs.

Selon la distinction précédemment faite à la Section 1.3.2.2, une fois les vérifications effectuées, les dispositifs peuvent réagir de deux manières :

- En autorisant ou en rejetant une activité dans le cas d'un dispositif de type interverrouillage de sécurité. Cela suppose que le système est dans un état sûr et que l'interverrouillage va rejeter une requête si celle-ci le conduirait dans un état catastrophique. Cela s'applique à Ranger, Elektra, R2C et MARAE. En effet, au sein de Ranger, si les commandes envoyées par les deux DMU au LPU et au PMU sont différentes, chaque LPU et PMU autorise la commande la plus sûre et rejette la moins sûre. Dans Elektra, le système expert va autoriser ou rejeter la commande envoyée par la chaîne de commande. R2C effectue des vérifications puis rejette ou transfert la requête envoyée par la couche décisionnelle ; quant à MARAE, la couche fonctionnelle est développée de telle sorte qu'une commande est rejetée si l'exécution de celle-ci viole une des propriétés de sécurité.

- En enclenchant une action de sécurité dans le cas d'un dispositif de type moniteur de sécurité. Cela suppose que le système est dans un état dangereux pour la sécurité et que le moniteur de sécurité doit lancer une action de recouvrement afin d'empêcher que le système atteigne un état catastrophique. Cela est le cas pour le DLR Co-Worker qui, lorsque la gravité d'une collision atteint un certain niveau de gravité, enclenche une action de sécurité spécifique à ce niveau de gravité (arrêt d'urgence par exemple).

Une autre caractéristique importante est l'indépendance entre le dispositif de sécurité et le système surveillé (voir Table 1.1). L'indépendance permet de se prémunir des défaillances de modes communs et de la propagation des erreurs. Néanmoins, les contraintes financières ou physiques peuvent limiter le degré d'indépendance qui peut être assuré ; un compromis entre les objectifs de sécurité et les contraintes du système

| | Observation | Réaction | Type de dispositif | Indépendance physique | Processus fonctionnel et de surveillance indépendants ? |
|---|---|---|---|---|---|
| **Ranger** | Commandes internes (envoyées par le Main DMU et Monitor DMU au LPU et PMU pour la commande des axes). Variables observables (position des axes, vitesse de déplacement, état des outils de préhension, force exercée sur les objets, etc.) | Maintient du système dans l'état le plus sûr en comparant les commandes et en les autorisant ou rejetant | Interverrouillage | partielle | oui |
| **Elektra** | Commandes internes. Variables observables | Rejet de la commande, signal lumineux etc. | Interverrouillage | oui | oui |
| **R2C** | Commandes internes (requêtes émises par le superviseur). Variables observables (état du système) | Rejet des requêtes ou interruption des activités | Interverrouillage | non | non |
| **MARAE** | Commandes internes (requêtes émises par les clients). Variables observables (état du système) | Rejet des requêtes ou interruption des activités | Interverrouillage | non | non |
| **DLR Co-Worker** | Variables observables (position de l'homme, force exercée sur les articulations (collision), etc.) | Déclenchement des tâches de sécurité (freins d'urgence, passage mode zéro gravité etc.) selon le niveau de gravité de la collision | Moniteur de sécurité | non | non |

TABLE 1.1 – Résumé des niveaux d'observation et de réaction ainsi que des niveaux d'indépendance des exemples étudiés

doit être trouvé. Cette indépendance peut se décliner de deux manières :

- Indépendance logicielle : dans la Section 1.3.2.2 de ce chapitre, deux types de dispositifs de sécurité on été définis. En effet le *moniteur "in-line"* représente les dispositifs de vérification inclus dans le code ; cela s'avère le cas pour R2C, MARAE et le DLR Co-Worker. Le *moniteur "out-line"* représente un dispositif de sécurité exécuté dans un processus indépendant du système surveillé comme Elektra et Ranger. Dans Elektra, il existe deux chaînes indépendantes de traitement (la chaîne de commande et la chaîne de sécurité) reliées grâce à différents canaux de communication. De la même manière, Ranger se compose de deux unités de traitement des données (DMU) : la DMU principale (ou *Main DMU)* produit les commandes et effectue des vérifications de sécurité alors que le DMU moniteur (ou *Monitor DMU)* n'effectue que les vérifications de sécurité.

- Indépendance physique : dans Elektra, le *safety bag* est un *moniteur diversifié* exécuté sur une chaîne de traitement totalement indépendante ; dans Ranger l'indépendance est partielle car une copie du moniteur s'exécute sur le calculateur principal et l'autre sur le calculateur indépendant.

Les dispositifs de sécurité peuvent aussi se distinguer par le processus d'*elicitation* des contraintes de sécurité vérifiées, comment ces dernières sont *exprimées* et comment elles sont *surveillées* en ligne. La Table 1.2 illustre les cinq dispositifs étudiés selon ces trois caractéristiques. A notre connaissance, il n'existe pas de travaux qui traitent spécifiquement le problème de la définition de règles ou de contraintes de sécurité malgré l'abondance des recherches autour de la surveillance en ligne. Dans la majeure partie des travaux, les règles de sécurité sont basées sur des standards et/ou sur l'expérience des experts du domaine. Aucun processus systématique pour la définition de règles ou de contraintes n'a été identifié dans les travaux étudiés.

Concernant l'expression de ces contraintes de sécurité dans un langage particulier, il n'y a pas d'approche universelle. Le choix du langage dépend du niveau d'expressivité requis. Elektra, R2C et MARAE utilisent un langage dédié pour l'expression des règles de sécurité (PAMELA, le graphe acyclique dirigé généré par ExoGen et les connecteurs BIP respectivement).

La mise en œuvre de la vérification en ligne est, quant à elle, très peu documentée excepté pour R2C où les règles de sécurité sont transformées en un graphe OCRD (en utilisant ExoGen). Dans Ranger, le moteur d'inférence présent au sein du système expert de la chaîne de sécurité permet de choisir la règle applicable, et selon l'état courant du système décide de transmettre la requête ou de la bloquer. En réorganisant la couche fonctionnelle (précédemment développée en GenoM) en utilisant l'architecture BIP, l'approche MARAE permet de construire une couche fonctionnelle qui respecte, par construction, un ensemble de contraintes de sécurité. De la même façon que dans MARAE, Ranger et le DLR Co-Worker effectuent les vérifications lors de l'exécution du code.

| | Elektra | Ranger | R2C | MARAE | DLR Co-Worker |
|---|---|---|---|---|---|
| **Elicitation des règles/contraintes de sécurité** | Experts en ferroviaire, standards | Analyse de risque | Experts en robotique | Experts en robotique | Experts en robotique |
| **Expression des règles/contraintes** | PAMELA | Langage naturel | ExoGen | Connecteurs BIP | Incluses dans le code |
| **Vérification des règles/contraintes** | Moteur d'inférence de la chaîne de sécurité | Lors de l'exécution | OCRD | Lors de l'exécution | Lors de l'exécution |

TABLE 1.2 – Moyen d'elicitation, d'expression et de vérification des règles/contraintes de sécurité

## 1.5 Conclusion

Les systèmes autonomes, de par leur nature, sont des systèmes critiques dont toute défaillance peut avoir un effet catastrophique sur les humains et l'environnement dans lequel ils opèrent. Dans ce chapitre, nous avons présenté de façon générale les systèmes autonomes ainsi que les menaces dont ils peuvent faire l'objet. Nous avons par la suite présenté les notions de base et la terminologie de la sûreté de fonctionnement utilisée dans la suite de ce manuscrit.

Nous nous sommes particulièrement intéressée aux méthodes pour assurer la sécurité. Les méthodes hors-ligne telles que la vérification basée modèle ou la preuve de théorème visent à s'assurer que le système est sûr avant la phase opérationnelle. Néanmoins, ces méthodes ne sont pas suffisantes pour les systèmes autonomes vu la complexité de tels systèmes et la difficulté de vérifier un nombre d'états potentiellement très grand. De plus, ces méthodes sont appliquées à un modèle du système et en faisant des hypothèses sur l'environnement ; ce qui peut différer du système implémenté et de l'environnement réel d'exécution.

Les méthodes en-ligne représentent une approche complémentaire aux méthodes précédentes et permettent de traiter des fautes qui se manifesteraient lors de l'exécution, avant qu'elles ne causent des dommages à l'homme et à l'environnement. L'utilisation d'un dispositif de surveillance indépendant est fortement recommandée pour les systèmes autonomes, car il permet de surveiller le système lors de l'exécution et enclencher des actions de blocage ou de mise en sécurité afin d'éviter les défaillances catastrophiques.

Nous avons présenté différents travaux implémentant des dispositifs de sécurité. La spécification et la mise en œuvre d'un dispositif de sécurité diffère d'un système à un autre. Nous avons mis en exergue les différents niveaux d'indépendance des systèmes étudiés et avons positionné leurs activités d'observation et de réaction au sein des architectures étudiées.

De plus, nous avons identifié les processus de définition des contraintes ou propriétés de sécurité vérifiées par les moniteurs étudiés ainsi que leurs langages d'expression et leur implémentation. Lors de notre étude de l'existant, nous avons mis en évidence

l'absence d'un processus systématique pour la définition de telles propriétés vérifiables en ligne. Les propriétés utilisées lors de la vérification émanent des experts des domaines d'application et des normes, et sont en général exprimées dans des langages dédiés ou implémentées au sein même du code du système surveillé.

L'objectif de cette thèse est de pallier ce déficit de méthode de spécification des propriétés de sécurité en proposant une approche partant d'une identification des dangers et produisant un modèle cohérent de règles de sécurité implémentables dans des moniteurs de sécurité et/ou des systèmes d'interverrouillage.

Dans le chapitre suivant, nous nous intéressons aux méthodes d'identification des dangers. Nous introduisons les notions de risque et de gestion de risque avant d'aborder les méthodes d'analyse de risque et le processus qui permet d'en extraire des contraintes de sécurité.

## Ce qu'il faut retenir

1. Les mécanismes décisionnels, cœur des systèmes autonomes, sont particulièrement vulnérables et induisent de nombreuses sources d'erreurs.
2. La sûreté de fonctionnement offre différentes approches qui, appliquées de façon combinée, permettent d'avoir une confiance justifiée en le système.
3. L'environnement inconnu et dynamique dans lequel opèrent les systèmes autonomes ainsi que leur complexité font que les méthodes de vérification hors-ligne sont particulièrement délicates à mettre en œuvre.
4. La vérification en-ligne par un dispositif indépendant permet de gérer les fautes se manifestant lors de l'exécution et d'engager des actions de recouvrement lorsque cela est nécessaire. L'indépendance du dispositif de surveillance permet de se prémunir contre la propagation des erreurs et des défaillances de mode commun.
5. La définition des contraintes ou propriétés de sécurité est une problématique non abordée et devrait faire l'objet de recherches approfondies.

# Chapitre 2

# Identification de contraintes de sécurité

## Introduction

Lors du projet Apollo 1, des calculs des risques potentiels ont été effectués et la probabilité de succès de la mission s'est avérée extrêmement faible. Malgré cela, la NASA n'a pas remis en question la mission (la course vers la lune contre les soviétiques étant déclarée). En 1967, lors d'un exercice au sol d'Apollo 1, un incendie s'est déclaré dans la cabine tuant les 3 astronautes à bord. La mission, et toutes celles en cours, furent arrêtées pendant vingt et un mois. La NASA a alors chargé Boeing de mettre sur pied un projet dédié exclusivement à l'étude de la sécurité-innocuité du projet Apollo. Des analyses de risque par Arbre de Fautes ont été effectuées sur tout le système Apollo. Cet étude a été un élément fondateur de l'intérêt pour les analyses de risque [SV02].

L'objectif de nos recherches est de spécifier des contraintes de sécurité de façon systématique à partir des dangers potentiels induits par l'utilisation du système. Nous nous sommes donc particulièrement intéressée aux méthodes qui permettent l'identification des dangers. Il existe aujourd'hui de nombreux travaux de recherche dans ce domaine ; les techniques d'analyse de risque apparaissent comme le moyen permettant d'identifier les dangers et pour certaines d'évaluer leur gravité ou leur probabilité par exemple.

Ce chapitre présente une sélection de travaux et de concepts autour de la notion de risque qui nous permettront par la suite de déployer notre méthode. En premier lieu, nous présentons les notions de risque, de gestion des risques ainsi que quelques méthodes d'analyse de risque. Puis, nous décrirons une méthode particulière d'analyse de risque qui applique la méthode HazOp (Hazard Operability) au langage de modélisation UML (Unified Modeling Language). Enfin, nous présentons notre méthodologie d'extraction de contraintes de sécurité à partir des résultats de cette méthode d'analyse de risque.

## 2.1 La gestion du risque

La gestion du risque est aujourd'hui une discipline à part entière , standardisée dans de nombreux domaines. C'est une approche complémentaire aux concepts de la sûreté de fonctionnement présentés au Chapitre 1, car elle s'intéresse au processus d'analyse permettant par la suite de sélectionner les techniques de sûreté de fonctionnement.

### 2.1.1 Le risque

La norme ISO Guide 51 [IG99] définit la notion de risque comme étant : *la combinaison de la probabilité d'occurrence d'un dommage et de sa gravité*. La notion de *risque* est étroitement liée à la *sécurité-innocuité* (en anglais : *safety*), cette dernière étant définie par le guide comme étant : *l'absence de risques inacceptables*.

Cette définition met en évidence que la notion du "risque zéro" est utopique. Il y apparaît également une notion de *seuil* au delà duquel le risque devient inacceptable. En effet, le risque est dit *tolérable* s'il est *accepté dans un contexte particulier se basant sur les valeurs courantes de la société* [IG99].

De ce fait, un processus de gestion de risque a été défini afin de permettre d'identifier les risques potentiels, d'évaluer l'acceptabilité de ces risques et de permettre de statuer sur la nécessité de mesures correctives.

### 2.1.2 Les étapes de la gestion du risque

La gestion du risque est un processus itératif qui doit s'effectuer tout au long du cycle de vie du système. La Figure 2.1 illustre les étapes du processus de gestion du risque, qui se compose en trois parties :

1. L'analyse de risque : est *l'utilisation systématique des informations disponibles afin d'identifier les dangers et d'estimer le risque* [IG99].

   Rappelons que le risque est la combinaison de la probabilité d'occurrence d'un dommage et de sa gravité. La Table 2.1 illustre une matrice de risque à deux paramètres "Fréquence" et "Gravité". Si ces deux paramètres sont divisés en 6 niveaux chacun, nous obtenons 36 combinaisons possibles de la criticité du dommage. Plus la fréquence d'occurrence ou la gravité est importante, plus le risque est élevé. Ces différentes combinaisons sont ensuite catégorisées en niveaux de risque. On distingue dans cet exemple 4 niveaux : Haute, Intermédiaire, Basse et Nulle (H = High, I = Intermediate, L = Low, N = None).

2. L'évaluation du risque : se base sur l'étape précédente pour évaluer les risques et détermine si le risque est tolérable.

   Une fois le risque estimé, il convient de fixer les seuils d'acceptabilité du risque. A titre d'exemple, pour un système donné, dans un contexte donné et une utilisation spécifique, tous les risques de niveau H (High) seront considérés comme

inacceptables (et tous les les autres acceptables [1]).

3. La réduction de risque : permet, dans le cas où le risque est considéré comme inacceptable, de déployer les mesures nécessaires à la réduction des risques. Afin d'atteindre le seuil tolérable, il faudra agir sur les paramètres "Fréquence" et/ou "Gravité" : les mesures qui consistent à réduire la probabilité d'occurrence du dommage sont appelées mesures de *prévention* et les mesures qui consistent à réduire la gravité du dommage sont appelées mesures de *protection*.

   Suite aux actions de réduction du risque, il est alors nécessaire de refaire une analyse pour s'assurer que de nouveaux dangers n'ont pas été introduits ainsi que pour évaluer le risque résiduel.

| **Fréquence d'occurrence** | **Gravité** | | | | | | |
|---|---|---|---|---|---|---|---|
| | 6 Fatale | 5 Critique | 4 Sévère | 3 Sérieuse | 2 Modérée | 1 Mineure | 0 Néant |
| Fréquente | H | H | H | H | H | I | N |
| Probable | H | H | H | H | I | L | N |
| Occasionnelle | H | H | H | I | L | N | N |
| Rare | H | H | I | L | N | N | N |
| Invraisemblable | I | I | L | N | N | N | N |
| Impossible | L | L | N | N | N | N | N |

TABLE 2.1 – Exemple d'une matrice de risque

## 2.2 Techniques d'analyse de risque

Dans cette section, nous allons nous focaliser sur l'étape de l'analyse de risque, point de départ de notre méthodologie.

### 2.2.1 Analyse par Arbre de Fautes (AdF)

L'analyse par Arbre de Fautes AdF (ou *Fault Tree Analysis* FTA) est une méthode d'analyse de risque qui a émergé dans les années soixante aux Etats-Unis, à la suite de l'incident Apollo 1 décrit dans l'introduction, dans des missions liées à l'aérospatiale et la défense. Le concept fondamental des arbres de fautes est qu'à partir d'un événement indésirable, on détermine ses causes d'une manière descendante et systématique.

L'Analyse par Arbre de Fautes est une méthode *descendante* (top-down) dans laquelle un événement indésirable du système, du point de vue de la sécurité ou de la fiabilité, est analysé. Cet événement indésirable est considéré comme la racine de l'arbre,

1. Certaines normes distinguent aussi un niveau intermédiaire de risque nommé *tolérable*, où on doit chercher à mettre en œuvre une réduction de risque si cela peut être effectué avec un coût raisonnable (notion ALARP : As Low as Reasonably Practicable)

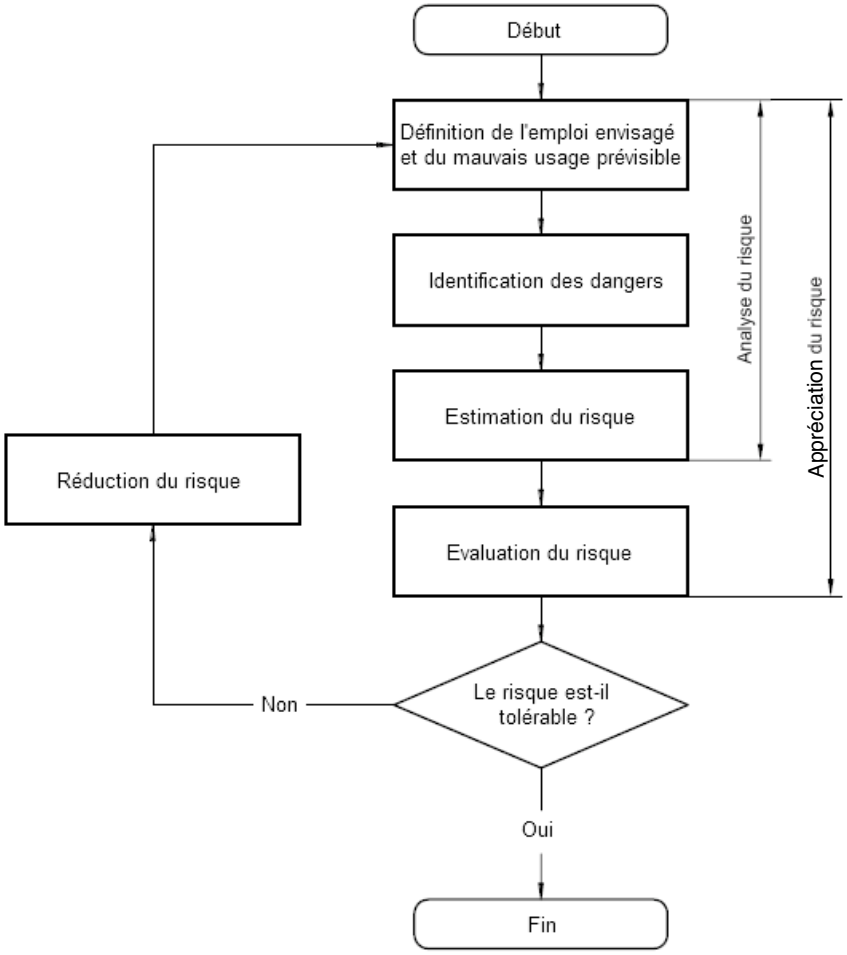


FIGURE 2.1 – Processus de gestion du risque, adapté de [IG99]

à partir duquel les *causes* qui mènent vers cet événement sont analysées. Chaque situation qui peut mener à cet effet est ajoutée à l'arbre sous forme d'expressions logiques. L'arbre est dessiné à l'aide de portes logiques conventionnelles. Chaque nouveau nœud de l'arbre est étudié à son tour et chaque situation qui peut mener à ce nouvel effet est ajoutée à l'arbre. L'opération est répétée jusqu'à ce que les effets ne puissent plus être décomposés [EI99]. L'arbre de fautes est un modèle graphique des combinaisons séquentielles ou parallèles de fautes qui résulteront en l'occurrence de l'effet indésirable pré-défini. Les fautes peuvent être des événements associés à des défaillances matérielles, des erreurs humaines, des erreurs logicielles, ou tout autre événement pertinent qui mènerait à l'événement indésirable. Il illustre les inter-relations logiques des événements de base qui conduisent vers l'événement indésirable, la racine de l'arbre (voir Figure 2.2).

L'Analyse par Arbre de Fautes permet de contribuer à prévenir les défaillances potentielles, et ainsi éviter des changements coûteux ultérieurs à la conception du

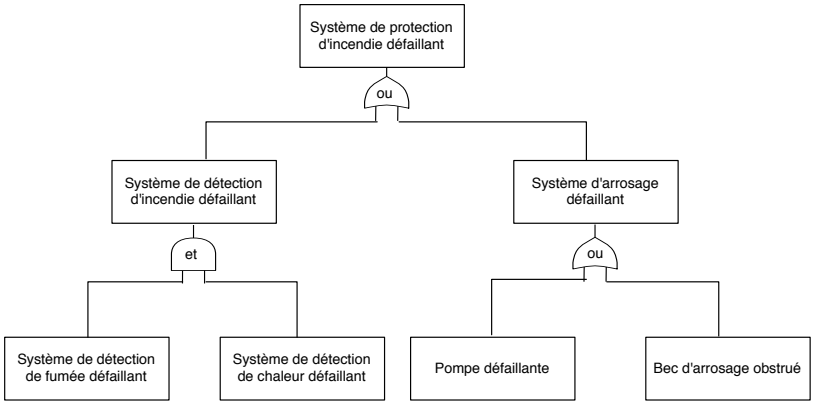


FIGURE 2.2 – Exemple d'un arbre de fautes

système. Elle peut également être utilisée comme outil de diagnostic afin de déduire les causes possibles d'un événement indésirable observé.

L'Analyse par Arbre de Fautes est une méthode essentiellement *qualitative*, dans le sens où elle renseigne sur les causes d'un événement indésirable. L'arbre peut être utilisé pour identifier ses coupes minimales. Une coupe est un ensemble d'événements pouvant provoquer l'événement indésirable situé à la racine de l'arbre. Une coupe est minimale lorsqu'elle ne contient aucune autre. Ainsi, les coupes minimales d'ordre un correspondent à des événements très critiques dans le sens qu'ils conduisent directement à l'événement redouté. Les coupes minimales d'ordre supérieur correspondent à des combinaisons d'événements qui doivent survenir ensemble pour conduire à l'événement redouté. On peut aussi évaluer la probabilité de l'événement redouté s'il est possible d'associer des probabilités d'occurrence aux feuilles de l'arbre et que les événements sont indépendants.

### 2.2.2 Analyse des Modes de Défaillance, de leurs Effets et leur Criticité (AMDEC)

Dans les années soixante, parallèlement à l'utilisation des arbres de fautes, l'Analyse des Modes de Défaillance, de leurs Effets et leur Criticité (AMDEC) (ou *Failure Mode, Effects and Criticality Analysis*, FMECA) était utilisée dans l'industrie automobile et spatiale. D'après la norme IEC61508-7 [IEC10], le but d'une analyse AMDEC est « *d'établir un ordre de criticité des composants qui pourraient être à l'origine d'une blessure, d'un préjudice ou d'une dégradation du système par le biais de défaillances uniques, afin de déterminer quels composants peuvent nécessiter une attention particulière et des mesures de surveillance nécessaires pendant la conception ou l'exploitation* ».

En effet, l'AMDEC est une évaluation du risque *ascendante (bottom-up)* qui examine les possibles modes de défaillance d'un sous système et de ses équipements, afin de déterminer les effets de telles défaillances sur les équipements et la performance du système. L'AMDEC est composée de deux analyses qui sont l'AMDE et l'Evaluation

de la Criticité. L'AMDE est une démarche ascendante consistant à identifier au niveau d'un système ou d'un de ses sous-ensembles, les modes potentiels de défaillance de ses éléments, leurs causes et leurs effets. L'Evaluation de la Criticité des modes de défaillance passe habituellement par la prise en considération croisée de la probabilité d'occurrence du mode, la possibilité de mettre en place une détection, et la gravité des conséquences.

L'AMDEC suit globalement les étapes suivantes :

- Définition et partitionnement du système en systèmes/sous-systèmes, ou bien équipement/unités, ou bien sous-ensembles/pièces élémentaires, avec une description fonctionnelle de ces sous-systèmes.
- Pour chaque élément de base, une liste complète des modes de défaillance est développée. Une matrice est construite pour chaque élément de base ; les différents modes de défaillance constituent alors les lignes de la matrice.
- Les effets de chaque défaillance sont déterminés et enregistrés dans la matrice AMDEC. Ils sont généralement classés par catégorie et sont adaptés au domaine d'application par les ingénieurs,
- Une classification de gravité est assignée à chaque mode de défaillance de chaque élément de base et est enregistrée dans la matrice AMDEC.
- Pour chaque mode de défaillance de chaque composant, la capacité du système à détecter et à signaler la défaillance est analysée.
- Pour chaque composant, la criticité du mode de défaillance est calculée à partir de différents paramètres (par exemple : indice de fréquence, la durée de la mission, indice de gravité, indice de détection).
- Selon le niveau de criticité, les éléments et les défaillances critiques sont mis en exergue afin qu'une réduction des risques y soit appliquée. La Figure 2.3 illustre une matrice AMDEC.

| N° | Sous-système | Composants | Mode de défaillance | Effet | | | HN | Causes possibles | Avant | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Système | Personne | Env | | | Gravité | Probabilité | Détectabilité | Criticité |
| 1 | BRAS-POIGNEE | Moteur | Perte de fonction | Pas de verticalisation<br>Pas de déverticalisation | | | HN2 | | 3 | 1 | 1 | 3 |
| 2 | | | Fonctionnement intempestif | Mouvement des bras non désiré | | | HN6 | | 4 | 1 | 5 | 20 |

FIGURE 2.3 – Extrait d'une matrice AMDEC avant réduction des risques

Il existe différents types d'analyse AMDEC, chacune donnant en sortie un type particulier de document de travail. En général, le résultat produit par cette analyse consiste en un rapport décrivant le système et les recommandations visant à réduire les conséquences des défaillances critiques (sélection de composants plus fiables, ajout de la redondance, intégration de dispositifs de surveillance, etc.). L'analyse AMDEC doit faire partie intégrale du processus de modélisation du système dès la phase conceptuelle. Lors de chaque révision de la conception, les recommandations doivent être intégrées afin de modifier, corriger et mettre à jour la conception du système.

### 2.2.3 Étude de danger et d'opérabilité (HazOp)

L'Étude de danger et d'opérabilité (ou *Hazard and Operability analysis*, HazOp) a pour but l'identification des dangers d'un système pour la sécurité-innocuité, leurs causes possibles, leurs conséquences ainsi que les actions recommandées pour minimiser leurs probabilités d'occurrence [IEC10, BAA+06].

A l'origine, HazOp a été développée dans les années 1970 par Imperial Chemical Industries (ICI) pour traiter les systèmes thermohydrauliques. Cette méthode est basée sur la collaboration d'un groupe d'experts dont les connaissances doivent couvrir suffisamment le système et ses applications. HazOp nécessite de la créativité afin d'imaginer les déviations possibles. Elle reste néanmoins un processus systématique du fait qu'elle applique à chaque *paramètre* du système (par exemple : la température, la vitesse, la pression) un *mot guide* (par exemple : pas de, plus de, trop de etc., voir Figure 2.4). Le résultat de cette conjonction est une *déviation* stockée dans une table nommée *table de déviations* (voir Figure 2.5). Pour chaque déviation, le groupe de travail doit déterminer les causes et les conséquences potentielles et doit identifier les moyens d'en prévenir l'occurrence ou en limiter les effets.

| Guideword | Possible interpretation |
|---|---|
| Missing | The output is not produced |
| Too high | The magnitude of the output is higher than intended |
| Too low | The magnitude of the output is lower than intended |
| As well as | Intended output but with additional result |
| Part of | Only part of the intended activity occurs |
| Reverse | The opposite of what is intended occurs |
| Other than | The intended does not happen but something else happens |
| Early | Something happens earlier than what is intended |
| Late | Something happens later than what is intended |
| Before | Something precedes something else that it should succeed |
| After | Something succeeds something else that it should precede |
| Inadvertent | Something happens when it should not |
| Stuck | The output does not change value |

FIGURE 2.4 – Exemple de mots guides HazOp [IEC01]

HazOp peut être considérée comme une méthode qualitative car, initialement, aucune estimation des probabilités d'occurrence des déviations ni l'estimation de leurs gravités n'ont été prévues. Cependant, HazOp peut être complétée par une analyse de criticité des risques, ce qui fait d'elle une méthode *quantitative* [IEC01]. HazOp a été adaptée à différents domaines et peut se décliner sous différentes formes en se focalisant sur le processus, le logiciel, les erreurs humaines, etc.

Néanmoins, avant toute analyse de ce type, le groupe de travail doit avoir assez de connaissances sur l'utilisation et le fonctionnement du système afin de pouvoir définir les paramètres sur lesquels les mots guides devront être appliqués. De plus, la liste des mots guides devra être revue et adaptée par les experts au domaine d'application de l'analyse.

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| **TITRE DE L'ÉTUDE:** EXEMPLE DE PROCÉDÉ | | | | | | | **FEUILLE: 1 de 4** | | |
| **N° du dessin** | | | **N° DE RÉVISION:** | | | | **DATE:** 17 décembre 1998 | | |
| **COMPOSITION DE L'ÉQUIPE:** | | | LB, DH, EK, NE, MG, JK | | | | **DATE DE LA RÉUNION:** 15 décembre 1998 | | |
| **PARTIE CONSIDÉRÉE:** | | | **Conduit de transfert du réservoir d'approvisionnement A au réacteur** | | | | | | |
| **INTENTION DE CONCEPTION:** | | | **Matériau:** A **Activité:** Transférer en continu à un débit supérieur à B<br>**Source:** Réservoir pour A **Destination:** Réacteur | | | | | | |
| **N°** | **Mot-guide** | **Elément** | **Déviation** | **Causes possibles** | **Conséquences** | **Protections** | **Commentaires** | **Mesures à prendre** | **Responsable mesures** |
| 1 | NE PAS FAIRE | Matériau A | Absence du matériau A | Réservoir d'approvision-nement A vide | Pas d'écoulement de A dans le réacteur<br>Explosion | Aucune apparente | Situation inacceptable | Prévoir l'installation sur le réservoir A d'une alarme de niveau bas, ainsi que d'un déclencheur à seuil bas pour arrêter la pompe B | MG |
| 2 | NE PAS FAIRE | Transférer A (à un débit >B) | Aucun transfert de A n'a lieu | Pompe A arrêtée, conduit obstrué | Explosion | Aucune apparente | Situation inacceptable | Mesurage du débit du matériau A, ainsi qu'une alarme de niveau bas, et un déclenchement de la pompe B en cas d'écoulement faible | JK |
| 3 | PLUS | Matériau A | Plus de matériau A: réservoir d'approvisionnement trop plein | Remplissage du réservoir à partir du camion-citerne alors que la capacité est insuffisante | Le réservoir va dépasser la limite de remplissage | Aucune apparente | Remarque: Ceci aurait dû être identifié durant l'examen du réservoir | Prévoir une alarme de niveau haut si non identifié précédemment | EK |

FIGURE 2.5 – Exemple d'une table de déviation HazOp [IEC01]

### 2.2.4 Discussion

La figure 2.6 récapitule les méthodes présentées, qui se scindent en deux sortes :

– les méthodes *ascendantes*, qui progressent des causes vers les effets et qui visent à déterminer les conséquences au niveau du système des défaillances des composants (AMDEC et HazOp),

– les méthodes *descendantes*, qui cheminent des effets vers les causes et qui visent à identifier les origines au niveau composant, des défaillances du système (Arbre de Fautes).

| **Méthodes** | **Phrase-clé** | **Sens de l'analyse** |
|---|---|---|
| **AMDEC** | Recenser les conséquences des défaillances et les évaluer | Ascendante |
| **AdF** | Organiser les éléments pouvant contribuer à un événement redouté | Descendante |
| **HazOp** | Recencer des déviations de comportement par l'application de mots guide et analyser leurs conséquences | Ascendante |

FIGURE 2.6 – Classement des méthodes d'analyse de risque

Rappelons que notre objectif est de spécifier des contraintes de sécurité à partir d'une liste de dangers. La méthode AMDEC permet d'identifier les modes de défaillance d'un sous-système et de ses équipements et de déterminer leurs causes ainsi que leurs effets. Afin d'obtenir ces modes de défaillance, il faut décomposer le système en sous-systèmes, les sous-systèmes en composants élémentaires avec la description fonctionnelle de ces derniers. Pour effectuer une telle analyse, il est essentiel d'avoir une connaissance précise de l'architecture du système étudié. Or, dans notre cas, nous souhaitons identifier tous les scénarios dangereux sans pour autant avoir fixé l'intégralité de l'architecture. Il est possible d'utiliser l'AMDEC à un haut niveau d'abstraction, ce qui revient à générer des tables d'analyses équivalentes à celles obtenues par l'analyse HazOp mais sans l'outil des mots guides.

A l'inverse de l'AMDEC, l'analyse par Arbres de Fautes permet de déterminer la combinaison d'événements pouvant mener à l'événement indésirable. Cependant, elle ne permet pas de lister l'ensemble des dangers redoutés. Elle nécessite d'identifier au préalable l'événement redouté et d'avoir, là aussi, une grande connaissance du système afin de pouvoir déduire les causes de l'événement redouté et de pouvoir les raffiner au maximum jusqu'à arriver à des événements non décomposables. L'AdF ne permet pas non plus d'identifier un enchaînement d'événements pouvant mener à une situation dangereuse, ce qui est souvent le cas dans un système complexe tel qu'un système autonome.

De plus, les analyses de types AdF et AMDEC sont, la plupart du temps, basées sur des représentations du système de type diagramme de bloc pour les structures fonctionnelles et de type automate pour les structures dynamiques. Or, ces modèles ne sont pas très bien adaptés aux systèmes autonomes. Tout d'abord, la structure décisionnelle de tels systèmes est difficilement décomposable en blocs fonctionnels. Ensuite, les automates ne sont pas adéquats pour modéliser la dynamique de systèmes qui opèrent de façon autonome dans un environnement non structuré, qui plus est en présence d'humains. Ces environnements non structurés impliquent un nombre infini de conditions d'opération, ce qui mène à une explosion combinatoire lors de l'utilisation de méthodes d'analyse des risques classiques.

La méthode HazOp a l'avantage d'identifier les dangers d'un système de façon systématique en appliquant à chaque paramètre d'un modèle système un mot guide, produisant une déviation. Il est également possible d'identifier des combinaisons d'événements, ou des séquences d'actions menant à une situation dangereuse. Afin de tirer avantage de cette méthodologie, Guiochet *et al.* [GMGP10, MGGPZ10, MGGP10] proposent d'adapter la méthode HazOp à des modèles d'utilisation du système, et non à l'analyse de l'architecture ou d'un modèle comportemental prescriptif. La connaissance détaillée de l'architecture du système étudié ne devient plus un obstacle.

## 2.3 L'analyse de risque HazOp/UML

L'approche est basée sur deux points importants : elle est focalisée sur les phases initiales du processus de développement (phase d'élicitation des exigences et phase de spécification), et elle adapte un processus d'analyse de risque standardisé qui est

HazOp. L'approche est conçue pour s'appliquer aux mêmes modèles que ceux utilisés pour le développement du système. Pour cela, l'analyse de risque HazOp a été adaptée pour être appliquée à une approche basée modèle répandue, qui intègre des activités humaines, qui est UML (Unified Modeling Language) [OMG03].

HazOp/UML est basée sur une description UML des scénarios d'utilisation et des interactions humain-robot. Il y a trois avantages à utiliser UML : 1) UML est un standard pour la description des systèmes ; 2) il est compréhensible par les non-experts ; 3) il est adapté aux premières phases de développement. Les deux diagrammes communément utilisés pour la description de l'utilisation du système sont les *cas d'utilisation* et les *diagrammes de séquences*.

## Diagramme de cas d'utilisation UML

Les cas d'utilisation interviennent très tôt dans la conception, et doivent permettre de concevoir, et de construire un système adapté aux besoins de l'utilisateur. Ils permettent de modéliser les besoins des clients d'un système. Leur objectif n'est pas de lister toutes les fonctionnalités du système, mais de clarifier, filtrer et organiser les besoins. Ces besoins définissent le contour du système à modéliser (ils précisent le but à atteindre) et permettent d'identifier les fonctionnalités principales du système.

Les cas d'utilisation énumèrent toutes les interactions envisageables entre le système et son environnement extérieur (voir Figure 2.7). Pour cela, on définit des acteurs (initiateurs d'actions) et le cas d'utilisation lui-même :

- le cas d'utilisation est représenté par une ellipse, dont le contenu est un texte décrivant le cas d'utilisation,
- les acteurs sont des entités externes qui interagissent avec le système (personne humaine, robot, etc.). Une même personne peut représenter plusieurs acteurs pour un même système ; pour cela les acteurs doivent être décrits par leur rôle.

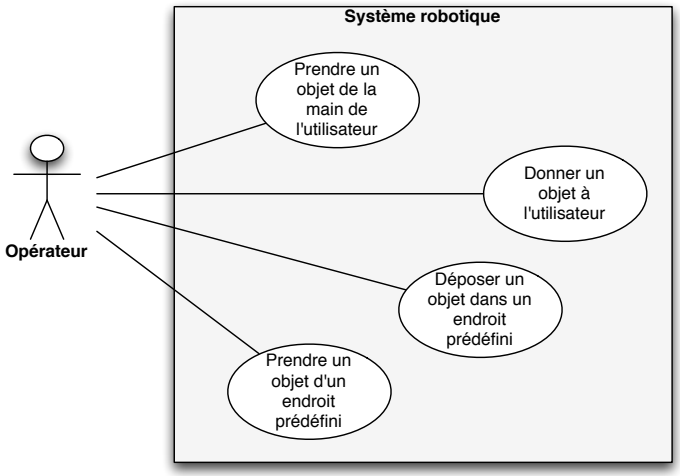


FIGURE 2.7 – Exemple d'un cas d'utilisation

Les cas d'utilisation interviennent à tous les niveaux de la conception. Ils per-

mettent de définir les fonctionnalités indispensables, et ainsi d'éviter l'introduction de fonctions inutiles ou inappropriées. Ils servent ainsi de base à la spécification, plus tard dans la conception, ils permettent de vérifier si les classes prévues remplissent effectivement les cas d'utilisation que l'on a définis ; et en fin de projet, ils servent de base aux tests, en indiquant quelles sont les fonctions à tester.

Les cas d'utilisation sont complétés par des conditions textuelles, appelées *attributs*. Les attributs représentent des propriétés physiques ou logiques d'un diagramme. Les attributs des cas d'utilisation qui ont été choisis dans le contexte d'HazOp/UML : les *pré-conditions* à remplir avant qu'une action du cas d'utilisation ne soit initiée, les *post-conditions* à satisfaire à la fin du cas d'utilisation et les *invariants* qui doivent être respectés lors du cas d'utilisation. Ces attributs permettront ultérieurement d'appliquer la méthode HazOp.

### Diagramme de séquence

Les diagrammes de séquences permettent de représenter des collaborations entre objets selon un point de vue temporel, en mettant l'accent sur la chronologie des envois de messages. L'ordre d'envoi d'un message est déterminé par sa position sur l'axe vertical du diagramme ; le temps s'écoule "de haut en bas" de cet axe. La disposition des objets sur l'axe horizontal n'a pas de conséquence pour la sémantique du diagramme. Chaque diagramme de séquence représente un scénario particulier d'un cas d'utilisation (voir Figure 2.8).

Les attributs des diagrammes de séquence considérés dans HazOp/UML sont les suivants :

- ordonnancement général : ordre de présence des messages au sein d'une interaction,
- événements temporels : dates des événements d'envoi et de réception des messages,
- objets de l'interaction : instances présentes lors de l'interaction,
- condition de garde des messages : clause (expression booléenne) qui conditionne l'envoi des messages,
- paramètres des messages.

La méthode HazOp est appliquée en sélectionnant les *attributs* des diagrammes et en y appliquant les *mots guides*. Une adaptation des mots guides a été proposée afin de pouvoir les appliquer aux attributs des cas d'utilisation et des diagrammes de séquences. La Table 2.2 illustre l'adaptation des mots guides pour les diagrammes de cas d'utilisation. La Table 2.3 illustre l'adaptation des mots guides pour les diagrammes de séquences. Une interprétation des déviations est donnée afin de guider le processus d'analyse. Tous les mots guides sont ainsi appliqués à tous les attributs afin d'éliciter les déviations.

Pour chaque déviation, l'analyste identifie l'effet sur le cas d'utilisation et sa conséquence sur le système. Les résultats de l'analyse sont une **liste des dangers potentiels** et les déviations qui peuvent y mener ainsi qu'une **liste de recommandations**

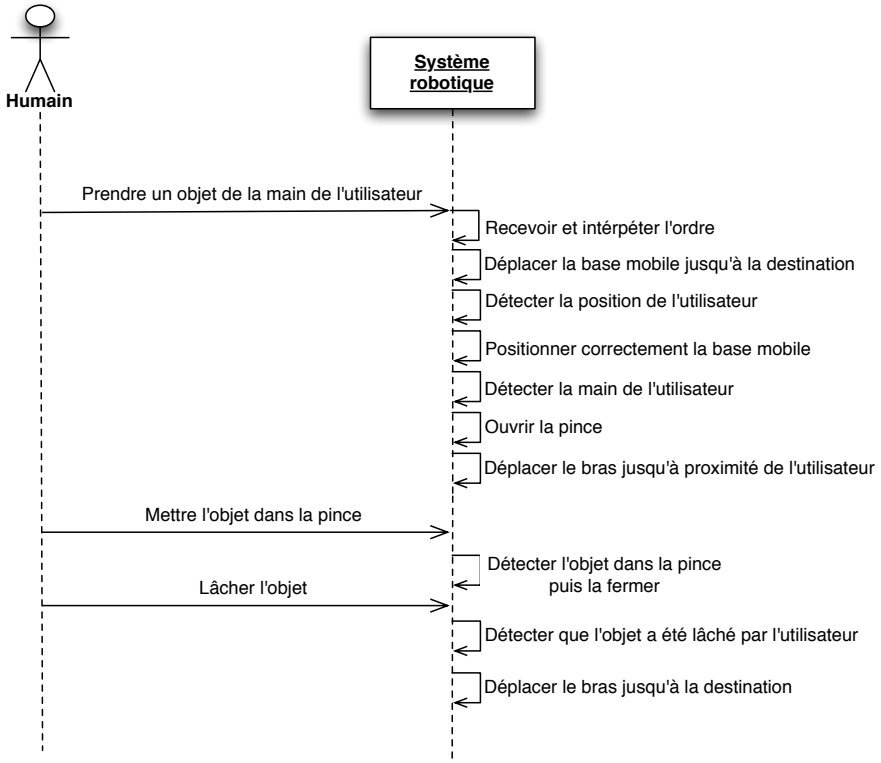


FIGURE 2.8 – Exemple d'un diagramme de séquence

| Entity = Use Case | | |
|---|---|---|
| **Attribute** | **Guideword** | **Interpretation** |
| Preconditions / Postconditions / Invariants | No/none | The condition is not evaluated and can have any value |
| | Other than | The condition is evaluated true whereas it is false<br>The condition is evaluated false whereas it is true |
| | As well as | The condition is correctly evaluated but other unexpected conditions are true |
| | Part of | The condition is partially evaluated<br>Some conditions are missing |
| | Early | The condition is evaluated earlier than required (other condition(s) should be tested before)<br>The condition is evaluated earlier than required for correct synchronization with the environment |
| | Late | The condition is evaluated later than required (condition(s) depending on this one should have already been tested)<br>The condition is evaluated later than required for correct synchronization with the environment |

TABLE 2.2 – Attributs, mots guides et interprétations pour le diagramme de cas d'utilisation, extrait de [GMGP10]

permettant, si implémentées, de réduire les risques (soit en réduisant la gravité du danger soit en réduisant sa probabilité d'occurrence).

La Table 2.4 donne un exemple d'une table de déviations résultant d'une analyse

| Entity = Sequence Diagram | | |
|---|---|---|
| **Attribute** | **Guideword** | **Interpretation** |
| Predecessors / successors during interaction | No | Message is not sent |
| | Other than | Unexpected message is sent |
| | As well as | Message is sent as well as another message |
| | More than | Message sent more often than intended |
| | Less than | Message sent less often than intended |
| | Before | Message sent before intended |
| | After | Message sent after intended |
| | Part of | Only a part of a set of messages is sent |
| | Reverse | Reverse order of expected messages |
| Message timing | As well as | Message sent at correct time and also at incorrect time |
| | Early | Message sent earlier than intended time |
| | Later | Message sent later than intended time |
| Sender / receiver objects | No | Message sent to but never received by intended object |
| | Other than | Message sent to wrong object |
| | As well as | Message sent to correct object and also an incorrect object |
| | Reverse | Source and destination objects are reversed |
| | More | Message sent to more objects than intended |
| | Less | Message sent to fewer objects than intended |
| Message guard condition | No/none | The condition is not evaluated and can have any value (omission) |
| | Other than | The condition is evaluated true whereas it is false, or vice versa (commission) |
| | As well as | The condition is well evaluated but other unexpected conditions are true |
| | Part of | Only a part of condition is correctly evaluated |
| | Late | The condition is evaluated later than required (other dependent condition(s) have been tested before)<br>The condition is evaluated later than correct synchronization with the environment |
| Message parameters / return parameters | No/None | Expected parameters are never set / returned |
| | More | Parameters values are higher than intended |
| | Less | Parameters values are lower than intended |
| | As Well As | Parameters are also transmitted with unexpected ones |
| | Part of | Only some parameters are transmitted<br>Some parameters are missing |
| | Other than | Parameter type / number are different from those expected by the receiver |

TABLE 2.3 – Attributs, mots guides et interprétations pour le diagramme de séquence, extrait de [GMGP10]

HazOp/UML. L'entête de la table contient les champs suivants :

1. l'entité UML à laquelle est appliqué l'analyse,
2. l'attribut ou élément considéré,
3. le mot guide appliqué,
4. la déviation résultant de l'interprétation de la combinaison d'un attribut et d'un mot guide,
5. l'effet au niveau du cas d'utilisation,
6. l'effet possible sur le système,
7. la gravité : estimation de l'effet du pire scénario sur le système,
8. les causes possibles de la déviation (matériel, logiciel, humain, etc.),
9. l'exigence du niveau d'intégrité : est la spécification préliminaire d'un niveau d'intégrité de sécurité pour certains éléments dans le but de spécifier des techniques adéquates de prévision et d'élimination de fautes afin d'éviter la déviation,
10. les nouvelles exigences de sécurité : si la déviation ne peut être évitée, une nouvelle exigence de sécurité est spécifiée,
11. les remarques : explication de l'analyse, recommandations supplémentaires, etc.,
12. le numéro du danger : les effets sur le système sont identifiés comme des dangers et un numéro est attribué à chacun d'entre eux.

TABLE 2.4 – Extrait d'une table de déviation de l'analyse HazOp/UML du diagramme de séquence "*prendre un objet de la main de l'utilisateur*" (projet PHRIENDS)

| **Projet :** PHRIENDS<br>**Table HazOp:** UC4<br>**Entité :** Diagramme de séquence UC1 : "Prendre un objet de la main de l'utilisateur" | | | | **Description du cas d'utilisation**<br>**Nom du cas d'utilisation: "Prendre un objet de la main de l'utilisateur"**<br>**Pré condition :** - aucun objet dans la pince<br>- emplacement atteignable<br>- l'objet peut être déplacé<br>**Post condition :** - la base du robot est à l'arrêt lors de l'interaction avec l'utilisateur<br>- l'objet est dans la pince<br>- le bras du robot est en mode "transport"<br>**Invariant :** aucun | | | | **Date :** 11/07/08<br>**Préparé par :** Ofaina Taofifenua<br>**Revu par :** Jérémie Guiochet<br>**Approuvé par :** | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Numéro de la ligne** | **Elément** | **Mot guide** | **Déviation** | **Effet sur le cas d'utilisation** | **Effet sur le système** | **Gravité** | **Causes Possibles** | **Exigence du niveau d'intégrité** | **Nouvelles exigences de sécurité** | **Remarques** | **Numéro du danger** |
| 1 | Aucun objet dans la pince | No/None | On suppose qu'il n'y a aucun objet dans la pince alors qu'il y en a un | Le robot va circuler avec un objet dans la pince et le lâche en voulant prendre un autre, l'objet tombe et heurte le second objet (si l'utilisateur n'intervient pas) | Impact physique / L'objet tombe / problème de performance | Critique | Défaillance logiciel / erreur humaine | Aucune | Intervention de l'utilisateur pour arrêter le robot | | 3,5,19 |
| | | | | | | | | | | | |
| 3 | | Other than | Un objet est détecté dans la pince alors qu'il y'en a aucun | Le robot n'exécute pas la tâche | Interruption de la séquence d'exécution / incompréhension entre l'utilisateur et le robot | Modérée | Défaillance physique (capteurs anti collision) \ défaillance logiciel | Aucune | Si le niveau d'intégrité n'est pas atteint, prévoir une redondance des capteurs / possibilité d'afficher l'état interne du système | afficher l'état interne du système permet de s'assurer que le robot et l'utilisateur ont la même perception du système | |
| | | | | | | | | | | | |
| 8 | L'objet peut être déplacé | Other than | L'objet est détecté comme déplaçable alors qu'il ne l'est pas | Le robot va faire tomber l'objet ou va le heurter (objet trop grand ou top petit) | Impact physique / L'objet tombe | Critique | Défaillance physique (capteurs) / défaillance logiciel / erreur humaine | Aucune | Si le niveau d'intégrité n'est pas atteint, redondances des capteurs | | 2 |
| | | | | | | | | | | | |
| 24 | La base du robot est à l'arrêt lors de l'interaction avec l'utilisateur | No/None | La base du robot n'est pas à l'arrêt lors de l'interaction avec l'utilisateur | Le robot bouge dangereusement | Impact physique / perte de confiance en le robot | Critique | Défaillance physique / Défaillance logiciel | Aucune | Arrêt d'urgence / Si le niveau d'intégrité n'est pas atteint, redondance matériel | | 1 |

| Projet :<br>Table HazOp:<br>Entité : | | | | | Description du cas d'utilisation<br>Nom du cas d'utilisation:<br>Pré condition :<br>Post condition :<br>Invariant : | | | | Date :<br>Préparé par :<br>Revu par :<br>Approuvé par : | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **Numéro de la ligne** | **Elément** | **Mot guide** | **Déviation** | **Effet sur le cas d'utilisation** | **Effet sur le système** | **Gravité** | **Causes Possibles** | **Exigence du niveau d'intégrité** | **Nouvelles exigences de sécurité** | **Remarques** | **Numéro du danger** |
| | | | | | | | | | | | |

⇓

Extraction de contraintes de sécurité

TABLE 2.5 – Colonnes concernées par l'extraction de contraintes de sécurité

## 2.4 Expression de contraintes de sécurité à partir de l'analyse HazOp/UML

Une fois l'analyse de risque HazOp/UML du système effectuée, le but est de spécifier à partir des résultats de cette analyse un ensemble de contraintes de sécurité qui serviront de base, par la suite, pour la spécification des propriétés qui seront surveillées par le dispositif de sécurité. L'objectif est de détecter une situation potentiellement dangereuse et d'agir afin de remettre le système dans un état sûr avant d'atteindre un état catastrophique.

### 2.4.1 Expression des contraintes de sécurité en langage naturel

Tout d'abord, nous définissons une contrainte de sécurité comme étant : *une condition* ***suffisante*** *pour empêcher l'occurrence d'une situation dangereuse* (nous reviendrons sur cette définition au Chapitre 3, Section 3.1). Les tables de déviations obtenues par l'analyse HazOp/UML sont parcourues une à une. A partir de chaque ligne d'une table HazOp dont la valuation de la colonne Gravité est différente de *nulle*, c'est-à-dire, Gravité $\in$ {*mineure, modérée, sérieuse, sévère, critique, fatale*}, nous tentons d'exprimer des contraintes de sécurité. A priori, ces contraintes pourraient être déduites à partir de la négation des affirmations portées, en langage naturel, dans les trois colonnes *Déviation*, *Effet sur le cas d'utilisation* et *Effet sur le système* (voir Figure 2.5).

Cependant, après l'étude de nombreux exemples de tables de déviations, nous avons conclu que :

1. Nous ne pouvons pas nous servir des conséquences décrites dans la colonne "Effet sur le système" pour exprimer des contraintes de sécurité car ce sont des effets décrits à haut niveau d'abstraction d'où il est très difficile de détecter la situation potentiellement dangereuse (en amont de la catastrophe) pour pouvoir agir en conséquence.

2. La colonne "Effet sur le cas d'utilisation" liste les conséquences des déviations sur le cas d'utilisation. Certains effets décrits dans cette colonne sont aussi de haut niveau d'abstraction et ne peuvent donc servir à exprimer une contrainte de sécurité à surveiller en ligne.

3. Les définitions obtenues dans la colonne "Déviation" sont, quant à elles, de bas niveau d'abstraction. Généralement, à partir de ces définitions, nous pouvons déceler relativement facilement les paramètres à observer et définir des contraintes de sécurité s'exprimant en fonction des valeurs de ces paramètres.

Spécifier les contraintes de sécurité à partir de la colonne *Déviation* permet de traiter les causes et prévenir les conséquences décrites dans les deux autres colonnes. Pour cette raison, dans le reste de ce manuscrit, nous nous focaliserons sur la colonne *Déviation* pour spécifier les contraintes de sécurité.

Pour illustrer cela, nous reprenons la table de déviations présentée dans la Table 2.4 qui est issue de l'analyse de risque HazOp/UML du bras robotisé mobile du projet PHRIENDS (Physical Human-Robot Interaction Dependability and Safety). Ce bras motorisé est monté sur une base mobile et opère dans un atelier au sein d'une usine et cela en présence d'humains (voir Figure 2.9). Un travail collaboratif peut avoir lieu entre l'homme et le robot (le robot peut donner un objet à l'humain par exemple). Le robot déambule librement dans cet environnement dynamique où d'autres objets sont en mouvement (humains, d'autres robots, etc.). La Table 2.6 liste les douze contraintes de sécurité extraites à partir de la Table 2.4.


FIGURE 2.9 – Bras motorisé du robot PHRIENDS

Considérons "l'Effet sur le système" de la déviation n° 2 : "Interruption de la séquence d'exécution / incompréhension entre l'utilisateur et le robot". A partir de cet effet, il est difficile d'exprimer une contrainte de sécurité utilisable. Typiquement, la négation d'un tel effet donnerait lieu à la contrainte de sécurité suivante : "Pas d'interruption de la séquence d'exécution / pas d'incompréhension entre l'utilisateur et le robot". A partir de cette contrainte très générale, nous ne pouvons pas déduire

| | Déviation | Effet sur le cas d'utilisation | Effet sur le système |
|---|---|---|---|
| | On suppose qu'il n'y a aucun objet dans la pince alors qu'il y en a un | Le robot va circuler avec un objet dans la pince et le lâche en voulant prendre un autre | Impact physique / L'objet tombe / problème de performance |
| ***Contraintes de sécurité en langage naturel*** | *Il ne faut pas qu'il y ait déjà un objet dans la pince* | *En allant prendre un objet, le robot ne doit pas circuler avec un objet dans la pince* | *Il ne faut pas que le robot lâche un objet* |
| | Un objet est détecté dans la pince alors qu'il y'en a aucun | Le robot n'exécute pas la tâche | Interruption de la séquence d'exécution / incompréhension entre l'utilisateur et le robot |
| ***Contraintes de sécurité en langage naturel*** | *Le robot ne doit pas détecter un objet dans la pince alors qu'il y'en a aucun* | *Le robot doit exécuter la tâche* | *Pas d'interruption de la séquence d'exécution / pas d'incompréhension entre l'utilisateur et le robot* |
| | L'objet est détecté comme déplaçable alors qu'il ne l'est pas | Le robot va faire tomber l'objet ou va le heurter (objet trop grand ou trop petit) | Impact physique / L'objet tombe |
| ***Contraintes de sécurité en langage naturel*** | *L'objet ne doit pas être détecté comme déplaçable alors qu'il ne l'est pas* | *Le robot ne doit pas faire tomber l'objet ou le heurter* | *Il ne faut pas que le robot cause un impact physique ou qu'il lâche un objet* |
| | La base du robot n'est pas à l'arrêt lors de l'interaction avec l'utilisateur | Le robot bouge dangereusement | Impact physique / perte de confiance en le robot |
| ***Contraintes de sécurité en langage naturel*** | *La base du robot doit être à l'arrêt lors de l'interaction avec l'utilisateur* | *Le robot ne doit pas bouger dangereusement* | *Il ne faut pas que le robot cause un impact physique* |

TABLE 2.6 – Contraintes de sécurité exprimées en langage naturel

les paramètres à observer afin d'exprimer une contrainte de sécurité qui pourra être surveillée en ligne.

Considérons l'effet de déviation n°4 sur le cas d'utilisation : "Le robot bouge dangereusement". Là aussi, nous ne pouvons pas en extraire une contrainte de sécurité exploitable vu la généralité de l'effet de la déviation.

Considérons la déviation n° 4, qui stipule que : "La base du robot n'est pas à l'arrêt lors de l'interaction avec l'utilisateur". L'interaction avec l'utilisateur étant définie par les concepteurs comme étant "Le bras du robot est en mouvement", nous pouvons conclure que pour traiter cette déviation, il faudra surveiller "la vitesse de la base du robot" lorsque "la vitesse du bras" n'est pas nulle.

### 2.4.2 Expression des contraintes de sécurité en langage formel

Nous tentons, par la suite, d'exprimer les contraintes de sécurité issues de la colonne Déviation sous forme de prédicats en utilisant des *variables pertinentes à la sécurité*, c'est-à-dire, des variables dont les changements de valeur ont un impact sur la sécurité du système.

Après l'étude de nombreux exemples d'analyse HazOp/UML, nous avons trouvé qu'il n'est pas possible de définir des contraintes de sécurité exploitables par un dispositif de sécurité dans les cas suivants :

1. Contraintes de sécurité non observables, et cela dans deux cas :

   (a) Variables pertinentes à la sécurité non observables : lorsque la contrainte de sécurité est exprimée à l'aide de variables pertinentes à la sécurité non observables par le dispositif de sécurité, ces contraintes ne peuvent être exploitées. De telles contraintes pointent les éléments critiques du système et

permettent de formuler des recommandations d'implémentation de moyens de perception supplémentaires afin d'élargir le champ de couverture du dispositif de sécurité, et donc augmenter la sécurité.

(b) La mise en oeuvre de certaines contraintes de sécurité dépend du type de dispositif de sécurité et à quel niveau de l'architecture du système surveillé il effectue les observations et réagit. Les contraintes de sécurité s'expriment parfois sous la forme de pré-conditions représentant des gardes à l'exécution de certaines actions. De telles contraintes ne peuvent être exploitées facilement que si le dispositif de sécurité peut intervenir au niveau de l'interface de requêtes d'exécution des actions, ce qui ne peut être le cas pour un dispositif de sécurité indépendant mis en œuvre de façon externe au système fonctionnel. En l'absence d'un tel accès, il faudrait faire appel à des moyens externes pour bloquer le déroulement des actions.

2. Certaines contraintes de sécurité indiquent la nécessité d'assurer une forte intégrité matérielle ou logicielle. Ce type de contraintes ne représentent pas des propriétés vérifiables en ligne ; néanmoins elles offrent une indication sur le besoin d'appliquer des mécanismes améliorant l'intégrité de l'élément considéré (intégrité des capteurs de température, intégrité du calcul du mode fonctionnel courant, etc.).

Pour illustrer les cas mis en exergue, considérons les contraintes identifiées dans la Table 2.6, colonne *Déviation* :

1. La 1ère contrainte stipule que : "Il ne faut pas qu'il y ait déjà un objet dans la pince". Cette contrainte exprime une pré-condition au déroulement du cas d'utilisation. Pour l'exploiter, le dispositif de sécurité doit pouvoir, d'une part, observer l'état de la pince (pleine ou vide) et, d'autre part, intervenir le cas échéant pour *empêcher* le déroulement du cas d'utilisation. Une fois le cas d'utilisation lancé, cette contrainte n'est plus pertinente car, justement, la pince sera occupée par l'objet pris de la main de l'utilisateur. Il s'agit donc d'une contrainte qui ne peut être mise en vigueur que par un mécanisme d'interverrouillage. Cet interverrouillage pourrait être réalisé par un dispositif interne ayant accès à l'interface d'appels des actions pour bloquer la requête "prendre objet de la main de l'utilisateur". Un dispositif externe nécessiterait la mise en œuvre d'un moyen de perception de l'état de la pince et d'un moyen de blocage physique ou électrique du déroulement du cas d'utilisation.
2. La 2ème contrainte stipule que : "Le robot ne doit pas détecter un objet dans la pince alors qu'il n'y en a aucun". Là aussi, l'observation concernée est l'état de la pince. Nous ne pouvons pas spécifier une contrainte de sécurité à vérifier en ligne car cette contrainte indique la nécessité que l'observation de l'état de la pince soit correcte. Afin de traiter cette contrainte, le seul recours est de réduire la probabilité d'occurrence d'une mauvaise observation ou en exigeant un haut niveau d'intégrité pour le capteur de la pince, par exemple, en implémentant une

redondance de celui-ci.

3. La 3ème contrainte stipule que : "L'objet ne doit pas être détecté comme déplaçable alors qu'il ne l'est pas". Afin de vérifier cette contrainte, il est nécessaire de pouvoir détecter l'objet ainsi que de pouvoir calculer ses dimensions afin que ses dernières soient comparées à la capacité de la prise en main de la pince du robot. Le dispositif de sécurité doit pouvoir percevoir l'objet et calculer, grâce à un algorithme, ses dimensions. Le dispositif doit aussi avoir connaissance des dimensions des objets que la pince peut porter. Ainsi, le dispositif pourra vérifier si le robot est en mesure de déplacer l'objet à l'aide de sa pince. En absence de moyens de perception appropriés, le dispositif ne sera pas capable de vérifier une telle contrainte de sécurité.

4. La 4ème contrainte stipule que : "La base du robot doit être à l'arrêt lors de l'interaction avec l'utilisateur". L'interaction avec l'utilisateur correspond au fait que le bras du robot soit en mouvement. Dans ce cas, les observations à effectuer pour la surveillance de cette contrainte sont : la vitesse de la base du robot et la vitesse du bras. Si ces deux variables sont observables par le dispositif de sécurité, nous pouvons exprimer la contrainte sous forme de prédicat comme suit : $(v\_base = 0) \vee (v\_bras = 0)$ ; ce qui signifie que la base du robot et le bras ne peuvent pas bouger en même temps. Dans le cas où une de ces deux variables n'est pas observable par le dispositif, il est nécessaire d'implémenter un mécanisme de perception spécifique. Si l'implémentation d'un tel mécanisme s'avère infaisable (par exemple, pour des raisons de coût), le système fonctionnel pourra éventuellement communiquer ses propres observations au dispositif de sécurité. En quel cas, il faut être conscient que ce dernier n'est pas à l'abri de défaillances de mode commun.

La Table 2.7 récapitule les contraintes de sécurité étudiées.

| | **Contraintes de sécurité en langage naturel** | **Classification de la contrainte** | **Recommandation** |
|---|---|---|---|
| **1** | Il ne faut pas qu'il y ait déjà un objet dans la pince | *Exploitable par un dispositif d'interverrouillage* | *Intervention à l'interface d'appel pour bloquer la requête "prendre objet" si "la pince est pleine"* |
| **2** | Le robot ne doit pas détecter un objet dans la pince alors qu'il y'en a aucun | *Exigence d'intégrité* | *Redondance des capteurs de la pince* |
| **3** | L'objet ne doit pas être détecté comme déplaçable alors qu'il ne l'est pas | *Variable pertinente à la sécurité non observable* | *Moyens de perception supplémentaires* |
| **4** | La base du robot doit être à l'arrêt lors de l'interaction avec l'utilisateur | Exploitable par un moniteur de sécurité | Surveillance continue de : (v_base=0) v (v_bras=0) |

TABLE 2.7 – Traitement des contraintes de sécurité exprimées en langage naturel

## 2.5 Conclusion

Dans ce chapitre nous avons présenté différentes méthodes d'analyse de risque qui ont pour but d'identifier des dangers afin de pouvoir effectuer par la suite l'évaluation des risques correspondants. L'évaluation des risques permettra de procéder à leur réduction, si nécessaire, afin de passer sous le seuil de tolérance et donc assurer la sécurité-innocuité du système.

Dans le but de pallier les limitations des méthodes classiques d'analyse de risque que sont l'Analyse par Arbre de Fautes et l'AMDEC, nous avons retenu une adaptation de la méthode standardisée HazOp aux modèles UML du système ; proposée par Guiochet *et al.* dans [GMGP10, MGGPZ10, MGGP10]. Cette méthode est particulièrement adaptée à l'analyse des interactions homme-machine ; de plus, elle est applicable dès les premières phases de développement du système. L'analyse de risque HazOp/UML est systématique et permet de générer une liste de dangers potentiels ainsi que les déviations qui peuvent y mener. L'explosion du nombre de tables HazOp et des déviations reste le plus souvent gérable car la méthode ne s'intéresse qu'aux cas d'utilisation et aux diagrammes de séquence modélisant les acteurs et le système. Cette méthode ne permet pas d'identifier les dangers issus de situations non prévues dans les scénarios d'interaction, néanmoins elle reste performante quant aux risques opérationnels.

Nous avons proposé dans ce chapitre une manière systématique d'extraire des contraintes de sécurité à partir des tables HazOp/UML de l'analyse de risque. Ces contraintes de sécurité représentent les données de base qui permettront de spécifier les propriétés à surveiller par un moniteur de sécurité. D'après l'application que nous avons effectuée sur le bras du robot PHRIENDS ainsi que sur le déambulateur robotisé MIRAS (présenté au Chapitre 4), il est généralement difficile d'obtenir des contraintes exploitables à partir des effets obtenus dans les colonnes "Effet sur le cas d'utilisation" et "Effets sur le système" et cela à cause de leur haut niveau d'abstraction. En revanche, la colonne "Déviation" permet, de façon plus importante que les deux autres, de spécifier des contraintes de sécurité exploitables et cela grâce à la nature bas-niveau des déviations qui y sont listées. En effet, les déviations obtenues dans cette colonne résultent de l'application d'un mot guide à un élément du cas d'utilisation qui correspond à un élément concret du système.

Nous avons identifié trois situations où la mise en oeuvres des contraintes identifiées n'est possible que sous conditions (conditions liées à l'architecture du moniteur de sécurité, au moyen d'observations disponibles, etc.). Cependant, de telles contraintes permettent d'effectuer des recommandations d'implémentation, qui une fois mises en œuvre, accroîtront le nombre de contraintes à surveiller et par conséquent amélioreront l'efficacité du moniteur de sécurité. Il est néanmoins nécessaire d'appliquer la méthode à de nombreuses analyses HazOp/UML afin de tester la complétude de cette liste et l'enrichir.

Dans le chapitre suivant, nous allons présenter un processus systématique de génération de règles de sécurité à partir d'une liste de contraintes de sécurité obtenues par la méthodologie exposée dans le présent chapitre.

## Ce qu'il faut retenir

1. Les méthodes d'analyse de risque classiques n'étant pas adaptées aux systèmes autonomes ; la méthode HazOp/UML a été développée afin de pallier ce manque.
2. Les tables de déviations obtenues par cette analyse de risque sont utilisées pour exprimer une liste de contraintes de sécurité.
3. Les colonnes "Effet sur le cas d'utilisation" et "Effet sur le système" listent des contraintes de sécurité de haut-niveau qui ne se prêtent pas à la spécification de contraintes de sécurité.
4. A partir d'une partie des éléments de la colonne "Déviation", et grâce à la démarche proposée, nous pouvons exprimer des contraintes de sécurité exploitables lors de la spécification d'un moniteur de sécurité.
5. Les éléments de la colonne "Déviation" restants permettent d'effectuer des recommandations aux experts du système dans le but d'augmenter la sécurité du système.

# Chapitre 3

# Spécification des règles de sécurité

## Introduction

Après avoir identifié un ensemble de contraintes de sécurité à partir de l'analyse des risques HazOp/UML du système fonctionnel, l'objectif est de définir à partir de ces contraintes un ensemble de propriétés que le moniteur de sécurité devra vérifier en ligne. Ce dernier doit être capable d'identifier que le système fonctionnel entre dans un état dangereux pour la sécurité-innocuité au sein duquel une action de recouvrement devra être exécutée, et cela afin d'empêcher l'occurrence d'une ou plusieurs situations catastrophiques.

Dans ce chapitre, nous introduisons le concept d'*états d'alerte* dans lesquels des actions de recouvrement (ou *actions de sécurité*) doivent être enclenchées. L'ensemble des états d'alerte représentent la *marge de sécurité* entre les états *nominaux* et les états *catastrophiques*, c'est-à-dire les états du système correspondants aux situations dangereuses.

Dans ce chapitre, nous proposons un processus systématique d'identification des états d'alerte à partir de la liste de contraintes de sécurité. Tout d'abord, les contraintes de sécurité seront formalisées par des prédicats sur des variables *environnementales* (que nous appellerons par la suite *variables pertinentes à la sécurité*). Ensuite, pour chaque contrainte de sécurité, nous présenterons une méthodologie de raffinement des invariants de sécurité et de définition, si possible, des marges de sécurité et par conséquent de l'ensemble des états d'alerte. Si une marge de sécurité ne peut pas être définie pour une variable en particulier, la contrainte de sécurité devra être traitée à l'aide d'un autre mécanisme de sécurité. Nous discuterons par la suite du choix des actions de sécurité qui seront engagées afin de remettre le système fonctionnel dans un état sûr. Pour finir, nous proposerons une approche qui permet de vérifier la cohérence des actions de sécurité qui pourraient être enclenchées simultanément.

## 3.1 Concepts de base

Dans la littérature parcourue au Chapitre 1, les auteurs ont souvent recours aux mêmes termes dans leurs travaux, comme par exemple : *contrainte de sécurité*, *propriété de sécurité*, *règle de sécurité*, *exigence de sécurité* etc., avec des significations différentes. Ces termes sont cependant rarement définis, ce qui donne lieu à différentes interprétations. Il nous a paru essentiel de lister les termes dont nous aurons besoin lors de nos travaux, ainsi que de proposer des définitions claires afin d'éviter toute ambiguïté lors de la présentation de notre méthodologie.

Comme définie dans le Chapitre 2, Section 2.1.1, la sécurité-innocuité correspond à *l'absence de risques inacceptables* [IG99]. Lors du processus de gestion de risque, des *objectifs de sécurité* sont formulés afin d'éviter des dommages, du moins avec un risque résiduel acceptable. Nous définissons un objectif de sécurité comme suit :

**Objectif de sécurité (ou Safety Requirement SR)** est une exigence générale de haut niveau d'abstraction de ce que signifie pour le système d'être sûr.

*Exemple : "le robot ne doit pas causer la chute du patient".*

La notion de contrainte de sécurité revêt plusieurs significations différentes dans la littérature. Parfois elle correspond à une contrainte absolue qui ne doit *jamais* être violée. Sa violation implique l'occurrence d'une catastrophe [Fir04]. Dans d'autres cas, il s'agit d'une contrainte plus "molle" spécifiant un seuil au-delà duquel le système n'est plus dans un état nominal et donc pour lequel des actions de sauvegarde devront être envisagées [MP09].

Nous préférons rendre explicite ces différentes significations au moyen de quatre concepts de base : *contrainte de sécurité*, *invariant de sécurité*, *action de sécurité* et *condition de déclenchement de sécurité*.

**Contrainte de Sécurité (ou Safety Constraint SC)** est une condition **suffisante** pour empêcher l'occurrence d'une situation dangereuse.

*Exemple : "le robot est stationné et n'est pas utilisé par le patient".*

**Invariant de Sécurité (ou Safety Invariant SI)** est une contrainte de sécurité **nécessaire**, c'est à dire, la violation d'un invariant de sécurité est intolérable car elle implique un risque inacceptable et la non satisfaction d'un objectif de sécurité.

*Exemple : "la vitesse du robot ne doit pas excéder* $3\ m/s$*" (où* $3\ m/s$ *est la vitesse au delà de laquelle le dommage est inéluctable).*

**Action de sécurité (ou Safety Action SA)** est une action explicite exercée pour mettre le système dans un état sûr.

*Exemple : "appliquer l'arrêt d'urgence".*

**Condition de Déclenchement de Sécurité (ou Safety Trigger Condition STC)** est une condition qui, lorsque valuée à vrai, enclenche une action de sécurité.

*Exemple : "la vitesse du robot excède* 2 $m/s$*".*

En s'appuyant sur ces quatre définitions, nous introduisons deux concepts supplémentaires :

**Marge de Sécurité (ou Safety Margin)** est la "distance" entre une condition de déclenchement de sécurité et la négation de l'invariant de sécurité.

*Exemple : dans l'exemple précédent, la marge de sécurité entre la condition de déclenchement de sécurité (c.-à-d., la vitesse du robot excède* 2 $m/s$*) et la négation de l'invariant de sécurité (c.-à-d., la vitesse du robot dépasse* 3 $m/s$*) est égale à* 1 $m/s$*.*

Une contrainte de sécurité et une condition de déclenchement de sécurité sont deux notions étroitement liées. En effet, une contrainte de sécurité peut être considérée soit comme un invariant de sécurité (donc condition nécessaire et suffisante), à partir duquel une marge de sécurité est *déduite* et une condition de déclenchement de sécurité est formulée. La contrainte de sécurité peut aussi être relâchée dans le cas où elle peut être temporairement violée. Dans ce cas, une marge de sécurité y est *ajoutée* afin d'exprimer la condition de déclenchement de sécurité.

**Règle de Sécurité (ou Safety Rule SR)** définit une façon de réagir en réponse à une situation dangereuse. Une règle de sécurité peut être exprimée comme une règle *si-alors* :
Règle de sécurité $\triangleq$ *si* [condition de déclenchement de sécurité] *alors* [action de sécurité].

*Exemple : "si la vitesse du robot dépasse les* 2 $m/s$ *alors procéder à l'arrêt d'urgence".*

La partie gauche de la Figure 3.1 illustre les concepts présentés ci-dessus en terme de partition des états possibles du système fonctionnel en états : *non-catastrophiques* et *catastrophiques*. Les états non-catastrophique sont réduits de la marge de sécurité pour constituer les états *sûrs* (voir Figure 3.1, partie droite). La marge de sécurité représente les états d'*alerte*. Une condition de déclenchement de sécurité doit être valuée à vrai quand le système passe d'un état sûr (exemple $\mathbf{x}_s$ dans la Figure 3.1) à un état d'alerte (exemple $\mathbf{x}_w$). Si le système est dans un état d'alerte, le moniteur de sécurité doit enclencher l'action de sécurité correspondante afin de remettre le système dans un état sûr. Si le moniteur de sécurité échoue et ne met pas le système dans un état sûr, il peut atteindre un état catastrophique (exemple $\mathbf{x}_c$).

La marge de sécurité est définie de telle sorte qu'elle puisse permettre un recouvrement. Selon les contraintes physiques qui régissent le système (la décélération maximale par exemple), la marge de sécurité doit permettre un recouvrement dans un temps imparti. Le choix de la marge et de l'action de sécurité doit rendre la transition de $x_w$ à $x_c$ improbable.

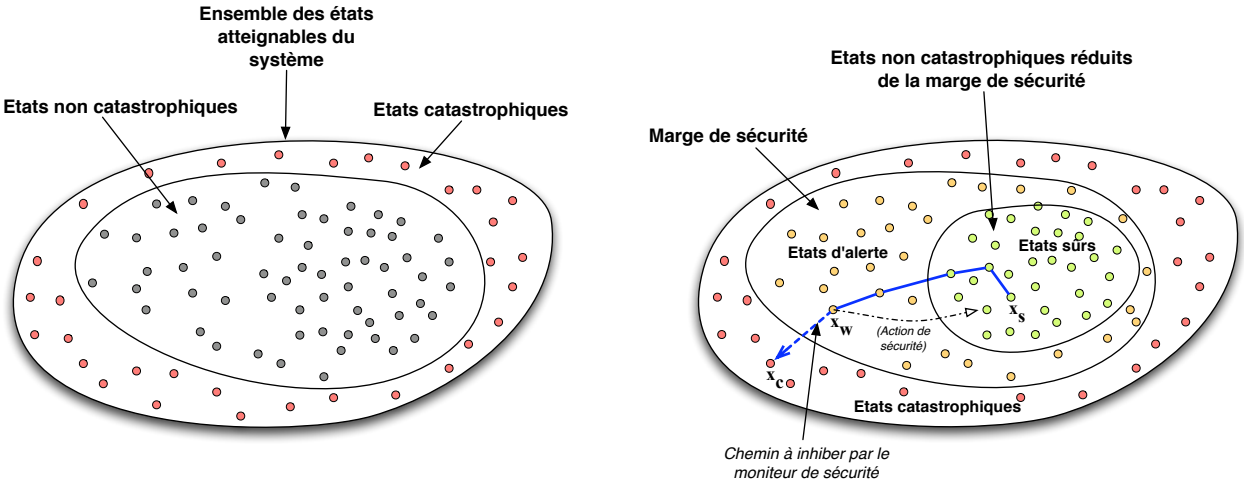


FIGURE 3.1 – Cas n°1 : une marge de sécurité globale peut être définie

De plus, la marge de sécurité ne doit pas être trop restrictive, c'est à dire, elle doit être définie de telle façon que le système puisse accomplir sa mission. Le calcul de la marge de sécurité est une étape cruciale car un compromis entre la sécurité et la disponibilité doit être trouvé. Cette étape nécessite une étroite collaboration des experts en sûreté de fonctionnement et les experts du domaine d'application du système.

Dans le cas où une marge de sécurité globale ne peut pas être définie, c'est à dire qu'il existe un chemin d'un état sûr à un état catastrophique qui ne passe pas par un état d'alerte, un interverrouillage de sécurité est nécessaire afin de supprimer ce chemin (voir Figure 3.2).

Dans le cadre des moniteurs de sécurité indépendants, il est souvent plus coûteux en terme de conception de mettre en place un interverrouillage plutôt que d'ajouter une action de sécurité au niveau du moniteur. De plus, l'interverrouillage est parfois impossible à implémenter car il existe des transitions qui ne peuvent pas être inhibées instantanément. Nous préférons utiliser ce mécanisme de sécurité comme dernier recours.

## 3.2 Invariants de sécurité et conditions de déclenchement de sécurité

Dans ce qui suit, nous définissons formellement les termes précédents afin de pouvoir déterminer les conditions pour lesquelles une marge de sécurité peut être identifiée.

Soit $\vartheta =< v_1, v_2, \ldots, v_n >$ le tuple des variables pertinentes à la sécurité et $dom(v_i)$ est le domaine de valeurs possibles pour $v_i, i \in [1, n]$.

Soit $\mathbf{x} \in X$ une valuation de $\vartheta$ tel que $X = dom(v_1)$ x $dom(v_2)$ x ... x $dom(v_n)$ représente l'ensemble des états atteignables du système.

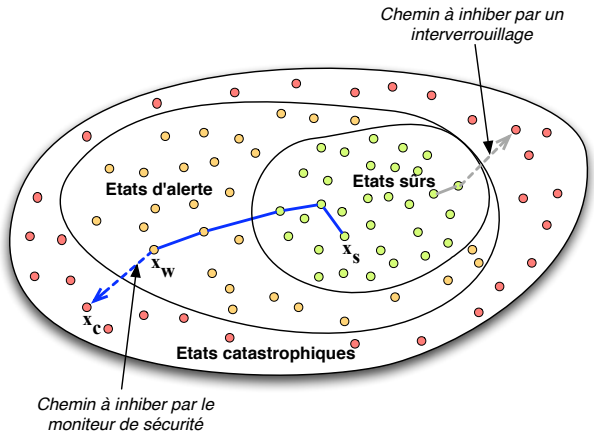


FIGURE 3.2 – Cas n°2 : une marge de sécurité globale ne peut pas toujours être définie

**Définition 3** (Invariant de sécurité)**.** *Un invariant de sécurité est un prédicat* $SI : X \rightarrow \mathbb{B}$ *tel que* $SI$ *est valué à vrai dans tout état non-catastrophique.*

L'ensemble des états catastrophiques $X_{cata}$ du système fonctionnel est défini comme suit :

$$X_{\text{cata}} = \{\mathbf{x} \in X \mid \overline{SI(\mathbf{x})}\} \tag{3.1}$$

Un moniteur de sécurité observe le tuple des variables pertinentes à la sécurité et évalue l'état du système à l'aide d'une condition de déclenchement de sécurité $STC : X \rightarrow \mathbb{B}$ qui est valuée à vrai lorsque le moniteur détecte que le système n'est pas dans un état sûr. Par conséquent, l'ensemble des états sûrs du système fonctionnel, tel que perçu par le moniteur de sécurité, est défini comme suit :

$$X_{\text{safe}} = \{\mathbf{x} \in X \mid \overline{STC(\mathbf{x})}\} \tag{3.2}$$

L'ensemble des états sûrs étant disjoint de l'ensemble des états catastrophiques, c.-à-d., $X_{\text{safe}} \cap X_{\text{cata}} = \emptyset$, on obtient :

$$\forall \mathbf{x} \in X : SI(\mathbf{x}) \vee STC(\mathbf{x}) \tag{3.3}$$

L'ensemble des états du systèmes qui ne sont ni sûrs ni catastrophiques constituent un sous-ensemble des états du système dans lesquels le moniteur de sécurité doit enclencher une action de sécurité afin que le système n'atteigne pas un état catastrophique et le ramener à un état sûr. Nous appelons ce sous ensemble, l'ensemble des *états d'alerte* (ou warning states). Il est défini comme suit :

$$X_{\text{warn}} = \{\mathbf{x} \in X \mid SI(\mathbf{x}) \wedge STC(\mathbf{x})\} \tag{3.4}$$

La Figure 3.3 illustre les états du système *sûrs*, *alerte* et *catastrophiques* avec les fonctions définies ci dessus.

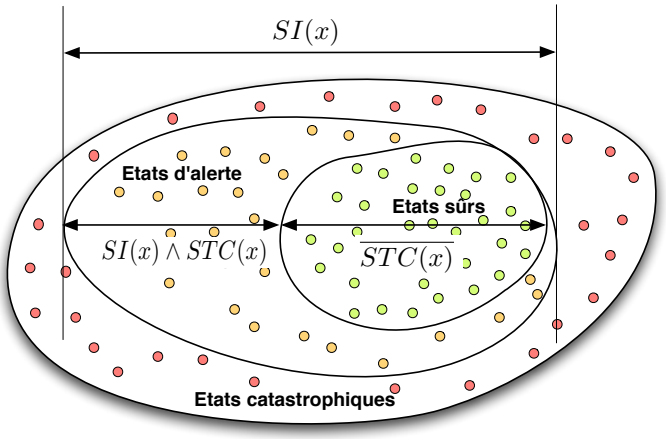


FIGURE 3.3 – Illustration des fonctions $SI$ et $STC$

Si l'on considère l'ensemble des transitions possibles entre les états du système, pour implémenter un moniteur de sécurité, nous exigeons que chaque *chemin* depuis un état sûr vers un état catastrophique passe par au moins un état d'alerte. Pour formaliser cette contrainte, considérons un état sûr $\mathbf{x}_s$, un état catastrophique $\mathbf{x}_c$ et un chemin de $\mathbf{x}_s$ vers $\mathbf{x}_c$ défini par la fonction $\pi$ :

$$\pi \begin{cases} [0,1] \rightarrow X \\ 0 \mapsto \mathbf{x}_s \\ 1 \mapsto \mathbf{x}_c \end{cases}$$

tel que $\pi$ est une fonction continue pour des variables continues, et monotone pour des variables discrètes. Nous définissons $\Pi(\mathbf{x}_s, \mathbf{x}_c) = \{\pi |\ \pi$ est un chemin de $\mathbf{x}_s$ à $\mathbf{x}_c\}$ comme étant l'ensemble de tous les chemins possibles. La contrainte qui stipule que chaque *chemin* depuis un état sûr vers un état catastrophique doit passer par au moins un état d'alerte est exprimée comme suit :

$$\begin{gathered} \forall \mathbf{x}_s, \mathbf{x}_c, \overline{STC(\mathbf{x}_s)}, \overline{SI(\mathbf{x}_c)}, \\ \forall \pi \in \Pi(\mathbf{x}_s, \mathbf{x}_c) : \exists\ t,\ \mathbf{x}_w = \pi(t),\ STC(\mathbf{x}_w) \wedge SI(\mathbf{x}_w) \end{gathered} \tag{3.5}$$

Il faut noter que cette contrainte impose qu'il n'existe pas d'état sûr *adjacent* à un état catastrophique. Si ceci n'est pas vérifié, un interverrouillage de sécurité serait nécessaire pour la suppression de la transition correspondante.

La condition de déclenchement de sécurité doit satisfaire les deux contraintes (3.3) et (3.5), ce qui mène à la définition suivante :

**Définition 4** (Condition de déclenchement de sécurité)**.** *Une* condition de déclenchement de sécurité *est un prédicat* $STC : X \rightarrow \mathbb{B}$ *tel que* $STC$ *est* fausse *dans les états sûrs,* vraie *dans les états catastrophiques, et passe à vrai* avant *d'atteindre un état*

*catastrophique sur tout chemin vers ce dernier depuis un état sûr. Formellement :*

$$\begin{aligned}\forall \mathbf{x}_c, \overline{SI(\mathbf{x}_c)} : \; & STC(\mathbf{x}_c) \\ & \wedge \left[\forall \mathbf{x}_s, \overline{STC(\mathbf{x}_s)}, \forall \pi \in \Pi(\mathbf{x}_s, \mathbf{x}_c) : \right. \\ & \quad \left. \exists\, t,\ \mathbf{x}_w = \pi(t),\ STC(\mathbf{x}_w) \wedge SI(\mathbf{x}_w)\right]\end{aligned}$$

## 3.3 Identification des conditions de déclenchement de sécurité

Dans cette section, nous allons décrire en détail comment, en partant d'une contrainte de sécurité, on peut définir une condition de déclenchement de sécurité ou bien, lorsque cela n'est pas possible, identifier la nécessité d'un interverrouillage de sécurité. Le but est que le moniteur de sécurité vérifie en ligne les conditions de déclenchement de sécurité. Pour cela, la contrainte de sécurité devra être exprimée à l'aide de variables pertinentes à la sécurité *observables*. Dans le cas où une ou plusieurs de ces variables ne sont pas observables, il faudra intégrer de nouveaux moyens d'observation.

### 3.3.1 Identification des états d'alerte

Dans un premier temps, nous faisons l'hypothèse que la contrainte de sécurité n'est pas seulement suffisante mais nécessaire, c'est-à-dire, qu'elle correspond à un invariant de sécurité.

Nous faisons le choix d'exprimer un invariant de sécurité global $SI$ dans une forme normale conjonctive : $SI = SI_1 \wedge \ldots \wedge SI_m$ afin de traiter par la suite chaque $SI_i$ séparément. Nous exprimons chaque invariant de sécurité élémentaire $SI_i$ comme une disjonction de termes : $SI_i = SI_{i(a)}(\mathbf{x}_a) \vee SI_{i(b)}(\mathbf{x}_b) \vee \ldots$ ($\mathbf{x}_a$ étant le tuple des variables utilisées dans $SI_{i(a)}$) de telle sorte que les variables pertinentes à la sécurité présentes dans chaque terme soient indépendantes et disjointes. Cette indépendance est nécessaire afin de pouvoir traiter la variation de chaque terme séparément (nous appellerons ces termes "atomes"). Si deux variables ne sont pas indépendantes (le volume et la température d'un gaz par exemple), l'invariant de sécurité devra être ré-exprimé pour qu'elles apparaissent dans le même atome. Une notation similaire est utilisée pour les conditions de déclenchement correspondantes, par exemple, $STC_{i(a)}$.

L'expression disjonctive d'un $SI_i$ est traitée comme suit (nous avons simplifié la notation en utilisant $a$, $b$, etc. pour désigner $SI_{i(a)}$, $SI_{i(b)}$, etc.) :

1. Construire le graphe correspondant à l'expression : $SI_i \equiv a \vee b \vee \ldots$ tel que les noeuds sont définis par des *minterms*[1] sur les atomes $a$, $b$, ..., et les transitions sont étiquetées par la valeur de l'atome provoquant la transition. Par exemple, dans la Figure 3.4, l'invariant de sécurité est $SI \equiv a \vee b$, ce qui donne lieu à

1. Un minterm représente le ET logique d'un ensemble de variables. Pour $n$ variables, il existe $2^n$ minterms.

4 noeuds ainsi que les transitions associées. Pour des raisons de simplification, nous limitons l'analyse aux transitions induisant un seul changement de valeur d'atome (les transitions induisant deux changements de valeur se traduisent en deux transitions successives correspondant au changement d'une seule valeur à la fois). Par exemple, la transition à partir du noeud $\bar{a}b$ (qui signifie $\bar{a} \wedge b$) vers le noeud $\bar{a}\bar{b}$ est étiquetée par $\bar{b}$, correspondant à l'événement $b \rightarrow \bar{b}$. Par analogie avec les automates temporisés, nous appelons le graphe issu de cette étape un graphe de région. Chaque noeud définit un ensemble d'états du système qui satisfait une condition (par exemple, la région 1 est l'ensemble des états du système qui satisfont $ab$). Une fois ce graphe construit, et en utilisant les définitions de la Section 3.2, nous pouvons calculer l'ensemble des régions *non-catastrophiques* et l'ensemble des régions *catastrophiques*.

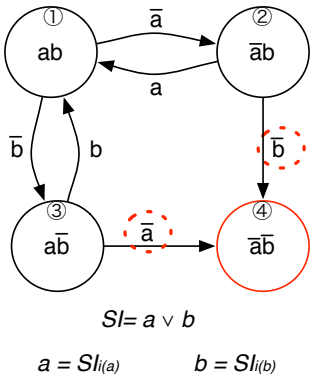


FIGURE 3.4 – Exemple d'un graphe de région

2. Nous nous intéressons tout particulièrement aux transitions entre une région *non-catastrophique* et une région *catastrophique*. Une telle transition sera nommée *transition critique* (par exemple les transitions $\bar{b}$ et $\bar{a}$ entourées de pointillés dans la Figure 3.4) et les régions non-catastrophiques correspondantes seront nommées *régions critiques* (par exemple dans la Figure 3.4, les régions (2) et (3) sont critiques). Nous analyserons chaque transition critique séparément pour déterminer si la région critique qui y correspond peut être scindée en une région *sûre* et une région d'*alerte*.

En se basant sur la Définition 4, nous pouvons poser une condition nécessaire à l'existence d'une marge de sécurité comme suit :

**Condition 1** (Existence d'une région d'alerte)**.** *Une région d'alerte existe pour la transition critique si et seulement s'il existe une partition de la région critique associée en deux sous-régions non vides.*

L'existence de la marge de sécurité est validée lorsque la condition de déclenchement de sécurité est spécifiée et que l'on a pu définir une action de sécurité correspondante pouvant réaliser un recouvrement effectif.

**Théorème :**
Soit $SI_i(\mathbf{x}) = SI_{i(a)}(\mathbf{x}_a) \vee SI_{i(b)}(\mathbf{x}_b)$ l'invariant de sécurité à traiter.

Une région d'alerte existe pour la transition critique $\overline{SI_{i(a)}}(\mathbf{x}_a)$ si et seulement s'il existe une partition de l'ensemble $X^{SI}_{(a)} = \{ \mathbf{x}_a \in X_{(a)} | SI_{i(a)}(\mathbf{x}_a)\}$ en deux sous-ensembles non vides. L'ensemble $X^{SI}_{(a)}$ est appelé l'extension de l'atome $SI_{i(a)}(\mathbf{x}_a)$.

Soit $X^{\overline{SI}}_{(a)}$ l'ensemble des états catastrophiques correspondants : $X^{\overline{SI}}_{(a)} = \{ \mathbf{x}_a \in X_{(a)} | \overline{SI_{i(a)}}(\mathbf{x}_a)\}$.

Soient l'ensemble $adherence(X^{SI}_{(a)})$ l'adhérence de $X^{SI}_{(a)}$ et $adherence(X^{\overline{SI}}_{(a)})$ l'adhérence de $X^{\overline{SI}}_{(a)}$.

L'ensemble des états de la *frontière* entre l'ensemble $X^{SI}_{(a)}$ et l'ensemble $X^{\overline{SI}}_{(a)}$ est nommé $X_F$ et est défini comme suit : $X_F = adherence\ (X^{SI}_{(a)}) \cap\ adherence\ (X^{\overline{SI}}_{(a)})$.

Une partition de l'ensemble $X^{SI}_{(a)}$ en deux ensembles non vides existe pour la transition critique $\overline{SI_{i(a)}}(\mathbf{x}_a)$ si et seulement si :

$$Card(\{X^{SI}_{(a)} \setminus X_F\}) > 1 \tag{3.6}$$

N. B. : L'ensemble à partitionner $X^{SI}_{(a)}$ est réduit de sa frontière afin de pouvoir traiter indifféremment les deux cas : l'inégalité **stricte** et l'inégalité **large** dans les conditions de définition de $X^{SI}_{(a)}$.

*Preuve :*
Rappelons que le lien entre les régions d'alerte et la condition de déclenchement de sécurité STC a été exprimé dans la Condition 3.5 :

$$\forall \mathbf{x}_s, \mathbf{x}_c, \overline{STC(\mathbf{x}_s)}, \overline{SI(\mathbf{x}_c)},$$
$$\forall \pi \in \Pi(\mathbf{x}_s, \mathbf{x}_c) : \exists\ t,\ \mathbf{x}_w = \pi(t),\ STC(\mathbf{x}_w) \wedge SI(\mathbf{x}_w)$$

*Cas* $\Rightarrow$ : en utilisant les notations ci dessus, nous supposons que la partition de $X^{SI}_{(a)}$ en deux sous-ensembles $X_1$ et $X_2$ non vides existe : $X_1 = \{ \mathbf{x}_a \in X_{(a)} | SI_{i(a)}(\mathbf{x}_a) \wedge P(\mathbf{x}_a)\}$ et $X_2 = \{\mathbf{x}_a \in X_{(a)} | SI_{i(a)}(\mathbf{x}_a) \wedge \overline{P}(\mathbf{x}_a)\}$ tel que $P(\mathbf{x}_a) \Rightarrow SI_{i(a)}(\mathbf{x}_a)$, $P$ étant un prédicat sur $\mathbf{x}_a$.

Soit $\mathbf{x}_1 \in X_1$ et soit $\mathbf{x}_2 \in X^{\overline{SI}}_{(a)}$. Soit un chemin $\pi$ qui va de $\mathbf{x}_1$ à $\mathbf{x}_2$ et qui passe par la transition critique $\overline{SI_{i(a)}}$. Il existe donc un $t_0$ tel que $\pi(t_0)$ satisfait $\overline{SI_{i(a)}}$ et $\pi(t_0 - \epsilon)$ satisfait $SI_{i(a)}$ pour un $\epsilon$ petit ($\{X^{SI}_{(a)} \setminus X_F\}$ est ouvert à gauche sur le chemin $\pi$, voir Figure 3.5).

Par conséquent, le $Card(\{X^{SI}_{(a)} \setminus X_F\})$ est supérieur à 1 car l'ensemble $\{X^{SI}_{(a)} \setminus X_F\}$ contient au moins deux éléments qui sont : $\mathbf{x}_1$ et $\pi(t_0 - \epsilon)$.

Dans ce cas, la condition de déclenchement de sécurité STC est $\overline{P}(\mathbf{x}_a)$.

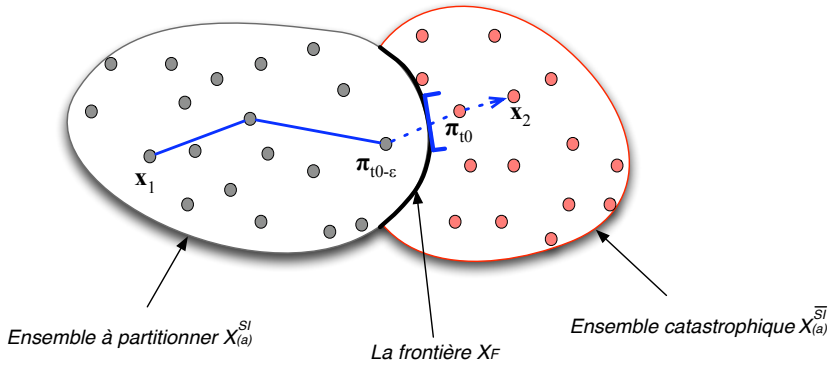


FIGURE 3.5 – Illustration de la possibilité de partitionner l'ensemble non-catastrophique

*Cas* $\Leftarrow$ : Supposons que la Condition 3.6 est satisfaite. Par conséquent, soit ($\mathbf{x}_1$ et $\mathbf{x}_2$) $\in \{X_{(a)}^{SI} \setminus X_F\}$ (tel que $\mathbf{x}_1 \neq \mathbf{x}_2$). Soit $\{\mathbf{x}_1\}$ le premier ensemble. Il existe donc un second ensemble $X_{(a)}^{SI} \setminus X_F \setminus \{x_1\}$ qui contient au moins un élément. L'ensemble de départ $\{X_{(a)}^{SI} \setminus X_F\}$ peut donc être partitionné en deux ensembles non vides. □

La Figure 3.6 illustre le cas nominal du partionnement où l'invariant de sécurité $SI_i$ est un atome $a$. La région critique $\{x \in X | a\}$ est partitionnée en deux sous régions : $\{x \in X | a \wedge a'\} \in X_{\text{safe}}$ et $\{x \in X \mid a \wedge \bar{a}'\} \in X_{\text{warn}}$. Dans ce graphe, lors de la violation de $a'$, le système entre dans une région d'alerte $a\bar{a}'$, ce qui enclenche une action de sécurité.

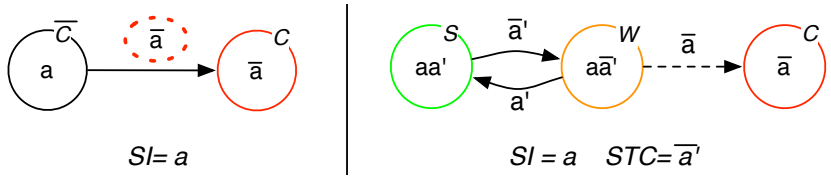


FIGURE 3.6 – Le graphe de région avant et après la spécification de la marge de sécurité

### 3.3.2 Impossibilité de définir les états d'alerte

Dans le cas où la Condition 3.6 n'est pas vérifiée pour une région critique, ou bien la partition proposée et la condition de déclenchement de sécurité correspondante ne sont pas validées par les experts du domaine, deux solutions peuvent être envisagées :

1. Considérer que la contrainte de sécurité n'est pas une condition nécessaire. Cela revient à définir un invariant de sécurité qui est moins contraignant. Ce relâchement de la contrainte doit évidemment être accompagné par une ré-evaluation des risques par les experts du domaine.

Une façon de faire cela est d'envisager que la contrainte de sécurité puisse être violée temporairement.

La Figure 3.7 illustre le graphe de région obtenu après modification de l'invariant de sécurité, où $a'' = (t_{\bar{a}} < \tau)$, $t_{\bar{a}}$ est le temps écoulé depuis la violation de $a$ et $\tau$ est la marge de sécurité temporelle. La nouvelle région catastrophique est définie par le prédicat $\bar{a}\bar{a''}$ qui représente la violation de l'ancien invariant $a$ pendant un laps de temps trop grand (défini par $\bar{a''}$). La région critique devient $\{x \in X | a \vee a''\}$ composée de deux sous régions : $\{x \in X | a\} \in X_{\text{safe}}$ et $\{x \in X \mid \bar{a}\ a''\} \in X_{\text{warn}}$. Une action de sécurité devra être engagée lorsque le prédicat $a$ est violé.

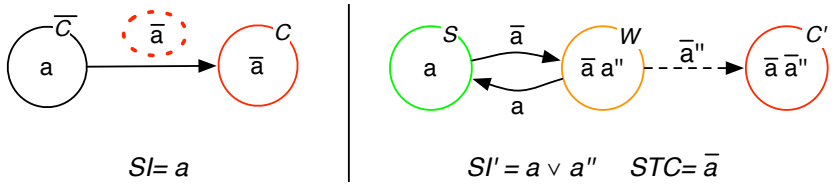


FIGURE 3.7 – Le graphe de région après redéfinition de l'invariant de sécurité et spécification de la marge de sécurité

2. Implémenter un *interverrouillage de sécurité* afin d'empêcher la transition critique. L'interverrouillage peut se décliner sous deux formes : 1) un interverrouillage logiciel, sa mise en œuvre nécessiterait une modification du logiciel du système surveillé ; 2) un interverrouillage physique qui nécessite une modification de l'électromécanique du système surveillé et qui peut être lourd à mettre en œuvre. C'est pour ces raisons que nous considérons l'implémentation d'un interverrouillage qu'en dernier recours.

   La figure 3.8 illustre l'inhibition de la transition $\bar{a}$ par un mécanisme d'interverrouillage.

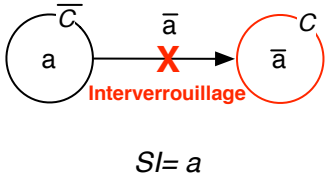


FIGURE 3.8 – Le graphe de région après implémentation d'un interverrouillage

### 3.3.3 Calcul des marges de sécurité pour les variables continues

Considérons, dans ce paragraphe, le cas particulier (assez fréquent en pratique) de la théorie développée en Section 3.2, où le tuple des $n$ variables pertinentes à la sécurité prend ses valeurs dans $\mathbb{R}^n$. Dans ce cas, on peut définir une méthode constructive pour spécifier une marge de sécurité multi-dimensionnelle.

L'invariant de sécurité et la condition de déclenchement de sécurité peuvent être définis comme des relations d'inégalité sur une ou plusieurs variables continues pertinentes à la sécurité :

$$SI(\mathbf{x}) = (f(\mathbf{x}) < 0) \tag{3.7}$$
$$STC(\mathbf{x}) = (g(\mathbf{x}) \geq 0) \tag{3.8}$$

où $f, g : X \to \mathbb{R}$.

Dans ce cas, il est possible de reformuler 3.1 et 3.2 comme suit :

$$X_{\text{cata}} = \{\mathbf{x} \in X \mid f(\mathbf{x}) \geq 0\}$$

$$X_{\text{safe}} = \{\mathbf{x} \in X \mid g(\mathbf{x}) < 0\}$$

La Définition 4 devient alors :

**Définition 5.** *Lorsqu'un invariant de sécurité et la condition de déclenchement de sécurité correspondante sont définis en termes d'inégalités de fonctions de variables pertinentes à la sécurité continues :* $SI(\mathbf{x}) = (f(\mathbf{x}) < 0)$ *and* $STC(\mathbf{x}) = (g(\mathbf{x}) \geq 0)$*, la fonction* $g(\mathbf{x})$ *doit satisfaire ce qui suit :*

$$\begin{aligned}\forall \mathbf{x}_c, (f(\mathbf{x}_c) \geq 0) : (g(\mathbf{x}_c) \geq 0) \\ \wedge [\forall \mathbf{x}_s, (g(\mathbf{x}_s) < 0), \forall \pi \in \Pi(\mathbf{x}_s, \mathbf{x}_c) : \\ \exists\ t,\ \mathbf{x}_w = \pi(t), \\ (g(\mathbf{x}_w) \geq 0) \wedge (f(\mathbf{x}_w) < 0)]\end{aligned}$$

Dans un état d'alerte $\mathbf{x}_w$, d'après (3.4), (3.7) et (3.8) nous avons $f(\mathbf{x}_w) < 0$ et $g(\mathbf{x}_w) \geqslant 0$. Nous reformulons 3.4 comme suit :

$$X_{\text{warn}} = \{\mathbf{x} \in X \mid f(\mathbf{x}_w) < 0 \wedge g(\mathbf{x}_w)\}$$

Afin de pouvoir représenter l'évolution de $f(\mathbf{x})$ et $g(\mathbf{x})$ tout au long du chemin $\pi(\mathbf{x})$ entre deux états, nous définissons deux fonctions $f_\pi(\mathbf{x})$ et $g_\pi(\mathbf{x})$ comme suit :

$$f_\pi(t) = f(\mathbf{x})|_{\mathbf{x}=\pi(t)} \tag{3.9}$$
$$g_\pi(t) = g(\mathbf{x})|_{\mathbf{x}=\pi(t)} \tag{3.10}$$

Lorsque nous considérons un chemin $\pi(t)$ tel que $\mathbf{x}_s = \pi(0)$ et $\mathbf{x}_c = \pi(1)$, les Figure 3.9 et 3.10 illustrent deux exemples d'expression de la fonction $g_\pi(t)$.

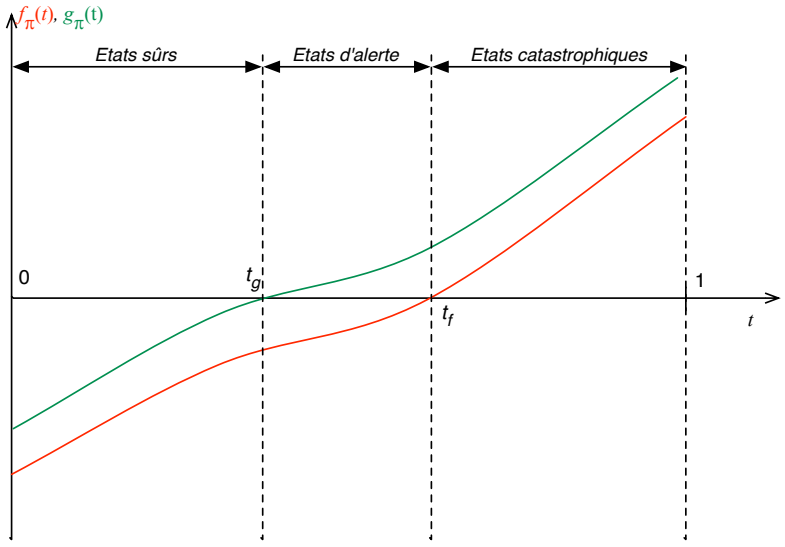


FIGURE 3.9 – Cas simple : $g_\pi(t) = f_\pi(t) + \theta$

**Définition 6.** *Un chemin $\pi$ est* f-monotone *sur $[t_1, t_2] \subseteq [0, 1]$ si, pour un $\epsilon$ infinitésimal :*

$$\forall t \in [t_1, t_2], f(t + \epsilon) > f(t)$$

Par exemple, le chemin $\pi$ de la Figure 3.9 est *f-monotone* sur $[0, 1]$. Un segment f-monotone $[t', t_f]$ du chemin $\pi$, où $t_f$ représente un point tel que $f(t_f) = 0$ et $t' \in [0, t_f]$, est un segment du chemin au long duquel le système s'approche continuellement de l'état catastrophique puis y entre.

Il faudra noter qu'il n'y a pas d'exigence stricte quant à la monotonie de la fonction $g_\pi(t)$ sur un segment du chemin f-monotone. La Figure 3.10 représente une condition de déclenchement de sécurité $g_\pi(t)$ parfaitement valide par rapport à $f_\pi(t)$. Cependant, il serait intéressant de choisir la fonction $g(\mathbf{x})$ telle que $g_\pi(t)$ soit effectivement monotone sur un segment du chemin $[t', t_f]$ f-monotone (voir Figure 3.9), car dans ce cas, la fonction de déclenchement de sécurité $g(\mathbf{x})$ révèle un danger croissant. Nous appelons une telle fonction de déclenchement de sécurité une fonction *révélatrice*.

Une façon simple de définir une fonction de déclenchement de sécurité ayant cette propriété est de considérer que :

$$g(\mathbf{x}) = f(\mathbf{x}) + \theta \tag{3.11}$$

pour une constante $\theta$, telle que $\{\mathbf{x} : g(\mathbf{x}) < 0\} \neq \emptyset$ (c'est-à-dire qu'il ne faut pas que la marge choisie soit tellement grande que la région sûre devient vide).

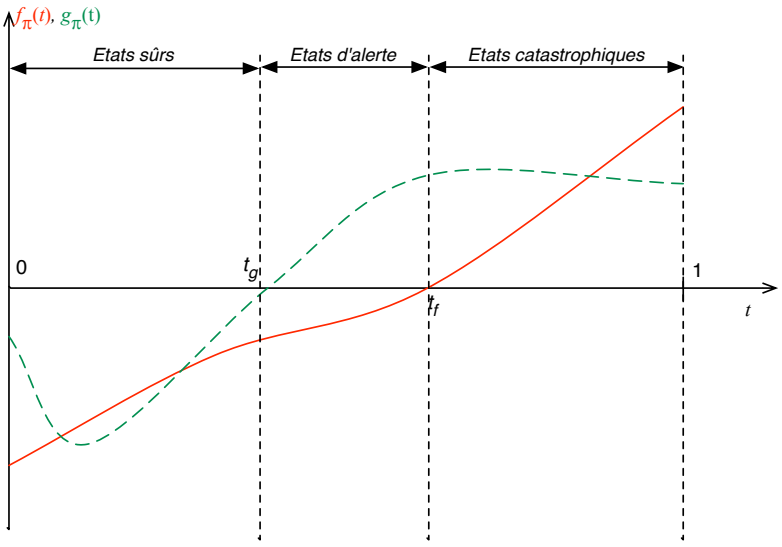


FIGURE 3.10 – Cas où $g_\pi(t)$ est non monotone

## 3.4 Actions de sécurité et dimensionnement des marges

Nous avons défini dans la Section 3.2 une action de sécurité comme étant : *une action explicite exercée pour mettre le système dans un état sûr*. Les actions de sécurité choisies en réponse à une condition de déclenchement de sécurité dépendent fortement de moyens d'actions disponibles au sein du système fonctionnel existant ainsi qu'à la réponse de ce système à une telle action de sécurité. La dynamique du système ainsi que les dispositifs de recouvrement disponibles ont un impact sur le choix d'une action de sécurité. Ce choix doit être effectué par les experts du domaine.

De plus, les actions de sécurité n'agissent pas forcément directement sur les valeurs des variables qui ont conduit au passage vers l'état d'alerte car :

- la variable correspondante peut ne pas être contrôlable,
- l'action directe sur la variable correspondante peut être déconseillée, voire interdite pour des raisons opérationnelles ou économiques (par exemple, l'action conduisant à un arrêt prolongé voire définitif du système).

Dans la section suivante, nous supposons que les actions de sécurité ont été définies par les experts du domaine. Nous les intégrons dans nos modèles afin de détecter la possibilité d'actions concomitantes et incohérentes.

## 3.5 Identification des actions de sécurité potentiellement concomitantes

Imaginons une voiture équipée d'un système régulateur de vitesse. La voiture est en mode déplacement lorsque le système de contrôle détecte une vitesse s'approchant

de la vitesse limite. Le système de contrôle de vitesse va donner l'ordre de freiner pour décélérer. Pendant ce temps, le système de détection de collision détecte que le véhicule de derrière s'approche dangereusement et ordonne d'accélérer ou du moins maintenir la vitesse actuelle afin d'éviter une collision. Dans ce cas, nous remarquons que les deux actions de sécurité sont complètement contradictoires. Il est donc essentiel de pouvoir détecter de telles situations.

Il est nécessaire au préalable d'identifier les états du système dans lesquels les actions de sécurité sont concomitantes. Pour cela, nous spécifions les actions de sécurité sur le graphe que nous avons obtenu, et nous proposons l'utilisation des machines de Mealy [2], qui sont un cas particulier des automates finis avec des *sorties* spécifiées sur les transitions. Ceci nous permet de spécifier, pour une transition particulière qui satisfait une condition de déclenchement de sécurité, l'action de sécurité correspondante (voir Figure 3.11).

Le nouveau graphe est défini par l'alphabet d'entrée $\mathbf{x}$, l'ensemble des régions du système $R_{\mathbf{x}}$, l'alphabet de sortie $O = \{a_1, a_2, ..., a_n\}$ correspondant à la liste des actions de sécurité disponibles, $\tau$ la fonction de transition et $\xi$ la fonction de sortie : $\xi : S_{\mathbf{x}} \times \mathbf{x} \rightarrow O$ qui associe à une paire (région du système, entrée), une action de sécurité.

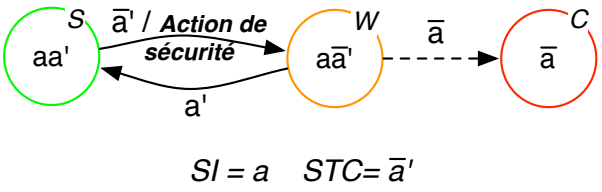


FIGURE 3.11 – Graphe de région du système représenté à l'aide d'une machine de Mealy

Rappelons que l'invariant de sécurité global est exprimé sous la forme d'une conjonction : $SI_1 \wedge \ldots \wedge SI_m$. Lorsque tous les termes $SI_i$ ont été traités et que les marges de sécurité et/ou les interverrouillages ont été spécifiés, les graphes correspondants peuvent être composés en faisant leur produit cartésien.

Considérons le cas de la conjonction $SI \wedge SI'$, où le graphe de régions correspondant à $SI$ est défini ci-dessus et le graphe de régions correspondant à $SI'$ est défini par l'alphabet d'entrée $\mathbf{x}'$, l'ensemble des régions du système $R_{\mathbf{x}'}$, l'alphabet de sortie $O' = \{a'_1, a'_2, ..., a_m\}$ correspondant à la liste des actions de sécurité disponibles, $\tau$ la fonction de transition et $\xi$ la fonction de sortie.

Le graphe de région correspondant à la composition des deux graphes précédents est défini par : $\sum = \mathbf{x} \cup \mathbf{x}'$ est l'alphabet d'entrée, $R_{\sum}$ est l'ensemble des régions du système, défini comme le produit cartésien de l'ensemble de régions $R_{\mathbf{x}}$ et $R_{\mathbf{x}'}$ en utilisant la fonction de composition $\mathcal{C}$ définie ci-dessous, et $O \cup O'$ est l'alphabet de sortie.

Nous pouvons définir les régions sûres, d'alerte et catastrophiques comme suit :

2. http ://fr.wikipedia.org/wiki/Machine_de_Mealy. Consulté le 18 juillet 2012

Région *sûre* $\mathcal{C}$ région *sûre* $\longrightarrow$ région *sûre*,

Région *sûre* $\mathcal{C}$ région d'*alerte* $\longrightarrow$ région d'*alerte*,

Région d'*alerte* $\mathcal{C}$ région d'*alerte* $\longrightarrow$ région d'*alerte*,

$\forall y \in$ {sûre, alerte, catastrophique}, région *catastrophique* $\mathcal{C}$ région $y \longrightarrow$ région *catastrophique*.

La figure 3.12 illustre la composition de deux graphes de région. Le premier, en haut de la figure représente le graphe de région du système sous l'invariant de sécurité : $SI = a$ et la condition de déclenchement de sécurité : $STC = \bar{a}'$. Le second, au milieu de la figure représente le graphe de région du système sous l'invariant de sécurité $SI' = b$, dans lequel aucune marge n'a pu être définie, ce qui a nécessité l'inhibition de la transition critique par un interverrouillage. Le graphe du bas de la figure illustre le résultat de la composition des deux graphes précédents.

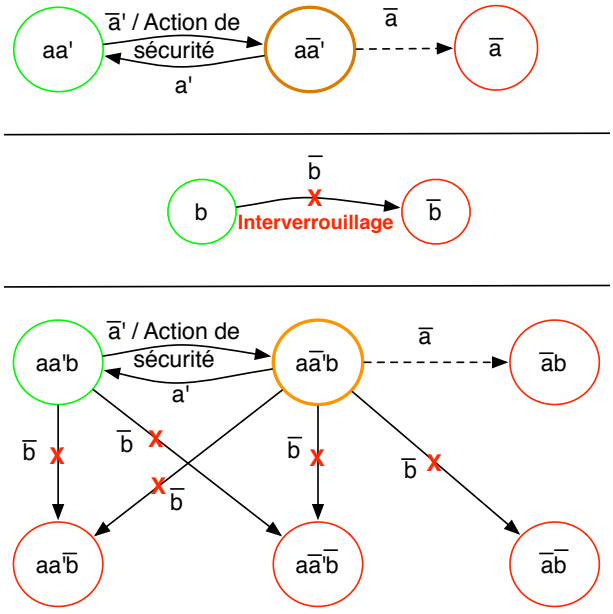


FIGURE 3.12 – Composition de deux graphes de régions, exemple n°1

La Figure 3.13 illustre la composition de deux autres graphes de région, chacun possédant une région d'alerte et par conséquent une condition de déclenchement de sécurité.

Le graphe du bas de la Figure 3.13 possède 3 régions d'alerte. Nous pouvons, grâce à ce graphe constater que, à partir de la région sûre (1), le système peut atteindre soit la région d'alerte (2) lorsque la condition de déclenchement de sécurité $STC' = \bar{b}'$ est vraie soit la région d'alerte (3) lorsque la condition de déclenchement de sécurité $STC = \bar{a}'$ est vraie. Le système peut atteindre la région d'alerte (4) à partir de la région sûre (1) soit en passant par les régions d'alerte (2) ou (3). Lorsque la transition $\bar{a}'$ de la région (2) à la région (4) est activée, cela enclenche l'action de sécurité $SA1$. Cependant, cette action de sécurité intervient alors qu'une autre action de sécurité $SA2$ est déjà enclenchée. Il faudra alors étudier la cohérence des deux actions successives $SA2$ et $SA1$.

Une solution possible serait de définir une priorité entre les actions de sécurité, ce qui permettrait, le cas échéant, d'activer l'action de sécurité ayant le plus haut niveau de priorité. De manière générale, c'est aux experts du domaine, ceux de la sécurité et les autorités compétentes de définir la stratégie de résolution du conflit en fonction des caractéristiques dynamiques du système.

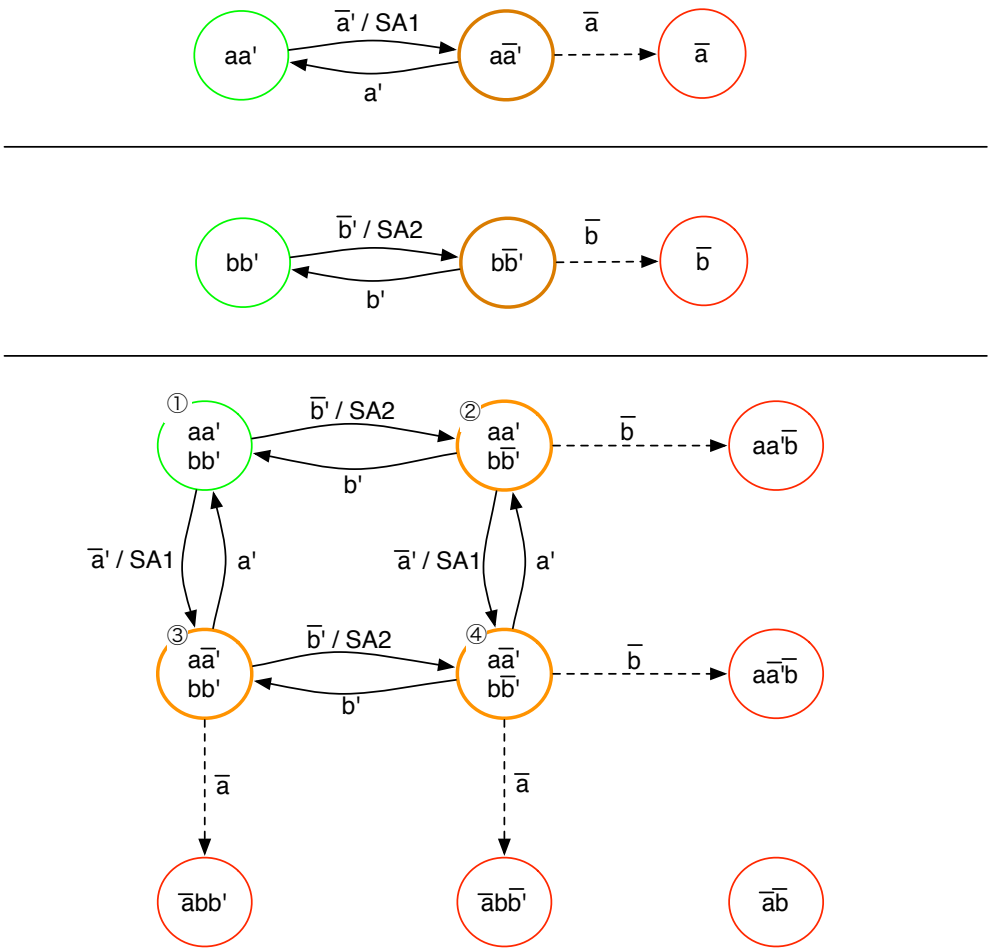


FIGURE 3.13 – Composition de deux graphes de région, exemple n°2

## 3.6 Récapitulatif de la méthode proposée

La Figure 3.14 décrit les différentes étapes du processus de définition des règles de sécurité. Rappelons que l'objectif est de définir des règles de sécurité qui seront exécutées en ligne par un moniteur de sécurité indépendant.

Le point de départ de notre processus est une analyse de risque HazOp/UML du système fonctionnel introduite au Chapitre 2, Section 2.3. Elle permet d'identifier les risques majeurs grâce à l'analyse de déviations des scénarios d'utilisation du système. Le résultat obtenu est un ensemble de tables de déviations à partir desquelles des

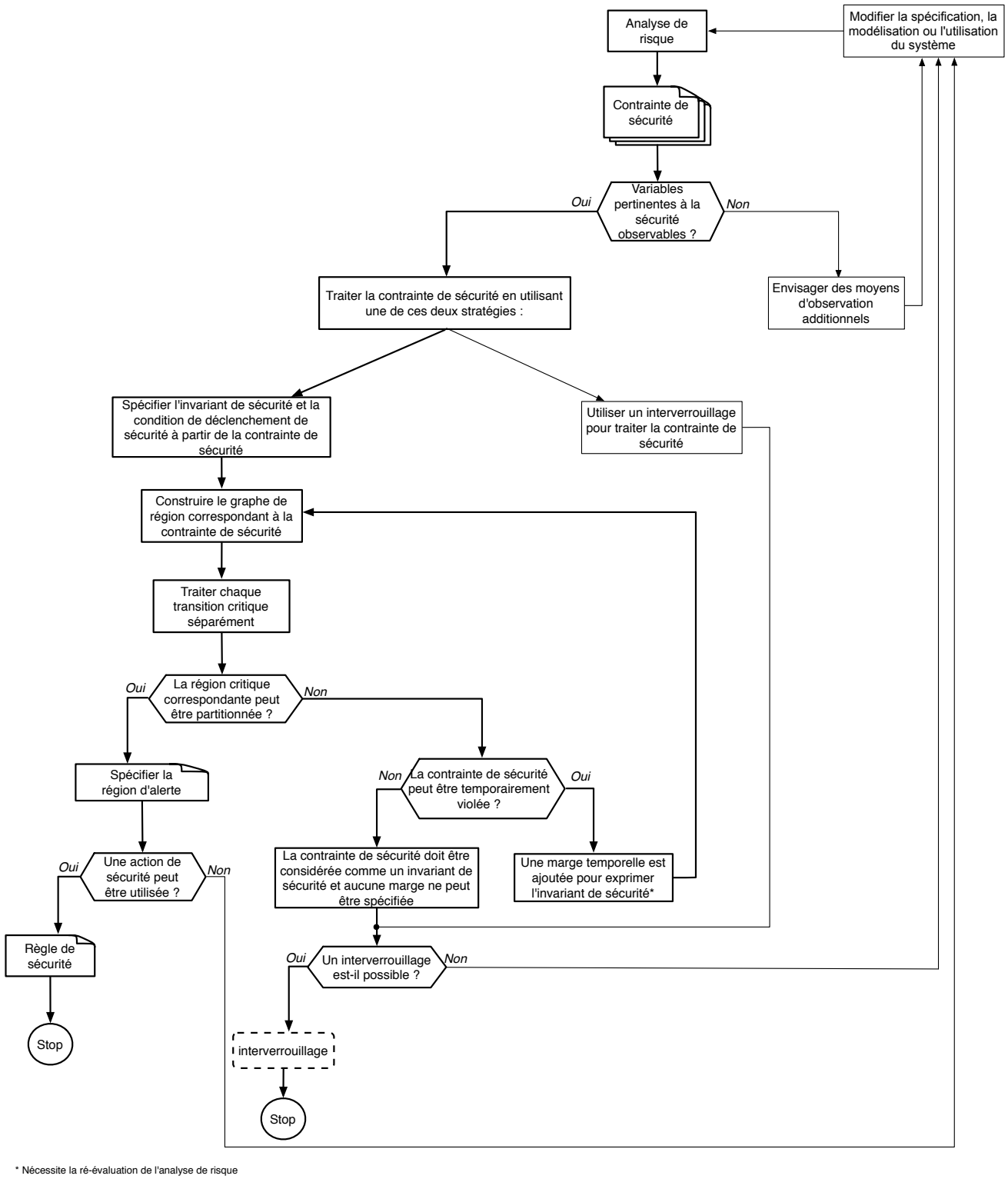


FIGURE 3.14 – Processus de définition des règles de sécurité

contraintes de sécurité sont identifiées. Les contraintes de sécurité sont ensuite traitées une à une.

L'étape suivante est d'étudier la nature des variables pertinentes à la sécurité utilisées pour exprimer une contrainte de sécurité. Si une des ces variables n'est pas observable par le moniteur de sécurité, il faudra envisager de nouveaux moyens d'observation. Dans le cas où ces variables sont observables, deux façons de traiter la contrainte de sécurité peuvent être envisagées :

- tenter de spécifier une marge de sécurité sur ces variables observables, ceci correspond à la partie gauche en gras de la Figure 3.14 et représente le processus développé lors de nos travaux,
- utiliser directement un interverrouillage pour traiter la contrainte de sécurité (voir discussion Section 3.2 et Section 3.3.2).

Afin de spécifier la marge de sécurité et de ce fait la condition de déclenchement de sécurité, nous tentons de partitionner les régions non-catastrophiques en deux sous régions : une région d'alerte qui consiste en la marge de sécurité, et une région sûre. Si pour une transition critique, aucune marge ne peut être spécifiée, nous pouvons tenter de spécifier une marge temporelle en considérant que la contrainte de sécurité puisse être temporairement violée. Dans le cas où la marge de sécurité peut être définie, l'étape finale est de spécifier l'action de sécurité qui devra être enclenchée afin d'amener le système dans un état sûr.

Si la contrainte de sécurité ne peut être temporairement violée, aucune règle de sécurité exécutable par un moniteur ne peut être définie à partir de cette contrainte. Elle devra donc être traitée en utilisant un interverrouillage. Si un interverrouillage ne peut pas être défini, il faudra alors modifier la spécification, la modélisation ou l'utilisation du système.

Une fois toutes les contraintes de sécurité traitées, l'étape finale du processus est d'identifier les états du système où plusieurs actions de sécurité sont enclenchées (voir Figure 3.15). Ceci permet, par la suite, à l'expert de définir les actions de sécurité à enclencher prioritairement afin d'éviter tout cas d'incohérence.

## 3.7 Conclusion

Dans ce chapitre, nous avons proposé un processus systématique pour la spécification de règles de sécurité à partir des déviations étudiées dans le cadre de l'analyse des risques du système et ce dans le but de les implémenter dans un moniteur de sécurité indépendant. Ce processus repose principalement sur la possibilité de détecter une situation potentiellement dangereuse grâce aux variables observables et de procéder à un recouvrement avant que le système n'atteigne un état catastrophique.

Le processus proposé est un processus *de bout en bout*, qui génère des contraintes de sécurité à partir d'une analyse des risques, puis traite ces contraintes une à une afin de spécifier des conditions de déclenchement de sécurité. L'étape finale du processus consiste en l'identification des actions de sécurité potentiellement concomitantes. L'ajout de marges de sécurité permet d'agir *avant* qu'il ne soit trop tard. Néanmoins,

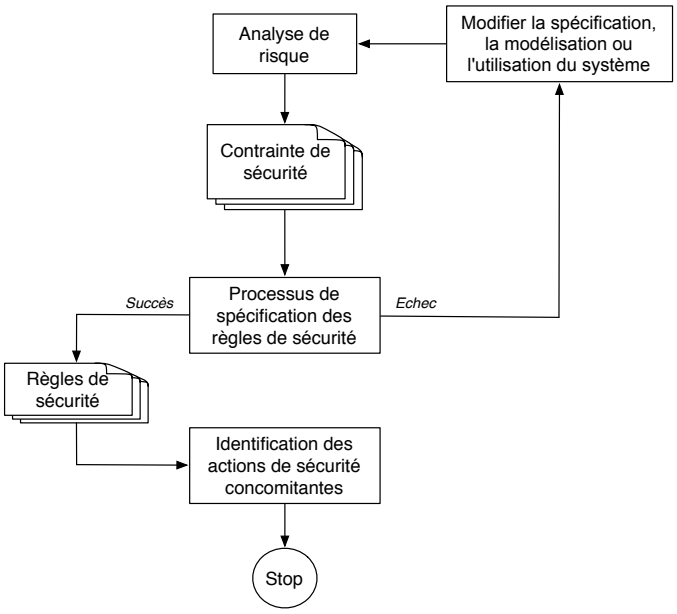


FIGURE 3.15 – Récapitulatif de la méthode proposée

cela affecte inévitablement la disponibilité du système ; impact qui devra faire l'objet d'études complémentaires.

Nous avons présenté dans ce chapitre des définitions claires et concises de concepts de base relatifs à la surveillance en ligne et nous les avons formalisés. Nous avons aussi présenté des méthodes mathématiques qui aident à vérifier l'existence d'une marge de sécurité. Nous avons proposé une méthode constructive pour définir une marge de sécurité lorsque les variables pertinentes à la sécurité prennent leurs valeurs dans l'ensemble des réels. Cependant, la marge définie de cette façon n'est pas toujours appropriée et devra être validée par l'expert selon le cas étudié. Néanmoins, la formulation mathématique présentée dans ce chapitre pour identifier l'existence d'une marge est plus générale et couvre aussi bien les variables booléennes, discrètes, continues, etc. Les méthodes mathématiques présentées constituent un support pour l'expert en charge de la spécification des conditions de déclenchement de sécurité.

La méthodologie proposée permet de modéliser les invariants de sécurité disjonctifs sous forme de graphe de régions afin de localiser les transitions critiques. Par la suite, chaque transition critique est traitée séparément, ce qui réduit fortement la complexité du processus. La génération de tels graphes peut être aisément automatisée.

Une stratégie différente pour l'expression de l'invariant de sécurité globale aurait pu être envisagée. L'invariant de sécurité pourrait être factorisé en $SI'_i$ tel que les tuples des variables dans chaque $SI'_i$ soient disjoints, puis traiter les $SI'_i$ séparément (éventuellement en les décomposant en atomes). Cette approche serait plus optimale (une seule condition de déclenchement pour la vitesse par exemple) mais passerait moins bien à l'échelle.

Afin de modéliser les actions de sécurité sur les graphes, nous avons retenu les machines de Mealy qui permettent d'associer une action à une transition dans le graphe

(la transition représente la condition de déclenchement de sécurité). Cependant, un autre choix de modélisation pourrait être fait afin d'avoir une meilleure visibilité sur les actions de sécurité, notamment leurs durées. La composition des graphes de régions obtenus permet de spécifier le comportement du système sous l'invariant de sécurité global. De plus, la composition permet de détecter les actions de sécurité qui pourraient être exécutées simultanément afin de vérifier leur cohérence. La composition des graphes pourrait poser des problèmes d'explosion du nombre de régions du graphe final. Néanmoins, selon notre expérience, le nombre de régions reste gérable vu la relative simplicité des graphes primaires dans des cas concrets. S'ajoute à cela la possibilité d'automatiser la composition des graphes.

Dans le chapitre suivant, nous allons appliquer notre processus à un déambulateur robotisé.

## Ce qu'il faut retenir

1. Dans le but d'assurer la sécurité-innocuité de systèmes autonomes opérant dans des environnements variables, complexes et non prédictibles, nous proposons un processus systématique de spécification de conditions de déclenchement de sécurité nécessaires pour que le moniteur de sécurité effectue une surveillance en ligne du système.
2. S'inspirant de la littérature du domaine ainsi que de notre propre vision de la problématique, nous proposons un ensemble de définitions des concepts de base pour la surveillance en ligne.
3. Nous avons choisi d'exprimer l'invariant de sécurité global sous forme conjonctive afin de pouvoir traiter chaque élément de la conjonction séparément, ce qui réduit de façon conséquente la complexité du processus.
4. Les transitions critiques vers des régions catastrophiques du système sont mises en évidence et traitées une à une avec, soit la possibilité de définir une région d'alerte, soit la redéfinition de l'invariant de sécurité afin de spécifier une marge de sécurité temporelle, soit le cas échéant la nécessité d'implémenter un interverrouillage.
5. Nous avons étudié le cas particulier où la marge de sécurité à définir concerne des variables continues. Nous avons proposé une façon de définir de telles marges de sécurité de manière constructive.
6. Les états du système où des actions de sécurité sont enclenchées simultanément sont détectées grâce à la composition des graphes de régions afin d'être revues et corrigées, en cas de besoin, par l'expert du système.

# Chapitre 4

# Validation de l'approche

## Introduction

Dans le chapitre précédent nous avons proposé une approche qui permet de spécifier des conditions de déclenchement de sécurité à partir d'un ensemble de contraintes de sécurité. Afin d'évaluer la méthodologie proposée, nous présentons dans ce chapitre son application à un cas concret qui est un robot déambulateur.

Nous présenterons tout d'abord le projet MIRAS ainsi que son analyse de risque, puis nous appliquerons la démarche proposée aux résultats de cette analyse de risque.

## 4.1 Le cas d'étude MIRAS

Dans cette section, nous allons présenter le projet en question ainsi que les modèles UML utilisés.

### 4.1.1 Présentation

MIRAS pour *Multimodal Interactive Robot for Assistance in Strolling* est un projet national financé par l'ANR (Agence Nationale pour la Recherche) soutenu par le programme TecSan (Technologies pour la Santé). Le projet a regroupé, en plus du LAAS (au sein duquel s'effectue cette thèse), les partenaires suivants :

- Robosoft [1] : qui propose des solutions robotiques dans le domaine du transport, de la santé, de la sécurité etc.,
- Médialis [2] : centre d'expertise médico-social qui édite des logiciels dédiés à la prise en charge des personnes âgées et des handicapés,
- ISIR (UPMC-CNRS) [3] : l'Institut des Systèmes Intelligents et de Robotique

1. http ://www.robosoft.com
2. http ://www.medialis.info
3. http ://www.isir.upmc.fr

(ISIR) est un laboratoire de recherche pluridisciplinaire arborant les disciplines des sciences de l'ingénieur et de l'information ainsi que des sciences du vivant pour des applications à la robotique médicale, la bio-science etc.,

- Trois hôpitaux qui sont : Henri Mondor (Unité Biomécanique - mouvement et système nerveux), La Grave (Laboratoire Gérontechnologie - Pôle Gériatrie) et Charles Foix.

L'objectif de ce projet est la conception et la mise au point d'un robot semi-autonome pour l'aide à la verticalisation et à la déambulation en milieu hospitalier des personnes âgées atteintes de troubles de l'équilibre et de l'orientation. Les déambulateurs existants, comme celui illustré dans la Figure 4.1 (a), ne permettent pas à ces patients d'être autonomes. Les bénéfices d'un système tel que MIRAS (voir Figure 4.1 (d)) sont :

- rendre l'autonomie de la marche aux patients concernés,
- adapter et contrôler l'activité de marche pour l'entrainement et l'amélioration de la condition physique du patient,
- libérer les personnels hospitaliers pour des actes plus techniques.

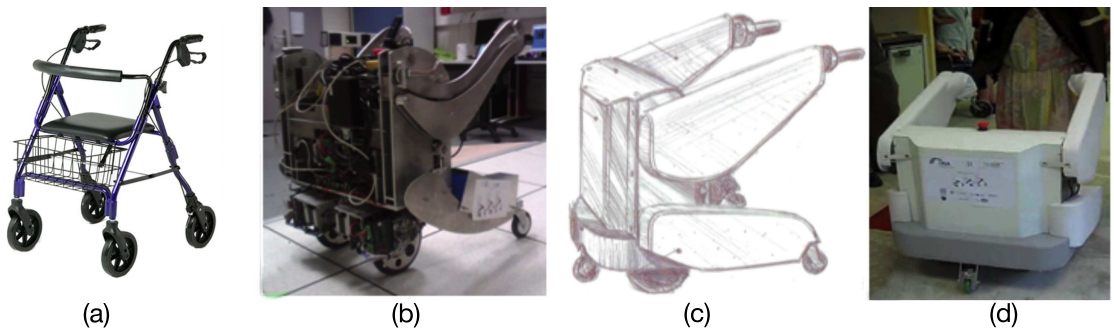


FIGURE 4.1 – (a) Déambulateur classique, (b) Robot MIRAS expérimental, (c) Modélisation avec la coque, (d) Prototype utilisé lors des essais cliniques

Le déambulateur possède d'autres fonctionnalités comme la surveillance de l'état physiologique du patient ainsi que sa posture. Il peut aussi se déplacer jusqu'au patient lorsque ce dernier l'appelle (fonction "hello").

### 4.1.2 Structure du robot

Le robot MIRAS (voir Figure 4.2) se présente comme un déambulateur articulé et actionné, doté de divers capteurs qui assurent la stabilisation posturale dans les phases de verticalisation et de déambulation des patients. La commande est permise par l'analyse conjointe de différents types de capteurs pour permettre à la personne âgée d'utiliser le robot de façon intuitive. Le robot doit exécuter les actions voulues par l'utilisateur sans que celui-ci ait à utiliser un protocole de communication. Un exemple typique est l'accompagnement aux toilettes : la personne ne pouvant ni se

lever seule de son fauteuil ni marcher seule quelques mètres, appelle un soignant qui ne fait que la soutenir. Le robot remplace ici avantageusement ce dernier. Pour cela, il doit exécuter un certain nombre d'actions qui dépendent de l'utilisateur (il ne "sait" pas a priori que la personne veut aller aux toilettes).

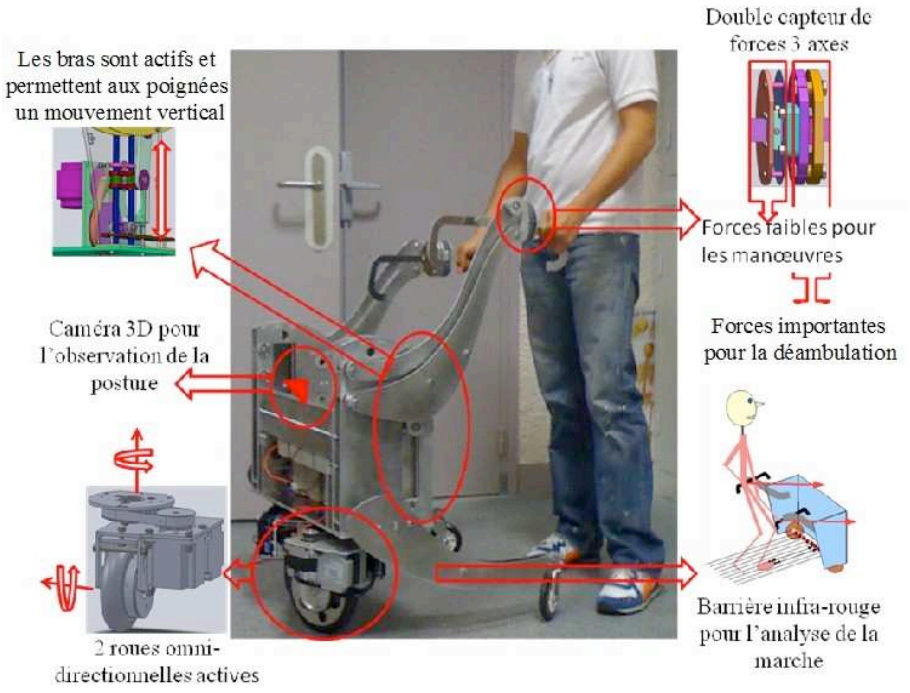


FIGURE 4.2 – Prototype du robot MIRAS

Le déambulateur est composé des éléments suivants :

- capteurs physiologiques : électrocardiographie ECG permettant de détecter la fatigue sur les variations du rythme cardiaque, en corrélation avec la surveillance du nombre de pas,
- capteurs d'effort au niveau des poignées,
- deux caméras : une caméra dynamique infrarouge pour détecter la posture du patient et une caméra environnement pour détecter un appel ("hello"),
- deux lasers : un tourné vers l'arrière pour surveiller la marche (information supplémentaire pour la fatigue) et un vers l'avant pour la navigation,
- ultra son pour la sécurité : balayage du sol sur 40 cm pour la détection d'obstacles,
- vérins électriques pour la verticalisation,
- moteurs sur les roues,
- calculateurs.

### 4.1.3 Environnement d'utilisation

L'environnement d'utilisation du robot en milieu hospitalier comprend :

- la chambre (simple ou double)
- les sanitaires de la chambre ou du service

- le couloir pour aller à la salle à manger
- la salle à manger
- la salle de bain
- l'accueil du service

Pour chacun de ces éléments de l'environnement, une description des espaces et de l'encombrement (autres appareils, ameublement), a été donnée. La Figure 4.3 représente un plan des chambres. Il permet d'indiquer les distances que les patients doivent être en mesure de parcourir pour aller de la chambre à la salle à manger par exemple.

### 4.1.4 Description des utilisateurs

D'un point de vue global, les futurs utilisateurs du déambulateur sont capables de se tenir brièvement debout en statique. En revanche, ils ne peuvent se lever ou s'asseoir seuls. Ils sont également capables de lâcher momentanément une poignée du déambulateur (pour saluer quelqu'un ou attraper un objet). Ils peuvent en moyenne déambuler une vingtaine de minutes avec des déambulateurs classiques. Cinq profils de patients ont été identifiés :

1. les patients âgés en rééducation après chirurgie du membre inférieur, notamment prothèse de hanche ou de genou,
2. les patients âgés cardiaques reprenant une activité de marche après une complication aigüe (poussée d'insuffisance cardiaque, les accidents cardio-vasculaires etc.),
3. les patients déments, notamment atteints d'une maladie d'Alzheimer dont l'évolution naturelle de la maladie s'accompagne de la perte de la marche,
4. les patients âgés parkinsoniens, non améliorés par les traitements médicaux ou chirurgicaux, qui ont des troubles marqués de l'équilibre postural responsable d'une limitation de leur autonomie de marche et de chutes,
5. les patients ayant subi un accident vasculaire cérébral de moyenne importance, qui n'entraîne pas d'hémiplégie complète mais une diminution de la force musculaire dans un territoire, une fatigabilité et des troubles sensitifs ou de l'équilibre.

### 4.1.5 Scénario nominal de l'utilisation

Le scénario nominal d'utilisation que l'on va présenter est le déplacement du patient de sa chambre vers les toilettes. La Figure 4.4 récapitule les actions effectuées par le patient et le robot. Le patient émet le souhait d'utiliser le déambulateur en l'appelant par un geste naturel, un "hello". Le robot s'éveille, détecte la position qu'il doit atteindre (position du patient) puis se déplace jusqu'à atteindre cette position. Une fois le robot à portée de main, le patient positionne le robot face à lui en tirant sur une poignée. Le robot suit alors le mouvement et se place en face du patient. Le patient appuie sur les poignées du robot, celui-ci détecte alors la volonté de verticalision et déploie les poignées. Une fois la verticalisation terminée, le patient marche jusqu'aux toilettes. Le robot est en mode déambulation, il exécute la marche dans la direction donnée par le patient sans que celui-ci ait la sensation de pousser le robot. Arrivé aux

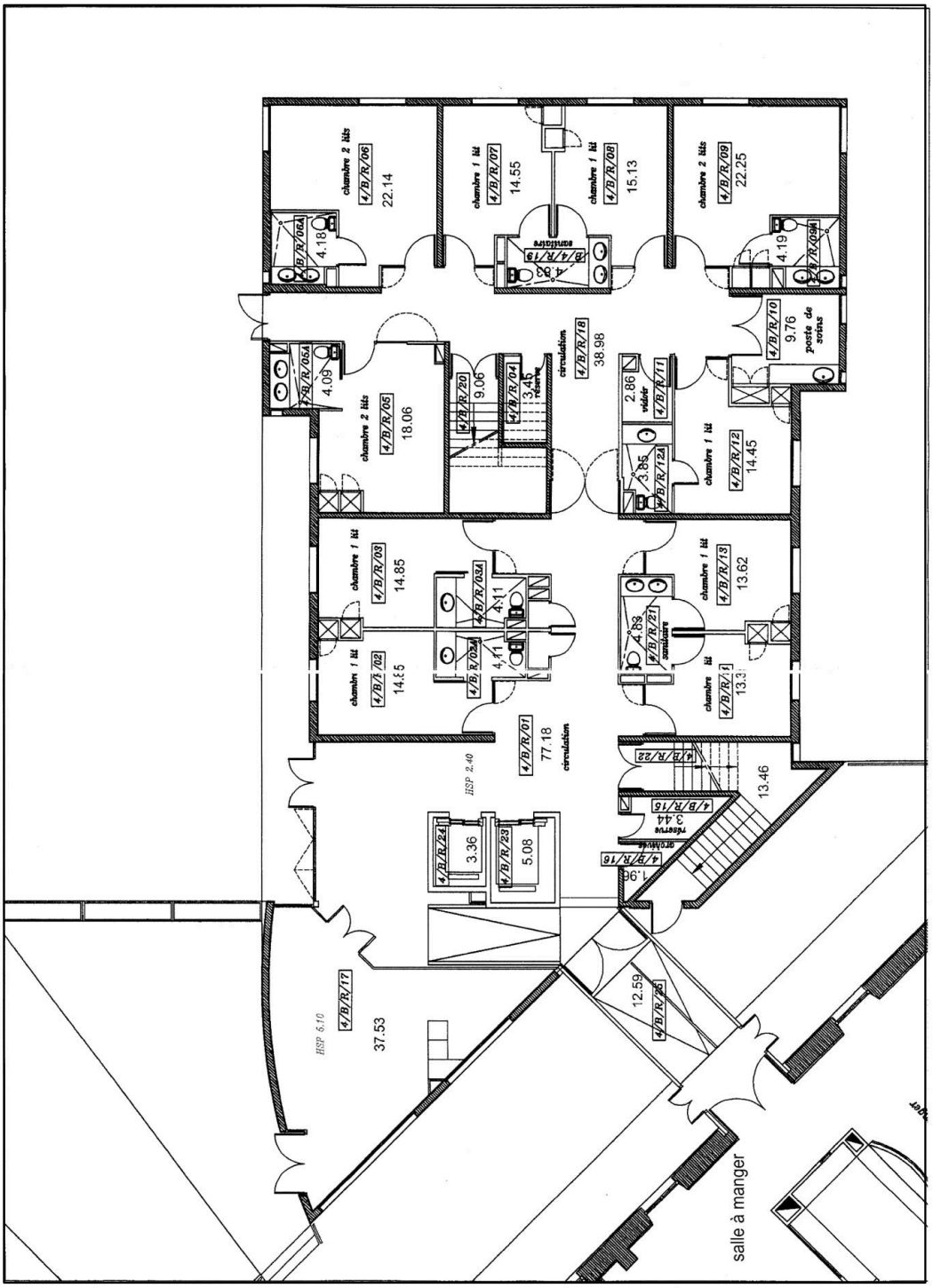


FIGURE 4.3 – Plan d'un environnement type d'utilisation du robot MIRAS

toilettes, la patient s'assoit sur les toilettes aidé par le robot qui s'est mis en mode deverticalisation pour aider le patient à s'asseoir correctement. Le patient écarte le robot. Le patient replace le robot encore une fois en face de lui, se lève en s'aidant du robot et déambule jusqu'à sa chambre.

| Action patient | Action Robot |
|---|---|
| Appel du robot : geste naturel ("hello") + voix | Réveil et détection de la position à atteindre |
| | Déplacement du robot à portée de la main |
| Placer le robot face à soi en le tirant par la poignée | Le robot se déplace dans la position correcte pour la verticalisation |
| Se lever du fauteuil | Le robot détecte la volonté de verticalisation et déploie les poignées |
| Marcher jusqu'aux toilettes | Le robot est en mode déambulation, il exécute la marche dans la direction imprimée par l'utilisateur sans que celui-ci ait la sensation de pousser |
| S'asseoir sur les toilettes | Le robot manœuvre pour aider la personne à être dans la bonne position |
| Écarter le robot | Le robot suit le mouvement |
| Placer le robot face à soi | Le robot se déplace dans la position correcte pour la verticalisation |
| Se lever du fauteuil | Le robot détecte la volonté de verticalisation et déploie les poignées |

FIGURE 4.4 – Le patient désire se rendre aux toilettes

### 4.1.6 Cas d'utilisation et diagrammes de séquences

La Figure 4.1 illustre les 15 cas d'utilisation modélisant les fonctionnalités du robot MIRAS :

- *Gestion de la déambulation* (UC01) : lorsque le patient désire marcher en s'aidant du déambulateur, il prend appui sur les poignées. Cet état va conduire le robot à déclencher le mode déambulation. Dans ce mode de fonctionnement les bras bougent dans un intervalle pré-défini. La base mobile du robot est mise en mouvement par quatre moteurs (2 par roue). Le robot avance dans la direction choisie par l'utilisateur.

- *Gestion de la verticalisation* (UC02) : lorsqu'il veut se lever en s'aidant du déambulateur, il s'appuie sur les poignées et se lève, c'est-à-dire que sa posture change. La connaissance en temps réel de la posture du patient et de l'effort exercé sur les poignées permet de déterminer si l'utilisateur désire se lever. Lors de la verticalisation, les moteurs permettent de monter les poignées à un niveau prédéterminé (hauteur des poignées exigée pour la déambulation). Lors de la verticalisation, il est possible qu'il y ait des déséquilibres (avant/arrière) du patient. Le système va compenser ou aider le patient à revenir en position assise.

- *Gestion de la déverticalisation* (UC03) : lorsque le patient désire s'asseoir, il va s'aider du robot pour passer de la position debout à la position assise. Le patient est moins anxieux que lors de verticalisation, il va même jusqu'à « se laisser tomber » parfois ; le robot va alors seulement accompagner le mouvement du patient.

- *Gestion de la perte d'équilibre* (UC04) : lors de la déambulation ou pendant que le patient se lève ou s'assied, le système rattrape les pertes d'équilibre de ce dernier.

- *Appel et déplacement autonome du robot* (UC05) : le robot est en attente dans un espace réservé, il détecte un signal de l'utilisateur puis planifie une trajectoire en évitant les obstacles, et enfin l'exécute.

- *Détection de la fin d'utilisation et déplacement en position d'attente* (UC06) : à l'inverse du cas précédent, un signal de l'utilisateur est transmis au robot qui planifie une trajectoire jusqu'à un endroit pré-défini (docking pour recharge batterie) puis la réalise.

- *Positionner le robot à la main* (UC07) : le patient peut déplacer le robot en position et en hauteur pour le positionner dans un état acceptable pour se lever ou pour l'éloigner.

- *Gestion des alarmes* (UC08) : le robot permet de détecter les situations suivantes : fatigue/problème physiologique, demande d'aide (vocale), perte de l'équilibre ou chute. En réponse le robot alerte le personnel médical (et peut lui transmettre sa position).

- *Programmation du profil patient* (UC09) : avant l'utilisation du robot, le personnel médical doit programmer le profil du patient : taux de handicap, taille, poids, type de pathologie, etc..

- *Apprentissage du profil patient* (UC10) : avant l'utilisation, un protocole d'apprentissage permet d'identifier le temps de verticalisation, les niveaux haut et bas des poignées, etc., du patient en fonction de sa pathologie.

- *Installation/Lancement* (UC11) : la mise en place, l'alimentation en énergie (préparation du lieu pour le "docking", la maintenance, etc..

- *Se lever à partir du siège robot* (UC12) : le robot assiste le patient quand il se lève à partir du siège intégré au robot.

- *S'asseoir sur le siège robot* (UC13) : le robot assiste le patient quand il s'assied sur le siège intégré au robot.

- *Pousser un objet avec le robot pendant la déambulation* (UC14) : le robot fournit un effort pour pousser un obstacle déplaçable qui se trouve devant (par exemple une porte).

- *Se déplacer en mode fauteuil roulant* (UC15) : le patient se place sur le siège intégré du robot et utilise les poignées pour commander le déplacement du robot.

Chacun de ces cas d'utilisation a été décrit par un ou plusieurs diagrammes de séquence. Nous allons nous intéresser plus particulièrement aux deux premiers cas

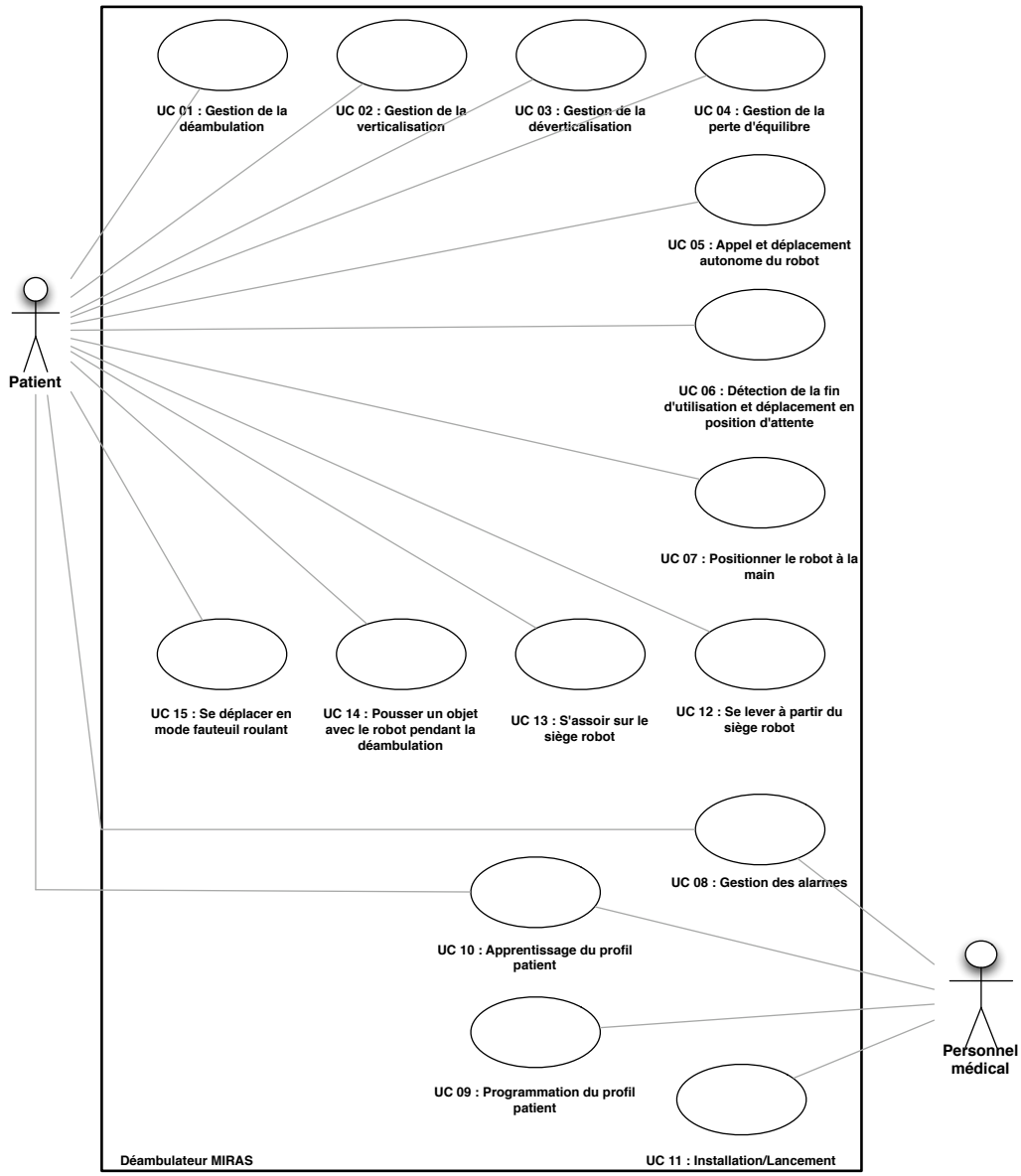


FIGURE 4.5 – Cas d'utilisation du déambulateur MIRAS

d'utilisation, pour lesquels les attributs sont résumés sur les Tables 4.1 et 4.2.

## 4.2 HazOp/UML appliquée à MIRAS

L'analyse de risque, présentée au Chapitre 2, Section 2.3 a été appliquée aux modèles de MIRAS. L'analyse a généré 395 déviations qui ont mené à l'identification de 15 dangers et 84 recommandations.

L'analyse de risque appliquée au cas d'utilisation *Gestion de la déambulation* a généré 19 déviations dont 16 d'une gravité différente de nulle. L'analyse de risques appliquée au cas d'utilisation *Gestion de la verticalisation* a généré 29 déviations dont

| Nom du cas d'utilisation | UC01 : Gestion de la déambulation |
|---|---|
| **Résumé** | La patient déambule à l'aide du robot |
| **Pré-conditions** | Les poignées sont à la bonne hauteur<br>Les batteries sont suffisamment chargées pour aller et revenir au point le plus loin et générer des alarmes |
| **Post-conditions** | Aucune |
| **Invariant** | Le patient est debout<br>Le patient tient le robot par au moins une poignée<br>Le robot est en mode déambulation<br>Les indicateurs physiologiques indiquent un état acceptable<br>La vitesse des roues ne doit pas excéder Vmax_bras m.s-1 |

TABLE 4.1 – UC01 : Gestion de la déambulation

| Nom du cas d'utilisation | UC02 : Gestion de la verticalisation |
|---|---|
| **Résumé** | Le patient se met debout avec l'aide du robot |
| **Pré-conditions** | Le patient est assis<br>Le robot est en attente de verticalisation<br>Les batteries sont suffisamment chargées pour effectuer cette tâche + la déverticalisation<br>Le robot est en face du patient |
| **Post-conditions** | Le patient est debout<br>Le robot est en mode déambulation |
| **Invariant** | Le patient tient le robot par les deux poignées<br>Le robot est en mode verticalisation<br>Les indicateurs physiologiques indiquent un état acceptable<br>La vitesse des bras ne doit pas excéder Vmax m.s-1 |

TABLE 4.2 – UC02 : Gestion de la verticalisation

16 d'une gravité différente de nulle. Les Tables 4.3 et 4.4 récapitulent ces 32 déviations ainsi que leurs effets sur le cas d'utilisation et dans le monde réel pour chaque cas d'utilisation.

## 4.3 Elicitation des règles de sécurité

Pour les raisons citées au Chapitre 2, section 2.4.1, nous allons nous intéresser plus particulièrement à l'élicitation de règles de sécurité à partir de la colonne *Deviation* de l'analyse de risque de MIRAS. Tout d'abord, nous allons générer des contraintes de sécurité à partir de cette colonne. Ensuite, nous tenterons de spécifier les conditions de déclenchement de sécurité correspondantes.

TABLE 4.3 – Les déviations générées par l'analyse de risque du cas d'utilisation *Gestion de la déambulation*

| N° | Déviation | Effet sur le cas d'utilisation | Effet sur le terrain | Gravité |
|---|---|---|---|---|
| 1 | Les poignées ne sont pas à la bonne hauteur pour la déambulation. | Le patient déambule dans une mauvaise position. | Chute du patient<br>Problèmes de santé dus à la posture | Sérieuse |
| 2 | Les batteries ne sont pas suffisamment chargées pour aller et revenir au point le plus loin et générer des alarmes | Le robot s'arrête pendant l'utilisation | Désorientation du patient<br>Fatigue du patient<br>Impossibilité de prévenir le personnel médical | Sérieuse |
| 3 | Le patient n'est pas en position pour déambuler | Le robot va se mouvoir sans le patient ou avec le patient dans une mauvaise position | Collision, déséquilibre ou chute du patient s'il utilise le robot | Sérieuse |
| 4 | Le robot n'est pas tenu par le patient | Le robot va se mouvoir sans le patient | Déséquilibre du patient s'il a relâché les poignées temporairement ou chute sans alarme | Sérieuse/Sévère |
| 5 | Le patient tient le robot mais le robot ne le détecte pas | Le robot ne va pas autoriser la déambulation | Erreur dans les séquences d'exécution / Patient désorienté/énervé/fatigué | Mineure |
| 6 | Le robot n'est pas en mode déambulation | Le patient attend de pouvoir déambuler mais le robot n'est pas disposé à le faire | Fatigue du patient | Mineure |
| 7 | Le robot passe dans un autre mode (verticalisation, etc…) de manière non désirée | Mouvement non attendu du robot | Chute/perte d'équilibre du patient | Sérieuse |
| 8 | L'état physiologique du patient n'est pas acceptable pour la déambulation | Le robot continue de déambuler malgré les difficultés du patient | Chute/Problème physiologique | Sérieuse |
| 9 | Certains indicateurs physiologiques sont incorrects | Difficulté d'interpréter le niveau de santé du patient | Problème physiologique sans détection | Sérieuse |
| 10 | Les indicateurs physiologiques sont interprétés trop tard | Le patient a déjà chuté ou eu un malaise et le robot ne déclenche l'alarme que trop tard | Malaise ou chute du patient sans intervention médicale | Sévère |
| 11 | Le robot entre en collision avec un objet | Collision avec le patient utilisant le robot | Chute/perte d'équilibre du patient | Sérieuse |
| 12 | Une collision est détectée sans qu'elle n'ait eu lieu | Le robot s'arrête sans raison | Fatigue du patient | Sérieuse |
| 13 | La collision est détectée avant qu'elle ait eu lieu | Collision | Chute/perte d'équilibre du patient | Sérieuse |
| 14 | Le robot sort des zones autorisées pour la déambulation | Le robot avec le patient se retrouve en milieu dangereux | Risques divers : collision avec l'environnement, blocage dans l'ascenseur, chute du patient | Sévère |
| 15 | Le robot pense qu'il est sorti d'une zone autorisée sans que cela ne soit le cas | Le robot s'arrête sans raison | Fatigue du patient | Mineure |
| 16 | La vitesse des roues dépasse Vmax m/s | Le robot tire le patient/Le robot déambule sans le patient | Deséquilibre/Chute du patient | Sévère |

| N° | Déviation | Effet sur le cas d'utilisation | Effet sur le terrain | Gravité |
|---|---|---|---|---|
| 1 | Le patient n'est pas assis | Le robot démarre la verticalisation sans le patient. Incohérence dans la séquence d'exécution. | Le patient attrape les poignées pendant qu'elles montent. Chute du patient | Sérieuse |
| 2 | Le patient est assis mais dans une position incorrecte pour la verticalisation | Le robot exécute la verticalisation | Déséquilibre/Chute du patient | Sérieuse |
| 3 | Les batteries ne sont pas suffisamment chargées | Le robot s'arrête pendant la verticalisation | Déséquilibre/Chute du patient | Sérieuse |
| 4 | Les batteries sont suffisamment chargées pour la verticalisation mais pas pour la déverticalisation | Le robot exécute la verticalisation et s'arrête pendant la suite des opérations | Fatigue du patient lors de l'arrêt. Impossible de signaler le problème. | Sérieuse/ Sévère |
| 5 | Le robot n'est pas en face du patient | Le robot démarre la verticalisation sans que ce dernier ne soit en position. | Déséquilibre/Chute du patient | Modéré |
| 6 | Le robot n'est pas en mode déambulation à la fin de la verticalisation | Le robot est dans un mode de fonctionnement incorrect | Déséquilibre/Chute du patient | Sérieuse |
| 7 | Le robot passe en mode déambulation trop tôt | Mauvaise synchronisation homme/robot | Déséquilibre/Chute du patient | Sérieuse |
| 8 | Le robot passe en mode déambulation trop tard | Mauvaise synchronisation homme/robot | Déséquilibre/Chute du patient | Sérieuse |
| 9 | Le patient ne tient pas le robot | Le robot continue la verticalisation malgré l'interruption | Déséquilibre/Chute du patient | Sérieuse |
| 10 | Le patient tient le robot mais le robot ne le détecte pas | Le robot s'arrête pendant la verticalisation | Déséquilibre/Chute du patient | Sérieuse |
| 11 | Le patient tient le robot par une seule poignée | Le robot continue la verticalisation mais le patient n'est plus en position | Déséquilibre/Chute du patient | Sérieuse |
| 12 | Le robot n'est pas en mode verticalisation | Changement de mode brutal du robot | Chute du patient due à un mauvais mode | Sérieuse |
| 13 | L'état physiologique du patient n'est pas acceptable pour la déambulation | Le robot continue la verticalisation malgré les difficultés du patient | Chute/Problème physiologique | Sérieuse |
| 14 | Certains indicateurs physiologiques sont incorrects | Le robot verticalise alors que l'état physiologique du patient n'est pas acceptable | Chute/Problème physiologique | Sérieuse |
| 15 | Les mauvais indicateurs physiologiques sont interprétés trop tard | Le patient a déjà chuté ou eu un malaise et le robot ne déclenche l'alarme que trop tard | Malaise ou chute du patient sans intervention médicale | Sévère |
| 16 | La vitesse des bras dépasse Vmax_bras m/s | Le robot tire le patient vers le haut / Le patient lâche les poignées | Déséquilibre/Chute du patient | Sévère |

TABLE 4.4 – Les déviations générées par l'analyse de risque du cas d'utilisation *Gestion de la verticalisation*

| | Déviation | Contrainte de sécurité |
|---|---|---|
| **1** | Les poignées ne sont pas à la bonne hauteur pour la déambulation. | Les poignées doivent être à la bonne hauteur pour la déambulation. |
| **2** | Les batteries ne sont pas suffisamment chargées pour aller et revenir au point le plus loin et générer des alarmes | Les batteries doivent être suffisamment chargées pour aller et revenir au point le plus loin et générer des alarmes |
| **3** | Le patient n'est pas en position pour déambuler | Le patient doit être en bonne position pour pouvoir déambuler |
| **4** | Le robot n'est pas tenu par le patient | Le robot doit être tenu par le patient |
| **5** | Le patient tient le robot mais le robot ne le détecte pas | Le robot doit détecter si le patient le tient |
| **6** | Le robot n'est pas en mode déambulation | Le robot doit être en mode déambulation |
| **7** | Le robot passe dans un autre mode (verticalisation, etc....) de manière non désirée | Le robot ne doit pas passer dans un autre mode sans raison valable |
| **8** | L'état physiologique du patient n'est pas acceptable pour la déambulation | L'état physiologique du patient doit être acceptable lors de la déambulation |
| **9** | Certains indicateurs physiologiques sont incorrects | Les indicateurs physiologiques ne doivent pas être incorrects |
| **10** | Les indicateurs physiologiques sont interprétés trop tard | Les indicateurs physiologiques ne doivent pas être interprétés trop tard |
| **11** | Le robot entre en collision avec un objet | Le robot ne doit pas entrer en collision avec un objet |
| **12** | Une collision est détectée sans qu'elle n'ait eu lieu | Le robot ne doit pas détecter une collision qui n'en est pas une |
| **13** | La collision est détectée après qu'elle ait eu lieu | Le robot doit détecter l'imminence une collision |
| **14** | Le robot sort des zones autorisées pour la déambulation | Le robot ne doit pas sortir des zones autorisées pour la déambulation |
| **15** | Le robot pense qu'il est sorti d'une zone autorisée sans que cela ne soit le cas | Le robot doit se localiser correctement |
| **16** | La vitesse de la base dépasse Vmax m/s | La vitesse de la base ne doit pas dépasser Vmax m/s |

TABLE 4.5 – Les contraintes de sécurité générées à partir de la colonne *Deviation* du cas d'utilisation *Gestion de la déambulation*

### 4.3.1 Génération des contraintes de sécurité

Comme présenté dans le Chapitre 2, Section 2.4, les contraintes de sécurité représentent la négation des déviations obtenues par l'analyse de risque du système. En parcourant la colonne *Deviation* des Tables 4.3 et 4.4 ligne par ligne et en effectuant la négation, nous obtenons une première liste de contraintes de sécurité. Les Tables 4.5 et 4.6 récapitulent ces contraintes, exprimées en langage naturel.

La seconde étape est d'exprimer les contraintes de sécurité sous forme de prédicats en utilisant des variables pertinentes à la sécurité observables par un moniteur de sécurité. La Table 4.7 récapitule l'ensemble des variables disponibles dans le système. Elles sont classées en quatre catégories : variables *observables* et *commandables* par le déambulateur et variables *observables* et *commandables* par le moniteur de sécurité. Ce dernier peut observer ces variables soit par des moyens d'observation propres à lui, soit par les moyens d'observation existants au sein du déambulateur (ce qui posera des problèmes de défaillances de mode commun). Les variables commandables par le moniteur représentent les valeurs sur lesquelles le moniteur peut agir pour remettre le système dans un état sûr (moyens de recouvrement).

La Table 4.8 illustre les contraintes de sécurité, du cas d'utilisation *déambulation*, qui ont pu être exprimées en langage formel.

Sur les 16 contraintes de sécurité exprimées en langage naturel :

| | Déviation | Contrainte de sécurité |
|---|---|---|
| **1** | Le patient n'est pas assis | Le patient doit être assis pour la verticalisation |
| **2** | Le patient est assis mais dans une position incorrecte pour la verticalisation | Le patient doit être assis correctement pour la verticalisation |
| **3** | Les batteries ne sont pas suffisamment chargées | Les batteries doivent être suffisamment chargées pour la verticalisation |
| **4** | Les batteries sont suffisamment chargées pour la verticalisation mais pas pour la déverticalisation | Les batteries doivent être suffisamment chargées pour la verticalisation et la déverticalisation |
| **5** | Le robot n'est pas en face du patient | Le robot doit être en face du patient pour la verticalisation |
| **6** | Le robot n'est pas en mode déambulation à la fin de la verticalisation | Le robot doit être en mode déambulation à la fin de la verticalisation |
| **7** | Le robot passe en mode déambulation trop tôt | Le robot ne doit pas passer en mode déambulation trop tôt |
| **8** | Le robot passe en mode déambulation trop tard | Le robot ne doit pas passer en mode déambulation trop tard |
| **9** | Le patient ne tient pas le robot | Le patient doit tenir le robot |
| **10** | Le patient tient le robot mais le robot ne le détecte pas | Le robot doit détecter si le patient le tient |
| **11** | Le patient tient le robot par une seule poignée | Le patient doit tenir le robot par les deux poignées lors de la verticalisation |
| **12** | Le robot n'est pas en mode verticalisation | Le robot doit être en mode verticalisation |
| **13** | L'état physiologique du patient n'est pas acceptable pour la déambulation | Le robot ne doit pas verticaliser si l'état physiologique du patient n'est pas acceptable |
| **14** | Certains indicateurs physiologiques sont incorrects | Les indicateurs physiologiques du patient doivent être corrects |
| **15** | Les mauvais indicateurs physiologiques sont interprétés trop tard | Les indicateurs physiologiques ne doivent pas être interprétés trop tard |
| **16** | La vitesse des bras dépasse Vmax_bras m/s | La vitesse des bras ne doit pas dépasser Vmax_bras m/s |

TABLE 4.6 – Les contraintes de sécurité générées à partir de la colonne *Deviation* du cas d'utilisation *Gestion de la verticalisation*

| Variable | Description | Observable par le déambulateur | Commandable par le déambulateur | Observable par le moniteur | Commandable par le moniteur |
|---|---|---|---|---|---|
| *Freq_pas* | Fréquence de marche (pas/min) | √ | | | |
| *Dist_pieds_robot* | Distances pieds patient/robot | √ | | | |
| *Lg_pas* | Longueur de pas | √ | | | |
| *Fr_poig* | Force sur les poignées | √ | | √ | |
| *Siege* | Etat du siège (occupé ou pas) | √ | | √ | |
| *Charge* | Autonomie de batterie restante | √ | | √ | |
| *Contact_poignees* | Contacts des mains avec les poignées {vrai, faux} | √ | | √ | |
| *Pulsation* | Rythme cardiaque (pulsations/mn) | √ | | √ | |
| *Temperature* | La température corporelle du patient (3 valeurs) | √ | | √ | |
| *Posture* | Posture du tronc du patient {debout, déséquilibre, chute} | √ | | √ | |
| *Dist_tronc_robot* | Distances tronc/robot (3 valeurs) | √ | | | |
| *Contact* | Contact avec obstacle (bumper) | √ | | | |
| *d* | Distance robot/objet le plus proche | √ | √ | √ | √ |
| *h* | Hauteurs des poignées (moyenne de 2 valeurs) | √ | √ | √ | |
| *v* | Vitesse de la base | √ | √ | √ | √ |
| *Acc_base* | Accélération de la base | √ | √ | √ | √ |
| *Orientation_roue* | Orientations des roues | √ | √ | | |
| *v_bras* | Vitesse des bras | √ | √ | | |
| *Niv_siege* | Niveau de montée du siège | √ | √ | | |
| *Alarme* | Alarme pour l'alerte | √ | √ | √ | √ |
| *Freins* | Décélération de la vitesse du robot | √ | √ | √ | √ |
| *Arret_urgence* | Arrêt du robot | | | √ | √ |

TABLE 4.7 – Liste des variables observables et commandables par le déambulateur et par le moniteur

1. Huit contraintes ont pu être exprimées sous forme de prédicats (1, 3, 4, 8, 10, 11, 13 et 16) :
   - La contrainte n° 1 stipule que : "Les poignées doivent être à la bonne hauteur pour la déambulation". Cette contrainte découle de la déviation qui stipule que : "les poignées ne sont pas à la bonne hauteur pour la déambulation". La bonne hauteur pour la déambulation est spécifiée par les experts du système comme étant un intervalle $I = ]hmin, hmax[$ au sein duquel les poignées peuvent varier lors de la déambulation. Cet intervalle est défini lors de l'apprentissage du profil du patient avant le lancement du déambulateur, il est donc spécifique à chaque patient. La hauteur des poignées $h$ et la vitesse de la base du robot $v$ sont, selon la Table 4.7, des variables observables par le moniteur de sécurité, nous pouvons donc spécifier la déviation sous forme de prédicat comme suit : $(v > 0) \wedge (h \geq hmax \vee h \leq hmin)$ qui équivaut à dire que le système n'est plus sûr lorsque le robot déambule $(v > 0)$ et que la hauteur des poignées est en dehors de l'intervalle $I$. La contrainte de sécurité est donc exprimée comme suit : $(v = 0) \vee (h < hmax \wedge h > hmin)$.

   - La contrainte n° 3 stipule que : "Le patient doit être en bonne position pour pouvoir déambuler". Cette contrainte découle de la déviation qui stipule que : "le patient n'est pas en bonne position pour pouvoir déambuler". La posture du patient est observable grâce à 3 capteurs de position qui permettent de calculer la valeur de la variable *posture* qui indique si le patient est *debout*, en *déséquilibre* ou s'il a *chuté*. La position *déséquilibre* n'est pas consi-

dérée comme un état catastrophique car le déambulateur possède une fonction de gestion de la perte d'équilibre (voir UC 04) qui permet de soutenir le patient en cas de déséquilibre et l'aide à retrouver une posture correcte. Nous pouvons donc spécifier la déviation sous forme de prédicat comme suit : $(v > 0) \wedge (posture = chute)$. La contrainte de sécurité est donc exprimée comme suit : $(v = 0) \vee (posture \neq chute)$ qui équivaut à dire qu'il ne faut pas que le robot déambule alors que le patient est tombé.

- La contrainte n° 4 de la Table stipule que : "Le robot doit être tenu par le patient avec les deux poignées". Cette contrainte découle de la déviation qui stipule que : "le robot n'est pas tenu par le patient avec les deux poignées". Le robot détecte que le patient le tient grâce à des capteurs installés au niveau de chaque poignée. La variable *contact_poignées* est une variable observable qui prend les valeurs {vrai,faux} pour indiquer si le patient tient le robot par les deux poignées ou pas. Nous pouvons donc spécifier la déviation sous forme de prédicat comme suit : $(v > 0) \wedge (contact\_poignees = faux)$. La contrainte de sécurité est donc exprimée comme suit : $(v = 0) \vee (contact\_poignees = vrai)$.

- Les contraintes n° 8 et 10 stipulent que : "L'état physiologique du patient doit être acceptable ou ne doit pas être interprété trop tard lors de la déambulation". Ces contraintes découlent des déviations qui stipulent que : "l'état physiologique du patient n'est pas acceptable ou est interprété trop tard lors de la déambulation". L'état physiologique du patient est surveillé grâce à deux mécanismes : trois capteurs de température et un capteur du rythme cardiaque. Selon le profil du patient, l'équipe médicale indique l'intervalle dans lequel la moyenne des trois températures prélevées du patient peut varier ([*Temp_min,Temp_max*]) ainsi que l'intervalle dans lequel le nombre de pulsations cardiques/minute peut varier ([*Puls_min,Puls_max*]). Les variables $Temperature$ et $Pulsation$ étant observables par le système, nous pouvons exprimer les deux déviations sous forme de prédicat comme suit : $(v > 0) \wedge ((Temperature < Temp\_min \vee Temperature > Temp\_max) \vee (Pulsation < Puls\_min \vee Pulsation > Puls\_max))$.
  La négation de cette déviation permet d'obtenir la contrainte de sécurité suivante : $(v = 0) \vee ((Temperature \geq Temp\_min \wedge Temperature \leq Temp\_max) \wedge (Pulsation \geq Puls\_min \wedge Pulsation \leq Puls\_max))$.

- Les contraintes n° 11 et 13 stipulent que : "Le robot ne doit pas entrer en collision avec un objet / Le robot doit détecter l'imminence d'une collision". Ces contraintes découlent des déviations qui stipulent que : "le robot entre en collision avec un objet / le robot ne détecte pas l'imminence d'une collision". Pour la détection d'obstacle, le déambulateur est équipé d'un télémètre laser pour calculer la distance entre le robot et l'objet le plus proche, appelée $d$. Lors de la déambulation, une collision se produit lorsque la distance de freinage du robot est supérieure à sa distance de l'objet le plus proche, c'est à dire : $distance\ de\ freinage \geq d$. La distance de freinage jusqu'à l'arrêt total du robot est calculée comme suit : $distance\ de\ freinage = \frac{v^2}{2\alpha}$, où $\alpha$ représente la décélération minimale du déambulateur (fournie par les concepteurs).

Après transformation, nous obtenons que la déviation est exprimée comme suit : $v \geq \sqrt{2\alpha d}$. Par conséquent, afin d'éviter une collision avec l'objet le plus proche, la contrainte de sécurité suivante doit être respectée : $v < \sqrt{2\alpha d}$.

- La contrainte n° 16 stipule que : "La vitesse des roues ne doit pas dépasser $Vmax$ m/s" où $Vmax$ est la vitesse maximale autorisée lors de la déambulation. De façon triviale, nous pouvons donc spécifier la contrainte de sécurité sous forme de prédicat comme suit : $v < Vmax$.

2. Huit contraintes de sécurité n'ont pas pu être exprimées sous forme de prédicats (2, 5, 6, 7, 9, 12, 15, 16) :
   - La contrainte n° 2 stipule que : "Les batteries doivent être suffisamment chargées pour aller et revenir au point le plus loin et générer des alarmes". Les experts ont défini que 15% de la charge maximale du robot est nécessaire pour aller et revenir au point le plus loin par rapport au lit du patient ainsi que pour générer l'alarme en cas d'incident. La charge maximale $charge\_max$ du robot est un paramètre connu du système. Le niveau de la charge du robot, appelée $charge$ est une variable observable par le moniteur. Néanmoins, même si nous pouvons spécifier la contrainte de sécurité sous forme de prédicat (comme suit : $(v = 0) \vee (charge > 0,15 * charge\_max)$), elle représente le cas où la contrainte ne peut être mise en œuvre que si le moniteur de sécurité peut observer les requêtes et agir dessus (voir Chapitre 2, Section 2.4.2). Si le moniteur peut effectuer une telle observation alors il ne permettra pas au déambulateur de commencer à déambuler si le charge des batteries n'est pas suffisante. Dans le cas où le moniteur de sécurité ne peut observer au niveau des requêtes, un interverrouillage externe serait nécessaire afin d'empêcher le robot de déambuler lorsque les batteries ne sont pas suffisamment chargées.

   - Trois contraintes de sécurité n° 5, 9 et 12 ne peuvent pas servir pour spécifier une règle de sécurité car ils expriment la nécessité d'une haute intégrité de l'observation. Une redondance des mécanismes d'observation pourrait être envisagée pour la détection du patient au niveau des poignées (contrainte n° 5), pour les capteurs de température et de pulsations cardiaques (contraintes n° 9) et pour la détection de collision (contrainte n° 12).

   - Les contraintes n° 14 et 15 stipulent que : "Le robot ne doit pas sortir des zones autorisées pour la déambulation" et que "Le robot doit pouvoir se localiser correctement". La fonction de localisation du déambulateur n'a pas été implémentée délibérément de la part des concepteurs. Le risque encouru (le déambulateur navigue dans une zone dangereuse) a été jugé tolérable car l'environnement de déambulation actuel du robot est limité et les zones à risques ont été isolées (escaliers par exemple). Dans le cas où le déambulateur serait déployé dans un autre environnement, le risque devra être réévalué et en conséquence, une implémentation de la géolocalisation pourrait s'avérer nécessaire afin de réduire le risque au seuil tolérable.

   - Les contraintes n° 6 et 7 indiquent, quant à elles, la nécessité d'assurer une

forte intégrité du calcul du mode actuel du démabulateur.

| | Contrainte de sécurité en langage naturel | Classification de la contrainte de sécurité |
|---|---|---|
| **1** | Les poignées doivent être à la bonne hauteur pour la déambulation. | *Exploitable par un moniteur de sécurité :* (v=0) ∨ (h>hmin ∧ h < hmax) |
| **2** | Les batteries doivent être suffisamment chargées pour aller et revenir au point le plus loin et générer des alarmes | *Exploitable par un dispositif d'interverrouillage* |
| **3** | Le patient doit être en bonne position pour pouvoir déambuler | *Exploitable par un moniteur de sécurité :* (v=0) ∨ (posture≠chute) |
| **4** | Le robot doit être tenu par le patient avec les deux poignées | *Exploitable par un moniteur de sécurité :* (v=0) ∨ (contact_poignées=2) |
| **5** | Le robot doit détecter que le patient le tient | *intégrité matérielle requise* |
| **6** | Le robot doit être en mode déambulation | *intégrité logicielle requise* |
| **7** | Le robot ne doit pas passer dans un autre mode sans raison valable | *intégrité logicielle requise* |
| **8** | L'état physiologique du patient doit être acceptable lors de la déambulation | *Exploitable par un moniteur de sécurité :* (v=0) ∨ ((Température≥Temp_min ∧ Température≤ Trempâmes) ∧ (Pulsation≥Puls_min ∧ Pulsation ≤ Puls_max)) |
| **9** | Les indicateurs physiologiques ne doivent pas être incorrects | *intégrité matérielle requise* |
| **10** | Les indicateurs physiologiques ne doivent pas être interprétés trop tard | *Exploitable par un moniteur de sécurité :* (v=0) ∨ ((Température≥Temp_min ∧ Température≤ Temp_max) ∧ (Pulsation≥Puls_min ∧ Pulsation ≤ Puls_max)) |
| **11** | Le robot ne doit pas entrer en collision avec un objet | *Exploitable par un moniteur de sécurité :* $(v < (2*\alpha*d)^{1/2})$ |
| **12** | Le robot ne doit pas détecter une collision qui n'en est pas une | *intégrité matérielle requise* |
| **13** | Le robot doit détecter l'imminence d'une collision | *Exploitable par un moniteur de sécurité :* $(v < (2*\alpha*d)^{1/2})$ |
| **14** | Le robot ne doit pas sortir des zones autorisées pour la déambulation | *Variable pertinente à la sécurité non observable* |
| **15** | Le robot doit pouvoir se localiser correctement | *Variable pertinente à la sécurité non observable* |
| **16** | La vitesse des roues ne doit pas dépasser Vmax m/s | *Exploitable par un moniteur de sécurité :* (v < Vmax) |

TABLE 4.8 – Les contraintes de sécurité du cas d'utilisation *Gestion de la déambulation* exprimées en langage formel

La Table 4.9 illustre les contraintes de sécurité, du cas d'utilisation *Gestion de la verticalisation*, qui ont pu être exprimées en langage formel.

Sur les 16 contraintes de sécurité exprimées en langage naturel :

1. Cinq ont pu être exprimées sous forme de prédicats (9, 11, 13, 15 et 16) et ont été traitées identiquement à ceux du cas d'utilisation précédent.

2. Onze contraintes n'ont pas pu être exprimées sous forme de prédicats (1, 2, 3, 4, 5, 6, 7, 8, 10, 12, 14) :

   - Les contraintes 1, 2 et 5 ne sont pas exploitables à défaut de moyens d'observation. Elles stipulent que le patient doit être assis correctement pour commencer la verticalisation, or les capteurs de position inclus dans le déambulateur détectent si le patient est bien debout, s'il est en déséquilibre ou bien s'il a chuté.

Ces capteurs ne détectent pas la position assise du patient. Afin de surveiller une telle contrainte, il est nécessaire d'implémenter de nouveaux algorithmes pour la détection de la posture assise. Il serait aussi possible d'implémenter des capteurs supplémentaires. Concrètement, c'est le patient qui positionne le robot en face de lui et qui estime qu'il est bien en face de lui (voir cas d'utilisation n° 07 : *Positionner le robot à la main*).

- La contrainte 3 est englobée par la contrainte 4 stipule que : "Les batteries doivent être suffisamment chargées pour la verticalisation et la déverticalisation". Cette contrainte a été traitée identiquement à la contrainte n° 2 du cas d'utilisation *Gestion de la déambulation*.

- Les contraintes n° 6, 7, 8 et 12 ont été traitées identiquement aux contraintes n° 6 et 7 du cas d'utilisation *Gestion de la déambulation*.

- Les contraintes 10 et 14 ont été traitées identiquement aux contraintes n° 5 et 9 respectivement, du cas d'utilisation *Gestion de la déambulation*.

L'étape suivante est de spécifier à partir de ces contraintes de sécurité formelles, les conditions de déclenchement de sécurité correspondantes.

### 4.3.2 Spécification des règles de sécurité

Afin d'illustrer la démarche, nous allons traiter quatre exemples : dans les deux premiers, la contrainte de sécurité est exprimée à l'aide d'une variable pertinente à la sécurité *continue* ; dans le troisième exemple, la contrainte de sécurité est exprimée à l'aide d'une variable pertinente à la sécurité *booléenne* et un dernier exemple où la contrainte de sécurité est exprimée à l'aide d'une variable pertinente à la sécurité de type *énumération*.

#### Exemple 1 : "le robot ne doit pas entrer en collision avec un objet"

Considérons la contrainte de sécurité n°11 de la Table 4.8 : "*le robot ne doit pas entrer en collision avec un objet*. Afin d'éviter une collision, la vitesse du robot doit être toujours inférieure à $\sqrt{2\alpha d}$, où $d$ représente la distance du robot de l'objet le plus proche et $\alpha$ représente la décélération minimale lors du freinage. Par conséquent, la contrainte de sécurité est exprimée formellement comme suit : $v < \sqrt{2\alpha d}$ où $v \in \mathbb{R}^+$ représente la vitesse des roues. $\vartheta =< v, d >$ est le vecteur des variables pertinentes à la sécurité. Ces deux variables étant observables par le moniteur, nous pouvons faire l'hypothèse que cette contrainte de sécurité est l'invariant de sécurité à traiter : $SI(\mathbf{x}) = v < \sqrt{2\alpha d}$.

La partie gauche de la Figure 4.6 représente le graphe de région correspondant à cet invariant de sécurité, d'où l'on peut constater que :

– la transition depuis l'état catastrophique vers l'état non-catastrophique a été

| | Contrainte de sécurité en langage naturel | Classification de la contrainte de sécurité |
|---|---|---|
| **1** | Le patient doit être assis pour la verticalisation | *Exploitable par un dispositif d'interverrouillage* |
| **2** | Le patient doit être assis correctement pour la verticalisation | |
| **3** | Les batteries doivent être suffisamment chargées pour la verticalisation | *Exploitable par un dispositif d'interverrouillage* |
| **4** | Les batteries doivent être suffisamment chargées pour la verticalisation et la déverticalisation | *Exploitable par un dispositif d'interverrouillage* |
| **5** | Le robot doit être en face du patient pour la verticalisation | *Variable pertinente à la sécurité non observable* |
| **6** | Le robot doit être en mode déambulation à la fin de la verticalisation | *intégrité logicielle requise* |
| **7** | Le robot ne doit pas passer en mode déambulation trop tôt | *intégrité logicielle requise* |
| **8** | Le robot ne doit pas passer en mode déambulation trop tard | *intégrité logicielle requise* |
| **9** | Le patient doit tenir le robot | *Exploitable par un moniteur de sécurité :* (v_bras=0) ∨ (contact_poignées=2) |
| **10** | Le robot doit détecter que le patient le tient | *intégrité matérielle requise* |
| **11** | Le patient doit tenir le robot par les deux poignées lors de la verticalisation | *Exploitable par un moniteur de sécurité :* (v_bras=0) ∨ (contact_poignées=2) |
| **12** | Le robot doit être en mode verticalisation | *intégrité logicielle requise* |
| **13** | Le robot ne doit pas verticaliser si les indicateurs physiologiques du patient ne sont pas acceptables | *Exploitable par un moniteur de sécurité :* (v_bras=0) ∨ ((Température>Temp_min ∧ Température< Temp_max) ∧ (Pulsation>Puls_min ∧ Pulsation < Puls_max)) |
| **14** | Les indicateurs physiologiques du patient doivent être corrects | *intégrité matérielle requise* |
| **15** | Les indicateurs physiologiques ne doivent pas être interprétés trop tard | *Exploitable par un moniteur de sécurité :* (v_bras=0) ∨ ((Température>Temp_min ∧ Température< Temp_max) ∧ (Pulsation>Puls_min ∧ Pulsation < Puls_max)) |
| **16** | La vitesse des bras ne doit pas dépasser Vmax_bras m/s | *Exploitable par un moniteur de sécurité :* (v_bras < Vmax_bras) |

TABLE 4.9 – Les contraintes de sécurité du cas d'utilisation *Gestion de la verticalisation* exprimées en langage formel

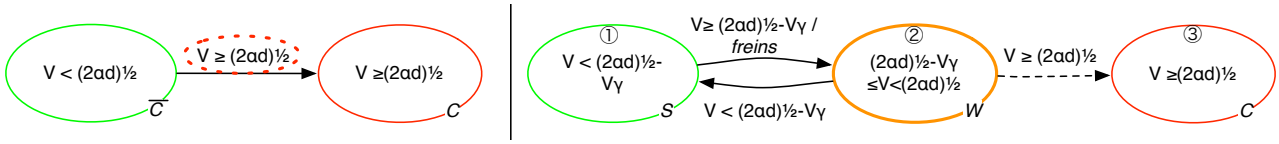


FIGURE 4.6 – Traitement de la contrainte de sécurité : "*le robot ne doit pas entrer en collision avec un objet*"

omise car nous considérons qu'une fois que le système atteint la région catastrophique, la catastrophe est alors inévitable. Par conséquent, il n'est pas possible de revenir vers un état non-catastrophique depuis un état catastrophique.
– il existe une transition critique (marquée par des pointillés rouges) : $v \geq \sqrt{2\alpha d}$.

Comme le stipule le théorème vu au Chapitre 3, une région d'alerte existe pour la transition critique ($v \geq \sqrt{2\alpha d}$) si et seulement s'il existe une partition de l'ensemble $\{ v \in \mathbb{R}^+ | v < \sqrt{2\alpha d}\}$ en deux sous-ensembles non vides. Nous allons donc tester si la condition d'existence de la marge (voir Chapitre 3, Condition 3.6) est vérifiée ou pas.

Calculons l'ensemble des états de la frontière entre l'ensemble $\{ v \in \mathbb{R}^+ | v < \sqrt{2\alpha d}\}$ et l'ensemble $\{ v \in \mathbb{R}^+ | v \geq \sqrt{2\alpha d}\}$ :

$$\begin{aligned} X_F &= adherence(\{v \in \mathbb{R}^+ | v < \sqrt{2\alpha d}\}) \cap \\ &\quad adherence(\{v \in \mathbb{R}^+ | v \geq \sqrt{2\alpha d}\}) \\ &= ]0, \sqrt{2\alpha d}] \cap [\sqrt{2\alpha d}, \infty[ \\ &= \{\sqrt{2\alpha d}\} \end{aligned}$$

$$Card(\{v \in \mathbb{R}^+ | v < \sqrt{2\alpha d}\} \setminus \{\sqrt{2\alpha d}\}) = Card(]0, \sqrt{2\alpha d}[) > 1$$

si : $\alpha > 0$ et $d > 0$.

Par conséquent, l'ensemble de départ $\{ v \in \mathbb{R}^+ | v < \sqrt{2\alpha d}\}$ peut être partitionné en deux sous ensembles non vides. Une région d'alerte peut donc être spécifiée pour la transition critique $v \geq \sqrt{2\alpha d}$.

La variable $v$ étant continue, nous pouvons calculer la marge de sécurité comme présenté au Chapitre 3, Section 3.3.3.

Nous exprimons l'invariant de sécurité sous forme d'inégalité (voir Chapitre 3, Equation 3.7) :

$$SI(\mathbf{x}) = (v < \sqrt{2\alpha d}) = (f(\mathbf{x}) < 0)$$

où :

$$f(\mathbf{x}) = v - \sqrt{2\alpha d}$$

D'après l'Equation 3.11 du Chapitre 3, nous pouvons exprimer une fonction de déclenchement de sécurité *révélatrice* $g_b(\mathbf{x})$ comme suit :

$$g(\mathbf{x}) = f(\mathbf{x}) + \gamma = v - \sqrt{2\alpha d} + v_\gamma$$

ce qui correspond à la condition de déclenchement de sécurité définie par l'équation 3.8, Chapitre 3 :

$$STC(\mathbf{x}) = (v \geq \sqrt{2\alpha d} - v_\gamma)$$

La partie droite de la Figure 4.6 illustre le graphe résultant de la spécification de la région d'alerte :

- la région non-catastrophique du graphe de gauche à été partitionnée en deux régions (1) et (2),
- lorsque la vitesse du robot $v$ dépasse $\sqrt{2\alpha d} - v_\gamma$ le système se retrouve dans la région d'alerte (2) ce qui enclenche les *freins* afin de ralentir la vitesse du robot. Les freins enclenchés peuvent être un dispositif de freinage supplémentaire aux freins existants au sein du système. Dans le cas où l'implémentation d'un tel dispositif s'avère coûteuse (temps, effort, coût, etc.), le moniteur de sécurité peut aussi agir sur les freins existants et commander une décélération.

**Exemple 2 : "les poignées doivent être à la bonne hauteur pour la déambulation"**

Considérons la contrainte de sécurité n°1 de la Table 4.8 : "*les poignées doivent être à la bonne hauteur pour la déambulation*" exprimée formellement comme suit : $(v = 0) \vee (h > hmin \ \wedge \ h < hmax)$. La bonne hauteur pour la déambulation est spécifiée par les experts comme étant l'intervalle $I = ]hmin, hmax[$. Soit $\vartheta =< v, h >$ le vecteur des variables pertinentes à la sécurité où $v \in \mathbb{R}^+$ représente la vitesse du robot et $h \in \mathbb{R}^+$ représente la hauteur des poignées du déambulateur. Ces deux variables étant observables par le système, nous pouvons faire l'hypothèse que cette contrainte de sécurité est l'invariant de sécurité à traiter : $SI(\mathbf{x}) = ((v = 0) \vee (h \in I))$.

Cet invariant de sécurité est exprimé sous forme disjonctive où les deux variables $v$ et $h$ sont indépendantes ; nous pouvons alors construire le graphe de région correspondant. La partie gauche de la Figure 4.7 représente ce graphe de région d'où l'on peut retenir que :

- Les transitions depuis la région (4) vers les régions (1), (2) et (3) sont omises car il n'est pas possible de revenir vers un état non-catastrophique depuis un état catastrophique.
- La transition depuis la région (1) vers la région (4) et les transitions entre les régions (2) et (3) ne sont pas prises en considération car elles impliquent le changement simultané des valeurs des deux variables $v$ et $h$. Comme indiqué au Chapitre 3, Section 3.3.1, les transitions induisant deux changements de valeur se traduisent en deux transitions successives correspondant au changement d'une seule valeur à la fois.
- Deux transitions critiques apparaissent (marquées par des pointillés rouges) : $(v > 0)$ et $(h \notin I)$.

1. Nous allons tenter de définir une marge de sécurité afin d'insérer une région d'alerte sur la transition critique $(v > 0)$.

   Comme le stipule le théorème vu au Chapitre 3, une région d'alerte existe pour la transition critique $(v > 0)$ si et seulement s'il existe une partition de l'ensemble $\{ \ v \in \mathbb{R}^+ | (v = 0) \}$ en deux sous-ensembles non vides. De manière évidente, il

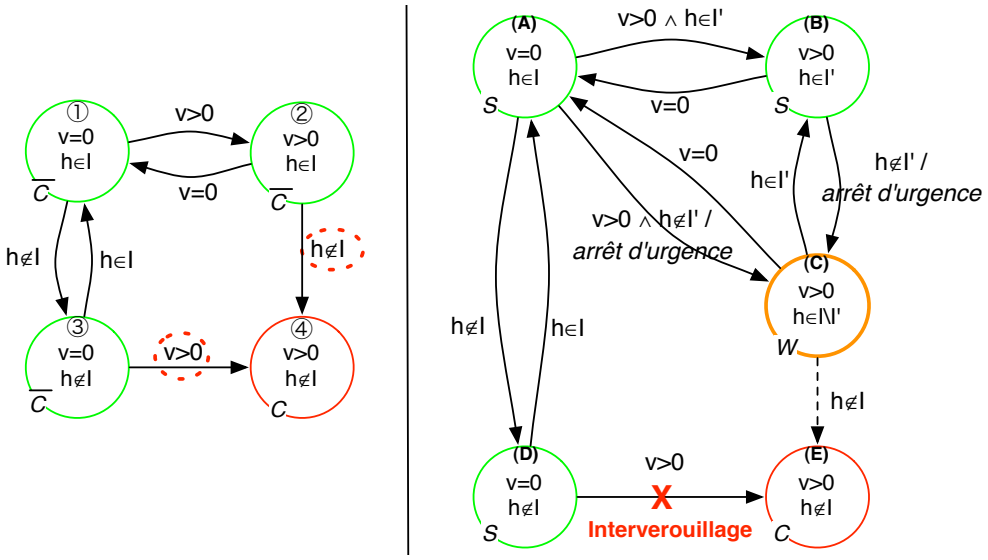


FIGURE 4.7 – Traitement de la contrainte de sécurité : "*les poignées doivent être à la bonne hauteur pour la déambulation*"

n'est pas possible de partitionner cet ensemble, néanmoins nous allons tester si la condition d'existence de la marge (voir Chapitre 3, Condition 3.6) est vérifiée ou pas.
Calculons l'ensemble des états de la frontière entre l'ensemble $\{v \in \mathbb{R}^+|(v = 0)\}$ et l'ensemble $\{v \in \mathbb{R}^+|(v > 0)\}$ :

$$\begin{aligned} X_F &= adherence(\{v \in \mathbb{R}^+|(v = 0)\}) \cap \\ &\quad adherence(\{v \in \mathbb{R}^+|(v > 0)\}) \\ &= \{0\} \cap [0, \infty[ \\ &= \{0\} \end{aligned}$$

$$Card(\{v \in \mathbb{R}^+|(v = 0)\} \setminus \{0\}) = Card(\emptyset) = 0$$

L'ensemble $\{v \in \mathbb{R}^+|(v = 0)\}$ ne peut donc pas être partitionné ; par conséquent aucune région d'alerte ne peut être spécifiée pour la transition critique. Cette transition devra donc être inhibée par un interverrouillage. Celui-ci doit empêcher le robot de bouger, par des moyens mécaniques ou électriques, tant que $h \notin I$.

2. Nous allons tenter de définir une marge de sécurité afin d'insérer une région d'alerte sur la transition critique ($h \notin I$). Calculons l'ensemble des états de la frontière entre l'ensemble $\{h \in \mathbb{R}^+|h \in I\}$ et l'ensemble $\{h \in \mathbb{R}^+|h \notin I\}$.

$$\begin{aligned} X_F &= adherence(\{h \in \mathbb{R}^+|h \in I\}) \ \cap \ adherence(\{h \in \mathbb{R}^+|h \notin I\}) \\ &= adherence(\{h \in \mathbb{R}^+|h \in ]hmin, hmax[\}) \cap \\ &\quad adherence(\{h \in \mathbb{R}^+|h \in ]\infty, hmin] \ \cup [hmax, \infty[\}) \\ &= \{[hmin, hmax]\} \cap \{]\infty, hmin] \cup [hmax, \infty[\} \\ &= \{\{hmin\}, \{hmax\}\} \end{aligned}$$

$Card(\{h \in \mathbb{R}^+ | h \in ]hmin, hmax[\} \backslash \{\{hmin\}, \{hmax\}\}) = Card(]hmin, hmax[) > 1$

si $hmax > hmin$.

Par conséquent, l'ensemble de départ $\{h \in \mathbb{R}^+ | h \in ]hmin, hmax[\}$ peut être partitionné en deux sous ensembles non vides. Une région d'alerte peut donc être spécifiée pour la transition critique ($h \notin I$).

La variable *hauteur des poignées* $h$ étant continue, nous pouvons calculer la marge de sécurité comme présenté au Chapitre 3, Section 3.3.3.
Nous exprimons l'invariant de sécurité sous forme d'inégalité (voir Chapitre 3, Equation 3.7) :

$$SI_{(b)} = (h \in I) = ((h > hmin) \wedge (h < hmax)) = (f_b(\mathbf{x}) < 0)$$

où :

$$f_b(\mathbf{x}) = max(hmin - h, h - hmax)$$

D'après l'Equation 3.11 du Chapitre 3, nous pouvons exprimer une fonction de déclenchement de sécurité *révélatrice* $g_b(\mathbf{x})$ comme suit :

$$g_b(\mathbf{x}) = f_b(\mathbf{x}) + \beta = max((hmin + \beta) - h, h - (hmax - \beta))$$

ce qui correspond à la condition de déclenchement de sécurité définie par l'équation 3.8, Chapitre 3 :

$$STC_b(\mathbf{x}) = (h \notin I') \; avec \; I' = [hmin + \beta, hmax - \beta]$$

La partie droite de la Figure 4.7 illustre le graphe résultant de la spécification de la région d'alerte :

– La région (2) du graphe de gauche a été partitionnée en deux régions (B) et (C).

– La transition vers la région partitionnée (2) a donné lieu à deux transitions : l'une vers la région (B) et l'autre vers la région (C).

– La transition sortant de la région partitionnée (2) vers la région (1) a donné lieu à deux transitions vers la région (1) : la première depuis la région (B) et la seconde depuis la région (C).

– Il n'y a pas de transition depuis la région (B) vers la région (E) car la variable $h$ varie de façon continue ; le système ne peut atteindre la région (E) depuis (B) qu'en passant par la région (C).

– Lorsque la hauteur des poignées $h$ quitte l'intervalle $I'$ alors que le robot est en déambulation (transition de (B) vers (C)), nous avons choisi d'enclencher l'*arrêt d'urgence* afin d'arrêter le robot et cela en ajoutant au robot un dispositif de freinage d'urgence commandable par le moniteur de sécurité par exemple. L'action de sécurité impacte sur la vitesse et non pas sur la hauteur des poignées car il n'est pas possible d'agir directement sur l'actionnement de ces dernières.

– Lorsque la vitesse du robot $v$ devient positive (le robot initie la déambulation) alors que la hauteur des poignées $h$ est en dehors de l'intervalle $I'$, le système se retrouve directement dans la région d'alerte (C). Dans cette situation, nous avons aussi choisi d'enclencher l'*arrêt d'urgence* afin d'arrêter le robot.

**Exemple 3 : "le robot doit être tenu par le patient avec les deux poignées"**

Considérons la contrainte de sécurité n°4 de la Table 4.8 : "*le robot doit être tenu par le patient avec les deux poignées*", exprimée formellement comme suit : $(v = 0) \vee (contact\_poignees = true)$. Soit $\vartheta =< v, contact\_poignees >$ le vecteur des variables pertinentes à la sécurité où $v \in \mathbb{R}^+$ représente la vitesse du robot et $contact\_poignees \in \mathbb{B}$ représente le fait que le déambulateur soit tenu par le patient par les deux poignées. Ces deux variables étant observables par le moniteur, nous pouvons faire l'hypothèse que cette contrainte de sécurité est l'invariant de sécurité à traiter : $SI(\mathbf{x}) = ((v = 0) \vee (contact\_poignees = true))$.

Cet invariant de sécurité est exprimé sous forme disjonctive où les deux variables $v$ et $contact\_poignees$ sont indépendantes ; nous pouvons alors construire le graphe de région correspondant. La Figure 4.8 représente ce graphe de région. Il y apparaît deux transitions critiques qui sont : $(v > 0)$ et $(contact\_poignees = false)$.

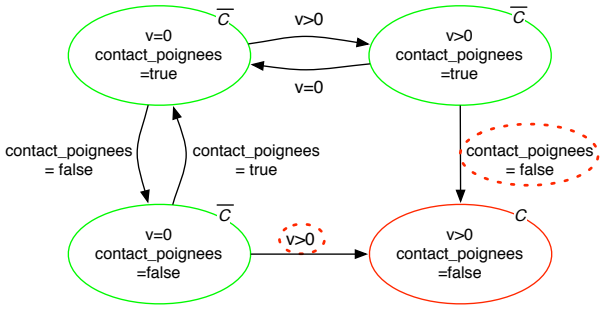


FIGURE 4.8 – Graphe de région correspondant à la contrainte de sécurité : "*le robot doit être tenu par le patient*"

1. La transition critique $(v > 0)$ est traitée comme dans l'exemple précédent, c'est-à-dire en implémentant un interverrouillage.

2. Nous allons tenter de définir une marge de sécurité afin d'insérer une région d'alerte sur la transition critique $(contact\_poignees = false)$. Nous étudions la

possibilité que l'ensemble $\{contact\_poignees \in \mathbb{B} | (contact\_poignees = true)\}$ soit partitionnable en deux ensembles non vides.
L'ensemble des états de la frontière est :

$$\begin{aligned} X_F &= adherence(\{contact\_poignees \in \mathbb{B} | (contact\_poignees = true)\}) \cap \\ &\quad adherence(\{contact\_poignees \in \mathbb{B} | (contact\_poignees = false)\}) \\ &= \{true\} \cap \{false\} \\ &= \emptyset \end{aligned}$$

Par conséquent,

$$Card(\{contact\_poignees \in \mathbb{B} | (contact\_poignees = true)\} \setminus \emptyset) = 1$$

La condition n'étant pas vérifiée, l'ensemble $\{contact\_poignees \in \mathbb{B} | (contact\_poignees = true)\}$ ne peut donc pas être partitionné.

Nous tentons de spécifier une marge de sécurité temporelle en étudiant si la contrainte de sécurité *"le robot doit être tenu par le patient"* peut être temporairement violée. Après discussion avec les experts en sécurité, il est apparu que le patient peut lâcher, sans danger, le robot lors de la déambulation pendant un petit laps de temps $T$. Le nouvel invariant de sécurité à traiter est : $SI'(\mathbf{x}) = ((v = 0) \vee (t\_perte\_contact < T))$ où $t\_perte\_contact \in \mathbb{R}^+$ est une variable observable par le système qui représente la durée pendant laquelle le patient lâche une des deux poignées du robot. La partie gauche de la Figure 4.9 illustre le graphe de région correspondant au nouvel invariant de sécurité.

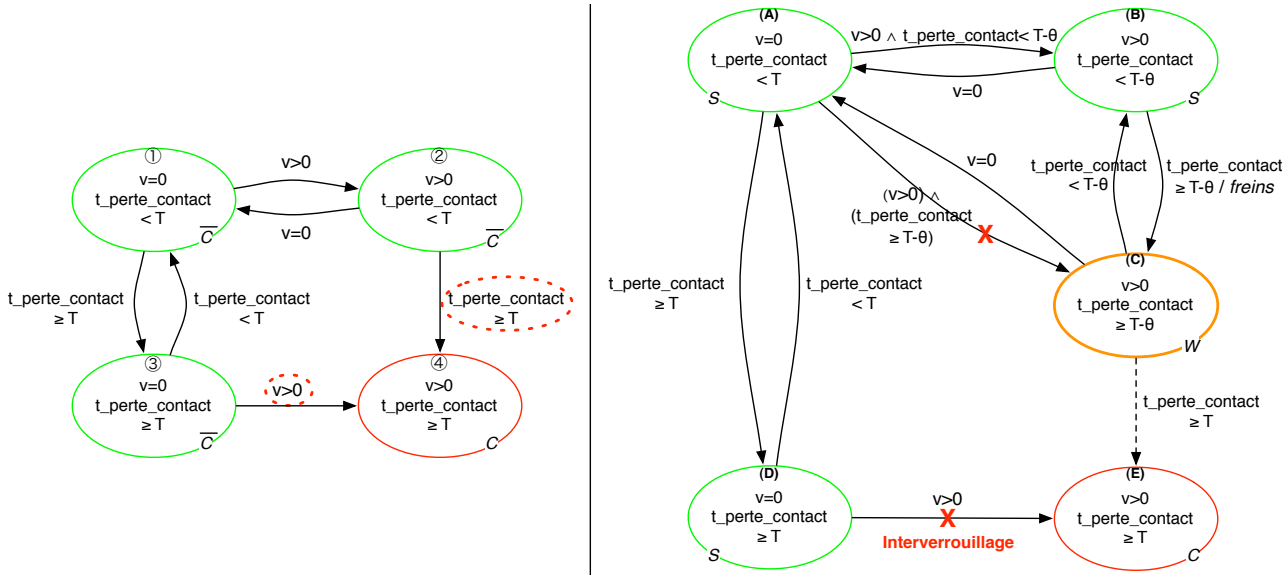


FIGURE 4.9 – Traitement de la contrainte de sécurité : "*le robot doit être tenu par le patient*" avec une marge temporelle

Nous allons essayer de spécifier une région d'alerte sur la transition critique $(t\_perte\_contact \geq T)$ en étudiant la possibilité de partitionner l'ensemble

$\{t\_perte\_contact \in \mathbb{R}^+ | (t\_perte\_contact \in ]0, T[)\}$. L'ensemble des états de la frontière est :

$$\begin{aligned} X_F &= adherence(\{t\_perte\_contact \in \mathbb{R}^+ | (t\_perte\_contact \in ]0, T[)\} \cap \\ &\quad adherence(\{t\_perte\_contact \in \mathbb{R}^+ | (t\_perte\_contact \in [T, \infty[)\} \\ &= \{[0, T]\} \cap \{[T, \infty[\} \\ &= \{T\} \end{aligned}$$

$$\begin{aligned} &Card(\{t\_perte\_contact \in \mathbb{R}^+ | (t\_perte\_contact \in ]0, T[)\} \setminus \{T\}) \; = \\ &Card(]0, T[) \; > \; 1 \end{aligned}$$

Par conséquent, l'ensemble $\{t\_perte\_contact \in \mathbb{R}^+ | (t\_perte\_contact \in ]0, T[)\}$ peut être partitionné en deux sous ensembles non vides. Comme dans l'exemple 1, vu que $t\_perte\_contact$ est une variable continue, nous pouvons calculer la marge de sécurité comme présenté au Chapitre 3, Section 3.3.3.

Nous exprimons l'invariant de sécurité sous forme d'inégalité $SI_b = (f_b(\mathbf{x}) < 0)$ où : $f_b(\mathbf{x}) = t\_perte\_contact - T$. Nous pouvons exprimer une fonction de déclenchement de sécurité *révélatrice* $g_b(\mathbf{x})$ comme suit :

$$g_b(\mathbf{x}) = f_b(\mathbf{x}) + \theta = t\_perte\_contact \; - \; T \; + \theta$$

ce qui correspond à la condition de déclenchement de sécurité :

$$STC_b = (t\_perte\_contact \geq \; T - \theta)$$

La partie droite de la Figure 4.9 illustre le graphe résultant de la spécification de la région d'alerte :

- Lorsque le patient relâche une des deux poignées lors de la déambulation, pendant plus de $T - \theta$ secondes, nous avons choisi d'enclencher les *freins* afin de ralentir la vitesse du robot jusqu'à son arrêt total. Nous ne pouvons pas agir sur la variable $t\_perte\_contact$ vu qu'elle n'est pas commandable ; elle dépend exclusivement du patient.

- Lorsque le patient initie la déambulation alors qu'il ne tient plus le robot par les deux poignées depuis plus de $T-\theta$ ms, le système se retrouve directement dans la région d'alerte. Nous avons choisi d'implémenter un interverrouillage afin de ne pas permettre au patient de commencer (ou recommencer) à déambuler s'il ne tient pas le robot par les deux poignées.

**Exemple 4 : "le patient doit être en bonne position pour déambuler"**

Considérons la contrainte de sécurité n°3 de la Table 4.8 : "*le patient doit être en bonne position pour déambuler*" correspondant à la contrainte formelle : $(v = 0) \vee (posture \neq chute)$. Soit $\vartheta =< v, posture >$ le vecteur des variables pertinentes à la sécurité où $v \in \mathbb{R}^+$ représente la vitesse du robot et $posture \in \{debout, desequilibre, chute\}$

représente la posture du patient lors de la déambulation. Ces deux variables étant observables par le système, nous pouvons considérer cette contrainte de sécurité comme étant l'invariant de sécurité à traiter : $SI(\mathbf{x}) = ((v = 0) \vee (posture \neq chute))$.

Cet invariant de sécurité est exprimé sous forme disjonctive où les deux variables $v$ et $posture$ sont indépendantes ; nous pouvons alors construire le graphe de région correspondant. La partie gauche de la Figure 4.10 représente ce graphe de région. Il y apparaît deux transitions critiques qui sont : $(v > 0)$ et $(posture = chute)$.

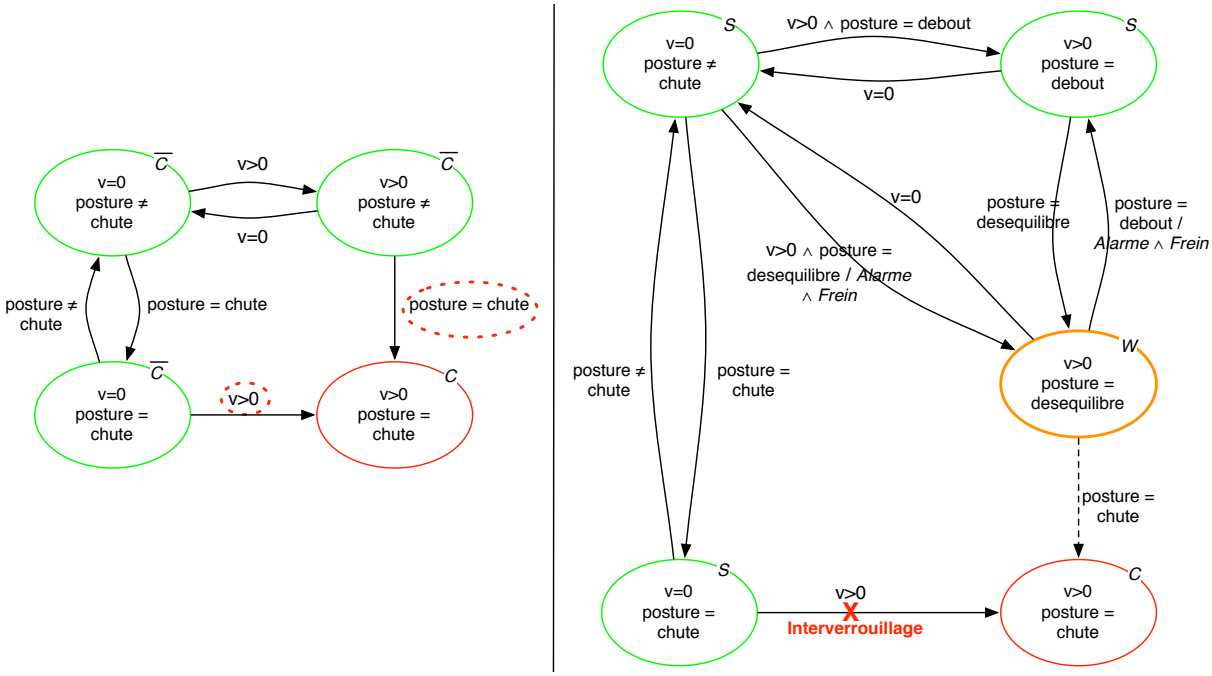


FIGURE 4.10 – Traitement de la contrainte de sécurité : "*le patient doit être en bonne position pour déambuler*"

1. La transition critique $(v > 0)$ est traitée comme précédemment, c'est-à-dire en implémentant un interverrouillage.

2. Nous allons tenter de définir une marge de sécurité afin d'insérer une région d'alerte sur la transition critique $(posture = chute)$. D'une manière évidente, l'ensemble $\{posture \in \{debout, desequilibre, chute\} | \ (posture \neq chute)\}$ peut être partitionné en deux sous ensembles non vides. Néanmoins, nous allons étudier la possibilité de partition en utilisant le théorème du Chapitre 3.
   L'ensemble des états de la frontière est :

$$\begin{aligned} X_F &= adherence(\{posture \in \{debout, desequilibre, chute\} | (posture \neq chute)\}) \cap \\ &\quad adherence(\{posture \in \{debout, desequilibre, chute\} | (posture = chute)\}) \\ &= \{debout, desequilibre\} \cap \{chute\} \\ &= \emptyset \end{aligned}$$

$$\begin{aligned} &Card(\{posture \in \{debout, desequilibre, chute\} | (posture \in \{debout, desequilibre\}\} \setminus \emptyset) \\ &= 2 \end{aligned}$$

| | **Invariant de sécurité (SI)** | **Condition de déclenchement de sécurité (STC)** | **Contrainte n°** |
|---|---|---|---|
| **1** | (v=0) ∨ (h > hmin ∧ h < hmax) | (v=0) ∧ (h ≥ hmin+β ∨ h ≤ hmax-β) | 1 |
| **2** | (v=0) ∨ (posture≠chute) | (v=0) ∧ (posture=desequilibre) | 3 |
| **3** | (v=0) ∨ (t_perte_contact < T) | (v=0) ∧ (t_perte_contact ≥ T-θ) | 4 |
| **4** | (v=0) ∨ ((Température>Temp_min ∧ Température< Temp_max) ∧ (Pulsation>Puls_min ∧ Pulsation < Puls_max)) | (v=0) ∧ ((Température≤Temp_min+$\omega_1$ ∨ Température≥ Temp_max-$\omega_2$) ∨ (Pulsation≤Puls_min+$\lambda_1$ ∨ Pulsation ≥ Puls_max-$\lambda_2$)) | 8,10 |
| **5** | (v < (2*α * d)$^{1/2}$) | (v ≥ (2*α * d)$^{1/2}$- $v_\gamma$) | 11,13 |
| **6** | (v < Vmax) | (v ≥ Vmax-δ) | 16 |

TABLE 4.10 – Invariants de sécurité et conditions de déclenchement de sécurité du cas d'utilisation *Gestion de la déambulation*

| | **Invariant de sécurité (SI)** | **Condition de déclenchement de sécurité (STC)** | **Contrainte n°** |
|---|---|---|---|
| **1** | (v_bras=0) ∨ (t_perte_contact < T) | (v_bras=0) ∧ (t_perte_contact ≥ T-λ) | 12,14 |
| **2** | (v_bras=0) ∨ (t_perte_contact < T) | | |
| **3** | (v_bras=0) ∨ ((Température>Temp_min ∧ Température< Temp_max) ∧ (Pulsation>Puls_min ∧ Pulsation < Puls_max)) | (v_bras=0) ∧ ((Température≤Temp_min+δ ∨ Température≥ Temp_max-δ) ∨ (Pulsation≤Puls_min+θ ∨ Pulsation ≥ Puls_max-θ)) | 16,18 |
| **4** | (v_bras=0) ∨ ((Température>Temp_min ∧ Température< Temp_max) ∧ (Pulsation>Puls_min ∧ Pulsation < Puls_max)) | | |
| **5** | (v_bras < Vmax_bras) | (v_bras ≥ Vmax_bras-ω) | 19 |

TABLE 4.11 – Invariants de sécurité et conditions de déclenchement de sécurité du cas d'utilisation *Gestion de la verticalisation*

La condition étant vérifiée, l'ensemble $\{posture \in \{debout, desequilibre, chute\} | (posture \neq chute)\}$ peut être partitionné en deux sous-ensembles non vides.

Une façon de définir la région d'alerte (et donc la marge de sécurité) sur des variables énumérées est d'avoir au préalable un graphe des transitions possibles. Typiquement, dans notre exemple le graphe des transitions indiquera que le patient passe de la position *debout* à la position *desequilibre* puis passe de cette dernière vers la position *chute*. Par conséquent, nous pouvons définir la région d'alerte de telle sorte qu'elle représente un état intermédiaire dans le graphe des transitions. La condition de déclenchement de sécurité serait : $STC_b = (posture = desequilibre)$. Nous considérons que si la $(posture = desequilibre)$, le déambulateur a tenté de redresser le patient sans succès.

La partie droite de la Figure 4.10 illustre le graphe résultant de la spécification de la région d'alerte. Lors de la déambulation, si le patient passe de la position *debout* à la position *desequilibre*, nous avons choisi d'enclencher l'*Alarme* afin d'avertir le personnel médical ainsi qu'engager les freins pour ralentir le déambulateur.

Les Tables 4.10 et 4.11 récapitulent l'ensemble des invariants de sécurité ainsi que les conditions de déclenchement correspondantes, des cas d'utilisation *Gestion de la déambulation* et *Gestion de la verticalisation* respectivement.

### 4.3.3 Identification des actions de sécurité concomitantes

Afin d'illustrer l'approche proposée en Section 3.5 du Chapitre 3, nous allons considérer le système sous deux invariants de sécurité qui sont : $(v < \sqrt{2\alpha d})$ (Table 4.10, Invariant n° 6) et $((v = 0) \vee (h \in I))$ (Table 4.10, Invariant n°1).

La Figure 4.11 récapitule les deux graphes de région correspondant aux invariants de sécurité, cités ci dessus, après spécification des marges de sécurité. La Figure 4.12 illustre la composition de ces deux graphes.

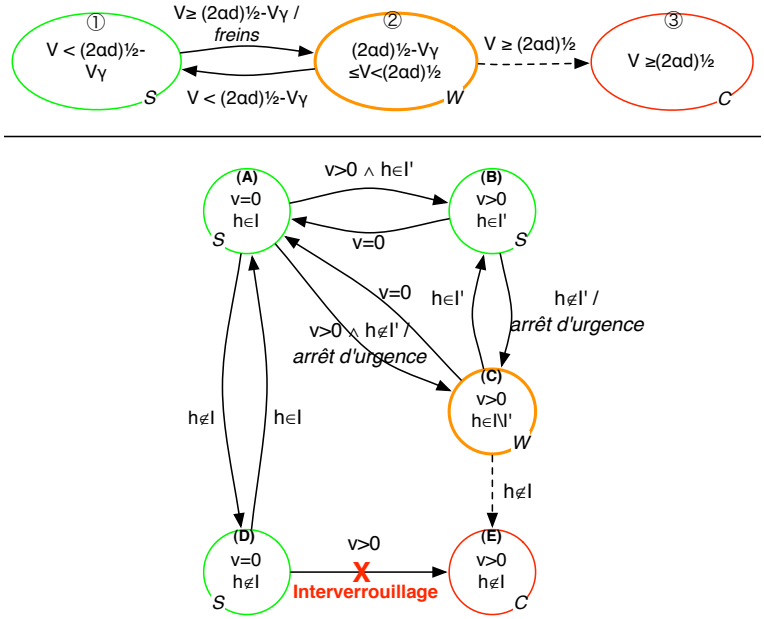


FIGURE 4.11 – Graphes de région pour la composition

Le graphe résultant de la composition comprend moins de régions que le produit de leurs cardinalités (soit 3 * 5 = 15). En effet, le produit cartésien de deux régions (qui est la valuation de la conjonction des deux prédicats définissant chacune des deux régions) peut donner lieu à une région vide. Par exemple, le produit cartésien de la région (2) et de la région (A) de la Figure 4.11 résulte en une région définie par le prédicat $(\sqrt{2\alpha d} - v_\gamma \leq v < \sqrt{2\alpha d}) \wedge (v = 0) \wedge (h \in I)$. Il n'existe aucun état du système qui puisse satisfaire ce prédicat. Par conséquent, le produit cartésien de la région (2) et de la région (A) donne lieu à une région vide. De ce fait le graphe issu de la composition contient 11 régions au lieu de 15.

Il n'existe aucune transition entre régions catastrophiques car nous considérons qu'une fois que le système atteint une région catastrophique, malgré les actions de sécurité enclenchées pour empêcher cela, la catastrophe est inévitable. Ce choix de conception fait que les transitions entre régions catastrophiques ne sont pas considérées (car il n'y a plus de recouvrement possible à ce stade). De ce fait, il n'y aucune transition qui mène vers la région (11). Cette région n'est atteignable que par les régions (8) et (10) qui sont des régions catastrophiques.

En analysant le graphe composé, on peut faire les observations suivantes :

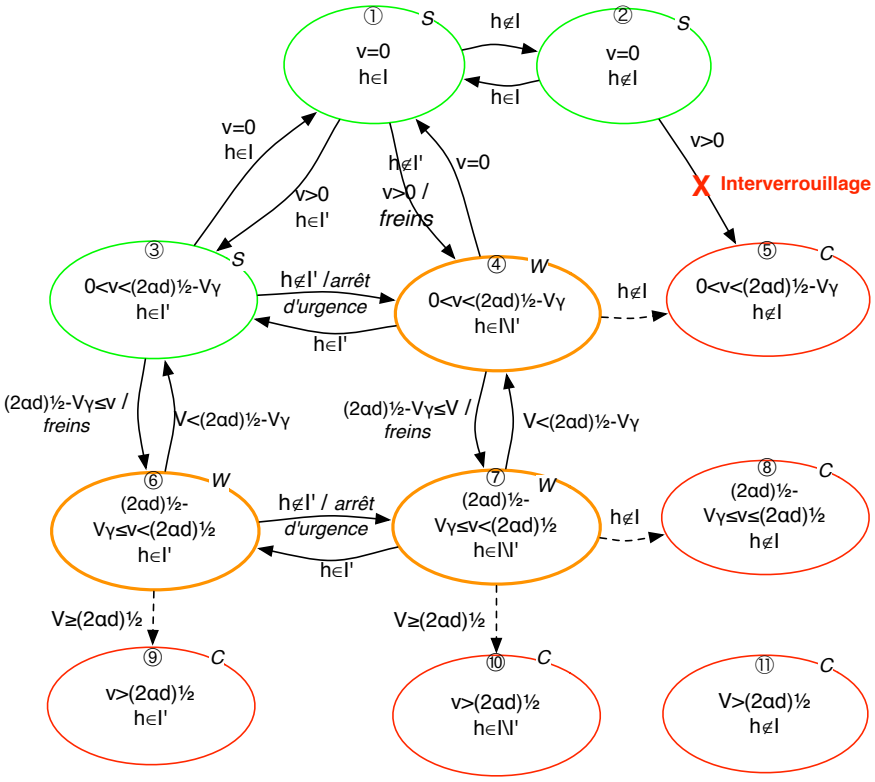


FIGURE 4.12 – Graphe de région résultant de la composition

– Il n'y pas de transition qui enclenche deux actions de sécurité simultanément.

– La transition de la région d'alerte (4) à la région d'alerte (7) enclenche les *freins* alors qu'une autre action, en l'occurrence *l'arrêt d'urgence*, a été précédemment enclenchée. Cela ne représente pas un cas d'incohérence car les deux actions ne sont pas contradictoires. Néanmoins, l'expert du système devra déterminer si engager les freins alors que l'arrêt d'urgence est enclenché ne provoque pas de dommage sur le système.

– La transition de la région d'alerte (6) à la région d'alerte (7) enclenche *l'arrêt d'urgence* alors qu'une autre action, en l'occurrence *les freins*, a été précédemment enclenchée. Là aussi, l'expert du système devra vérifier si cette succession d'actions de sécurité n'est pas dommageable au système.

– La transition de la région (2) à la région (5) est inhibée par un interverrouillage ; c'est-à-dire que le déambulateur ne peut pas déambuler si la hauteur des poignées est en dehors de l'intervalle requis $I$.

– Le choix des marges de sécurité et des mécanismes de recouvrement devrait permettre une réduction (en dessous du seuil de tolérance) soit de la probabilité d'occurrence de la situation catastrophique soit de sa gravité. Ces mécanismes de prévention et de protection devraient rendre les transitions des régions d'alerte

vers les régions catastrophiques improbables (illustrées en pointillées dans le graphe).

– Les transitions induisant le changement simultané des valeurs des deux variables $v$ et $h$ sont omises (par exemple les transitions entre la région (3) et (7)). Nous considérons que de telles transitions se traduisent par deux transitions successives correspondant au changement d'une seule valeur à la fois. Par exemple la transition de la région (3) à la région (7) représente la transition de la région (3) vers la région (6) suivi par la transition de la région (6) vers la région (7).

## 4.4 Conclusion

Nous avons proposé dans ce chapitre l'application de notre approche au robot déambulateur MIRAS. L'analyse de risque HazOp/UML précédemment effectuée au LAAS-CNRS a servi de base pour notre travail. La spécification de règles de sécurité à partir de l'analyse est un processus fastidieux. En effet, l'analyse de risque étant une approche non formelle, effectuée par un humain, certaines déviations peuvent être incompréhensibles et donc inutilisables. Aussi, passer des déviations exprimées en langage naturel à des contraintes de sécurité exprimées en logique formelle peut s'avérer difficile. Une connaissance détaillée du système peut être nécessaire dans certains cas.

Néanmoins, l'application de l'approche proposée à deux cas d'utilisation a permis de spécifier 32 contraintes de sécurité dont 13 ont abouti à des règles de sécurité. Les 19 contraintes restantes doivent être traitées par d'autres moyens de réduction de risque, autre que la surveillance par un moniteur de sécurité indépendant. En effet, la méthode a mis en évidence que certaines contraintes de sécurité ne peuvent pas être traitées pour générer des règles de sécurité mais signalent la nécessité d'une haute fiabilité des mécanismes d'observation. De plus, d'autres contraintes de sécurité pourraient donner lieu à des règles de sécurité si les mécanismes d'observation nécessaires sont implémentés au sein du système dans les versions ultérieures. La méthodologie devra être appliquée à plus de cas d'utilisation afin de statuer sur sa robustesse et en déceler les limites. Automatiser la composition de graphe serait un plus majeur afin de modéliser le système sous l'ensemble des invariants de sécurité et déceler, éventuellement, les actions concomitantes incohérentes.

## Ce qu'il faut retenir

1. L'application de l'approche proposée à l'analyse de risque de deux cas d'utilisation a permis de spécifier 32 contraintes de sécurité.
2. 13 règles de sécurité formelles ont pu être exprimées à partir des 32 contraintes informelles.
3. Des moyens de réduction de risque on été proposés pour les 19 contraintes restantes. En effet, la méthodologie a permis de mettre en évidence la nécessité d'implémenter la redondance de certains capteurs ainsi que la nécessité de nouveaux moyens d'observation.

4. La composition des graphes n'a détecté aucune incohérence des actions de sécurité pour l'exemple étudié.

# Chapitre 5

# Conclusion et Perspectives

Les progrès technologiques en matière de mécanismes décisionnels font que les systèmes autonomes sont de plus en plus performants et feront certainement partie intégrale de notre vie d'ici quelques années à condition que ces systèmes soient suffisamment sûrs pour ne pas causer des dommages à leurs utilisateurs ou à leur environnement. Ceci explique l'intérêt de la communauté scientifique à rendre de tels systèmes sûrs. La sûreté de fonctionnement offre une multitude d'outils afin de développer des systèmes sûrs, depuis la phase de spécification jusqu'à la phase opérationnelle. Notre contribution se situe au niveau du processus d'élicitation et de spécification de contraintes de sécurité à vérifier par un moniteur de sécurité indépendant.

Dans ce chapitre, nous allons récapituler la démarche suivie ainsi que les leçons apprises et nous dressons une liste non exhaustive des axes de recherche qui pourraient faire suite à ces travaux.

## 5.1 Démarche globale

Nos travaux de recherche s'articulent autour des systèmes autonomes et de la façon dont nous pouvons assurer leur mise en œuvre de façon sûre, du point de vue de la sécurité. Les systèmes autonomes, de part leur nature, doivent faire face à des menaces spécifiques. De plus, les mécanismes décisionnels, cœur des systèmes autonomes, sont particulièrement vulnérables et peuvent être à l'origine d'un certain nombre de fautes. Il est donc essentiel de développer des méthodes pour assurer la sécurité de ces systèmes, sous peine de voir leur utilisation limitée à des fonctionnalités non critiques.

Notre étude de l'état de l'art nous a conforté dans l'idée qu'il est nécessaire de développer des méthodes de vérification en ligne spécifiques aux systèmes autonomes pour pallier les fautes de développement résiduelles et les situations adverses imprévues. En effet, les méthodes hors ligne telles que la vérification basée modèle, la preuve de théorème et le test ont été éprouvées et restent incontournables pour l'élimination de fautes avant la mise en œuvre des systèmes. Néanmoins, ce sont des méthodes insuffisantes pour les systèmes autonomes et spécialement les mécanismes décisionnels. La vérification en ligne est apparue comme un complément indispensable aux méthodes hors ligne existantes et permet d'assurer la sécurité malgré la présence de fautes. Parmi

ces méthodes, de nombreux travaux [Sch95, LS09, GP10] et normes [IEC10] attestent de la nécessité d'un dispositif indépendant pour la surveillance du système lors de la phase d'exécution.

La surveillance en ligne par moniteur de sécurité permet de faire face aux particularités des systèmes autonomes (grand espace d'états, environnement incertain, etc.) en surveillant l'exécution courante du système et en vérifiant que celle-ci ne viole pas un certain nombre de propriétés de sécurité. De plus, elle permet d'engager des actions de recouvrement afin d'empêcher que le système n'atteigne un état catastrophique. Nous avons étudié de près différents travaux implémentant des dispositifs de sécurité. Ces travaux portent souvent sur la vérification et la mise en vigueur des propriétés de sécurité mais assez peu sur leur spécification. A l'issue de cette étude, nous avons mis en évidence plusieurs types de moniteurs de sécurité, leurs particularités vis-à-vis de l'observation et de la réaction au sein du système surveillé, l'étendue de leur indépendance par rapport à ce dernier, etc. Notre attention s'est portée tout particulièrement sur les propriétés de sécurité qui doivent être respectées tout au long de l'exécution et qui permettent de s'assurer que le système reste dans un état sûr. Nous avons remarqué que la problématique de définition de propriétés de sécurité n'a été jusqu'alors que très peu étudiée. En effet, à notre connaissance, il n'existe pas de travaux visant à définir une méthode claire de spécification de propriétés de sécurité vérifiables en ligne par un moniteur de sécurité. Nous avons donc décidé de nous focaliser sur la définition d'une telle méthode.

Nous avons proposé, dans un premier temps, de nous baser sur une méthode d'analyse de risque afin d'extraire des contraintes de sécurité [MMGP$^+$12]. Notre choix s'est porté sur l'analyse HazOp/UML et a été motivé par le fait que cette méthode permet de découvrir les comportements anormaux potentiels du système surveillé en se basant sur un modèle du système qui peut être défini dès les phases initiales du développement. Il s'agit d'une méthode systématique par laquelle des mots guides sont appliqués aux différents attributs des modèles. Nous avons utilisé les tables de déviations HazOp/UML pour extraire des contraintes de sécurité. Parmi les différentes informations portées dans ces tables pour chaque déviation, il s'avère que, dans les cas étudiés, seulement la cellule définissant la déviation contenait des informations à même de conduire à une contrainte de sécurité exploitable. D'autres cellules, analysant les effets de la déviation sur le cas d'utilisation et sur le système, permettent aussi d'exprimer des contraintes de sécurité mais, dans les cas étudiés, celles-ci sont de trop haut niveau pour être exploitables dans un moniteur de sécurité. De plus, nous avons identifié deux autres cas où les définitions des déviations ne peuvent être exploitables pour spécifier les contraintes recherchées : lorsque l'observation des variables pertinentes à la sécurité composant la contrainte de sécurité n'est pas possible et lorsque la contrainte de sécurité exprime une exigence d'intégrité.

Dans un second temps, nous avons proposé un processus de bout en bout qui spécifie des conditions de déclenchement de sécurité et des interverrouillages à partir des contraintes de sécurité issues de l'étape précédente [MMBG$^+$12]. Nous avons, tout d'abord, défini et formalisé de façon claire les concepts relatifs à la surveillance en ligne. Ensuite, nous avons proposé de spécifier les états dangereux que le système pourrait atteindre afin de réagir et lui permettre de revenir dans un état sûr avant d'atteindre un état catastrophique. Pour cela, nous avons proposé des méthodes mathématiques

qui permettent de prouver l'existence d'une marge de sécurité (et par conséquent la définition des états dangereux) et de la calculer. La méthode est générale dans le sens qu'elle s'applique à des variables de sécurité de types différents (réels, entiers, booléens, etc.) dans un cadre unifié. Pour finir, nous avons présenté une méthode qui permet de déceler l'existence d'actions de sécurité concomitantes afin qu'elles soient traitées en amont de la phase opérationnelle.

Dans le but de valider la démarche proposée, nous l'avons appliquée à un déambulateur robotisé développé au sein du projet MIRAS. Tout d'abord, un ensemble de contraintes de sécurité ont été spécifiées à partir de l'analyse HazOp/UML de deux cas d'utilisation du robot. L'application a permis de valider les cas, identifiés à la première étape de nos travaux, où les déviations ne peuvent pas servir à spécifier des contraintes de sécurité. Nous avons éprouvé avec succès les méthodes mathématiques proposés afin de calculer la marge de sécurité, et cela sur des exemples de natures diverses. Un ensemble d'états dangereux a pu être défini, et nous avons étudié différentes actions de sécurité possibles pour remettre le système dans un état sûr.

## 5.2 Leçons apprises

La méthodologie présentée dans ce manuscrit permet de définir un moniteur pour vérifier la cohérence de l'exécution du système surveillé par rapport à un ensemble de contraintes de sécurité. La performance de cette surveillance repose, entre autre, sur la complétude de l'ensemble des contraintes surveillées. Notre méthode s'appuie sur une analyse de risque qui génère de façon systématique une liste de déviations possibles. Par conséquent, la complétude des contraintes de sécurité repose sur celle des déviations produites par l'analyse de risque. Bien que HazOp/UML soit une méthode systématique permettant d'explorer des déviations de tous les attributs des modèles, elle reste bien évidemment impuissante par rapport à des interactions imprévues ou non-modélisées. Il parait clair que la surveillance en ligne ne se substitue pas à d'autres méthodes de la sûreté de fonctionnement, et notamment l'évitement et la suppression des fautes. Il est nécessaire que toutes ces méthodes soient exploitées de façon complémentaire afin de couvrir le plus de fautes possibles.

L'application de la méthodologie au robot MIRAS a permis de mettre en évidence toute la difficulté de passer d'une méthode non formelle, effectuée par des humains (qui est HazOp/UML) à des règles de sécurité formelles implémentables au sein d'un moniteur de sécurité. Concernant les deux cas d'utilisation étudiés, sur les 32 déviations étudiées, 13 règles de sécurité ont pu être spécifiées. Les 19 déviations restantes représentent soit le cas où les variables pertinentes à la sécurité ne sont pas observables soit le cas où les contraintes de sécurité issues des déviations expriment une exigence d'intégrité. Ceci représente un taux encourageant qui pourrait être amélioré dans le cas où les recommandations concernant l'implémentation des mécanismes de perception supplémentaires sont appliquées.

## 5.3 Perspectives

Nos travaux de recherche s'inscrivent dans le contexte de la vérification en ligne par moniteur de sécurité. Lors de ces travaux, nous avons identifié la possibilité de les étendre et les compléter des manières suivantes :

1. Il serait interessant d'appliquer la démarche proposée à d'autres analyses de risque HazOp/UML de systèmes autonomes afin de tester la complétude de la méthode de spécification de contraintes de sécurité à partir des déviations. Cela permettrait d'éprouver les concepts définis ainsi que la méthode de vérification et de calcul des marges de sécurité.

2. L'approche proposée pourrait être partiellement automatisée. En effet, la génération du graphe de région à partir d'un invariant de sécurité, l'identification des régions critiques à partitionner, le test de faisabilité d'effectuer une partition, et la génération du nouveau graphe issu de l'introduction de l'état d'alerte sont autant d'étapes qui peuvent être automatisées au sein d'un outil. Ceci permettrait de traiter efficacement et rapidement l'ensemble des contraintes de sécurité identifiées.

3. Dans nos travaux, nous nous sommes intéressés à la vérification de la cohérence des actions de sécurité concomitantes. La méthodologie proposée se base sur la composition des graphes résultant de chaque condition de déclenchement de sécurité. Cette composition pourrait être, elle aussi, automatisée, ce qui permettrait de composer tous les graphes issus de toutes les conditions de déclenchement de sécurité d'un cas d'utilisation. Il serait alors possible d'étudier la faisabilité d'une telle méthode avec un grand nombre d'états du graphe résultant ainsi que l'apport vis-à-vis de la détection d'actions exécutées simultanément. Toujours concernant les actions de sécurité, nous avons choisi de les modéliser sur les graphes de région en utilisant des machines de Mealy qui permettent d'associer à la garde de la transition, une action correspondante. Néanmoins, ce type de graphe n'est pas assez expressif dans le sens où il n'offre pas la possibilité de modéliser les durées des actions de sécurité enclenchées. Un choix de modèle suffisamment expressif permettrait d'étudier plus finement la concomitance éventuelle d'actions de sécurité multiples.

4. Il parait essentiel d'implémenter un moniteur de sécurité mettant en œuvre les règles de sécurité spécifiées à l'aide de notre processus. Ceci permettrait de valider l'approche complète allant de la spécification des règles jusqu'à leur vérification, et de mesurer l'impact d'un tel dispositif sur les performances du système. De plus, des procédures de test pourraient être définies afin de tester la couverture des règles de sécurité spécifiées.

5. La méthodologie présentée permet de définir une condition de déclenchement de sécurité pour chaque invariant de sécurité. Une fois cette condition valuée à vraie, elle enclenche une action de sécurité. Il est possible de définir un ensemble

de conditions de déclenchement de sécurité pour le même invariant de sécurité. La définition de conditions de déclenchement multiples permet d'envisager des paliers où des actions de sécurité différentes peuvent être enclenchées. Une telle approche permettrait de définir des actions de sécurité de plus en plus répressives à mesure que le système s'approche de l'état catastrophique. L'approche pourrait accroître la disponibilité du système en n'appliquant les actions les plus répressives qu'en ultime recours.

6. Dans notre approche, les règles de sécurité sont définies par rapport aux cas d'utilisation. Chaque cas d'utilisation possède un ensemble de règles de sécurité à vérifier. Il serait intéressant de pouvoir regrouper les règles obtenus au sein de mode fonctionnels. En effet, il existe des travaux sur des modes de ce type, appelés *mode de sécurité*, où les auteurs ont proposé un ensemble de concepts relatifs à cela [GPBB08]. Ils définissent un mode de sécurité par l'ensemble des intervalles ou valeurs que peut prendre chaque variable pertinente à la sécurité. Ce modèle de mise en œuvre permettrait peut-être de gérer de façon plus intuitive les problèmes liés aux changements de modes fonctionnels.

7. Dans notre approche, le calcul des marges de sécurité se fait sur des variables pertinentes à la sécurité, donnant lieu à des conditions de déclenchement de sécurité *statiques*. Il nous parait intéressant de pouvoir spécifier de propriétés de sécurité *dynamiques* telles que des propriétés temporelles ou des propriétés liées aux séquencements d'événements (conditions sur les changements de modes par exemple). Ceci permettrait de couvrir un plus grand nombre de propriétés de sécurité à respecter lors de l'exécution du système et donc avoir une meilleure couverture des situations dangereuses pour la sécurité.

8. Une fois les règles de sécurité et les interverrouillages définis, il peut exister de nombreuses implémentations possibles au sein de l'architecture d'un système autonome. Par exemple, une règle de sécurité pourrait être implémentée dans un dispositif de sécurité indépendant tel que c'est étudié dans cette thèse, mais aussi au sein de l'architecture du système autonome, au niveau de la couche décisionnelle, ou de la couche fonctionnelle, ou entre ces deux couches, etc. L'ensemble des possibilités doit être étudié et des méthodes établies pour choisir où implémenter ces points de surveillance, notamment pour que les règles de sécurité implémentées à différents niveaux de l'architecture soient cohérentes.

# Bibliographie


[ACF+98] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An Architecture for Autonomy. *The International Journal of Robotics Research*, 17(4) :315–337, 1998.

[ACMR03] K. Altisen, A. Clodic, F. Maraninchi, and E. Rutten. Using controller-synthesis techniques to build property-enforcing layers. In *European Symposium on Programming, ESOP 2003*, pages 174–188. Princeton University Press, 2003.

[ALRL04] A. Avižienis, J. C. Laprie, B. Randell, and C. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, 1 :11–33, 2004.

[BAA+06] O. Bridal, M. Amann, M. Auerswald, J. Edén, M. Graniou, C. Hellwig, B. Josko, T. Kimmeskamp, A. Kingston, R. Krzemien, M. Leeman, O. Lundkvist, A. Piovesan, C. Scheidler, P. Tiplady, and D. Ward. Electronic architecture and system engineering for integrated safety systems. Guidelines for establishing dependability requirements and performing hazard analysis. Technical report, Information Society, European Commission, Nov 2006.

[BBGP07] E. Baudin, J.P. Blanquart, J. Guiochet, and D. Powell. Independant safety systems for autonomy. Technical Report 07710, LAAS CNRS, 2007.

[BBS06] A. Basu, M. Bozga, and J. Sifakis. Modeling heterogeneous real-time components in BIP. In *Fourth IEEE International Conference on Software Engineering and Formal Methods*, SEFM '06, pages 3–12, Washington, DC, USA, 2006. IEEE Computer Society.

[BDSG+10] S. Bensalem, L. De Silva, M. Gallien, F. Ingrand, and R. Yan. 'Rock Solid' software : A verifiable and correct-by-construction controller for rover and spacecraft functional level. In *i-SAIRAS 2010, The 10th International Symposium on Artificial Intelligence, Robotics and Automation in Space*, Sapporo, Japan, 2010.

[BGR+96] D. E. Bernard, E. B. Gamble, N. F. Rouquette, B. Smith, Y. W. Tung, N. Muscettola, G.A. Dorias, B. Kanefsky, J. Kurien, W. Millar, P. Nayal, K. Rajan, and W. Taylor. Remote agent experiment DS1 technology validation report. Technical report, Ames Research Center and JPL, NASA, 1996.

[CGP99] E.M. Clarke, O. Grumberg, and D.A. Peled. Model checking. Technical report, The MIT Press, 1999.

[Chu11] H.N. Chu. *Test and Evaluation of the Robustness of the Functional Layer of an Autonomous Robot*. Doctorat de l'université de Toulouse. Institut National Polytechnique de Toulouse, Sep 2011.

[EI99] C. A. Ericson II. Fault tree analysis - a history. In *Erb17th International System Safety Conference*, Seattle, Washington, 1999. The Boeing Company.

[Erb89] A. Erb. Safety measures of the electronic interlocking system Elektra. In *IFAC Workshop, SAFECOMP '89*, pages 49–52. Pergamon Press, 1989.

[FD00] J. Fox and S. Das. *Safe and sound - Artificial Intelligence in Hazardous Applications*. AAAI Press - The MIT Press, 2000.

[FHC97] S. Fleury, M. Herrb, and R. Chatila. GenoM : a tool for the specification and the implementation of operating modules in a distributed robot architecture. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS '97*, volume 2, pages 842 –849, sep 1997.

[Fir04] D. Firesmith. Engineering safety requirements, safety constraints, and safety-critical requirements. In *Journal of Object technology*, 2004.

[Fox01] J. Fox. Designing Safety into Medical Decisions and Clinical Processes. In *International Conference on Computer Safety, Reliability and Security*, volume 1, 2001.

[FS99] M. S. Feather and B. Smith. Automatic generation of test oracles-from pilot studies to application. *26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*, 0 :63, 1999.

[Gat97] E. Gat. On three-layer architectures. In D. Kortenkamp, R.P. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*, pages 195–210. MIT/AAAI Press, 1997.

[GMGP10] J. Guiochet, D. Martin-Guillerez, and D. Powell. Experience with model-based user-centered risk assessment for service robots. In *IEEE 12th International Symposium on High-Assurance Systems Engineering*, HASE '10, pages 104–113, Washington, DC, USA, 2010. IEEE Computer Society.

[GMTB07] J. Guiochet, G. Motet, B. Tondu, and C. Baron. Sécurité des systèmes de la robotique médicale. *Techniques de l'ingénieur*, Sécurité et gestion des risques :1 –16, 2007.

[GP06] J. Guiochet and D. Powell. Étude et analyse de systèmes indépendants de sécurité-innocuité de type *safety bag*. Technical Report 05551, LAAS CNRS, 2006.

[GP10] A. Goodloe and L. Pike. Monitoring distributed real-time systems : A survey and future directions. Technical report, NASA Aviation Safety Program Office, 2010.

[GPBB08] J. Guiochet, D. Powell, E. Baudin, and J.P. Blanquart. Online safety monitoring using safety modes. In *6th IARP - IEEE/RAS - EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments*, Pasadena, CA, USA, 2008.

[HSF+09] S. Haddadin, M. Suppa, S. Fuchs, T. Bodenmüller, A. Albu-Schäffer, and G. Hirzinger. Towards the robotic Co-Worker. In *14th International Symposium on Robotics Research, ISSR '09*, Lucerne, Switzerland, sep 2009.

[Hua04] H. M. Huang. Autonomy Levels For Unmanned Systems (ALFUS) Framework. *Numéro NIST Special Publication 1011*, 2004.

[IEC01] IEC61882. Hazard and operability studies (HAZOP studies) : Application Guide. International Electrotechnical Commission, 2001.

[IEC10] IEC61508-7. Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, Part 7 : Overview of techniques and measures. International Organization for Standardization and International Electrotechnical Commission, 2010.

[IEE05] IEEE. Standard for software verification and validation, std 1012-2004. Institute of Electrical and Electronics Engineers IEEE, Revision of IEEE Std 1012-1998, 2005.

[IEE08] IEEE. Standard for software and system test documentation. *IEEE Std 829-2008*, pages 1 –118, 2008.

[IG99] ISO/IEC-Guide51. Safety aspects - Guidelines for their inclusion in standards. International Organization for Standardization and International Electrotechnical Commission, 1999.

[IP02] F. Ingrand and F. Py. Online execution control checking for autonomous systems. In *7th International Conference on Intelligent Autonomous Systems*, Marina del Rey, California, USA, 2002.

[Kle91] P. Klein. The safety bag expert system in the electronic railway interlocking system Elektra. *Journal of Expert Systems with Applications*, 3(4) :499–560, 1991.

[LAB+96] J.C. Laprie, J. Arlat, J.P. Blanquart, A. Costes, Y. Crouzet, Y. Deswarte, J.C. Fabre, H. Guillermain, M. Kaâniche, K. Kanoun, C. Mazet, D. Powell, C. Rabéjac, and P. Thévenod. *Guide de la Sûreté de Fonctionnement*. Cépaduès éditions, 1996.

[LCI+04] B. Lussier, R. Chatila, F. Ingrand, M.O. Killijian, and D. Powell. On Fault Tolerance and Robustness in Autonomous Systems. In *3rd IARP-IEEE/RAS-EURON Joint Workshop on Technical Challenges for Dependable Robots in Human Environments*, 2004.

[Lev91] N. G. Leveson. Software safety in embedded computer systems. *Communications of the ACM Magazine*, 34(2) :34 –46, feb 1991.

[LS09] M. Leucker and C. Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 78(5) :293 – 303, 2009.

[MGGP10] D. Martin-Guillerez, J. Guiochet, and D. Powell. Experience with a model-based safety analysis process for an autonomous service robot. In *IARP Workshop on Technical Challenges for Dependable Robots in Human Environments (DRHE 2010), Toulouse, France*, pages 1–8, 2010.

[MGGPZ10] D. Martin-Guillerez, J. Guiochet, D. Powell, and C. Zanon. A UML-based method for risk analysis of human-robot interactions. In *2nd International Workshop on Software Engineering for Resilient Systems, UK*. ACM, 2010.

[MMBG+12] A. Mekki Mokhtar, J.P. Blanquart, J. Guiochet, D. Powell, and M. Roy. Safety trigger conditions for critical autonomous systems. In *18th IEEE Pacific Rim International Symposium on Dependable Computing*, Niigata, Japan, 2012. IEEE Computer Society.

[MMGP+12] Amina Mekki Mokhtar, Jérémie Guiochet, David Powell, Jean-Paul Blanquart, and Matthieu Roy. Elicitation of executable safety rules for critical autonomous systems. In *Embedded Real Time Software and Systems (ERTS2), Toulouse, France*, 2012.

[MNPW98] N. Muscettola, P.P. Nayak, B. Pell, and B.C. Williams. Remote agent : to boldly go where no AI system has gone before. *Artificial Intelligence*, 103(1-2) :5 – 47, 1998.

[MP05] T. Menzies and C. Pecheur. Verification and validation and artificial intelligence. In M. Zelkowitz, editor, *Advances in Computers*, volume 65 of *Advances in Computers*, pages 153 – 201. Elsevier, 2005.

[MP09] B.S. Medikonda and S.R. Panchumarthy. An approach to modeling software safety in safety-critical systems. *Journal of Computer Science*, 5 :311–322, 2009.

[OMG03] OMG. 2nd revised submission to OMG RFP ad/00-09-02 - Unified Modeling Language : Superstructure - version 2.0. Object Management Group, 2003.

[PI04a] F. Py and F. Ingrand. Dependable execution control for autonomous robots. In *International Conference IEEE/RSJ on Intelligent Robots and Systems, IROS '04*, volume 2, pages 1136 – 1141 vol.2, sept.-2 oct. 2004.

[PI04b] F. Py and F. Ingrand. Real-time execution control for autonomous systems. In *2nd European Congress on Embedded Real Time Software and Systems ERTS2 '04*, pages 21–23, Toulouse, France, 2004.

[PNW11] L. Pike, S. Niller, and N. Wegmann. Runtime verification for ultra-critical systems. In *2nd International Conference on Runtime Verification*, San Francisco, California, USA, 2011.

[PRDS07] D. Pinard, S. Reynaud, P. Delpy, and S.E. Strandmoe. Accurate and autonomous navigation for the ATV. *Aerospace Science and Technology*, 11(6) :490 – 498, 2007.

[PS00] C. Pace and D. Seward. A safety integrated architecture for an autonomous safety excavator. In *International Symposium on Automation and Robotics in Construction*, 2000.

[PTF02] D. Powell and P. Thévenod-Fosse. Dependability issues in ai-based autonomous systems for space applications. In *2nd IARP-IEEE/RAS joint workshop on Technical Challenge for Dependable Robots in Human Environments*, pages 163–177, October 2002.

[RBQ96] C. Rabejac, J.-P. Blanquart, and J.-P. Queille. Executable assertions and timed traces for on-line software error detection. In *Proceedings of Annual Symposium on Fault Tolerant Computing*, pages 138–147. IEEE Computer Society Press, 1996.

[RRAA04] S. Roderick, B. Roberts, E. Atkins, and D. Akin. The Ranger robotic satellite servicer and its autonomous software-based safety system. *IEEE Intelligent Systems Journal*, 19 :12–19, 2004.

[RW89] P.J.G. Ramadge and W.M. Wonham. The control of discrete event systems. *Proceedings of the IEEE*, 77(1) :81 –98, 1989.

[Sai77] S.H. Saib. Executable Assertions - An Aid To Reliable Software. In *11th Asilomar Conference on Circuits, Systems and Computers. Conference Record.*, pages 277–281. IEEE, 1977.

[Sch95] B.A. Schroeder. On-line monitoring : a tutorial. *IEEE Computer Journal*, 28(6) :72 –78, June 1995.

[SV02] M. Stamatelatos and W. Vesely. Fault tree handbook with aerospace applicaions. Technical report, NASA Headquarters Office of Safety and Mission Assurance, NASA Langley Research Center, August, 2002.

[The86] N. Theuretzbacher. VOTRICS : Voting Triple Modular Computing System. In *16th IEEE symposium on fault-tolerant computing, FTCS'86*, pages 144–150, Austria, 1986. IEEE Computer Society.