Etude et analyse de différents dispositifs externes de sécurité-innocuité de type safety bag

Jérémie Guiochet, David Powell

Étude et analyse de différents dispositifs externes de sécurité-innocuité de type *safety bag*

Rapport interne LAAS CNRS n° 05551

Jérémie Guiochet David Powell

21 novembre 2005

Avant propos

Ce document a été rédigé dans le cadre d'une étude préliminaire d'une thèse commençant en octobre 2005, sur la *définition de contraintes de sécurité-innocuité vérifiables en ligne pour systèmes autonomes critiques*, en collaboration avec EADS-Astrium.

Table des matières

1 Introduction

L'utilisation élargie de systèmes informatisés conduit à une substitution de l'homme par la machine dans de nombreuses applications. Que ce soit pour des tâches difficiles, impossibles, ou dangereuses pour l'homme, de plus en plus de responsabilités sont déléguées à ces systèmes. À titre d'exemple, l'exploration de l'univers est aujourd'hui réalisée grâce à des sondes téléguidées par l'homme depuis la terre, mais dont les messages ne nous arrivent qu'avec un retard très important. Pour étendre la portée de ces missions d'exploration, les futurs systèmes doivent être capables d'évoluer sans l'intervention de l'homme et notamment être capables de réagir seuls face à des situations imprévues. De tels dispositifs, appelés *systèmes autonomes*, sont étudiés et développés depuis de nombreuses années en robotique, notamment grâce à des techniques d'intelligence artificielle. Leur utilisation semble aujourd'hui s'étendre à de nombreux autres domaines.

Si l'étendue des bénéfices est importante il n'en reste pas moins que l'autonomie est encore une fonctionnalité rarement utilisée dans des application critiques telles que le médical, le spatial, ou le transport. En effet, l'utilisation de ces systèmes dans ces applications pose le problème de leur sûreté de fonctionnement, notamment vis-à-vis des conséquences potentiellement désastreuses de leur défaillance (humaines et matérielles). Deux études de faisabilité [CM03, TTP+03] ne donnent pas des résultats très encourageants de ce point de vue : la première étude, menée pendant deux ans sur treize robots issus de sept modèles différents, donne un MTBF (temps moyen entre deux défaillances) de seulement 8 heures ; la seconde étude, menée pendant cinq mois sur une dizaine de robots autonomes d'un même modèle, donne un MTBF de 4,6 heures. Dans le DARPA Grand Challenge[1] organisé en mars 2004, quinze véhicules terrestres autonomes ont eu pour tâche de parcourir 228 kilomètres dans le désert Mojave ; les deux compétiteurs les plus performants n'ont pas dépassé le douzième kilomètre. Enfin, la norme IEC61508 [IEC01, part.3, clause 7.4.3], donne un exemple du peu de confiance accordée envers les mécanismes décisionnels à base d'intelligence artificielle : l'utilisation de tels mécanismes comme stratégie de tolérance aux fautes est admise au niveau de criticité le plus faible (appelé SIL1) mais déconseillée pour les niveaux d'intégrité correspondant aux fonctions les plus critiques (SIL2 à SIL4).

Ces quelques exemples montrent qu'un travail important doit être réalisé pour réduire les risques de défaillance des systèmes autonomes. On distingue, pour ces systèmes, les classes de menaces suivantes [PTF02] :

- les situations adverses de l'environnement dans lequel le système évolue ;
- les manques de connaissance ou les imprécisions dans la perception de cet environnement ;
- les fautes physiques affectant ses capteurs, actionneurs ou moyens de traitement ;
- les fautes introduites lors de la conception du système, et plus particulièrement, au niveau de son architecture informatique (qui comprend notamment les logiciels de prise de décision).

Lors de la conception d'un système autonome, malgré l'utilisation de techniques

[1] http ://www.darpa.mil/grandchallenge/

d'élimination et de prévention des fautes, il est impossible d'éviter toutes les fautes présentées ci-dessus. Il est donc nécessaire de concevoir des systèmes capables de remplir leur fonction malgré la présence de fautes (notion de fiabilité) et d'éviter les défaillances catastrophiques lorsque le système ne peut remplir sa fonction (notion de sécurité-innocuité) [ALRL04].

C'est dans cette logique d'acceptation des fautes que le rapport SPAAS (Software Product Assurance for Autonomy on-board Spacecraft) [2] conclut sur la nécessité d'avoir un sous-système permettant de vérifier en temps réel des propriétés de sécurité-innocuité du système global. Cette technique de tolérance aux fautes, que nous nommons *safety bag* en référence aux travaux sur Elektra [Kle91], est également définie de cette manière dans la norme IEC 61508 [IEC01, part.7, annexe C.3.4]. Elle fournit une protection, du point de vue de la sécurité, contre l'ensemble des classes de menaces présentées ci-dessus en détectant les situations potentiellement dangereuses afin d'éviter les défaillances catastrophiques. Elle s'appuie sur le principe de la diversification en distinguant deux chaînes de traitement : une chaîne fonctionnelle et une chaîne de contrôle. La première est dédiée à la réalisation des fonctions du système, et la seconde est destinée au contrôle de la première. La chaîne de contrôle est constituée par un ensemble d'assertions exécutables déduites des propriétés de sécurité que le système doit respecter pour éviter les défaillances catastrophiques.

Il est possible de retrouver le concept du *safety bag* dans de nombreux systèmes et dans des domaines variés comme : le ferroviaire (Elektra [Kle91]), le spatial (projet SPAAS [BHL+04]), le médical (*guardian agent* [FD00]) ou la robotique (R2C [PI04a]). Ces travaux de recherche se sont concentrés sur l'architecture même du *safety bag* et sur les choix d'implantation des règles de sécurité (langages de programmation des règles de sécurité et compilateurs de ces règles). En revanche, le processus de détermination et d'expression des règles de sécurité n'a été que très peu traité. Or, la sûreté de fonctionnement globale dépend de l'efficacité de ces règles, donc de la représentativité et de la complétude des hypothèses faites par rapport aux situations réelles. Avant de concevoir un *safety bag*, il est donc fondamental de mener une réflexion sur la méthode de détermination de ces règles de sécurité, et l'impact sur les choix d'architecture du *safety bag*. L'objet de l'étude est de définir une méthode permettant de spécifier puis de concevoir un mécanisme de tolérance aux faute particulier, celui que nous nommerons *safety bag*. Ce rapport, en tant qu'étude préliminaire, se concentre sur les éléments de bibliographie et propose également des orientations de recherche.

Il existe aujourd'hui de nombreuses solutions techniques pour la conception de ces systèmes de sécurité, notamment pour les choix d'architecture, des langages de programmation, des variables d'état surveillées, etc. Le but de ce rapport est de les identifier en se basant sur quelques cas d'études. Pour chaque système étudié le plan suivant sera utilisé :

- **Présentation** : une vue très générale du système étudié est présentée ainsi que le cadre d'étude (industriel/académique, commercialisation, etc.)
- **Exigences de sûreté de fonctionnement** : description des exigences de sûreté de fonctionnement et des types de fautes traitées.

[2] European Space Agency Project : 14898/01/NL/JA : SPAAS (Software Product Assurance for Autonomy on-board Spacecraft)

– **Architecture et fonctionnement** : description de l'architecture du système de contrôle et du *safety bag* intégré, et du fonctionnement dynamique de l'ensemble.
– **Règles de sécurité** : les méthodes de détermination des règles de sécurité contrôlées, et le langage d'expression associé.
– **Résultats et discussions** : les résultats de l'utilisation du système de *safety bag* en terme de fautes détectées et tolérées (fautes couvertes), et discussion sur le système utilisé.

Pour certains systèmes, des sections de ce plan sont absentes, soit par manque d'information, ou plus simplement parce que le sujet n'a pas été abordé dans le cadre du système en question (notamment pour les règles de sécurité).

2 Définitions du *safety bag*

Le dispositif appelé *safety bag* est un système de sécurité-innocuité[3] actif. Il doit être capable de détecter et éventuellement traiter toute évolution du système vers un état à risque. Il est important de noter que pour cette étude nous nous situerons dans le domaine des systèmes de commande et plus particulièrement ceux à sécurité critique. La diversité des réalisations de ce type de systèmes est telle qu'il n'existe aucune catégorisation et a fortiori aucune définition commune ou standardisée du terme *safety bag*. De plus on retrouve les mêmes concepts dans de nombreuses autres terminologies telles que : circuit de contrôle d'un composant autotestable, dispositif externe de sécurité, dispositif indépendant de sécurité, *external monitoring system*, *safety-related subsystem*, *safety kernel*, *safeguard*, *protective system*, *guardian agent*, *safety supervisor*, etc. Il existe également des concepts recoupant ces notions comme celles de *safety barrier* (en relation avec la notion de *defense in depth*), *wrapper*, ou même celle de test d'*oracle*. Pour chacun de ces termes on retrouve les principaux concepts présents dans l'approche du *safety bag* que l'on propose ici. Cependant, afin de ne pas se perdre dans des digressions terminologiques les rapports entre ces définitions ne seront pas détaillés ici.

Notons qu'il existe une définition de *safety bag* dans la norme IEC 61508-7 2000 (C.3.4.), traduite par *dispositif externe de sécurité* dans la version française :

> C.3.4 Dispositif externe de sécurité
> But *: Assurer la protection du logiciel contre les anomalies de spécification et d'implémentation résiduelles susceptibles d'affecter la sécurité.*
> Description *: Un dispositif externe de sécurité est un système de surveillance qui est mis en œuvre à partir d'un ordinateur indépendant selon une spécification différente. Ce dispositif a pour seul objectif de vérifier que l'ordinateur principal exécute des opérations sûres mais pas nécessairement correctes. Il surveille l'ordinateur principal de manière permanente et empêche tout état du système contraire à la sécurité. De plus, en cas de détection d'un état d'ordinateur potentiellement dangereux, il faut que le système soit ramené à l'état sûr soit par le dispositif*

[3]Dans la suite de ce document nous utiliserons le terme de sécurité à la place de sécurité-innocuité (*safety* en anglais), car il n'existe pas ici d'ambiguïté avec la sécurité-immunité (*security* en anglais) que nous n'aborderons pas

externe de sécurité soit par l'ordinateur principal.
Il convient que le matériel et le logiciel de sécurité externes soient classés et qualifiés suivant le SIL approprié.
Référence : Using AI Techniques to Improve Software Safety. Proc. IFAC SAFECOMP 88, Sarlat, France, Pergamon Press, October 1986.

Il est intéressant de noter que les versions en anglais et en français produites par les auteurs de la norme ne diffèrent que par l'appellation de la technique :

C.3.4 Safety bag
Aim *: To protect against residual specification and implementation faults in software which adversely affect safety.* Description *: A safety bag is an external monitor, implemented on an independent computer to a different specification. This safety bag is solely concerned with ensuring that the main computer performs safe, not necessarily correct, actions. The safety bag continuously monitors the main computer. The safety bag prevents the system from entering an unsafe state. In addition, if it detects that the main computer is entering a potentially hazardous state, the system has to be brought back to a safe state either by the safety bag or the main computer. Hardware and software of the safety bag should be classified and qualified according to the appropriate SIL.*
Reference : Using AI Techniques to Improve Software Safety. Proc. IFAC SAFECOMP 88, Sarlat, France, Pergamon Press, October 1986.

3 Elektra

Mots clés : système expert, safety bag, moteur d'inférence, base de connaissances, règles de sécurité, dual-channel, domaine ferroviaire.

3.1 Présentation

Elektra est un système d'aiguillage pour le transport ferroviaire, commercialisé par Alcatel Autriche sous le nom de *Alcatel 6131 LockTrac Electronic Interlocking System* (ELEKTRA). Il existe à ce jour (en 2005) 80 installations en Autriche et 20 en suisse.

Le système de *safety bag* intégré à Elektra est composé d'un canal logique (*logic channel*) et d'un canal de sécurité (*safety channel*). La technique utilisée pour le canal de sécurité est un système expert (base de connaissances et moteur d'inférence) effectuant des vérifications en-ligne (en temps réel) des calculs du canal logique.

Les informations présentées dans cette section proviennent des sources suivantes :

- Le site web d'Alcatel (http ://www.alcatel.fr)
- N. Theuretzbacher. Using AI techniques to improve software safety. In *Proc. IFAC SAFECOMP 86, Sarlat, France*, pages 99–105. Pergamon Press, October 1986 [The86]
- P. Klein. The safety-bag expert system in the electronic railway interlocking system Elektra. *Expert Systems with Applications*, 3 :499–506, 1991 [Kle91]

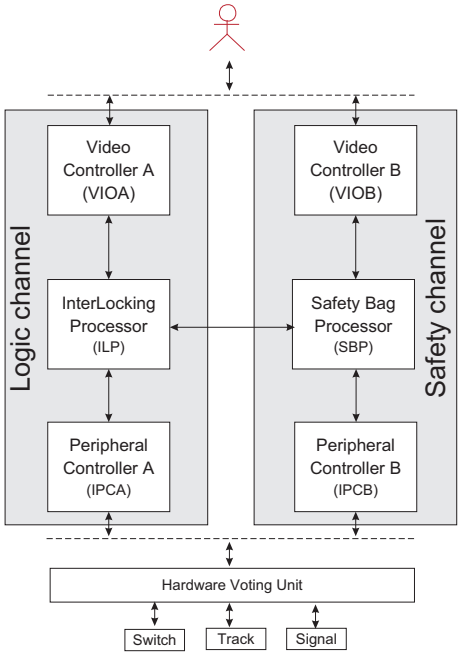

FIG. 1 – Modèle basique d'architecture d'Elektra

- A. Erb. Safety measures of the electronic interlocking system "Elektra". In *Proc. IFAC SAFECOMP 89, Vienna, Austria*, pages 49–52. Pergamon Press, 1989 [Erb89]
- H. Kantz and C. Koza. The Elektra railway signalling-system : Field experience with an actively replicated system with diversity. In *Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing (FTCS)*, pages 453–458. IEEE Publisher, 1995 [KK95]

3.2 Exigences de sûreté de fonctionnement

Dans le contexte de l'application ferroviaire, les auteurs mentionnent la présence d' erreurs potentiellement dangereuses et de situations dangereuses [The86] sans les décrire. Ils s'appuient sur les exigences du domaine ferroviaire, notamment sur les normes CENELEC recommandant moins de 10^{-9} défaillances critiques pour la sécurité par heure, ainsi que sur l'exigence de la *Austrian Federal Railways* qui est d'avoir moins d'une interruption de service en 10 ans [KK95].

Sur ces éléments peu présentés par les auteurs, deux classes de fautes sont identifiées : les fautes de conception et les fautes physiques [Erb89, KK95]. Ils s'attachent ainsi à assurer, malgré ces fautes, la sécurité-innocuité et la fiabilité du système.

3.3 Architecture et fonctionnement

Il existe au sein de l'architecture d'Elektra deux niveaux de redondance :

- le premier niveau est réalisé par un système à deux canaux, un pour l'aspect fonctionnel et l 'autre pour la vérification des sorties du premier (*safety bag*) ;
- le deuxième consiste en l'utilisation d'une couche logicielle appelée VOTRICS (*Voting Triple-Modular Redundancy Computer System*) implantée dans les calculateurs des deux canaux qui met en place un masquage d'erreurs par vote sur trois composants redondants.

L'utilisation de la couche VOTRICS ne concerne pas le *safety bag* et ne sera donc pas présenté ici. La figure 1 propose une version simplifiée de l'architecture d'Elektra. Le fonctionnement suit le diagramme de séquence de la figure 2 (modélisation résumant les descriptions du fonctionnement). Sur ce diagramme, le processeur principal (ILP) élabore une commande et l'envoie au processeur du safety bag (SBP). Ce dernier est composé d'un moteur d'inférence et d'une base de connaissances contenant les règles et une mémoire de travail. Le moteur d'inférence transforme tout d'abord la commande en un GOAL, qui est l'expression logique de l'autorisation de la commande. Puis il lit les données relatives à l'état du système, et sélectionne la ou les règles applicables (avec un système de gestion des conflits de ces règles). L'exécution de la règle consiste ensuite à transmettre la requête ou à la bloquer. Lorsque le calcul est effectué dans les deux processeurs un vote matériel est réalisé. Il consiste à ne pas transmettre la commande calculée par le canal logique et à mettre le système dans un état sûr lorsque les deux canaux de calcul ne sont pas d'accord (comme par exemple mettre les feux à rouge et bloquer tous les trains en attendant que la situation évolue).

3.4 Règles de sécurité

3.4.1 Méthode de détermination des règles de sécurité

Cet aspect fondamental de la conception du *safety bag* n'est que très peu présenté dans les articles. On trouve cependant les trois citations suivantes :

> *The knowledge base of the prototype was extended and refined by determining and demonstrating the system to railway signaling specialists at Alcatel Austria.* [Kle91]
>
> *The rule-based programming paradigm was very well suited to represent the functionality of the safety bag according to the operating requirements.* [Kle91]
>
> *The specification* [of the safety bag channel] *is derived from the operating regulations, e.g., of the Austrian Federal Railways* [KK95]

Il est mentionné dans [Erb89] que pour les premières versions du système sans logiciel (uniquement avec des relais), des analyses de types AMDEC du système électronique ont été effectuée. Mais à partir du moment où les concepteurs ont introduit du logiciel il semble que de telles analyses n'ont pas été réalisées.

On peut en conclure que ces règles ont été implantées grâce à l'expérience dans le ferroviaire des développeurs, la consultation d'experts, et l'utilisation de normes. Les auteurs mentionnent aussi l'importance du langage d'expression de ces règles

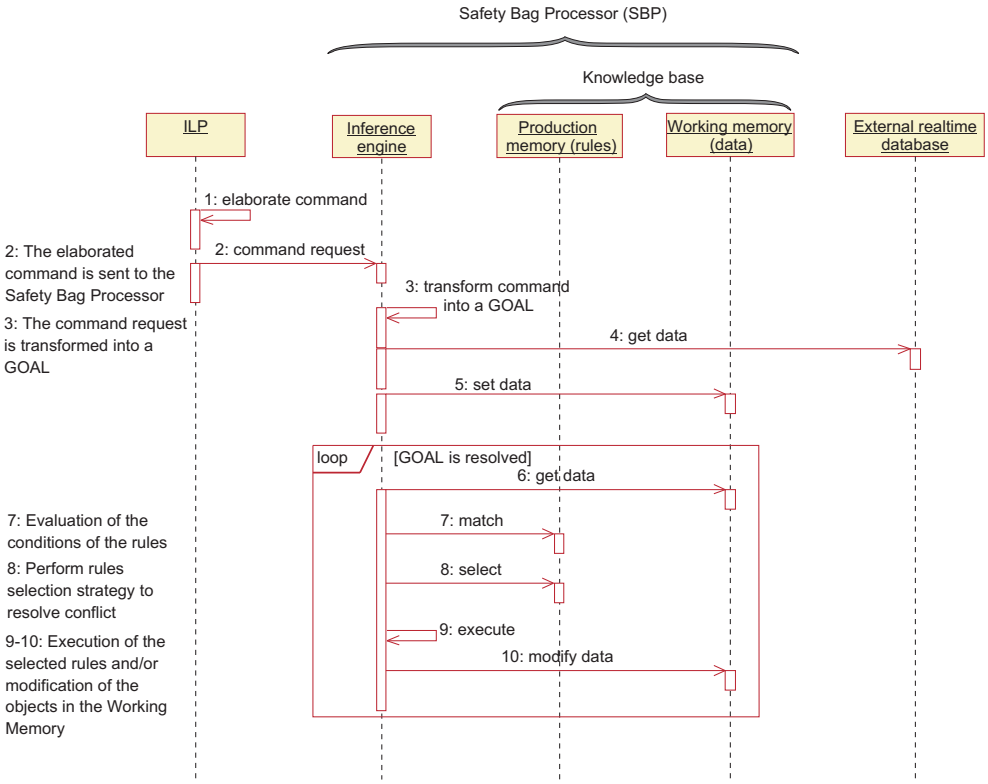

FIG. 2 – Diagramme de séquence illustrant le fonctionnement du *safety bag* d'Elektra

vis à vis des exigences fonctionnelles, sans pour autant entrer dans le détail. Enfin il est à noter que aucune méthode de détermination de ces règles n'est présentée.

3.4.2 Langage d'expression des règles de sécurité

Dans une première version d'Elektra les concepteurs utilisaient des règles décrites dans un langage machine, l'OPS5. Il s'agit d'un langage « orienté règles »utilisé dans les systèmes experts qui permet également de faire du procédural. La figure 3 présente le code d'une règle de sécurité dans ce langage. Pour pouvoir sélectionner cette règle, le programme se base sur la définition des deux objets : un *SWITCH* (position d'un relai de commande d'un aiguillage), et un *GOAL*. La règle est définie par une partie concernant les conditions de test et une deuxième concernant l'expression des actions à réaliser (`modify &1 (STATUS = check_ok) ;`).

Une version ultérieure d'Elektra se base sur un langage plus haut niveau, PAMELA, permettant de définir des règles à un niveau d'abstraction plus élevé. Les articles sur Elektra ne proposent pas d'exemple de règles en PAMELA mais uniquement la traduction en anglais présentée figure 4.

3.5 Conclusions

3.5.1 Résultats de l'utilisation du safety bag dans Elektra

Au vu des références bibliographiques dont nous disposons, les résultats de l'utilisation d'Elektra ne sont présentés que dans un seul article de 1995 [KK95]. Il est important de noter que le *safety bag* a été très utilisé lors des phases préliminaires à la mise en opération du système. Les concepteurs ont en effet définit des phases de mise en place du système global : *integration test phase, installation phase in the station, qualification test phase, pilot operation phase*, et *regular operation phase*. Sans entrer dans le détail de ces phases, on peut noter que depuis le laboratoire d'essai jusqu'à la gare ferroviaire, le système est modifié et affiné. Le *safety bag* a été activé lors des premières phases du développement où des erreurs de conception (comme par exemple des erreurs dans la traduction de la topologie du réseau ferroviaire propre à la station) ont été détectées et corrigées. Toujours d'après cet article, sur tous les systèmes installés et en mode opérationnel en 1995, aucune situation critique n'avait été détectée par les *safety bag*.

3.5.2 Discussion

Une des particularités d'Elektra est d'avoir implanté un système expert qui fonctionne encore aujourd'hui alors que de nombreux projets incluant des technologies de type d'intelligence artificielle n'ont pas aboutis. Ce système semble ainsi s'être affranchi de plusieurs menaces induites par l'utilisation de systèmes experts et notamment :

- Problèmes sur les applications de taille « réelle » :
  - Incomplétude de l'expertise
  - Erreurs dans les règles
  - Inconsistance des règles entre elles
- Comment modéliser/formaliser le sens commun ?

```
Type SWITCH = element
      (
      NUMBER:     integer;    -- number of the switch
      POSITION    :   symbol; -- left, right, undifined
      OCCUPANCY   :   symbol; -- free,occupied, undifined
      MANUAL_MOVE :   symbol; -- allowed, not allowed
      LOCKED :        symbol; -- yes, no
      CLOSE :         symbol; -- not used ,
                                 Used_in_main_route
      ROUTE_NR :      symbol; -- number of the route in
                                 which the switch is
                                 located
      );
```

```
Type GOAL = element
      (
      STATUS      :   symbol; -- safety_check, chekc_ok,
                                 danger
      COMMAND     :   symbol; -- move switch, ...
      COMMAND_ATR :   symbol; -- single_move,
                                 move_in_route,
                                 manual_move, ...
      OBJECT      :   integer;-- number of switch,
                                 signal, ...
      );
```

```
rule MOVE_SWITCH_IN_ROUTE
{
  &1 (GOAL                    -- Left hand-side
                              --  (match conditions),
                              -- the GOAL element controls
                              -- the operation of the
                              -- expert system

      STATUS      =   safety_check; -- Perform safety check
      COMMAND     =   move_switch;  -- of a 'switch_move'
      COMMAND_ATTR=   move_in_route;-- command during the
                                    -- ligne-up of route
      );

  &2  (SWITCH

      NUMBER      =   &1.OBJECT     -- The 'switch' data
      POSITION    <>  undefined;    -- object describes the
      OCCUPANCY   =   free;         -- status of the switch
      MANUAL_MOVE =   not_allowed;  -- to be moved
      LOCKED      =   no;
      CLOSE       =   no;
      );

-->                           -- Right hand-side (action part)
                              -- signal 'safety_check_ok'
                              -- if hte safety conditions
                              -- are met
      modify &1 (STATUS = check_ok);
}
```

FIG. 3 – Déclaration d'un type d'objet décrivant l'état d'un aiguillage (*SWITCH*), d'un objectif *GOAL*, et d'une règle de sécurité pour un aiguillage de voie ferrée. Cette règle contient les vérifications nécessaires avant de commuter un aiguillage après l'élaboration d'une route. Tiré de [The86].

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

FIG. 4 – Règles de sécurité de haut niveau du système Elektra

- Comment extraire/modéliser/formaliser les connaissances des experts ?
- Peut on réduire l'expertise à des règles ?
- Explosion combinatoire : nécessité d'heuristique

Dans le cadre du ferroviaire, mais aussi dans d'autres domaines, l'utilisation de règles provenant d'années d'expertise et de normes permet de s'affranchir de certaines de ces menaces. Cependant, une des limites évidente de cette démarche est l'application à des systèmes innovants, où la quantité d'information et l'expérience ne sont pas suffisants.

Le fonctionnement du *safety bag* repose sur la nécessité d'avoir une connaissance et une représentation à tout instant du système. Pour cela les auteurs mentionnent une base de données temps-réel permettant de reconstruire la topologie du système. Ils ne précisent pas cependant si les deux canaux (ILP et SBP) utilisent cette même base ou s'il s'agit d'une base dédiée à l'un ou l'autre des canaux. Dans une logique de diversité et d'efficacité du safety bag il est pourtant fondamental que les canaux soient entièrement indépendants et surtout au niveau de la représentation du système à l'instant t.

4 Ranger Robotic Satellite Servicer

Mots clés : Robotique, réplication active et passive, variables d'état critiques, domaine spatial

4.1 Présentation

Le Ranger est un système robotique devant permettre le réapprovisionnement en fuel, la réparation et la mise à jour de systèmes spatiaux en orbite. Il est constitué de deux bras à sept degrés de liberté comme le montre la figure 5. Il s'agit d'un

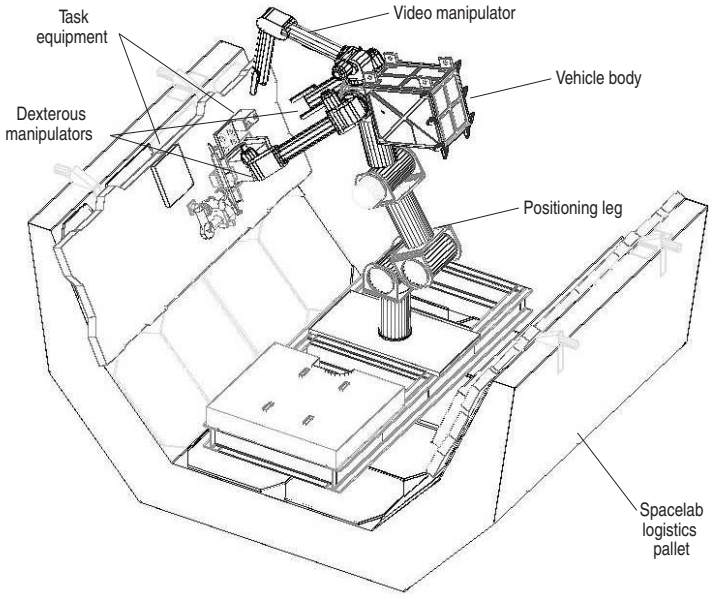

FIG. 5 – Vue globale du Ranger dans sa configuration actuelle

projet de recherche développé à l'université du Maryland (USA) et financé par la NASA. Il y a eu de nombreux tests en laboratoire et sous l'eau (quelques centaines d'heures).

Les informations utilisées pour cette section proviennent de trois sources (mais les publications ont été relativement nombreuses) :

- S. Roderick, B. Roberts, E. Atkins, and D. Akin. The Ranger robotic satellite servicer and its autonomous software-based safety system. *IEEE Intelligent Systems*, 19(5) :12–19, 2004 [RRAA04]
- S. Roderick. Validation of a computer-based hazard control system for a robotic payload on the space shuttle. Master of science, University of Maryland, USA, 2000 [Rod00]
- B. Bon and H. Seraji. Real-time model-based obstacle detection for the NASA ranger telerobot. In *Proc. Internation Conference on Robotics and Automation, Albuquerque, New Mexico*, pages 1580–1587. IEEE Publisher, April 1997 [BS97]

4.2 Exigences de Sûreté de fonctionnement

Les auteurs ont identifié les dommages potentiels suivants :

- dommage physique de la navette spatiale ;
- impossibilité d'un retour sur terre ;
- largage et perte d'un objet non solidaire à la navette ;
- force excessive appliquée à un objet.

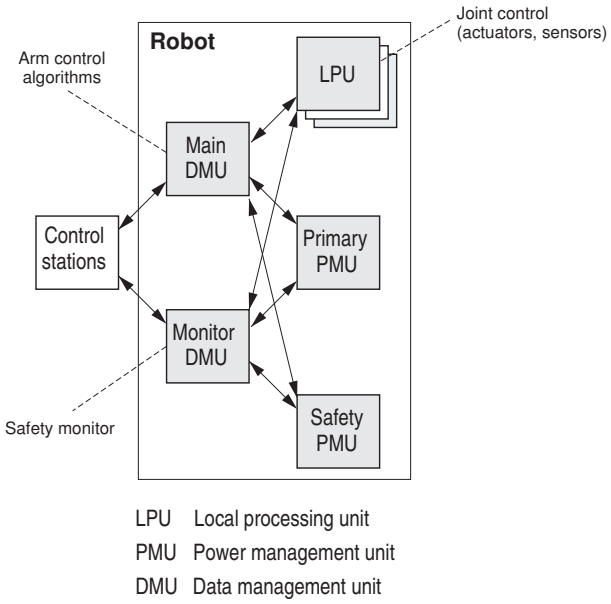

FIG. 6 – Architecture du contrôleur du *Ranger*

Ces dommages proviennent principalement de deux situations dangereuses : un déplacement causé par une commande erronée et un mouvement non contrôlé (avec une commande correcte). Les fautes induisant ces dangers ont été classées en défaillances matérielles, logicielles, de communications ou des opérateurs.

4.3 Architecture et fonctionnement

L'architecture choisie pour tolérer les classes de fautes présentées ci-dessus est celle présentée sur la figure 6. Le système est composé de plusieurs processeurs, dont *Main DMU* et *Monitor DMU*. Le *Main DMU* est en charge du commande du manipulateur mais effectue également une surveillance de la sécurité du système. Le *Monitor DMU* n'élabore aucune commande mais réalise les mêmes calculs de surveillance du systèmes que le processeur principal. Les deux DMU exécutent donc en parallèle les mêmes calculs de sécurité. Cette surveillance se base uniquement sur le contrôle de quatre données de l'état du système pour déterminer si un danger est imminent :

- positions des axes du robot (calcul de zones de sécurité et des distances minimums d'approche du bras robot) ;
- vitesses de déplacement (calcul des vitesses maximums des articulations pour limiter les impacts) ;
- états des outils de préhension (pour éviter tout largage non désiré d'un équipement) ;
- forces exercées sur les objets et couples des axes (calcul des forces maximums applicables au robot lui même et différents équipements) .

Ces quatre contrôles sont uniquement réalisés 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.

Dans la dernière version du Ranger, les composants logiciels en charge de ces vérifications sont identiques sur les deux processeurs, sauf le contrôle de la position qui a été développé de deux manières différentes [BS97][4]. Ces deux processeurs communiquent donc avec le LPU (*Local Processing Unit*) qui est en charge de piloter les axes avec les commandes calculées. Dans le cas nominal les deux processeurs communiquent la même information au LPU, mais en cas de désaccord le LPU ne sélectionne que la commande « la plus sûre »(technique de l'accord passif, ou *passive concurrence* en anglais). Les auteurs donnent, à titre d'exemple, l'état dans lequel doit se trouver le LPU, qui est calculé et transmis au LPU par les deux processeurs *Main* et *Monitor*. Sur la base de trois états possibles, *safe*, *halt*, et *running*, le LPU qui reçoit comme commande *safe* et *running* sélectionne alors l'état *safe*. Cependant, dans les articles à notre disposition les auteurs ne donnent pas d'exemple autres que cette commande de changement d'état.

4.4 Conclusions

L'architecture présentée ici permet de poser le problème de la définition d'un *safety bag*. En effet, certains éléments de la définition donnée dans la section 2 sont présents comme l'indépendance des processeurs ou la mise en état sûr. Cependant, le fait que des algorithmes de sécurité soient implantés dans le processeur principal (*Main DMU*) sur la base des mêmes spécifications que pour le processeur de contrôle (*Monitor DMU*), éloigne ce système de la définition première d'un *safety bag*. De plus pour trois des vérifications les algorithmes sont identiques, ce qui augmente la probabilité de défaillances de mode commun.

Un autre aspect qui différencie le cas du Ranger des autres exemples présentés dans ce rapport, est le fait que le processeur de surveillance n'effectue aucune vérification des commandes calculées par le processeur principal (il n'existe en effet aucune interaction entre le *Main DMU* et le *Monitor DMU*). Le *Monitor DMU* ne surveille donc que l'état global du système commandé par l'intermédiaire de capteurs, et ceci indépendamment des erreurs de commandes du *Main DMU*. Cette différence est encore plus flagrante si l'on se place dans le cadre des systèmes autonomes, où un dispositif comme un *safety bag* devrait pouvoir surveiller les décisions que prend le logiciel et agir en conséquence, et non pas attendre d'en détecter les effets sur le système.

Cependant, ce système permet de tolérer des fautes matérielles du processeur principal puisque le processeur de surveillance permet alors de maintenir le système dans un état non dangereux. Il permet également de tolérer un ensemble de fautes logicielles du processeur principal grâce aux vérifications effectuées à la fois sur ce même processeur, et également sur le processeur de surveillance. De par ces caractéristiques de tolérance aux fautes, ce système se place dans une catégorie de *safety bag* limités à l'observation du système commandé pour la détection de dangers.

[4]Cependant cette référence ne présente qu'un seul algorithme de contrôle de position

5 *Guardian agents*

mots clés : Intelligence artificielle, agent, règles de sécurité, langage de représentation de la connaissance, domaine médical.

5.1 Présentation générale

Dans leur ouvrage « *Safe and sound* »[FD00], les auteurs présentent les dangers liés à l'utilisation de systèmes à base d'intelligence artificielle (notamment pour le domaine médical), et proposent différents moyens pour les maîtriser. Parmi ces moyens, ils avancent le concept de système actif (en ligne) de gestion de la sécurité (*active safety management*) basé sur la notion d'*agent intelligent et autonome*[5], qui rejoint le concept de *safety bag*.

Il est important de noter que pour la définition du système actif de gestion de la sécurité les auteurs se sont appuyés sur les travaux réalisés dans le cadre d'Elektra (cf. précédemment section 3).

Les références utilisées dans cette section sont les suivantes :

- site web du projet PRO*forma* : http ://acl.icnet.uk/lab/proforma.html ;
- articles disponibles sur http ://acl.icnet.uk/lab/PUBLICATIONS traitant de PRO*forma* ;
- et principalement : J. Fox and S. Das. *Safe and sound - Artificial intelligence in hazardous applications*. AAAI Press - The MIT Press, 2000 [FD00].

5.2 Exigences de sûreté de fonctionnement

Les systèmes présentés dans ce livre reposent sur des technologies à base d'intelligence artificielle. Et notamment celles comportant des modules décisionnels constitués d'une base de connaissances et d'un moteur d'inférence. Ils présentent des dangers spécifiques à cette technologie [FD00][p.132 et p.246] :

- des incohérences, redondances, imprécisions de la base de connaissances du système ;
- des mises à jour statiques ou dynamiques incorrectes de la base de connaissances ;
- des imprécisions des procédures d'inférence (même si la base de connaissances est correcte) ;
- des situations adverses non prévues par les concepteurs entraînant un comportement non désiré ;
- le critère de décision construit de manière implicite ou explicite par le système peut ne pas être acceptable dans toutes les situations (par exemple provoquer des effets secondaires non prévus).

Les auteurs proposent d'utiliser des techniques classiques d'analyse du risque appliquées à l'analyse d'un système à base d'IA dans le chapitre 7. Ils identifient alors deux classes de source potentielle de dommage regroupant les dangers présentés :

- les fautes au sens de [Lap04] ;

[5] Ils définissent un agent intelligent et autonome par différentes fonctionnalités qui lui sont associées et que nous résumons ici [FD00, p.127] : opérations en autonomie (décision, acquisition et action), réactivité, interaction avec l'environnement, fonctions cognitives de planification.

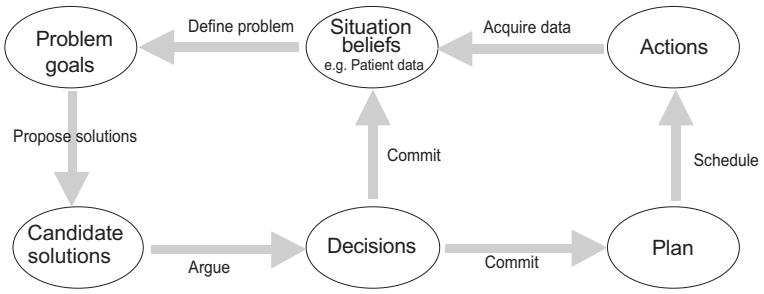

FIG. 7 – Le *Domino process model* tiré de [FD00]

– les dangers (*hazard*) qui sont l'équivalent des situations dangereuses [ISO99]. Cette séparation (qui peut sembler ambiguë car les dangers contiennent les fautes dans de nombreuses définitions) est motivée par le fait que la notion de faute n'est pas suffisante pour exprimer l'ensemble des menaces pesant sur de tels systèmes. En effet, il existe un ensemble de situations, qui ne sont pas des fautes mais qui peuvent entraîner le système dans des états dangereux. Néanmoins c'est sur cette base que les auteurs proposent des moyens d'évitement des fautes et des dangers par une maîtrise du processus de développement (langages graphiques et formels, analyse du risque, etc.). Cependant, malgré l'utilisation de ces techniques et, face à la complexité et au non-déterminisme de ces systèmes, ils concluent sur l'impossibilité de donner l'assurance que les systèmes à base d'intelligence artificielle sans mécanismes de contrôle supplémentaire sont sûrs (du point de vue de la sécurité). Ce constat permet alors d'argumenter en faveur d'un système de gestion de la sécurité en-ligne (ou actif), sous forme d'agent indépendant nommé *guardian agent*.

5.3 Architecture et fonctionnement

La proposition de Fox et Das n'est pas une architecture matérielle/logicielle à proprement parler. Ils se placent dans un contexte d'« agents intelligents ». Les auteurs ne présentent pas de vue structurelle de leur travaux mais s'attachent à décrire comment construire ces agents.

Le point de départ de leur approche est le modèle *Domino* présenté sur la figure 7, qui représente tout processus décisionnel (informatisé ou non). Les noeuds sont des bases de données et les flèches sont des fonctions ou des procédures d'inférence permettant de compléter les informations des noeuds. Un point important est que les éléments de ce modèle ne sont pas exclusivement réalisés par des humains ou par du logiciel, mais peuvent l'être par des réalisations mixtes. Cette représentation qui n'est ni fonctionnelle, ni orientée-objet, permet aux auteurs d'identifier et de modéliser pour un système donné les tâches et les données présentes dans les mécanismes décisionnels (principalement pour des diagnostiques médicaux). Ce modèle permet de montrer qu'à partir de connaissances du système (*Situation beliefs*), des objectifs sont fixés (*Problem goals*), et plusieurs options (*Candidate solutions*) sont proposées. Le choix de celle(s) qui sera (seront) mise(s) en application (*Decisions*) permet d'élaborer le cas échéant un plan (*Plan*) d'ac-

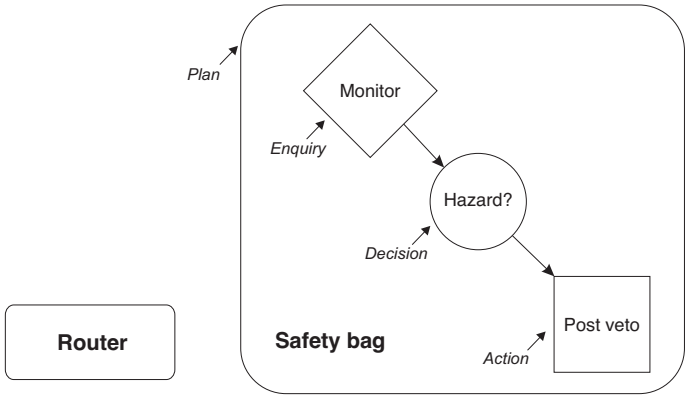

FIG. 8 – Une implémentation en PROforma du *plan Safety bag*, tiré de [FD00]

tions (*Actions*), en mettant à jour les connaissances de la situation ainsi modifiée (*Situation beliefs*).

Pour pouvoir ensuite développer un système informatisé réalisant concrètement ces tâches, les auteurs proposent de structurer les différents mécanismes décisionnels en quatre tâches élémentaires (données ici en version originale l'anglais, les symboles graphiques de ces tâches sont données dans la figure 8 présentée ultérieurement) :

- *enquiry*, actions retournant une information sur le système (comme l'acquisition de données d'état du système) ;
- *decision*, toute activité résultant en un choix ;
- *action*, réalisation d'une action sur le système ;
- *plan*, un ensemble ordonné des trois précédentes tâches afin de réaliser un but.

Ainsi, pour construire un système d'aide à la décision dans le domaine médical (diagnostique et choix du traitement), les modes de raisonnement des experts médicaux sont capturés avec le modèle Domino puis modélisés en tâches élémentaires (en utilisant les éléments présentés ci-dessus avec un outil appelé Proforma).

Pour la conception d'un *guardian agent*, la démarche est identique. À partir de la capture de raisonnements d'experts de la sécurité, il est possible de traduire la connaissance de ces experts en « règles de sécurité » représentés ici en *plans* (contenant des *actions*, *enquiries*, et *decisions* et d'autres *plans*). La figure 8 est une modélisation en PROforma proposée par Fox et Das du concept de *safety bag* d'Elektra (cf. précédemment section 3). Sur cette figure la partie logique effectuant le calcul des routes (commandes des aiguillages et des feux de signalisation) est modélisée comme un plan (*router*).

En résumé, pour Fox et Das, l'architecture et le fonctionnement d'un *guardian agent* est celui d'un agent intelligent et autonome. Même si ce n'est pas explicite on peut en déduire qu'il y a donc un moteur d'inférence dont les décisions sont guidées par des règles (donc une base de connaissances).

5.4 Règles de sécurité

La conception d'un agent suit le processus présenté dans la partie gauche de la figure 9. À partir d'un concept, on spécifie en PROforma les tâches élémentaires que l'agent aura à réaliser. Ceci forme la spécification de la base de connaissances de l'application, traduite ensuite en un langage formel R2L. Ce qui est ajouté ici par rapport à la figure extraite de l'ouvrage, correspond à des éléments présentés par les auteurs mais non intégrés dans la figure d'origine. Cette figure présente une autre dimension au processus : celle concernant les aspects de sécurité. En parallèle à l'analyse des tâches avec PROforma, il est possible de spécifier de manière séparée les contraintes de sécurité que l'agent aura à vérifier. Pour les identifier, le processus se basera sur des méthodes classiques de l'analyse du risque (HAZOP[6], FMEA[7], et FTA[8] sont citées p.137 et 247), ainsi que sur des règles de sécurité génériques proposées par les auteurs. Celles-ci, quinze au total, ont été déterminées de manière empirique sur la base de cas concrets (6 premières règles), mais également sur une analyse théorique du modèle Domino :

```
/* Rule 1 */
/* Détecter tout état anormal potentiellement dangereux, et créer */
/* un objectif permettant de le traiter */
        if      result of enquiry is State and
                State is not safe
        then    goal is remedy State

/* Rule 2 */
/* Si l'état anormal est un danger connu ayant une action curative */
/* connue, alors proposer cette action comme candidate */
/* à l'objectif précédent */
        if      goal is remedy State and
                known remedy for State is Action
        then    candidate for remedy of state is Action

/* Rule 3 */
/* S'engager pour l'action curative si l'agent peut établir qu'elle est */
/* permise au regard des règles du protocole */
        if      candidate for remedy of State is Action and
                decision status of Action is permitted and Action is safe
        then    decision status of Action is obligatory

/* Rule 4 */
/* Toute action potentiellement dangereuse doit avoir été autorisée */
/* au préalable par un agent de niveau supérieur*/
        if      candidate for remedy of State is Action and
                possible (Action causes Newstate) and
                NewState is not safe
        then    authorization of Action is obligatory
```

[6] Hazard and operability analysis
[7] Failure modes and effects analysis
[8] Fault tree analysis

```
/* Rule 5 */
/*  Quand une action a été autorisée, elle est permise */
        if      authorization of Action is obligatory and
                Action is authorized
        then    Action is permitted

/* Rule 6 */
/* Toute action n'ayant aucune conséquence dangereuse est permise */
        if      candidate for remedy of State is Action and
                not (possible (Action causes NewState)
                and NewState is not safe)
        then Action is permitted

/* Rule 7 */
/* Si une situation dangereuse peut être atteinte */
/* à partir de la situation courante, l'objectif devient l'évitement de la */
/* situation dangereuse */
        if      current situation is Situation1 and
                Situation1 can lead to Situation2 and
                Situation2 is not safe
        then    goal is prevent:Situation2

/* Rule 8 */
/* Si l'état courant peut entraîner une situation que l'on */
/* souhaite éviter, et qu'il existe un plan permettant de */
/* supprimer l'état courant, alors ce plan devient une solution /*
/* candidate à l'évitement de la situation */
        if      goal is prevent:Situation and
                Current_state is a cause of Situation and
                Plan is method for removing Current_state
        then    candidate for goal prevent:Situation is Plan

/* Rule 9 */
/* Si l'objectif est d'éviter un événement et qu'il existe un plan */
/* permettant de le bloquer, alors ce plan devient un candidat à l'objectif */
        if      goal is prevent:Event and
                Plan is a method for blocking Event
        then    candidate for goal prevent:Event is Plan

/* Rule 10 */
/* Si l'objectif principal est d'éviter une situation, et qu'il */
/* existe une action entraînant une situation capable de la */
/* compenser, alors cette action est un candidat à l'objectif principal */
        if      goal is prevent:Situation1 and
                Action causes Situation2 and
                Situation2 compensates for Situation1
        then    candidate for goal prevent:Situation1 is Action

/* Rule 11 */
/* Si l'objectif principal est d'éviter une situation, et qu'il */
/* existe une méthode permettant de détecter un précurseur */
```

```
/* de cette situation, alors cette méthode est un candidat */
/* à l'objectif principal */
        if      goal is prevent:Situation and
                Precursor is a precursor of Situation and
                Monitor is a method for detecting Precursor
        then    candidate for goal prevent:Situation is Monitor

/* Rule 12 */
/* Si une action est candidate à un objectif mais peut */
/* avoir comme effet secondaire une situation dangereuse, alors */
/* cette action est annotée comme étant défavorable avec */
/* la justification qu'elle est dangereuse */
        if      candidate for Goal is Action and
                Situation is a possible side-effect of Action and
                Situation is not safe
        then    Action is opposed on the grounds that: it is hazardous

/* Rule 13 */
/* Si une action est candidate à un objectif, et faisant partie */
/* d'un plan, lui même obligatoire pour un deuxième objectif, alors */
/* l'action est soutenue avec la justification qu'elle est */
/*nécessaire pour ce deuxième objectif */
        if      candidate for Goal1 is Action and
                Plan is obligatory for Goal2 and
                Action is a component of Plan
        then    Action is supported on the grounds that:
                it is required for Goal2

/* Rule 14 */
/* Si une action est obligatoire pour une raison et  qu'elle */
/* n'est pas exclue pour une autre raison, alors elle est permise */
        if      Action is obligatory on grounds G1 and
                Action is not excluded on other grounds G2
        then    Action is permitted

/* Rule 15 */
/* Si une action est candidate à l'objectif et le soutien en sa faveur */
/* est supérieur au soutien de toute autre action candidate, alors */
/* l'action est permise */
        if      candidate for Goal is Action1 and
                netsupport for Action1 is Degree1 and
                it is note the case that
                (       candidate for Goal is Action2 and
                        netsupport for Action2 is Degree2 and
                        Degree2 is greater than Degree1 )
        then    Action is permitted
```

Un langage formel à base de proposition, Lsafe, a été développé pour implanter ces règles. Les opérateurs logiques, ou modalités, sont présentés figure 10. Ces opérateurs sont utilisés dans des expressions logiques. Par exemple, $< oblg > \neg\varphi$ signifie que à tout instant la propriété φ ne doit pas être vraie, le système doit rendre

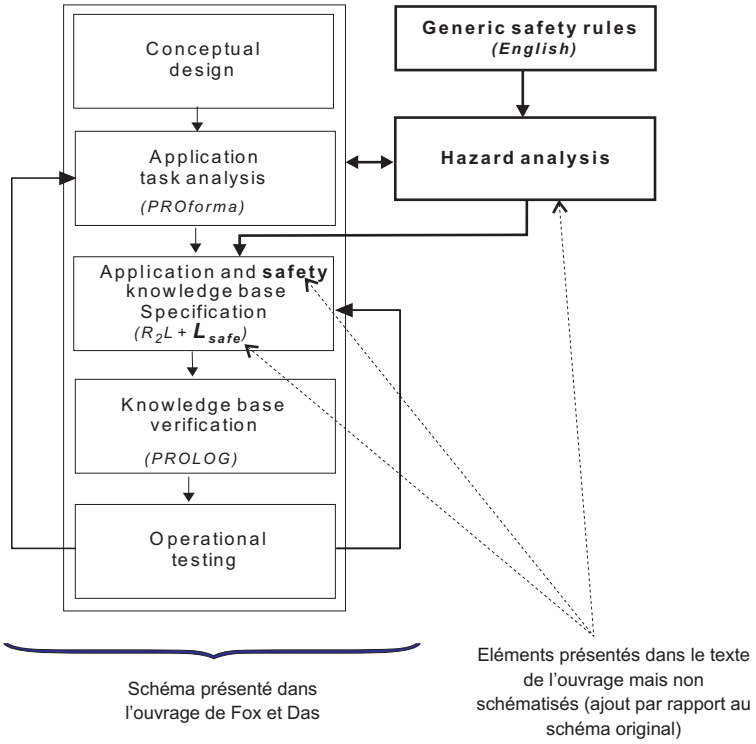

FIG. 9 – Cycle de vie pour des systèmes à base d'IA tiré de [FD00], et augmenté de leurs propositions

cette propriété fausse (à la différence de $\neg\varphi$ qui signifie qu'à tout instant φ est fausse et que c'est immuable). Si cette propriété devient vraie, alors une règle de sécurité doit être appliquée ce qui s'écrit : $\varphi \rightarrow < perm > Action1 \vee < safe > Action2$. La règle est donc de rendre vraies toutes les préconditions de l'Action1 ou de rendre *safe* l'Action2. Les Auteurs proposent de séparer alors ces informations en différentes sections comme présenté sur la figure 11. Cet exemple ne reprend que quelques éléments de celui présenté dans [FD00, p.249].

Ce processus permet donc d'obtenir un agent contenant à la fois une spécification des tâches à réaliser (en R2L) et des contraintes de sécurité (en Lsafe). Il est donc possible de séparer les aspects fonctionnels de ceux concernant la sécurité grâce à ces deux langages. Cependant, il est difficile de déterminer comment cette séparation sera mise en œuvre dans un agent dédié à la sécurité. En effet les auteurs ne donnent pas d'exemple concret de l'implantation des règles en Lsafe dans un agent composé de tâches PROforma ou non.

<safe>	safe	action α or property φ is safe
<auth>	authorized	action α is authorized by a superior agent
<pre>	preferred	action α is preferred to action β
<perm>	permitted	all obligatory preconditions of action α are satisfied
<oblg>	obligatory	action α (property φ) is obligatory
[t1,t2]		action α (property φ) is true in the interval t1 to t2

FIG. 10 – Ensemble des modalités du langage Lsafe

Property Symbols

asthma	the patient has asthma
vl_pef	the patient has very low peak expiratory flow
distressed	the patient is distressed

Action symbols

salbutamol	treatment with 5mg of salbutamol
terbutaline	treatment with 10mg of terbutaline plus oxygen
pef_measure	measure peak expiratory flow

Application knowledge

asthma → <*oblg*> *pef_measure*
vl_pef → <*perm*> *salbutamol* ∧ <*safe*>*salbutamol*
vl_pef → <*perm*> *terbutaline* ∧ <*auth*>*terbutaline*

Safety constraints

<*oblg*> ¬ *vl_pef*
vl_pef → <*oblg*> ¬ *distressed*

FIG. 11 – Exemple d'utilisation de Lsafe pour un système d'aide à la décision du traitement de l'asthme

5.5 Résultats et discussion

Le travail présenté dans cet ouvrage est très riche de par les différents domaines qu'il aborde, ainsi que par les propositions qui sont faites. Les points les plus intéressants dans le cadre de notre recherche sont la présence d'un langage dédié à l'expression de règles de sécurité (Lsafe) et d'éléments de réflexion sur le processus d'identification et de construction des règles (PROforma et analyse du danger).

La structure du *guardian agent* n'est pas différente d'autres systèmes à base d'IA (base de connaissances et moteur d'inférence) mais les langages de spécification sont originaux. Il est donc possible de réaliser des systèmes de type safety-bag en se basant sur les deux propositions de langages formels de specification R2L et Lsafe.

Cependant, il existe des aspects à prendre en compte pour la suite de notre recherche. L'utilisation de Lsafe pour la réalisation d'un *guardian agent* n'est pas illustrée par un cas concret. Il est difficile à travers l'ouvrage de déterminer si les auteurs pensent qu'un guardian agent doit être réalisé en R2L et Lsafe, ou uniquement avec Lsafe. La réponse se situe sans doute entre les deux. Certains cas sont possibles avec uniquement Lsafe, et d'autres plus complexes requièrent l'utilisation de R2L. Ainsi Lsafe n'est qu'une proposition qui n'a pas été validée expérimentalement et qui n'a pas été étudiée depuis la sortie du livre[Fox04]. Il existe d'autres langages pour les systèmes à base d'IA actuellement en développement ou ayant déjà été validés sur plusieurs applications. Il conviendrait de déterminer s'ils sont mieux adaptés à la réalisation de systèmes de type *safety bag*.

6 Projet SPAAS

6.1 Présentation générale

Il existe dans le domaine spatial de nombreux avantages à utiliser des systèmes autonomes notamment pour les tâches suivantes :

- accomplissement de missions d'observation (planification autonome en fonction des demandes des utilisateurs émanant de la terre) ;
- maintient de la position relative d'une navette spatiale dans un vol en formation ;
- exploration des profondeurs de l'espace (sondes capables d'enclencher des mécanismes d'auto-protection) ;
- accomplissement de missions robotisées (exploration, prélèvements, etc.).

Malgré ces avantages une seule expérience d'autonomie a été réalisée en contexte réel, celle de Deep Space One (voir la synthèse dans [PTF02]), où pendant plusieurs jours le système est resté en totale autonomie.

C'est dans ce contexte que le projet SPAAS (Software Product Assurance for Autonomy on-board Spacecraft) a été lancé. Il s'agit d'un projet ESA, pour lequel ont collaboré EADS-ASTRIUM, AXLOG, et le LAAS-CNRS. Les objectifs étaient de déterminer les moyens possibles et disponibles à mettre en œuvre pour maîtriser la sûreté de fonctionnement des logiciels autonomes présents dans les satellites du futur. Une des recommandations d'un rapport de l'état de l'art sur cette problématique [LPPTF01], montrait l'importance de la présence d'un système de

surveillance de type *safety bag*. Une expérimentation a suivi cette recommandation et un prototype a été développé. Il a été implanté sur une station de travail, et sur un système embarqué sous VxWorks, en utilisant des composants de planification présents sur d'autres applications spatiales.

Un travail a été également mené sur un *Plausibility Checker* qui est en fait une sorte de *safety bag* vérifiant au sol la cohérence des commandes envoyées au satellite. En dehors de la réalisation technique qui est différente de celle à bord, l'approche du problème est la même. Bien que le système au sol ne soit pas un système autonome, la problématique est très similaire. Certains éléments de ce travail seront présentés dans cette section en complément au *safety bag*, et notamment dans la section relative aux règles de sécurité.

Cette section s'appuie essentiellement sur les rapports techniques et un article de conférence suivants :

- N. Lécubin, J.C. Poncet, D. Powell, and P. Thévenod-Fosse. SPAAS : Software product assurance for autonomy on-board spacecraft. Lessons learnt from autonomous non-space applications. Technical Report 01267, LAAS-CNRS, Toulouse, July 2001 [LPPTF01]
- J.P. Blanquart, X. Méchin, and J.C. Poncet. SPAAS : Software Product Assurance for Autonomy on-board Spacecraft. Plan for implementing assurance software for autonomy functions. Technical Report SPAAS/TN5, EADS Astrium SAS, Toulouse, August 2002 [BMP02]
- J.P. Blanquart, F. Deladerrière, and C. Honvault. SPAAS : Software Product Assurance for Autonomy on-board Spacecraft. Experimentation of SPAAS reusable components. Technical Report SPAAS/TN6, EADS Astrium SAS, Toulouse, February 2004 [BDH04a]
- J.P. Blanquart, C. Honvault, N. Lécubin, J.C. Poncet, and G. Quach. SPAAS : Software Product Assurance for Autonomy on-board Spacecraft. Software Documentation Package of the SPAAS on-board reusable component (safety bag). Technical Report SPAAS/SDB, EADS Astrium SAS, Toulouse, Febrary 2004 [BHL$^+$04]
- E. Totel, B. Polle, and MC. Charmeau. Modelling an autonomous spacecraft architecture. In *Data Systems in Aerospace DASIA'2001, Nice, France*, June 2001 [TPC01]

6.2 Exigences de sûreté de fonctionnement

Le rapport [BMP02] mentionne deux catégories de menaces susceptibles d'affecter la sûreté de fonctionnement :

- des séquences d'événements inconnus liés à l'espace ;
- des fautes ou de défaillances du matériel et du logiciel.

Ce projet étant orienté vers la réalisation d'un prototype, les exigences de sûreté de fonctionnement n'ont pas été détaillées plus en avant. De plus, aucun lien entre ces exigences de haut niveau d'abstraction et des règles de sécurité implantées dans le prototype n'est présenté.

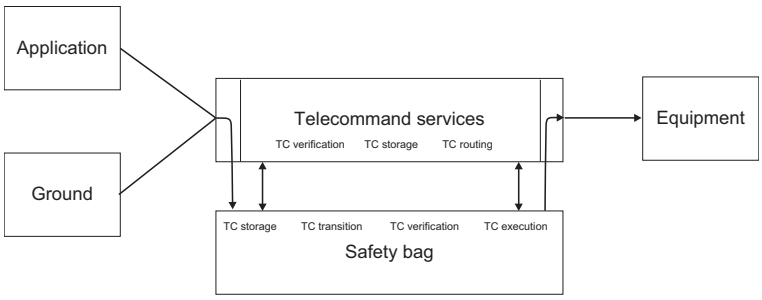

FIG. 12 – Système de gestion des télécommandes incluant le safety bag tiré de [BHL+04]

6.3 Architecture et fonctionnement

Le choix d'architecture d'implantation du *safety bag* a été largement guidé par la technologie déjà utilisée par EADS Astrium. Ainsi, toute action à bord du satellite est considérée comme une « télécommande » (notée TC par la suite). Par exemple, l'allumage d'une caméra et les messages échangés entre modules logiciels sont des TC. Ceci conduit à l'existence d'un module logiciel particulier fournissant divers services liées aux TC regroupé dans la boîte *TC services* de la figure 12. Les services dispensés sont la vérification d'homogénéité, la mémorisation, et l'exécution des TC devant être réalisées à un instant donné. Tout ce qui concerne la planification, ou les calculs de commandes, est effectué par d'autres programmes.

Le *safety bag* développé dans SPAAS présenté sur la figure 12 est connecté en entrée du module *TC services*, afin de stocker et vérifier la cohérence des TC émises soit par le sol soit par d'autres programmes du système. Les diagrammes de séquence des figures 13, 14, et 15 illustrent les différents fonctionnements du *safety bag*. Lorsqu'une application, ou le sol émet une *time tagged TC* (une télécommande à exécuter à un instant t), le traitement est décrit figure 13. Le module *TC Services* vérifie que la TC n'est pas corrompue (de type *checksum*). Puis cette TC est stockée par le *safety bag* jusqu'à l'instant t où la TC doit être exécutée (ceci est déclenché par le TC servicer qui maintient une liste). À ce moment, la séquence de la figure 14 décrit comment est traitée cette requête. Le safety bag effectue alors deux actions majeures :

- Transition : en fonction de la TC et de l'état actuel du système le *safety bag* évalue l'état suivant.
- Vérification : le *safety bag* évalue si cet état suivant n'est pas un état à risque, si c'est le cas l'exécution est bloquée.

La TC est ensuite envoyée à l'équipement concerné (par exemple les propulseurs). La figure 15 illustre le cas où une TC doit être exécutée immédiatement et est donc stockée et traitée dans la même séquence.

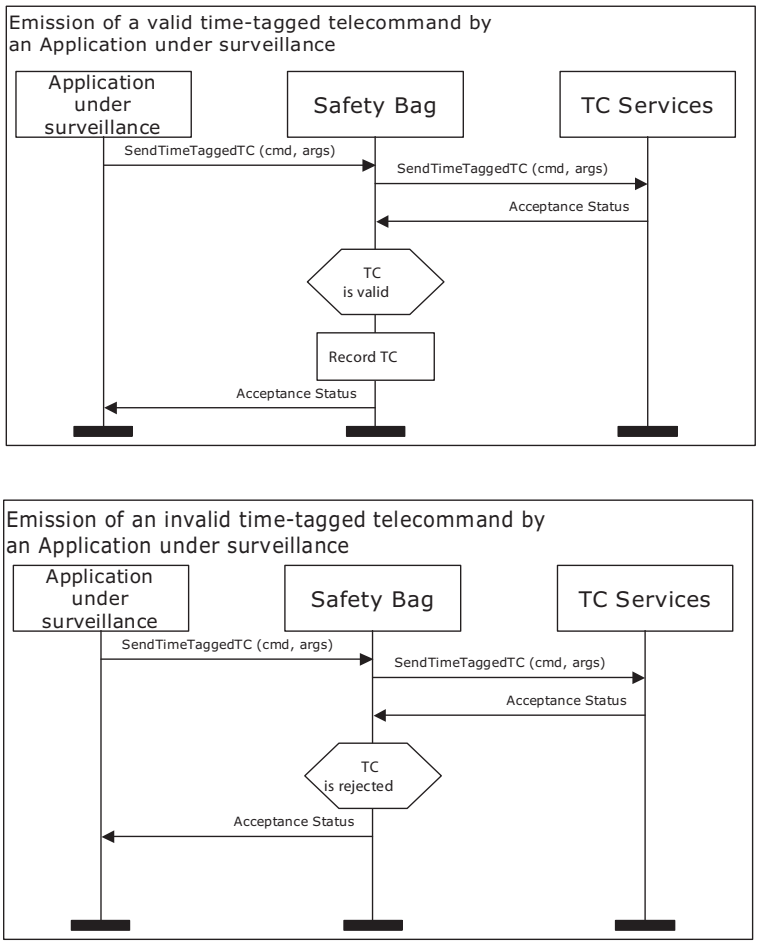

FIG. 13 – Émission d'un télécommande temporisée par une application surveillée tiré de [BHL+04]

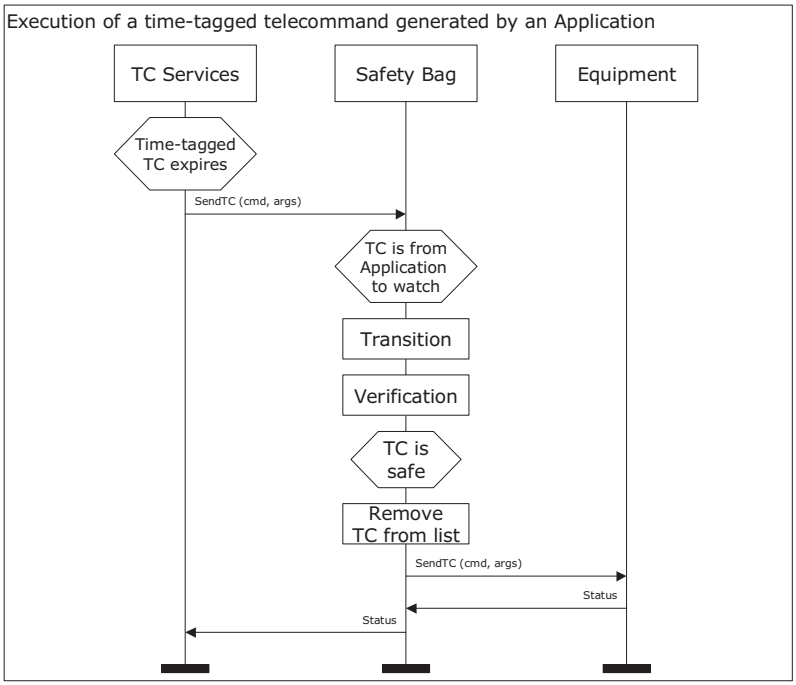

FIG. 14 – Exécution d'un télécommande temporisée mémorisée tiré de [BHL+04]

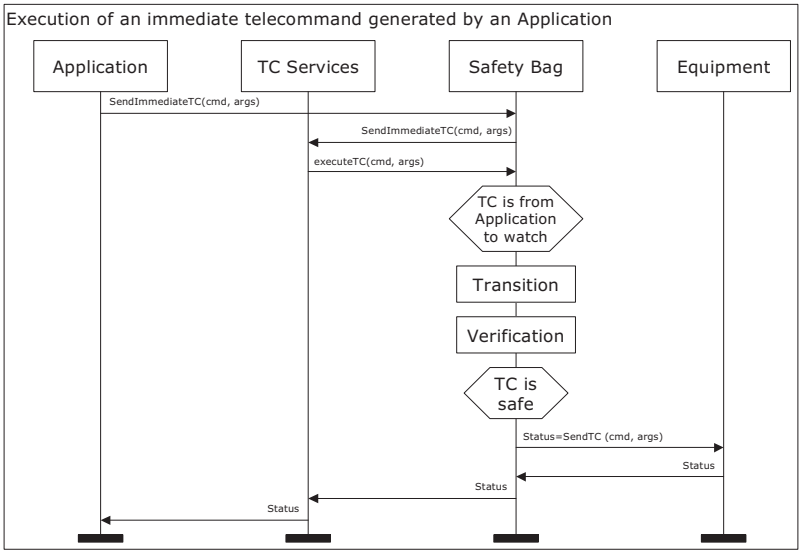

FIG. 15 – Exécution d'un télécommande temporisée immédiate tiré de [BHL+04]

6.4 Règles de sécurité

6.4.1 Au sein du *safety bag*

La détermination des règles de sécurité n'étant pas l'objectif du projet, il n'existe aucune section dédiée à ce sujet. On trouve tout au long du document des références à des contraintes de sécurité utilisées pour les tests et la validation du système, mais aucune démarche systématique n'a été effectuée. Les deux règles présentes au travers des différents documents sont :

- lors d'une manœuvre, l'énergie du système global ne doit passer en dessous d'un seuil critique ;
- après l'émission d'un plan (par le planificateur embarqué), pour tout signal de mise ON de consommation d'une ressource, il doit exister un signal de mise OFF.

Ces deux règles mises en œuvre dans le prototype de *safety bag* ont été intégrées au code source sans langage informatique spécifique. Le *safety bag* a été lui-même programmé en tant qu'un service (et non comme composant indépendant) du système de gestion des données du contrôleur de satellite.

6.4.2 Au sein du *plausibility checker*

Un travail technique différent du *safety bag* a été réalisé pour le *plausibility checker* bien que leur fonctionnement soit globalement le même. En effet le *plausibility checker* suit globalement le même protocole :

- interception de la télécommande ou de la procédure à émettre au satellite ;
- lecture de l'état actuel du satellite ;
- calcul de l'état suivant du satellite ;
- détermination de la dangerosité de l'état suivant en vérifiant des règles de sécurité.

Malgré les similarités avec le *safety bag*, les auteurs ont plus développé pour ce module la notion de règle et de son langage d'expression. Ainsi, les règles de vérification, regroupées en listes de règles (cf. figure 17) sont exprimées dans des fichiers textes suivant la syntaxe présentée sur la figure 16. Cette structure s'appuie sur la notation BNF[9]. Puis grâce à un parseur ces règles sont traduites automatiquement en langage machine. Il est intéressant de noter que la même approche a été utilisée pour représenter l'état du système, modélisé comme un agrégat de variables d'états dont la syntaxe est la même que pour les règles de sécurité (BNF).

Un travail important concerne le simulateur permettant de calculer l'état suivant du système. Ce module s'appuie sur trois sous-modules que sont les simulateurs :

- du satellite ;
- d'interpréteur de procédures du satellite (une copie du logiciel embarqué) ;
- et du système de gestion des données sur le satellite.

En fonction de ces trois simulateurs, le *plausibility checker* calcule l'état suivant du système.

[9] Backus-Naur Form : un draft de la version finale de la norme ISO sur le format BNF est disponible sur http ://www.cl.cam.ac.uk/~mgk25/iso-14977.pdf

```
rule_packet          →  FROM_ELEM INTEGER TO_ELEM INTEGER action
                        { check_rule_list }
check_rule_list      →  check_rule check_rule_list
                     |  ε
check_rule           →  forbidden_tc_rule
                     |  forbidden_modif_rule
                     |  value_rule
forbidden_tc_rule    →  FORBIDDEN_TC tc action
forbidden_modif_rule→  FORBIDDEN_MODIF var_name action
value_rule           →  VALUE var_name operation value action
variable_definition  →  IDENTIFIER type function
tc                   →  IDENTIFIER
var_name             →  IDENTIFIER
action               →  IDENTIFIER
                     |  ε
operation            →  EQUALS_SIGN
                     |  GREATER_SIGN
                     |  SMALLER_SIGN
                     |  DIFF_SIGN
                     |  SEQ_SIGN
                     |  GEQ_SIGN
                     |  AND_SIGN
                     |  OR_SIGN
                     |  function
value                →  IDENTIFIER
function             →  IDENTIFIER

Terminal symbols are represented in upper cases. Non terminal symbols
are represented in lower cases. The symbol ε stands for the empty word.
```

FIG. 16 – Syntaxe des règles de sécurité tiré de [BDH+04b] (en BNF, Backus Naur Form)

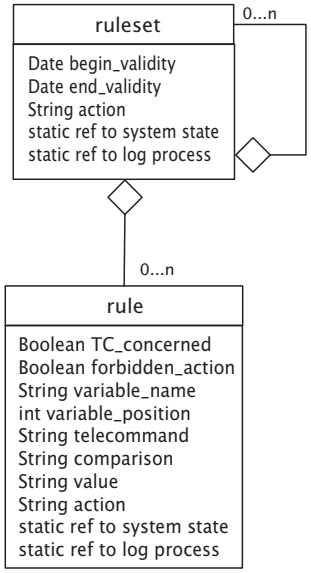

FIG. 17 – Diagramme de classes des ensembles de règles tiré de [BHL+04]

6.5 Résultats et discussion

Les expérimentations effectuées ont validé la mise en œuvre, et l'architecture du *safety bag* en tant que service du système, sur la base de quelques règles de sécurité. Ce travail a permis d'identifier un nombre important d'éléments à prendre en compte lors de la conception d'un *safety bag* pour une application autonome de type satellite.

Tout d'abord, le *safety bag*, en tant que composant supplémentaire aux composants fonctionnels, augmente la complexité du système et donc la probabilité de défaillance. Ce problème récurrent lors de la conception de système tolérants aux fautes, montre qu'il est nécessaire de démontrer qu'une erreur du *safety bag* ne peut entraîner de défaillance catastrophique. Or, dans SPAAS, le *safety bag* est vu comme un service, intégré au logiciel de calcul, ce qui peut entraîner des défaillances de mode commun. Ceci peut être évité en rendant le *safety bag* indépendant du système de commande. Il peut être externe (sur un autre processeur), ou plus simplement posséder au moins un fil d'exécution et une mémoire indépendants.

Un aspect important du *safety bag* de SPAAS, est la vérification du plan, basée sur la notion de *time tagged* TC. Celles-ci sont stockées à la fois par le système mais aussi par le *safety bag* pour effectuer des vérifications de cohérence entre elles. Ceci implique que le *safety bag* n'intercepte pas uniquement les TC envoyées aux équipements (caméra, réacteurs, etc.) mais aussi des TC internes au logiciel de commande comme les détails du plan. Ce type de *safety bag* ne se concentre donc pas uniquement sur les sorties du système de commande, mais aussi sur des éléments internes au logiciel et notamment à la planification, avant que les commandes ne soient envoyées aux équipements.

Enfin, lors du projet SPAAS, la détermination des règles de sécurité, que ce soit au niveau du *safety bag* ou du *plausibility checker* n'a pas suivi de processus systématique ou de démarche particulière. Cependant, dans le cadre du *plausibility checker*, des propositions ont été faites au niveau de l'expression des règles basées sur une expression en BNF et une encapsulation objet. Les auteurs ont également développé un *parser* pour traduire les règles de sécurité en un langage exécutable.

7 R2C

7.1 Présentation générale

L'équipe RIA du LAAS-CNRS développe depuis plusieurs années ses contrôleurs de robots autonomes sur la base d'un patron d'architecture implanté sur de nombreux robots. Il se décompose en deux niveaux, un niveau **fonctionnel** (pour tout ce qui a trait aux fonctions du système autonome, c.a.d. localisation, mobilité, charge utile, etc.) et un niveau **décisionnel** comprenant un planificateur et un superviseur. Une évolution de cette architecture est présentée figure 18. Elle inclut un troisième niveau, *Execution control level*, qui est une couche de sécurité vérifiant les requêtes transmises par le superviseur et également des flots de données échangés entre les modules fonctionnels.

Les quatre papiers publiés à ce jour sur le sujet sont :

- F. Py and F. Ingrand. Dependable execution control for autonomous robots. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems IROS2004, Sendai, Japan*, volume 2, pages 1136–1141. IEEE Publisher, September 2004 [PI04a]
- F. Py and F. Ingrand. Real-time execution control for autonomous systems. In *Proceedings 2nd European Congress ERTS, Embedded Real-Time Software*, Toulouse, France, January 2004 [PI04b]
- F. Ingrand and F. Py. An execution control system for autonomous robots. In *Proceedings IEEE International Conference on Robotics and Automation*, Washington D.C., USA, May 2004 [IP04]
- F. Ingrand and F. Py. Online execution control checking for autonomous systems. In *Proceedings of the 7th International Conference on Intelligent Autonomous Systems (IAS-7)*, Marina del Rey, California, USA, March 2002 [IP02]

7.2 Exigences de sûreté de fonctionnement

Les exigences de sûreté de fonctionnement sont directement liées aux classes de menaces, émergeant du contexte d'utilisation des robots, mais aussi de l'architecture LAAS en deux couches (fonctionnelle et décisionnelle) :

- les différents modèles utilisés dans le planificateur, le superviseur et la couche fonctionnelle sont hétérogènes du point de vue des niveaux d'abstraction, des langages utilisés, etc. Ces disparités sont une menace pour la complétude et la cohérence de la représentation des états du système à travers les différentes couches ;
- la complexité des mécanismes décisionnels est telle qu'il est difficile de certifier qu'un état non désiré ne sera pas atteint ;
- le développement indépendant des nombreux composants de la couche fonctionnelle peut entraîner des erreurs de spécification des interactions (notamment au niveau des interfaces des modules) ;
- certains modules ont des interdépendances (comme la consommation d'une ressource) qui peuvent être complexes à identifier ;
- les situations adverses de l'environnement ouvert dans lequel le système évolue ne peuvent toutes être identifiées et testées.

Face à ces différentes facteurs pouvant affecter la fiabilité et la sécurité, et à l'impossibilité de tous les éliminer, le système proposé est proche d'un *safety bag*. Son architecture et son fonctionnement sont présentés dans la section suivante.

7.3 Architecture et fonctionnement

La solution proposée est d'intégrer à l'architecture LAAS une couche de contrôle d'exécution contenant un module appelée R2C (cf. figure 18). Cette couche, située entre les couches décisionnelle et fonctionnelle, permet de vérifier des règles de sécurité à la fois sur les requêtes émanant de la couche décisionnelle (et plus précisément du superviseur) et sur les requêtes échangées entre les modules de la couche fonctionnelle.

L'architecture interne de R2C est donnée figure 19. Ce composant est constitué d'un *state checker* dont la tâche est de vérifier des assertions logiques de sécurité sur

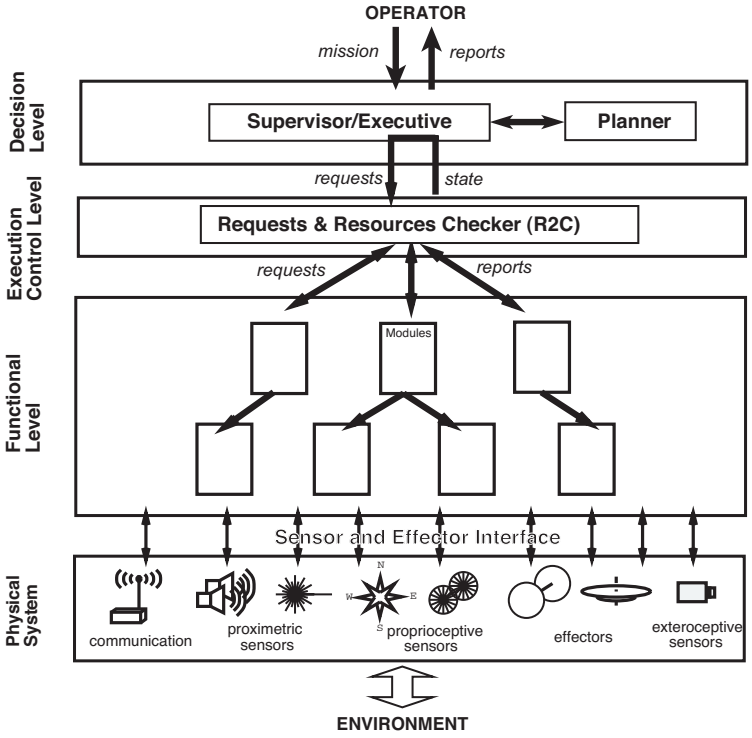

FIG. 18 – Architecture générique LAAS des contrôleurs de robots

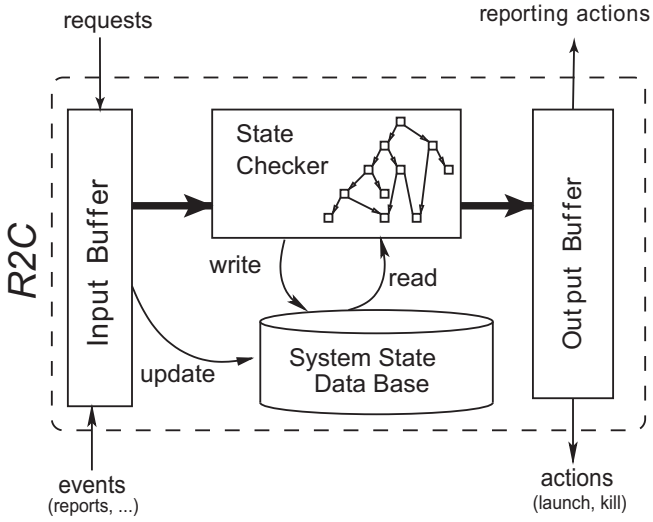

FIG. 19 – Architecture de R2C

les états du système. Ce *state checker* est un arbre logique de type OBDD (Ordered Binary Decision Diagrams, cf. [Bry86]) construit à partir des règles de sécurité en utilisant un parser/compilateur développé au LAAS : Exogen. Sur la base de cet arbre logique, le système vérifie si des conditions sont remplies et en cas d'échec deux options sont possibles : la requête est rejetée ou un processus est arrêté[10]. Par exemple, dans le cas d'une règle interdisant la prise de photo lorsque le robot est en déplacement, le parcours de l'arbre permet de vérifier si les différents éléments de cette règle sont vérifiés et de rejeter la requête de prise de photo le cas échéant. Des règles plus complexes requièrent l'utilisation de plusieurs tests, ainsi que de nombreuses variables systèmes directement lues dans la couche fonctionnelle et stockée dans la *system state database*. Notons que dans ces conditions l'arbre logique peut alors atteindre une taille importante et réduire la performance du système. D'après les auteurs, [Py05b], les premiers résultats de performance sont acceptables en laboratoire. Cependant, ils mentionnent que dans certaines utilisations où les actions devraient être réalisées dans des temps très courts, les délais induits par l'utilisation de R2C pourraient entraîner des problèmes importants pour la plate-forme robotique (aspect non présenté dans les papiers sur le sujet).

7.4 Règles de sécurité

Un travail important a été réalisé pour développer un langage de haut niveau de description des règles de sécurité. Quelques une de ces règles sont données figure 20. Ce langage d'expression est un sous ensemble d'un langage existant. Il est composé d'un ensemble d'opérateurs logiques tels que *always, never, running, last*, et de symboles logiques($=>, \&\&, ||$) usuels. Si certaines règles sont interprétables sans trop de difficultés pour le non spécialiste, d'autres font appel à une connaissance très fine des modules fonctionnels utilisés. Leur lecture est alors complexe comme celle de la règle 1. Ces règles sont ensuite traduites par Exogen pour former un arbre logique (ou arbre binaire) décomposant chaque élément en tests binaire.

La spécification de ces règles n'a pas suivi de méthode particulière. Les règles ont principalement été identifiées lors d'entretiens avec des spécialistes de la robotique et des modules fonctionnels utilisés. Comme pour les cas d'étude précédents c'est l'expertise des concepteurs qui est ici la garantie de la cohérence et de la complétude des règles de sécurité [Py05a].

7.5 Résultats et discussion

R2C a été implanté avec succès sur des robots du LAAS et a déjà permis de mettre en évidence au moins une erreur de conception (émission d'une requête sans attendre la fin d'une action nécessaire à cette requête). Des tests en laboratoire ont permis de vérifier expérimentalement l'efficacité de R2C.

L'approche du R2C est en certains points comparable à celle d'autres *safety bags*, mais il existe des différences importantes. Par exemple, le système R2C n'est pas un canal séparé du traitement logique comme dans Elektra et ne traite pas

[10] Une troisième option est présente dans les perspectives des travaux sur R2C [Py05a], elle consiste à réordonnancer les requêtes et les processus

```
check {

/* Règle 1 */
/* On ne peut attacher de Motion Estimator (ME) ou de Sensor Estimator (SE)  */
/* à POM que si le modèle géometrique du robot a déjà été défini
*/

 always: ( running(pom.addME) || running(pom.addSE) ) => last(pom.SetModel);

/* Règle 9
  * On ne peut communiquer via l'antenne si le robot bouge
  */
 never: running(antenna.Comunicate) && running(rflex.TrackSpeedStart);

 /* Règle 10
  * Quand le robot bouge la platine ne doit pas bouger
  */
 never: running(rflex.TrackSpeedStart) &&
                (
                running(platine.CmdPosCoord)
                || running(platine.CmdPosPan)
                || running(platine.CmdPosTilt)
                || running(platine.TrackPos)
                );

 /* Règle 11
  * le robot ne doit pas dépasser la vitesse de 0.9 m/s
 */
 never: running(rflex.TrackSpeedStart with arg.name.value.v>0.9);

 /* Règle 12
  * avant de pouvoir communiquer il faut que le module antenna ait
  * une connaissance des fenêtres de visibilité
  */
 always: running(antenna.Comunicate) => last(antenna.AddWindow);

 /* Règle 13
  * Avant d'indiquer les fenêtres de visibilité à antenna ce modéle
  * doit avoir été initialisé.
  */
 always: running(antenna.AddWindow) => last(antenna.Init);


 /* Règle 14
  * On ne peut prendre d'image si la camera n'a pas été
  * initialisée avec succès
  */
 always: running(camera.OneShot) => last(camera.Initialize);

 /* Règle 15
  * Lors d'une prise de vue la platine ne doit pas bouger.
  */
 never: running(camera.OneShot) &&
                (
                running(platine.CmdPosCoord)
                || running(platine.CmdPosPan)
                || running(platine.CmdPosTilt)
                || running(platine.TrackPos)
                );
};
```

FIG. 20 – Exemples de règles de sécurité de R2C avant compilation par EXOGEN

uniquement les sorties du système ; il est intégré entre les couches de l'architecture pour pouvoir étendre son champ d'action, mais fragilise par là même le contrôleur.

R2C utilise sa propre représentation des états du système, et introduit donc, dans le contrôleur de robot, un modèle supplémentaire avec un niveau d'abstraction différent des autres couches de l'architecture. Cette diversification du modèle peut être vue comme une technique de sûreté de fonctionnement et plus particulièrement de tolérance aux fautes. Paradoxalement, cela revient également à augmenter un des facteurs de menaces présentés au départ, qui est la consistance et cohérence entre les différents modèles et leurs niveaux d'abstraction. Plus généralement, l'introduction de la redondance induit une augmentation de la complexité, ce qui accroît la probabilité de la présence de fautes. Cet aspect lié à la complexité rejoint le problème de l'interaction de R2C avec les autres couches, tel que l'interprétation des rapports émis par R2C par les couches décisionnelle et fonctionnelle, qui peut être également source d'erreurs. Par exemple, R2C détecte le fait qu'une requête d'utilisation d'une caméra a été émise sans qu'il y ait eu préchauffage. R2C retourne alors une erreur au superviseur qui dans une première version de la conception interprétait ce message comme une caméra défaillante. Cet exemple illustre un problème de disponibilité (puisque le système n'utilisera plus la caméra) induit par l'ajout de la couche R2C. Comme pour la redondance, l'introduction de dispositifs supplémentaires peut conduire à réduire la disponibilité et la fiabilité, car la probabilité de présence de fautes est augmentée. En gagnant en sécurité, on réduit alors la fiabilité du système. Ainsi, l'adjonction d'un *safety bag* n'échappe pas au compromis classique en sûreté de fonctionnement entre fiabilité et sécurité.

Il est important de noter tout de même que l'utilisation de R2C, qui ne devrait pas conduire à réduire la sécurité, ne peut garantir l'absence de défaillances catastrophiques induites par ce *safety bag*, et notamment s'il existe d'autres mécanismes dédiés à la sécurité. Par exemple, R2C pourrait bloquer des processus en cours nécessaires à l'évitement d'un danger. Il est donc important lors de la conception de ce type de système de sécurité de justifier son utilisation par une analyse du risque induit par l'utilisation de ce sous-système.

Enfin, comme pour les autres cas d'étude de ce rapport, l'accent a été mis sur l'architecture, les choix de langages et de compilateurs, mais il n'y a pas eu d'utilisation de processus spécifique de détermination des règles de sécurité.

8 Conclusions

À travers les différents cas d'études présentés dans ce document, nous présentons dans cette section une synthèse de ce qu'est un système externe de sécurité de type *safety bag*, du point de vue des architectures et des fonctionnalités possibles, puis des éléments nécessaires au processus de développement. Nous proposerons en dernier point une définition de ce type de système, que nous adopterons pour la suite de nos travaux.

8.1 Architecture et fonctionnalités d'un *safety bag*

La figure 21 présente une schématisation des architectures des cas d'étude que nous avons présentés dans ce rapport. Selon la définition d'un *safety bag* de la norme IEC 61508 (voir section 1), l'architecture doit répondre à des exigences. Notamment, il doit être extérieur au système du point de vue matériel (« système de surveillance mis en œuvre à partir d'un ordinateur indépendant, selon une spécification différente »). Sur cette figure, on peut noter que seul Elektra et le Ranger disposent d'un processeur dédié à la surveillance. Pour les autres systèmes, le *safety bag* se trouve au cœur du système qu'il surveille, soit en tant qu'agent (guardian agent), composant (R2C), ou même service (SPAAS). Ceci s'oppose à la définition de la norme IEC 61508. Cependant, ces approches, présentant différent degrés d'indépendance vis-a-vis du système surveillé, caractériseront pour nous une échelle d'indépendance pour la définition d'un *safety bag* que nous présenterons ultérieurement (cf. section 8.3). Un autre aspect important est l'indépendance du système principal vis-à-vis du *safety bag* (pour l'instant nous nous sommes surtout intéressé à l'inverse). En effet, suivant l'architecture choisie, la défaillance du *safety bag* pourra être plus ou moins bien tolérée par le système global. Par exemple, on peut concevoir le système tel que l'arrêt du *safety bag* entraîne le blocage total du système. Ce choix est fait si l'on considère que la sécurité est plus importante que la disponibilité. À l'opposé, certains systèmes peuvent être conçus pour accomplir leur mission au delà des défaillances des modules de sécurité (comme des sondes explorant les profondeurs de l'espace). Un point de conception corrélé à ce problème est l'activation ou la désactivation par l'homme des règles de sécurité, voire du *safety bag* lui même. Il est difficile de dire quelle est la meilleure approche puisque l'histoire des accidents technologiques montre que la désactivation possible de système de sécurité a entraîné des dommages important. Cependant, pour des systèmes autonomes, il nous semble important que la manipulation des règles de sécurité (modification, suppression, ajout) en ligne soit possible, surtout lorsque le dispositif est éloigné, ou en mission comme une navette spatiale.

De ces cinq cas d'étude, il est possible de définir différents niveaux de connaissance de l'état du système par le *safety bag*. Tout d'abord, il peut être nécessaire que le système à surveiller ne soit pas vu comme une boîte noire par le *safety bag*. Par exemple dans le cas d'un système autonome, il semble important qu'il puisse vérifier avant l'exécution des actions du plan, que celles-ci respectent un ensemble de règles de sécurité. C'est cette approche qui est développé dans SPAAS et R2C. Il est donc possible de concevoir des *safety bag* qui voient le système à surveiller comme une boîte grise ou blanche. Dans le cas du Ranger l'unité de surveillance ne traite pas les sorties du contrôleur de robot lui-même (pas d'interaction entre ces deux unités). Ce n'est donc pas l'unité principale qui est surveillée, mais le système global, qui est donc vu comme une boîte noire. Notons que les spécifications et l'implantation des règles de sécurité au sein du Ranger sont les mêmes sur ces deux unités. Ceci s'écarte de la définition de départ du *safety bag*. Mais nous considérerons que cette application consiste tout de même en un *safety bag*, étant bien conscient qu'il s'agit ici d'un dispositif se limitant à l'observation de l'état global du système.

Le type d'actions possibles du *safety bag* impacte également sur l'architecture. Un *safety bag* peut se limiter à la détection (et déléguer l'action à un autre compo-

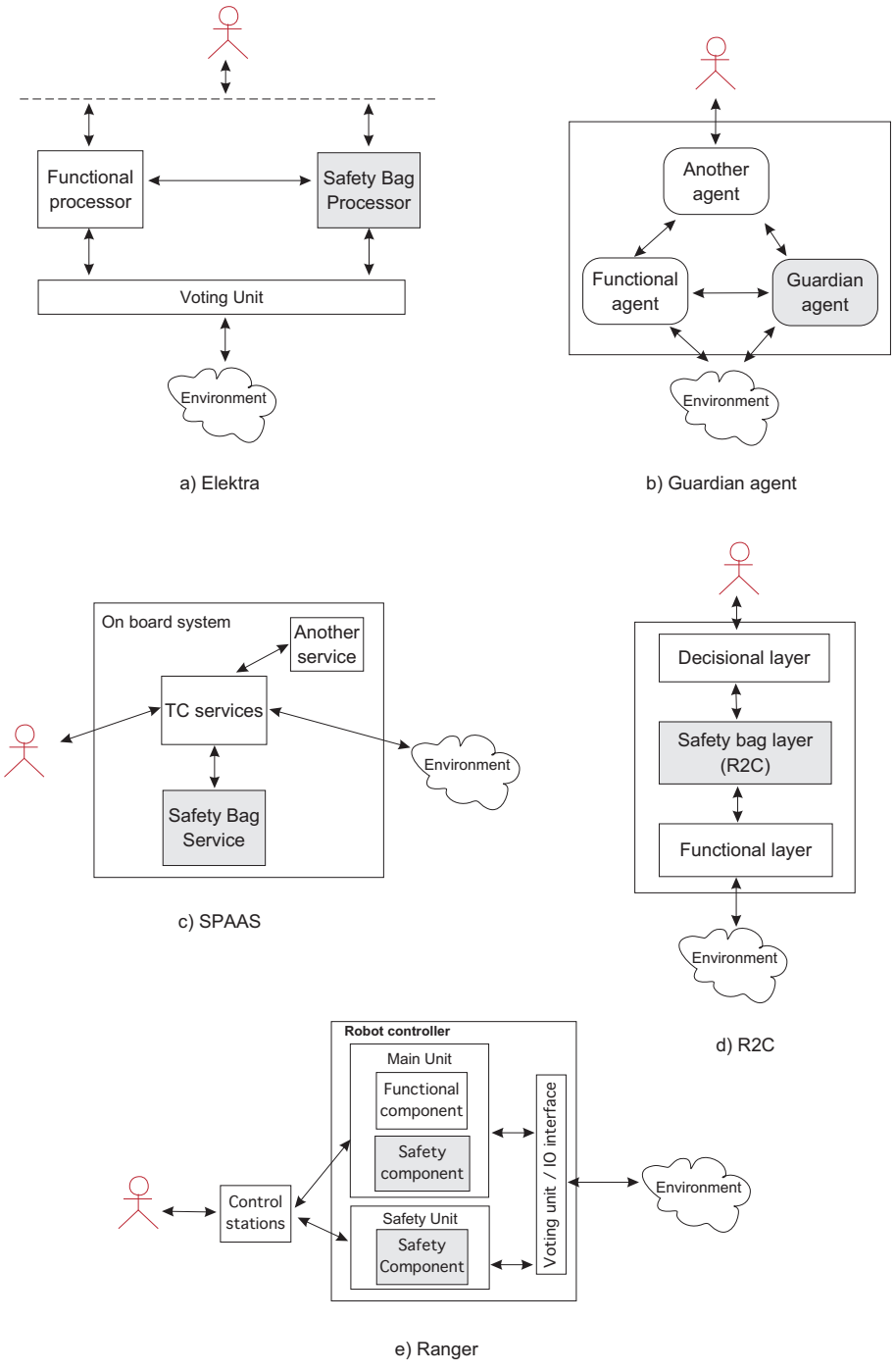

FIG. 21 – Les cinq architectures incluant les différentes approches du *safety bag*

sant), ou effectuer un recouvrement d'erreur : veto dans le *guardian agent* ou Elektra, blocage, ré-ordonnancement ou destruction de requête dans SPAAS et R2C. Dans chacun des ces cas l'architecture est différente puisqu'elle est dépendante du degré d'action que l'on donne au *safety bag* (entre simple transmission d'une détection ou recouvrement d'erreur). Il est également important de noter que la problématique commune de l'interprétation par le système de l'action du *safety bag* peut impliquer des modifications d'architecture, notamment au sein du système décisionnel (voir l'exemple de R2C dans la section 7.5).

8.2 Processus de développement d'un *safety bag*

Les techniques de développement des *safety bag* des cas d'étude présentées dans ce rapport sont résumées dans le tableau de la figure 22. On peut noter que seul Fox et Das pour le *Guardian Agent* font référence à des techniques d'analyse pour la détermination des règles de sécurité, mais sans en exposer les principes et l'adaptation nécessaire pour la réalisation d'un *safety bag*. Les principaux points d'étude qui sont abordés lors du développement d'un *safety bag* sont :

1. un modèle statique et dynamique du système, permettant également de représenter les situations dangereuses et les scénarios critiques. Une attention particulière devra être portée au modèle d'interaction du sous-système avec son environnement (c'est lors d'une interaction que le *safety bag* est activé) ;
2. des techniques de prévision de fautes permettant d'identifier les situations dangereuses (dues à des situations adverses) basées sur les modèles précédents ;
3. un langage de haut niveau d'expression des règles de sécurité ;
4. un modèle des états du système utilisable en ligne ;
5. un interpréteur, ou *parser* permettant d'intégrer ces règles de sécurité dans le code du *safety bag*.
6. une méthodologie, ou un « guide » de conception du *safey bag* indiquant quelles techniques utiliser, quand, comment, par qui ;
7. des patrons de conception de *safety bags*.

Ces différents points couvrent un très large spectre de travaux de recherche. Sans vouloir tout traiter, les paragraphes suivant montrent pour chacun de ces aspects des orientations possibles et quelques éléments de réflexion.

Point numéro 1 : Langage de modélisation

Les techniques actuelles de modélisation telles que UML, ou AADL permettent de donner des vues statiques et dynamiques des systèmes informatiques. La dimension « système » de ce projet (modélisation du système dans son environnement et réaction à des événements externes, capture des connaissances des experts pour les changements d'états du système, etc.) implique que le langage de modélisation soit capable d'appréhender cette dimension. Ainsi, UML , ou SysML[11], sembleraient

[11] *Systems Modeling Language* est un langage dérivé d' UML, http ://www.sysml.org

Cas d'étude	R2C	SPAAS	Guardian Agent	Elektra
Domaine	Robotique	Spatial	Médical	Ferroviaire
Présence d'un système décisionnel	Planificateur, superviseur	Planificateur	Agent intelligent (base de connaissances, moteur d'inférence)	–
Techniques utilisées dans le safety bag	Arbre binaire de décision	Règles de sécurité codées en dur	Agent intelligent (base de connaissance, moteur d'inférence)	Système expert
Menaces visées	Fautes de conception (hétérogénéité des modèles, complexité des mécanismes décisionnels, indépendance des modules, etc.) Situations adverses non prévues	Fautes matérielles et logicielles Situations adverses non prévues	Fautes de conception (dans la base de connaissances, et dans les mécanismes décisionnel) Situations adverses non prévues	Fautes de conception (complexité de la topologie du réseau ferroviaire) Fautes physiques
Langage des règles de sécurité	Modalités ajoutées à un langage logique + Exogen (parser/compilateur)	Structures de données (BNF) + *parser*	Lsafe (compilation en PROLOG)	PAMELA
Méthode de détermination des règles	Consultation d'experts	–	Méthode proposée mais non implémentée : analyse de la tâche couplée avec une analyse du danger (HAZOP, FMEA, FTA) Utilisation de règles génériques de sécurité	Consultation d'experts Utilisation de normes

FIG. 22 – Tableau récapitulatif de certains éléments des cas d'études

convenir. Pour l'aspect interaction, le diagramme de séquence (commun à de nombreux langages de modélisation) est un outil puissant par son pouvoir d'expression, son adaptabilité, et sa lisibilité.

En revanche la représentation d'une défaillance, ou d'un phénomène dangereux, ou plus généralement d'une situation adverse est un point qui n'est pas intégré actuellement à ces langages. Or il est fondamental que ces notions soient représentées pour exprimer par la suite les règles de sécurité. Cette représentation peut être faite de manière textuelle, ou modélisée avec les langages présentés ci-dessus, ou également de manière formelle. Ce dernier point est important si l'on souhaite obtenir une cohérence entre la modélisation du système et celle des situations adverses et des règles de sécurité. Ces dernières seront obligatoirement représentées à un moment donné de manière formelle pour être utilisées en ligne par le *safety bag*.

Ainsi, comme pour beaucoup d'applications, les techniques de modélisation nécessaires à la réalisation d'un *safety bag*, devront s'appuyer sur différentes technologies suivant les étapes du processus de développement et la vue du système considérée (statique ou dynamique, d'analyse ou de conception, etc.).

Point numéro 2 : Techniques de prévision des fautes

Le processus d'identification des règles de sécurité doit être ascendant et descendant comme lors d'une analyse du risque. Certaines règles propres à l'application, proviendront de l'analyse de l'architecture et des dangers possibles, d'autres seront déterminées grâce à des modèles de règles plus théoriques (comme dans le cas du *guardian agent*). La méthode devra permettre de confronter les règles de sécurité établies par les concepteurs aux spécialistes du domaine. Elle permettra également d'inclure lors du processus des normes de sécurité du domaine considéré. L'approche considérée pourra s'appuyer sur des techniques comme HAZOP, FMECA ou FTA, en interaction avec le langage d'expression choisi (langage de modélisation ou non, formel ou non).

Point numéro 3 : Langage d'expression des règles de sécurité

Le langage d'expression des règles de sécurité doit être d'un niveau d'abstraction suffisamment élevé pour être compréhensible et modifiable par des experts de la sécurité (et non spécialistes de l'informatique). Le langage naturel est évidemment le premier à choisir mais par la suite un langage formel sera nécessaire avant de passer à la compilation ou à l'exécution (si les règles sont sous forme de base de connaissance). Ce langage doit faire l'objet d'un choix suivant des critères de puissance d'expression, de lisibilité. Il est également important d'en choisir un qui soit évolutif et dont ses créateurs ou utilisateurs soient réactifs aux questions qui pourront émerger lors de la réalisation d'un *safety bag*..

Point numéro 4 : Modèle des états du système utilisable en ligne

Le *safety bag* doit interpréter une situation et déterminer l'état du système et de son environnement, ainsi que les états suivants possibles. Cet aspect a été très largement étudié dans le domaine de l'intelligence artificiel. Il convient donc de choisir une technique en se basant sur des critères comme ceux du point précédent.

Point numéro 5 : Compilateur ou *parser* des règles de sécurité

Cet aspect est directement lié au langage d'expression des règles de sécurité, et donc reprend les éléments présentés au point 3.

Point numéro 6 : Méthode de conception d'un *safety bag*

Il est important de proposer une méthode permettant de guider les concepteurs vers la solution et le type de *safety bag*. Elle doit prendre en compte la problématique du domaine considéré et la complexité du système et des dangers associés. Cette méthode pourrait être guidée par les techniques d'analyse du risque précédemment évoquées, mais l'expression des règles devrait conduire à développer des techniques spécifiques.

Sur la base des cas d'étude présentés dans ce rapport, il est possible de formuler quelques exigences pour la méthode de conception du *safety bag*. Tout d'abord, ll semble important qu'il existe des modèles communs entre cette méthode et le développement du système. Même si l'indépendance des techniques d'analyses est parfois intéressante pour réduire les fautes de développement (par la diversité), il n'en reste pas moins qu'un des objectifs de notre travail de recherche est de créer des articulations autour de modèles communs pour aboutir à des règles de sécurité cohérentes et complètes. En effet, la conception d'un *safety bag* implique l'emploi de techniques variées, appliquées à différents domaines, et couvrant plusieurs disciplines, et une des difficultés est de proposer une démarche permettant de coordonner tous ces aspects.

Un autre point important est la garantie de la cohérence des actions entreprises par des dispositifs de sécurité multiples. Même si cet aspect n'a pas été présenté dans les cas d'étude présentés ici, on peut imaginer que des systèmes externes capables de bloquer certaines actions peuvent engendrer des situations dangereuses.

Une idée que l'on retrouve uniquement dans les papiers sur Elektra est l'utilisation que l'on va faire du *safety-bag* tout au long du processus de développement. Par exemple, il est possible d'envisager des phases où le *safety bag* servira d'abord à détecter des fautes de conception pour les corriger, puis lors de la vie opérationnelle il permettra de recouvrir des erreurs. Ceci implique des utilisations différentes du *safety bag* suivant l'étape dans le processus de développement, et donc de prévoir différents modes de fonctionnement de ce dernier. Cet aspect peut être important pour les exigences de performance du système, directement liées au *safety bag*, qui ne sont pas les mêmes si l'on se trouve dans les étapes de tests en laboratoire ou de mise en opération. Par ailleurs, la méthode de conception doit également intégrer des estimations des performances du *safety bag* et du système modifié.

Point numéro 7 : Patrons de conception du *safety bag*

Des différents cas d'études il serait intéressant de dériver un ou plusieurs patrons de conception d'un système de sécurité de type *safety bag*. Ces patrons pourront être introduit dans la méthode de conception proposée.

8.3 Conclusion sur la définition d'un *safety bag*

Comme nous l'avons évoqué au début de ce document, il n'existe pas de définition commune au concept de *safety bag*. Cependant, en partant de la définition fournie par la norme IEC 61508, et à travers les différents cas d'étude considérés, nous proposons de traduire le terme *safety bag* par *système externe de sécurité* (*external safety system*), et répondant aux exigences suivantes :

- Ce système doit être externe au module fonctionnel principal. Externe ne signifie pas obligatoirement sur un processeur séparé, mais indépendant et donc possédant son propre fil d'exécution, ainsi qu'une mémoire protégée.
- Sa spécification doit être différente de celle du module fonctionnel principal, et notamment pour les règles de sécurité si elles existent déjà au sein de ce dernier.
- Il doit surveiller le module fonctionnel et le système global en permanence, et empêcher toute transition vers un état dangereux.
- En cas de détection de danger, ce dispositif doit amener le système principal dans un état sûr, le niveau « sûr » étant défini pour l'application considérée par les concepteurs.

Références

[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(1) :11–33, 2004.

[BDH04a] J.P. Blanquart, F. Deladerrière, and C. Honvault. SPAAS : Software Product Assurance for Autonomy on-board Spacecraft. Experimentation of SPAAS reusable components. Technical Report SPAAS/TN6, EADS Astrium SAS, Toulouse, February 2004.

[BDH+04b] J.P. Blanquart, F. Deladerrière, C. Honvault, N. Lécubin, J.C. Poncet, and G. Quach. SPAAS : Software Product Assurance for Autonomy on-board Spacecraft. Software Documentation Package of the SPAAS Ground reusable component (plausibility checker). Technical Report SPAAS/SGD, EADS Astrium SAS, Toulouse, February 2004.

[BHL+04] J.P. Blanquart, C. Honvault, N. Lécubin, J.C. Poncet, and G. Quach. SPAAS : Software Product Assurance for Autonomy on-board Spacecraft. Software Documentation Package of the SPAAS on-board reusable component (safety bag). Technical Report SPAAS/SDB, EADS Astrium SAS, Toulouse, Febrary 2004.

[BMP02] J.P. Blanquart, X. Méchin, and J.C. Poncet. SPAAS : Software Product Assurance for Autonomy on-board Spacecraft. Plan for implementing assurance software for autonomy functions. Technical Report SPAAS/TN5, EADS Astrium SAS, Toulouse, August 2002.

[Bry86] R.E. Bryant. Graph-based algorithms for boolean function manipulation. In *IEEE Transactions on Computers*, volume C-35, 8, pages 677–691, 1986.

[BS97] B. Bon and H. Seraji. Real-time model-based obstacle detection for the NASA ranger telerobot. In *Proc. Internation Conference on Robotics and Automation, Albuquerque, New Mexico*, pages 1580–1587. IEEE Publisher, April 1997.

[CM03] J. Carlson and R. R. Murphy. Reliability analysis of mobile robots. In *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, pages 274–281, Taipei, Taiwan, September 14-19 2003.

[Erb89] A. Erb. Safety measures of the electronic interlocking system "Elektra". In *Proc. IFAC SAFECOMP 89, Vienna, Austria*, 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.

[Fox04] J. Fox. Private communication, 2004.

[IEC01] IEC 61508. Functional safety of electrical/electronic/programmable electronic safety-related systems. International Electrotechnical Commission, 2001.

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

[IP04] F. Ingrand and F. Py. An execution control system for autonomous robots. In *Proceedings IEEE International Conference on Robotics and Automation*, Washington D.C., USA, May 2004.

[ISO99] ISO/CEI Guide 51. Aspects liés à la sécurité - Principes directeurs pour les inclure dans les normes. International Organization for Standardization, 1999.

[KK95] H. Kantz and C. Koza. The Elektra railway signalling-system : Field experience with an actively replicated system with diversity. In *Proceedings of the Twenty-Fifth International Symposium on Fault-Tolerant Computing (FTCS)*, pages 453–458. IEEE Publisher, 1995.

[Kle91] P. Klein. The safety-bag expert system in the electronic railway interlocking system Elektra. *Expert Systems with Applications*, 3 :499–506, 1991.

[Lap04] J.C. Laprie. Sûreté de fonctionnement informatique : Concepts de base et terminologie. *REE*, X(X) :XX, 2004.

[LPPTF01] N. Lécubin, J.C. Poncet, D. Powell, and P. Thévenod-Fosse. SPAAS : Software product assurance for autonomy on-board spacecraft. Lessons learnt from autonomous non-space applications. Technical Report 01267, LAAS-CNRS, Toulouse, July 2001.

[PI04a] F. Py and F. Ingrand. Dependable execution control for autonomous robots. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems IROS2004, Sendai, Japan*, volume 2, pages 1136–1141. IEEE Publisher, September 2004.

[PI04b] F. Py and F. Ingrand. Real-time execution control for autonomous systems. In *Proceedings 2nd European Congress ERTS, Embedded Real-Time Software*, Toulouse, France, January 2004.

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

[Py05a] F. Py. *Contrôle d'Exécution dans une Architecture Hiérarchisée pour Systèmes Autonomes*. PhD thesis, Université Paul Sabatier, Toulouse (Fr), october 2005.

[Py05b] F. Py. Private communication, 2005.

[Rod00] S. Roderick. Validation of a computer-based hazard control system for a robotic payload on the space shuttle. Master of science, University of Maryland, USA, 2000.

[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*, 19(5) :12–19, 2004.

[The86] N. Theuretzbacher. Using AI techniques to improve software safety. In *Proc. IFAC SAFECOMP 86, Sarlat, France*, pages 99–105. Pergamon Press, October 1986.

[TPC01] E. Totel, B. Polle, and MC. Charmeau. Modelling an autonomous spacecraft architecture. In *Data Systems in Aerospace DASIA'2001, Nice, France*, June 2001.

[TTP+03] N. Tomatis, G. Terrien, R. Piguet, D. Burnier, S. Bouabdallah, K. O. Arras, and R. Siegwart. Designing a secure and robust mobile interacting robot for the long term. In *Proceedings of the 2003 IEEE International Conference on Robotics & Automation*, pages 4246–4251, Taipei, Taiwan, September 14-19 2003.