Nancy-Université
Université Henri Poincaré

U.F.R. Sciences et Techniques Mathematiques, Informatique et Automatique
Ecole Doctorale IAEM Lorraine
DFD Automatique

Centre de Recherche en Automatique de Nancy

UMR 7039
NANCY-UNIVERSITE
CNRS

Thèse

Présentée pour l'obtention du titre de

Docteur de l'Université Henri Poincaré, Nancy I

En Automatique, Traitement du Signal, Génie Informatique

par **Dominique EVROT**

CONTRIBUTION A LA VERIFICATION D'EXIGENCES DE SECURITE : APPLICATION AU DOMAINE DE LA MACHINE INDUSTRIELLE

Soutenue publiquement le 17/07/2008 devant le jury composé de :

Rapporteurs :		
Vincent CHAPURLAT	Maître de Conférences/HdR à l'Ecole des Mines d'Ales	LGI2P
Bernard RIERA	Professeur à l'Université de Reims Champagne Ardenne	CReSTIC

Examinateurs :		
Jean-Marc FAURE	Professeur à SUPMECA	LURPA
Lothar LITZ	Professeur à l'Université Technique de Kaiserslautern	IAC
Pascal LAMY	Docteur de l'Université de Metz	INRS
Gérard MOREL	Professeur à l'Université H. Poincaré	CRAN
Jean-François PETIN	Maître de conférences/HDR à l'Université H. Poincaré,	CRAN

Remerciements

Ces remerciements sont sans doute la partie la plus ludique qu'il m'ait été donné de rédiger pour cette thèse. Non pas parce que j'y trouvais motifs à rire, mais simplement parce que remercier les personnes qui m'ont aidé, soutenu, encouragé et supporté pendant toutes ces années me fait vraiment plaisir. Encore faudrait-il que je n'oublie personne…

Je commence donc par remercier une personne sans qui il m'aurait été difficile de réaliser ce travail et qui je l'espère se reconnaîtra.

Plus sérieusement, je tiens particulièrement à remercier Jean-François PETIN qui a su gardé le cap lorsque personnellement j'étais perdu, en proie au doute ou au pessimisme qui caractérisent les aspirants docteur en fin de deuxième année. Jean-François je crois bien que sans ta bonne humeur, tes capacités de synthèse et tes conseils avisés je n'aurais pu atteindre ce résultat.

Sans vouloir dénoncer personne, je crois aussi très sincèrement que ce travail de thèse doit également beaucoup à l'inspiration du Professeur Gérard MOREL qui nous a orienté dans les bonnes directions.

Je remercie également Pascal LAMY et Philippe CHARPENTIER de l'INRS qui ont suivi et soutenu ce travail depuis le début, et qui sont parvenus à me faire respecter une certaine rigueur dans mes activités. Les mauvaises langues diront que ce n'était pas gagné d'avance…

Comme la recherche est un travail d'équipe, je tiens à remercier toutes les personnes qui m'ont soutenu au CRAN, à l'INRS et plus récemment à l'IUT d'Aix en Provence, notamment les professeurs Jean-Michel SPRAUEL, et Jean-Marc LINARES qui m'ont accueilli au sein de leur équipe EA(MS)² me permettant de terminer ce travail sous des latitudes plus clémentes.

Je me dois également de remercier tous les membres du jury qui ont bien voulu examiner ce travail de thèse.

Quand je suis arrivé à Nancy, je ne pensais pas rencontrer autant de gens d'horizons si différents, ni m'acclimater aussi facilement à cette ville et à cette région. Je tiens donc à tous vous remercier tous les doctorants et ex doctorants que j'ai rencontré durant ces trois années Ramy, Salah, Pierrot, Max, Rémi, David, Will, Salah (le loup), Bélynda. Merci à vous tous!

Enfin je voudrais remercier mes parents ainsi que mon frère et ma sœur pour avoir fait de moi ce que je suis à présent, et à Mathilde et Elise pour m’avoir imaginé un futur.

Préambule

Les travaux de recherche présentés dans ce document ont été réalisés dans le cadre d'une convention CIFRE entre l'Université Henry Poincaré (UHP) de Nancy, plus particulièrement le Centre de Recherche en Automatique de Nancy (CRAN), et l'Institut National de Recherche et de Sécurité (INRS).

L'INRS a été fondé en 1947 par les différents partenaires sociaux français, syndicats et patronats, avec comme mission de réduire le nombre d'accidents du travail, nombre qui était alors important. Pour répondre à cet objectif qui est toujours d'actualité, l'INRS s'est focalisé autour de 3 grands axes : le savoir, l'information et l'accompagnement.

Les agents de l'INRS sont en contact permanent avec les entreprises au travers des Caisses Régionales d'Assurance Maladie (CRAM) pour diffuser de l'information et du conseil visant à diminuer les risques d'accidents de travail dans ces entreprises. Dans ce contexte, une partie des activités de recherche menées à l'INRS consiste à anticiper les futures évolutions (technologiques, organisationnelles, humaines etc.) d'un secteur d'activité, afin d'en prévoir les conséquences sur la sécurité des personnes et de définir les messages et conseils qu'il sera alors nécessaire de faire passer aux entreprises.

Le travail présenté dans ce mémoire a été réalisé au sein du laboratoire Sûreté des Systèmes Automatisés (SSA), qui développe des activités d'études et d'assistance dans le secteur des systèmes manufacturiers de production et des machines industrielles. L'encadrement de ce travail a été assuré à l'INRS par M. Pascal Lamy et par M. Philippe Charpentier, responsable de ce laboratoire.

Au sein du cran ces travaux se sont déroulés dans le cadre du projet Sûreté de Fonctionnement et Diagnostic des Systèmes (SURFDIAG) en lien avec le Groupement d'Intérêt Scientifique Surveillance Sûreté et Sécurité des Grands Systèmes (GIS 3SGS). L'encadrement de ce travail a été assuré par Jean François Pétin, maître de conférences habilité à diriger les recherches et Gérard Morel, Professeur.

Table des matières

Table des figures

Introduction Générale

L'introduction des nouvelles technologies de l'information et de la communication dans les systèmes automatisés entraîne un accroissement de la complexité des fonctions qu'ils supportent (commande, surveillance, maintenance voire gestion technique). Une étude réalisée par (Johnson 2004) a montré que cet accroissement de la complexité a un impact sur la disponibilité, la sûreté de fonctionnement et la sécurité des systèmes (Figure 1). En effet, leurs propriétés ne sont plus réductibles aux propriétés de leurs constituants pris isolément (composants logiciels de commande, circuits électromécaniques ou électropneumatiques, actionneurs et capteurs, etc.) mais émergent d'un réseau d'interactions entre ces constituants qui peut être à l'origine de comportements néfastes et difficiles à prévoir (Chapurlat 2007). Dans le domaine de la sécurité des machines industrielles, cible applicative de l'INRS, cette évolution se traduit par le transfert des applications de commande intégrant des fonctions de sécurité traditionnellement réalisées en logique câblée vers des Automates Programmables Industriels de Sécurité (APIdS). Dans le cadre d'études prospectives visant à anticiper les besoins et trouver de nouvelles méthodes de conception, l'INRS a souhaité approfondir les mécanismes de preuve du code embarqué dans les APIdS.

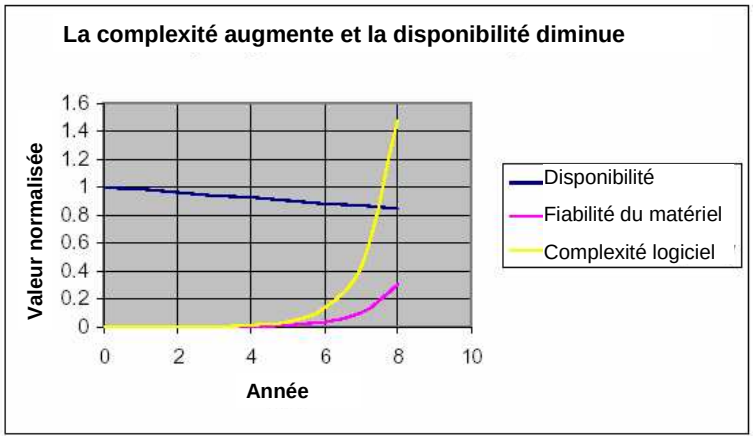


Figure 1 Scénario d'évolution de la complexité et de la disponibilité (Johnson, 2007)

Notre conviction est que le développement sûr de ces systèmes doit combiner des approches pragmatiques orientées « système », qui tiennent compte du facteur d'échelle réel d'une automatisation pour appréhender le fonctionnement global du système et son architecture, avec des approches plus formelles qui permettent de s'assurer que les propriétés intrinsèques des constituants contribuent efficacement au respect des exigences « système » formulées par les utilisateurs. Ce constat est renforcé par les référentiels normatifs dans le domaine de la sûreté de fonctionnement des systèmes embarquant du logiciel ou de l'électronique, tels que le standard IEC 61508, qui fournissent un certain nombre de recommandations relatives à :

- la nécessité de tenir compte de l'environnement de la commande afin d'appréhender les propriétés de sécurité au travers des différents constituants (partie opérative, interface avec l'opérateur, partie commande câblée, partie commande programmée, …) sollicités par les dispositifs de protection (barrière immatérielle, dispositifs de commande sécurisée, arrêts d'urgence, …),
- l'utilisation de composants logiciels tels que les composants sur étagère (COTS) permettant d'augmenter la lisibilité de l'application et de faciliter leur validation, en particulier pour les composants supportant une fonction de sécurité,
- l'utilisation de méthodes formelles de développement notamment pour les applications critiques.

Pour les aspects sécurité des machines en phase de conception, la Directive Machine 98/37/CE a été déclinée en un référentiel normatif, notamment ISO 12100 et CEI 62061, qui donne la démarche à suivre et reprend les deux premières recommandations énoncées ci-dessus. Dans ce contexte, le problème initial de vérification de code posé par l'INRS, a été élargi dans le cadre des processus techniques de l'ingénierie « système »[1], notamment des processus de définition du système et de validation & vérification (Sheard, 2006), afin de mieux prendre en compte l'analyse et la définition des exigences de sécurité préalablement à toute activité de vérification (IEEE 1220). Le travail présenté dans ce mémoire a donc pour objectif de définir une approche méthodologique permettant l'identification, la formalisation et la structuration d'exigences globales relatives à un système, puis leur projection, sous forme de propriétés invariantes, sur une architecture de composants. Cette approche devra intégrer des outils plus ou moins formels maîtrisables par les ingénieurs travaillant dans le

[1] Association Française d'Ingénierie Système : www.afis.fr

domaine de la machine. La vérification a priori des exigences, notamment de sécurité, repose alors, d'une part, sur un raffinement prouvé des exigences « système » permettant d'établir leur équivalence avec un ensemble de propriétés intrinsèques relatives à chacun des composants en interactions, et d'autre part, sur la vérification formelle de ces propriétés intrinsèques. D'un point de vue plus formel, l'objectif de ce travail est de mettre en œuvre des mécanismes de raffinement, d'allocation et de projection des exigences permettant d'établir le prédicat suivant :

$$\prod_{i=1}^{n} C_i \Rightarrow \{R_k\}, k \in 0..m \quad \textbf{(1)}$$

Où R_k représente les exigences « système » et C_i les comportements de ses constituants.

Ce mémoire, développé en quatre chapitres, s'organise de la façon suivante.

Le premier chapitre présente le contexte industriel de notre étude au travers des textes normatifs qui régissent le développement de systèmes électriques, électroniques, électroniques programmables soumis à de fortes contraintes de sécurité et de sûreté de fonctionnement et des pratiques industrielles en vigueur dans le domaine de la machine. Nous montrons en particulier que, si les APIdS offrent, de par leur architecture, un certain nombre de garanties quant à la fiabilité des composants matériels, le remplacement des chaînes de sécurité par des APIdS ne peut être envisagé que dans le cadre d'un processus sûr de développement permettant la formalisation, à un niveau « système » des exigences de sécurité et leur vérification par des techniques formelles.

Le chapitre 2 est consacré au positionnement scientifique de notre travail. Nous mettons en évidence la complémentarité des approches et modèles orientés « système » (Meinadier, 2002) et des approches formelles de vérification telles que le *model-checking* (Clarke *et al*, 1999) ou le *theorem-proving* (Hoare, 1969). Les premières sont parfaitement adaptées à l'identification et la structuration des exigences fonctionnelles et non fonctionnelles des systèmes mais souffrent d'un manque de formalisation excluant toute possibilité de vérification formelle. Les secondes offrent, quant à elles, des mécanismes efficaces pour la preuve de propriété mais sont souvent limitées à la vérification de propriétés relatives aux constituants élémentaires du système, et pour lesquelles l'identification et la formalisation restent délicates.

Le chapitre 3 présente notre contribution méthodologique basée sur des mécanismes de raffinement, de projection et d'allocation des exigences sur une architecture de composants matériels et/ou logiciels. Ces modèles, exprimés dans le formalisme de SysML[2] servent de base à la formalisation de propriétés invariantes, sous la forme d'axiomes en logique temporelle ou de post-conditions, que l'on cherchera à vérifier pour chacun des constituants à l'aide respectivement de techniques de *model-checking* ou de simulation. La cohérence entre les exigences énoncées à un niveau système et leur raffinement en termes de propriétés locales aux constituants est, quant à elle, vérifiée à l'aide de techniques de *theorem-proving*.

Enfin, le chapitre 4 illustre notre proposition sur un cas d'étude proposé par l'INRS relatif à l'automatisation en logique programmée des fonctions de sécurité d'une presse mécanique. Cette étude est complémentaire d'une première étude menée à l'INRS sur l'utilisation de la méthode B dans le domaine manufacturier (Abrial, 2006), dans laquelle la nécessité de poser un cadre méthodologique avait été mise en avant. Afin de faciliter la prise en compte des propriétés de sécurité identifiées lors de l'analyse SysML par les outils de simulation et de *model-checking*, nous avons développé un démonstrateur intégrant, autour d'une structure de données XML, les outils MagicDraw (SysML), ControlBuild (Simulation de Systèmes à Evénements Discrets) et UPPAAL (*model-checking*).

En conclusion de ce mémoire, nous soulignons l'intérêt de combiner les approches « système » et les techniques formelles pour identifier, formaliser, propager, vérifier, et tracer les diverses exigences de sécurité. Nous montrons que le champ d'application de notre proposition, centré initialement dans le cadre du travail de thèse sur la sécurité des machines industrielles, peut être élargi au développement sûr de systèmes complexes à logiciel prépondérant.

[2] SysML est un profil d'UML2 qui étend son champ d'application initial, le génie informatique, à l'ingénierie des systèmes. *Source OMG, http://www.sysml.org/*

Chapitre 1 : Contexte industriel

I - Introduction

Ce chapitre a pour objectif de présenter le contexte industriel de notre étude. Il s'appuie d'une part sur les différents textes normatifs relatifs au développement des systèmes manufacturiers incluant des fonctions de sécurité programmées, et d'autre part sur les pratiques industrielles mises en œuvre pour la conception de systèmes sûrs dans ce secteur d'activité. Au travers de ces deux aspects du monde industriel, nous cherchons à montrer qu'à l'heure actuelle, l'utilisation de fonctions de sécurité complexes réalisées en logique programmée dans le cadre de machines industrielles requiert la mise en place d'un processus sûr de développement couvrant les diverses phases classiques d'automatisation industrielle (spécification, conception, implantation) pour tracer les exigences de sécurité depuis leur expression à un niveau « système » incluant les éléments mécaniques, électriques et logiciels d'une machine jusqu'à leur allocation sur des composants de commande tout en proposant des moyens de validation et/ou de vérification permettant de s'assurer de leur respect.

I.1 La sécurité des personnes au travail

I.1.1 Définitions

La sécurité d'un système est définie comme l'absence de risques inacceptables (ISO CEI Guide 51, 1990). Cette définition reposant sur la notion de risque, l'objectif de mise en sécurité du système n'est atteignable que par le biais d'un processus de réduction du risque. Le risque est une combinaison de la probabilité d'un dommage (blessure physique ou atteinte à la santé) et de la gravité de ce dommage.

L'approche utilisée dans le domaine manufacturier consiste, lorsqu'il n'est pas possible de supprimer ou de réduire le risque à la source, à mettre en place des dispositifs de protections qui permettent de réduire la probabilité et ou la gravité du dommage. Pour cela on s'appuie sur l'identification des phénomènes dangereux et des zones dangereuses associées :

- Phénomène dangereux : source potentielle de dommage. Un phénomène dangereux peut être qualifié de manière à faire apparaître sa source (ex : phénomène dangereux mécanique, phénomène dangereux électrique)

- Zone dangereuse : tout espace, à l'intérieur et/ou autour d'une machine, dans lequel une personne peut être exposée à un phénomène dangereux.

La sécurité fonctionnelle (ISO CEI Guide 51, 1990) d'un système est un sous-ensemble de la sécurité globale qui dépend du bon fonctionnement des systèmes de commande relatifs à la sécurité qu'ils soient matériels ou logiciels, et du bon fonctionnement des dispositifs externes de réduction du risque. Ils doivent toutefois être associés à des exigences de sécurité fonctionnelle qui expriment le comportement que doit adopter le système dans son ensemble pour que les dispositifs de protection remplissent entièrement leur rôle. Ces exigences système de sécurité fonctionnelle doivent être réalisées en partie par le système de commande. Le problème est alors d'identifier et de définir précisément le rôle du système de commande dans la réalisation de ces exigences puis de concevoir les solutions réalisant ces exigences.

I.1.2 Dispositifs de protection

La plupart des dispositifs de protection ont pour objectif de restreindre l'accès à la zone dangereuse lorsque le phénomène dangereux y est présent. Pour se protéger contre les risques mécaniques, le secteur manufacturier en utilise un large éventail (Référence à ED 807 Sécurité des machines et des équipements de travail – Moyens de protection contre les risques mécaniques, brochure INRS 95 pages):

- les protecteurs de type capot qu'ils soient fixes ou mobiles, le choix étant conditionné par la fréquence d'accès à la zone dangereuse
- les commandes bimanuelles qui obligent l'opérateur à appuyer de manière synchrone sur deux commandes (une pour chaque main)
- les barrières immatérielles, qui détectent le passage au travers d'un plan de l'espace,
- ou encore des systèmes à faisceau laser pour détecter le franchissement des doigts dans la zone dangereuse.

Des dispositifs à base de caméra pour identifier les personnes dans une zone dangereuse et arrêter la machine sont en cours de développement.

Une des difficultés pour tous ces composants est d'avoir des temps de réponse suffisamment courts afin de ne pas avoir à éloigner exagérément l'opérateur de la zone de travail.

La mise en œuvre de ces dispositifs va dépendre de leur complexité. Par exemple un capot fixe peut simplement être encastré sur le bâti de la machine pour assurer sa fonction de sécurité alors que pour un protecteur mobile, on retient deux principes :

- Verrouillage : le protecteur doit être fermé pour que les fonctions dangereuses de la machine puissent s'accomplir et en cas d'ouverture, un ordre d'arrêt est donné.
- Interverrouillage : le protecteur doit être fermé et bloqué pour que les fonctions dangereuses de la machine puissent s'accomplir et le protecteur reste bloqué jusqu'à ce que le risque de blessure dû aux fonctions dangereuses de la machine ait disparu

Les fonctions de verrouillage et d'interverrouillage des protecteurs fixes et mobiles sont réalisées par la partie commande du système. Jusqu'à l'apparition des APIdS, elles étaient réalisées en logique câblée. De même pour les autres dispositifs à base de détection de personnes, les fonctions de sécurité étaient réalisées en logique câblée.

I.1.3 Fonctions de sécurité programmées

Les automates programmables industriels (APIs) sont apparus en France au début des années 1970. A cette époque les systèmes de commande des machines industrielles étaient réalisés en logique câblée, basée sur l'emploi de composants électromécaniques (contacteurs, relais etc.). Les perspectives offertes par les API, telles que la diminution des coûts liés au câblage, la facilité de modification et de maintenance du système de commande, ou encore la possibilité d'utiliser des fonctions plus complexes firent que rapidement ils furent utilisés dans les machines industrielles afin d'automatiser leurs fonctions. Il a été ainsi possible d'élargir le champ d'application, la complexité des traitements et, globalement, le niveau d'automatisation et de flexibilité de ces machines. A ceci près que seules les fonctions non sécuritaires étaient réalisées en logique programmée. Pour des raisons de sécurité du matériel (traitement monocanal), l'emploi d'un API pour implémenter les fonctions de sécurité était prohibé. Ainsi, jusqu'à maintenant, ces fonctions étaient toujours réalisées par la mise en place d'une chaîne électromécanique dite de sécurité. La donne a changé avec l'apparition à la fin des années 1990 des APIdS qui ont rendu possible sous certaines restrictions l'utilisation de fonctions de sécurité programmées dans les systèmes de commande des machines industrielles.

Il existe deux grandes familles d'APIdS (Kneppert et Dei-Svaldi, 2003), chacune étant conçue dans un but précis.

La première famille est composée d'automates conçus pour la commande de processus. Ils doivent assurer la disponibilité d'un processus en cours malgré une défaillance interne. Pour cela ils utilisent des architectures redondantes d'ordre 2 incluant des autotests pour la détection de défaillances, ou bien d'ordre 3 avec une procédure de vote 2 parmi 3. Ces architectures permettent de détecter la voie défaillante et de maintenir les sécurités du processus.

Les APIdS de la deuxième famille sont utilisés pour la commande de machines. A la différence des automates utilisés pour la commande de processus, ils doivent interrompre les mouvements dangereux dès qu'une voie de traitement est défaillante. Leur capacité à réagir rapidement à une défaillance est donc prépondérante. Leurs architectures sont donc étudiées pour permettre la détection de défaillances internes tout en assurant un temps de réponse très court. Le type d'architecture que l'on retrouve le plus souvent, est une architecture redondante d'ordre 2, avec comparateur permettant de s'assurer que les deux voies de traitements donnent bien le même résultat en sortie. Dans le cas contraire, les sorties prennent une valeur préprogrammée, correspondant à une mise en sécurité de la machine.

Ces automates sont les supports matériels pour les systèmes de commandes programmés incluant des fonctions de sécurité. Les logiciels qu'ils supportent peuvent donc réaliser à la fois des fonctions de sécurité et des fonctions nominales. Les blocs fonctionnels relatifs à la sécurité seront traités de manière redondante. Cette différence de traitement entre fonctions nominales et fonctions de sécurité doit être prise en compte dans la conception du logiciel de commande.

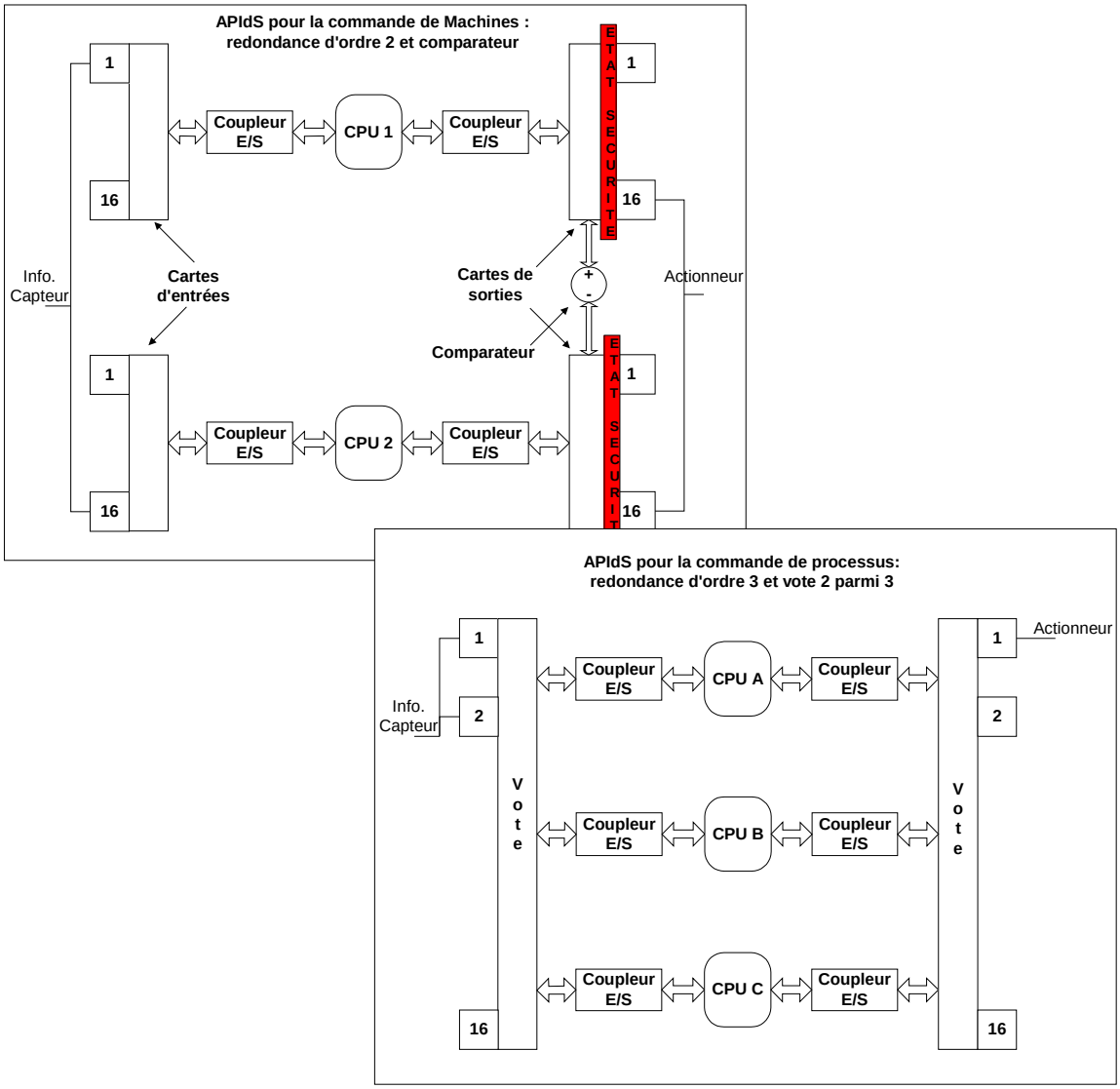


Figure 2 Structures d'APIdS

Le remplacement des chaînes de sécurité par des APIdS ne peut être envisagé qu'à la condition que le niveau de sécurité de l'ensemble automate plus logiciel soit au moins équivalent à celui des chaînes de sécurité câblées. Dans cette optique, certaines normes 'transversales' de sécurité du secteur manufacturier ont été modifiées pour inclure des prescriptions spécifiques à la réalisation de logiciel de commande intégrant des fonctions de sécurité.

I.2 Contexte normatif

La directive « machines » (Directive, 1998) concerne toutes les machines de l'industrie manufacturière. Elle définit des *exigences essentielles de santé et de sécurité* qui sont des

exigences très générales applicables à de nombreuses machines. Afin de simplifier le travail des constructeurs de machines, les normes harmonisées donnent des prescriptions de conception assurant présomption de conformité aux exigences de la directive.

Les normes relatives à la conception des machines présentent une architecture originale reposant sur des degrés d'imbrication (Charpentier et Ciccotelli, 2006). Elles sont classées selon quatre types : A, B1, B2 et C. Les normes de type A précisent les notions fondamentales, les principes de conception et les aspects généraux valables pour tous les types de machines. Les normes de type B1 traitent d'un aspect particulier de la sécurité (par exemple, distances de sécurité, température superficielle, bruit). Elles définissent les principes de base du sujet de sécurité traité et comment ces principes peuvent être appliqués aux normes de type C. Les normes de type B2 traitent d'un type de dispositif conditionnant la sécurité (par exemple, commandes bimanuelles, dispositifs de verrouillage, dispositifs sensibles à la pression, protecteurs etc.) valable pour une large gamme de machines. Elles donnent les prescriptions de performance pour la conception et la fabrication du moyen de protection considéré. Enfin les normes de type C indiquent des prescriptions de sécurité détaillées s'appliquant à une machine particulière ou à un groupe de machines. Elles traitent de tous les phénomènes dangereux significatifs concernant la machine considérée, en s'appuyant notamment sur les normes de type B pertinentes. Les normes de types B et C permettent d'orienter les constructeurs ou les rénovateurs de machines vers des solutions où l'identification et l'analyse du risque ont déjà été réalisées.

Les référentiels évoqués dans les paragraphes suivants concernent l'analyse de risque et la conception des systèmes de commande. Ils sont applicables à un grand nombre de machines de l'industrie manufacturière.

La Figure 3 donne les objectifs relatifs des textes liés à la sécurité des machines. Ces textes sont :

- La norme ISO 12100 (ISO 12100, 2004) qui donne les principes généraux de conception d'une application de sécurité. Elle propose un processus d'appréciation des risques destiné à déterminer les mesures à mettre en place pour leur réduction. C'est une norme harmonisée de type A.
- La norme ISO 14121 (ISO 14121, 1999) qui guide les concepteurs pour la réalisation de l'appréciation du risque. Elle n'aborde pas le sujet des mesures de réduction des risques. C'est également une norme harmonisée de type A.

– La norme CEI 61508 (CEI 61508, 2002) qui traite de la sécurité fonctionnelle des systèmes électriques, électroniques, électroniques programmables relatifs à la sécurité. Cette norme n'a pas été élaborée dans le cadre de la directive « machines ».

– La norme CEI 62061 (CEI 62061, 2005) qui est une déclinaison de la CEI 61508 qui est harmonisée à la directive « machines ».

– La norme ISO 13849 (ISO 13849, 2006) est une norme harmonisée de type B1 qui traite de la conception et de la réalisation des parties de systèmes de commande relatives à la sécurité.

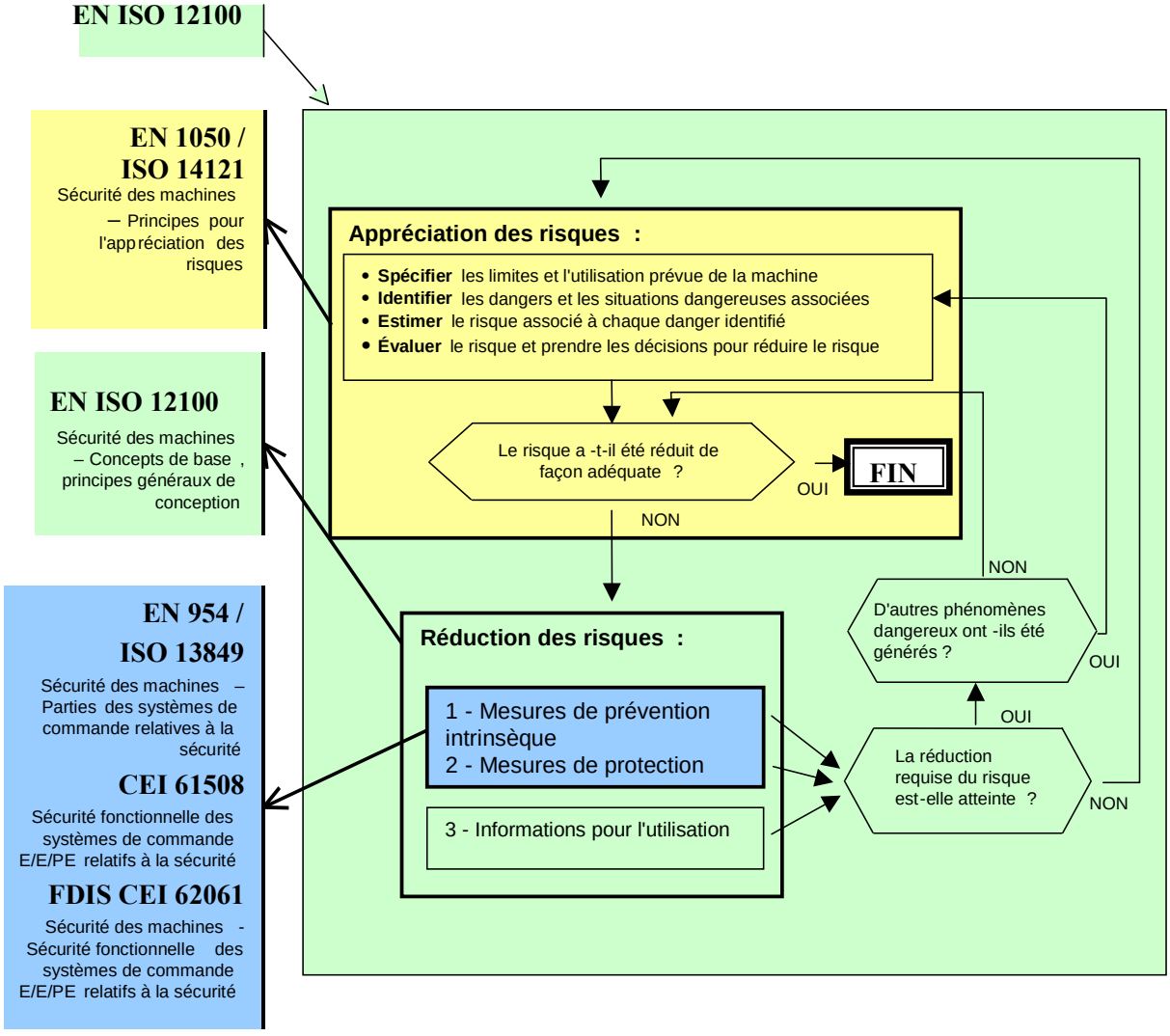


Figure 3 Normes liées à la sécurité des machines (Charpentier et Ciccotelli, 2006)

L'approche globale de prise en compte de la sécurité dans la phase de conception est définie par les normes de type A et B1. Dans le domaine de la machine, cette approche est décomposée en deux temps. Dans un premier temps les normes ISO 12100-1 et ISO 14121 précisent la démarche permettant d'identifier les mesures de réduction du risque à partir des spécifications fonctionnelles du système. Dans un deuxième temps, les normes CEI 62061 et ISO 13849 prescrivent des outils et méthodes pour la réalisation de ces mesures. Ces normes s'appuient sur un ensemble de termes relatifs à la sécurité des machines.

Il existe donc ainsi un certain nombre de normes traitant de la sécurité machine, si bien que l'on est en droit de se poser des questions sur la cohérence entre tous ces textes (Blaise et al, 2003) ainsi que sur le résultat obtenu. L'empilement de textes et de recommandations normatifs est il suffisant pour garantir la sécurité des systèmes manufacturiers ?

Les paragraphes suivants analysent l'impact de ces recommandations normatives, d'une part, sur le processus technique d'identification du risque et des exigences de sécurité, et d'autre part, sur le processus de conception et les mettent en regard des pratiques industrielles en vigueur. Enfin, au regard des précédents paragraphes, nous conclurons sur les différentes contraintes identifiées - normatives, technologiques, et techniques – que doit respecter la conception d'un système de commande intégrant des fonctions de sécurité réalisées en logique programmable pour une machine de l'industrie manufacturière.

I.3 Synthèse

L'évolution des automatismes permet d'avoir des machines de plus en plus performantes du point de vue des fonctions de sécurité. En effet, celles-ci sont de plus en plus intégrées dans la commande du système au point que les fabricants voient comme une nécessité le passage à des fonctions de sécurité programmées. De plus, ces fonctions sont de plus en plus complexes permettant ainsi une mise en œuvre plus fluide de la machine (commande des cycles par l'intermédiaire des barrières immatérielles, …)

Toutefois, malgré ces évolutions, on a répertorié durant l'année 2000 en France (Statistiques Financières, 2004) environ 700 000 accidents du travail entraînant un arrêt de travail supérieur à 3 jours, dont 300 000 pour les secteurs industriels utilisant des systèmes automatisés. Ces accidents ne sont pas tous dus à des problèmes de sécurité sur des systèmes automatisés, mais ce chiffre souligne tout de même, que malgré les évolutions technologiques, un certain nombre de risques subsiste.

Ils sont dus pour la plupart à 3 phénomènes : une utilisation non prévue du système, une mauvaise intégration de la sécurité dans le développement du système, et un manque d'utilisation des retours d'expérience (Fadier et De La Garza, 2006) auxquels il faut ajouter un facteur humain: l'opérateur intervient dorénavant de façon prépondérante dans des modes de fonctionnement dégradés, ce qui paradoxalement complexifie les interventions humaines et l'interface homme machine.

Ces travaux de thèse tentent d'apporter une contribution au point numéro 2, à savoir la prise en compte de la sécurité des personnes dans le développement des systèmes automatisés, dans le but d'anticiper les évolutions futures de ces systèmes.

En ce sens, les normes recommandent quasiment de façon unanime l'intégration et la prise en compte du risque lors de la conception des systèmes, ainsi que l'utilisation d'un certain nombre de méthodes et d'approches permettant d'atteindre cet objectif.

II - Processus de réduction du risque

L'identification des exigences système de sécurité fonctionnelle, ne peut être réalisée qu'à partir d'un processus d'analyse et de réduction du risque. Dans le domaine manufacturier, ce processus, qui doit être conduit en parallèle de la spécification et de la conception des parties nominales du système, est encadré par les normes ISO 14121 et ISO 12100 et supporté par des méthodes d'analyse classique. Dans la suite de nos travaux nous distinguerons les fonctions nominales d'un système qui remplissent les attentes du client, et les fonctions de sécurité, ou relatives à la sécurité qui ont pour but de réduire les risques liés au système tout en préservant au maximum les fonctions nominales.

II.1 Appréciation et réduction du risque : prescriptions normatives

L'analyse de risque consiste à identifier les phénomènes dangereux, afin de délimiter une zone dangereuse, puis d'estimer la probabilité et la gravité des dommages susceptibles d'être causés par ce phénomène dangereux. Ce processus d'analyse est donné par la norme ISO 14121 (Figure 4). Prenons comme exemple une presse mécanique, le phénomène dangereux mécanique principal est l'écrasement. La zone dangereuse se situe entre l'outil et la matrice. L'estimation du risque dépendra de la probabilité de présence de l'opérateur dans la zone dangereuse et de la gravité du dommage qu'il pourrait subir. Cette estimation ne donnera pas les mêmes résultats pour une presse à chargement/déchargement manuel ou pour une presse à chargement/déchargement automatique étant donné que pour ces deux cas les probabilités que l'opérateur pénètre dans la zone dangereuse sont différentes.

Si l'appréciation du risque met à jour un risque inacceptable, ce risque doit être réduit. La norme ISO 12100 donne les grandes lignes d'un processus de réduction du risque. L'objectif des mesures de conception qui résultent de ce processus est d'empêcher pour chacun des phénomènes dangereux, l'apparition d'un dommage. Pour cela, la définition même de risque donne deux possibilités.

La première est de réduire la gravité du dommage en rendant le phénomène dangereux inoffensif, voire en l'éliminant. On utilisera alors des méthodes de prévention intrinsèques. Ce sont des mesures qui, en modifiant la conception ou les caractéristiques de la machine, éliminent le phénomène dangereux.

La seconde est de réduire la probabilité du dommage. On utilisera alors des mesures de protection permettant d'empêcher que l'opérateur ne pénètre la zone dangereuse comme par exemple des protecteurs ou des barrières immatérielles, ou bien des mesures de protection permettant de préserver la personne du phénomène dangereux. Par exemple, des équipements individuels de protection tels que des lunettes permettent de protéger l'opérateur des phénomènes dangereux de projection.

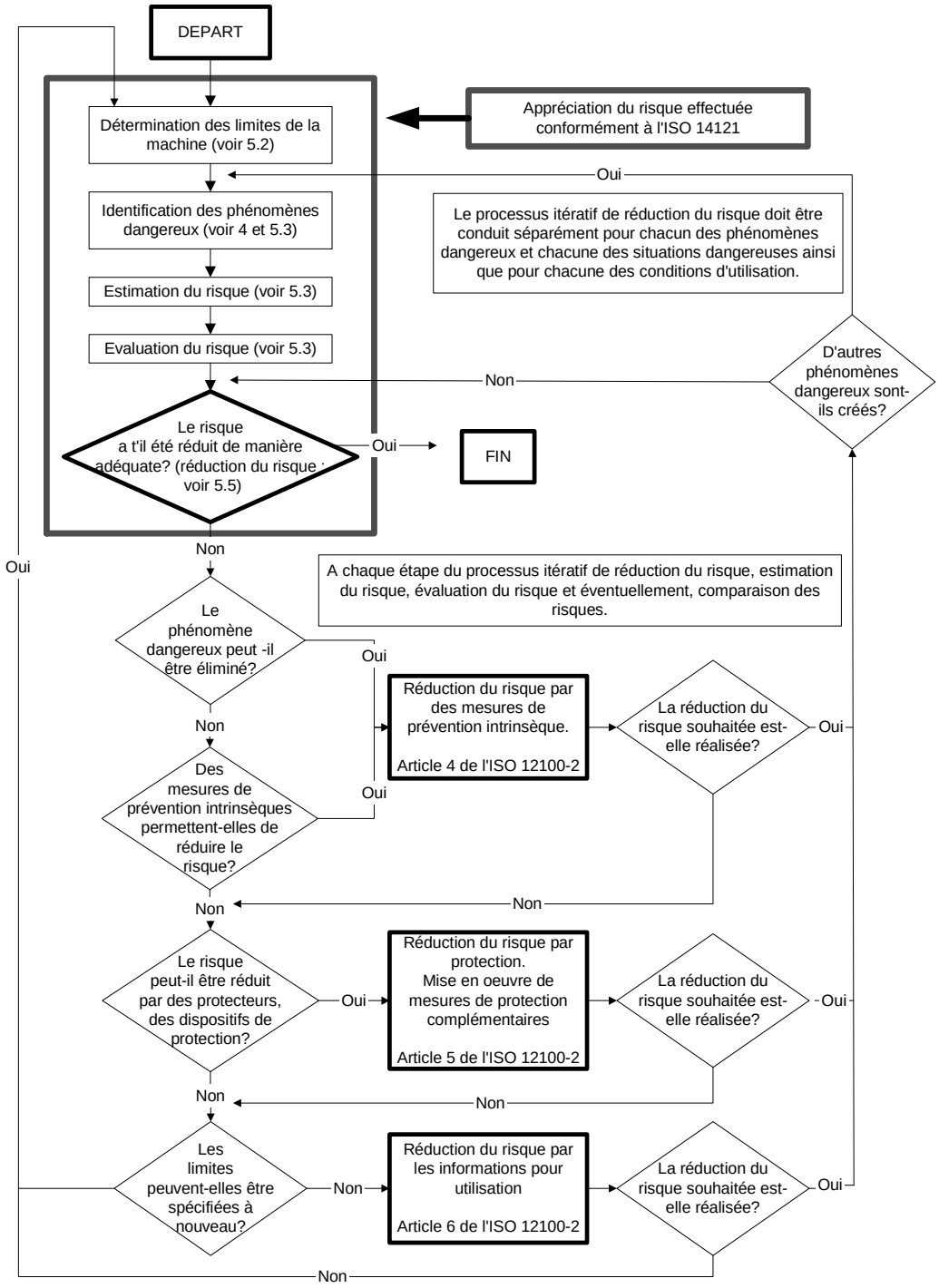


Figure 4 Processus d'analyse et de réduction du risque (normes ISO 12100 et 14121)

Le processus de réduction du risque défini par les normes ISO 12100 et ISO 14121 est donné Figure 4. Celui ci s'applique de manière itérative afin de s'assurer que les mesures de protection identifiées n'engendrent pas de nouveaux phénomènes dangereux et que le risque a été réduit de manière adéquate. Dans le cas contraire, de nouvelles mesures doivent être mises en place. On peut, à l'aide de cette méthode, préconiser de manière systématique l'emploi de dispositifs de réduction du risque pour chaque phénomène dangereux identifié. Toutefois, ces dispositifs ne sont pas à eux seuls suffisants pour assurer la sécurité du système. Ils doivent être correctement intégrés avec les autres composants et être associés à des exigences de sécurité fonctionnelle.

Ce processus de réduction de risque doit être intégré au processus global de développement du système, notamment pour les activités d'identification des phénomènes dangereux et de détermination des zones dangereuses, qui nécessitent d'avoir en entrée une connaissance *a priori* de l'architecture de la machine. C'est une activité essentielle de la spécification globale du système qui est traitée avec plus ou moins de méthode par les industriels.

II.2 Appréciation et réduction du risque : pratiques industrielles

Dans le secteur manufacturier, l'analyse des risques, le choix des protecteurs et l'identification des exigences de sécurité fonctionnelle associées est basée sur deux points : l'expérience et la réutilisation de solutions éprouvées d'une part, et les prescriptions des normes de type C d'autre part.

On peut en fait distinguer deux types de constructeurs de machines. Le premier fournit des machines sur catalogue. Le développement de ces machines est réalisé en reprenant à 95% des modèles existants et éprouvés. Dans ce cas là, l'analyse des risques est basée effectivement sur l'expérience du constructeur et sur les normes de type C relatives à la machine standard développée.

Le deuxième type de constructeurs fournit des machines spéciales. Le développement de chaque machine doit alors prendre en compte les exigences spécifiques des clients. Pour l'analyse de risque, le constructeur se base sur les prescriptions normatives de bas niveau, typiquement les normes de type B2. Ces normes sont en fait le résultat d'une analyse de risque réalisée sur une machine particulière (presse plieuse, presse mécanique etc.) ou un dispositif de protection connu (commande bimanuelle, barrage immatériel …). A partir des

résultats développés dans ces normes, le constructeur peut identifier les mesures de protection qu'il doit mettre en place pour sa machine « spéciale »

On peut donc dire que dans le secteur manufacturier, l'analyse des risques n'est pas vraiment mise en œuvre de manière systématique mais repose plus sur une combinaison de l'expérience des constructeurs et des spécifications normatives de type B2 et C.

II.3 Synthèse

Les normes d'appréciation et de réduction du risque donnent les principes de choix de dispositifs de prévention/protection pour limiter les risques jugés inacceptables lors de l'analyse de risque. Ces dispositifs sont associés à des exigences de sécurité fonctionnelle, qui contraignent le comportement global du système à réaliser.

Ces normes d'appréciation du risque sont mises en application à l'intérieur de normes plus spécifiques : les normes de type B2 et C. Ces normes, ajoutées aux retours d'expérience constituent pour les constructeurs les bases du choix des dispositifs de protection et de l'identification des exigences de sécurité fonctionnelle associées.

Ce type de méthode est suffisant dans un secteur où les évolutions sont relativement lentes, et dans lequel les normes sont mises à jour avec une fréquence suffisante pour qu'elles tiennent compte de ces évolutions. Toutefois, l'introduction d'une technologie programmable pour la réalisation des fonctions de sécurité nous amène à penser que le rythme des évolutions va s'accélérer, ce qui remettrait en cause le modèle d'analyse des risques basé sur l'expérience et les normes de bas niveau.

III - Processus de développement

L'appréciation globale du risque au niveau de la machine ayant été réalisée, il est nécessaire d'identifier les exigences « système » de sécurité fonctionnelle puis de spécifier et concevoir la contribution que doit apporter chacune des parties du système de commande à ces exigences. Dans le cas d'un système de commande programmé, cette activité est encadrée par un certain nombre de prescriptions normatives qui portent essentiellement sur les méthodes de développement à mettre en place pour assurer que le système de commande participe de manière efficace à la réalisation des exigences système de sécurité fonctionnelle. Ces méthodes doivent être adaptées aux spécificités de la cible finale : l'APIdS.

III.1 Cycles de développement

III.1.1 Cycles de vie classiques

Les processus de développement sont constitués de phases et correspondent aux premières contributions significatives pour organiser le développement de produits ou de systèmes. Ils mettent en évidence les activités nécessaires, leur ordonnancement, les produits obtenus. De nombreux travaux visant à proposer des structures pour ces cycles de développement ont été menés depuis les années 1970.

Le cycle en cascade prône une structuration séquentielle des activités de développement (Royce, 1970). Son usage est délicat en cas d'erreur ; il faut refaire la phase en cours ou la phase précédente, voire une phase plus en amont. Cette démarche simple est critiquée d'une part pour son impact sur le coût d'une erreur décelée trop tardivement dans le cycle de développement, et d'autre part pour l' « effet tunnel » qu'elle induit par un délai trop long entre l'expression des besoins et la mise en exploitation du système opérationnel.

Le cycle en V (Forsberg et Mooze, 1991) est le premier cycle de développement à avoir précisé les activités de vérification et de validation. Sa structure en V, permet d'associer pour chaque activité de spécification et de conception (branche gauche du V) une activité de vérification ou de validation (branche droite du V).

Le cycle en spirale généralise le principe d'incrément et évite « l'effet tunnel » où le temps est trop long entre les premières spécifications de besoins et la livraison de tout le système (Boehm, 1986). Il introduit la notion de prototype.

Le cycle en Y dissocie les aspects techniques (à droite) des aspects fonctionnels (à gauche) pour ensuite traiter les phases de conception puis de réalisation (au centre).

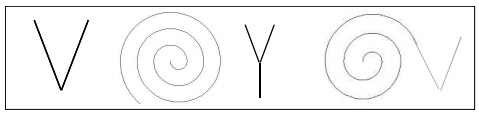


Figure 5 Cycle en V, en spirale, en Y et cycle de Figarol

On trouve ainsi dans la littérature un certain nombre de cycle de développement sur les bases des cycles en V et en spirale (Figarol *et al*, 2002). Dans les années 2000, des travaux ont été menés pour apporter une nouvelle dimension aux cycles de développement classiques, et ainsi les rendre plus adaptés aux contraintes de développement de systèmes complexes.

III.1.2 Cycles de vie pour systèmes complexes

Le développement de systèmes complexes nécessite la mise en place de cycles de développement spécifiques. Afin de faciliter l'élaboration et la compréhension du système global, ces cycles doivent proposer un développement incrémental.

Le cycle en spirale est fondé sur cette notion d'incrément, chaque nouveau cercle de la spirale correspondant à un pas supplémentaire dans la définition du système final. Le cycle de développement débute au centre de la spirale, puis se déroule le long de cette dernière en passant par un certain nombre d'activités de développement. Ces activités sont définies par des « secteurs angulaires » de la spirale. Ainsi à chaque nouveau pas du développement (comprendre à chaque nouveau cercle de la spirale), les mêmes activités sont effectuées.

Le cycle en spirale défini Figure 6 (Boehm et Hansen, 2001)(Boehm et Port, 2001) s'adresse tout particulièrement au développement de systèmes complexes intégrant des logiciels réalisant des fonctions de sécurité. On y retrouve les activités d'analyse des risques, de conception, de vérification et validation, de revue, toutes définies sur un secteur particulier du cycle. Plus on s'éloigne du centre de la spirale et plus les objets manipulés sont concrets et détaillés. Ce cycle de développement repose sur deux concepts : le développement incrémental, et la séquence d'activités nécessaire à la réalisation d'un incrément.

Le « dual vee-model » Figure 6 (Forsberg et al, 2005) propose une approche basée sur l'architecture du cycle en V mais plus proche du cycle en spirale. Il définit en fait deux types de V, un V pour les niveaux du système (système, sous-système, composants) et le cycle de développement en V classique qui sera appliqué à chacun des niveaux du système. Ce cycle permet d'appréhender à la fois l'architecture du système et son développement.

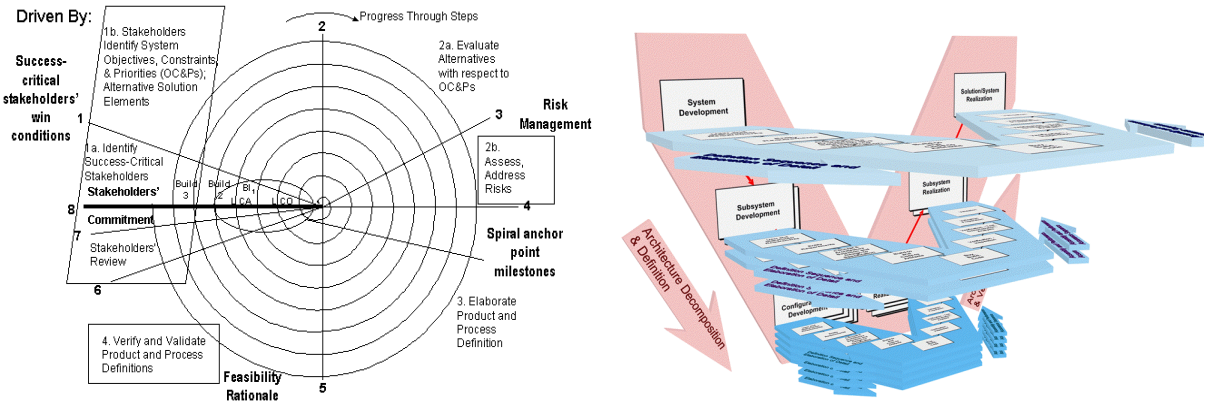


Figure 6 « Win Win Spiral Model » et Dual Vee Model

Les cycles de développement permettent de structurer l'ensemble des activités d'ingénierie d'un système. Dans le cas de systèmes simples, cette structure est en réalité un guide méthodologique pour le développement du système. Toutefois ces représentations sont trop contraignantes (Ménadier, 2002) et ne prennent pas en compte le fait que bien souvent les activités de développement recouvrent plusieurs phases du cycle de développement. On peut citer par exemple les activités de vérification et de validation des exigences de sécurité.

III.1.3 Approche par processus

L'approche par processus est basée sur le constat que quelle que soit la stratégie employée pour le développement d'un système, les activités permettant ce développement restent les mêmes. Les processus techniques sont donc basés sur les différentes activités de l'ingénierie système. Ils sont scindés en deux catégories, les processus de définition du système, et les processus d'IVV (intégration vérification et validation). Ils sont définis par les normes d'ingénierie système (IEEE 1220, EIA 632, IS0 15288).

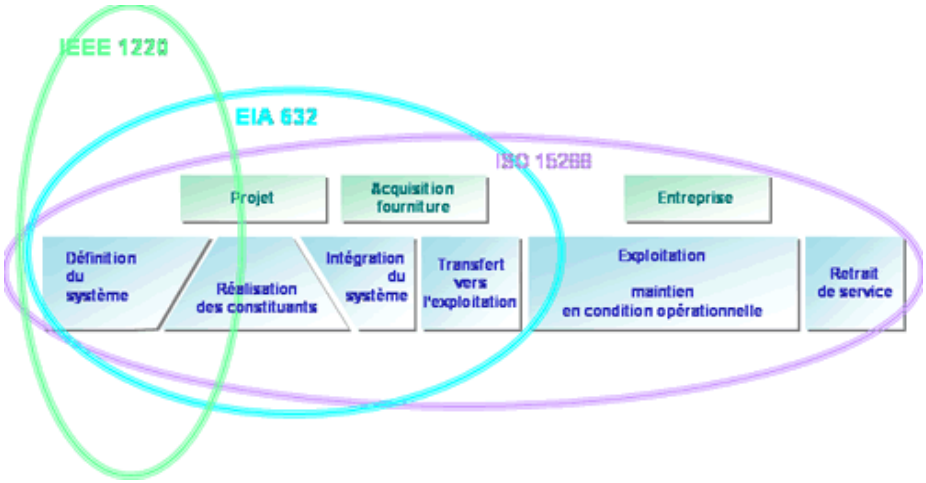


Figure 7 Couverture des normes d'ingénierie système (www.affis.fr)

En ce qui concerne les aspects ingénierie système, les travaux présentés dans ce mémoire se concentrent sur la définition du système, nous avons donc choisi de nous appuyer essentiellement sur la norme IEEE 1220.

L'approche par processus étant plus flexible que les cycles de développement en v et en spirale, elle s'adapte donc mieux aux différents contextes industriels du domaine manufacturier. De plus, du point de vue de l'INRS et de son rôle d'assistance, il est plus intéressant de conseiller les entreprises avec une vision processus qui ne contraint pas la séquence des activités de développement plutôt que d'imposer une structure de développement basée sur un cycle de développement particulier. Nous faisons donc le choix dans notre étude de ne plus parler de cycle de développement mais uniquement de processus et de boucles de développement.

Nous allons donc proposer un processus de développement global basé sur deux boucles : une boucle d'ingénierie système et une boucle de conception/réalisation. La boucle d'ingénierie système sera focalisée sur la définition des architectures d'exigences et de composants du système à faire, tandis que la boucle de conception-réalisation aura comme objectif la conception comportementale et la réalisation des composants.

III.2 Processus de définition du système

L'ensemble des processus participant à la définition du système forme la boucle d'ingénierie système. Cette boucle est réalisée de manière itérative sur les différents niveaux

de décomposition du système. Elle est composée du processus de définition des exigences et du processus de conception.

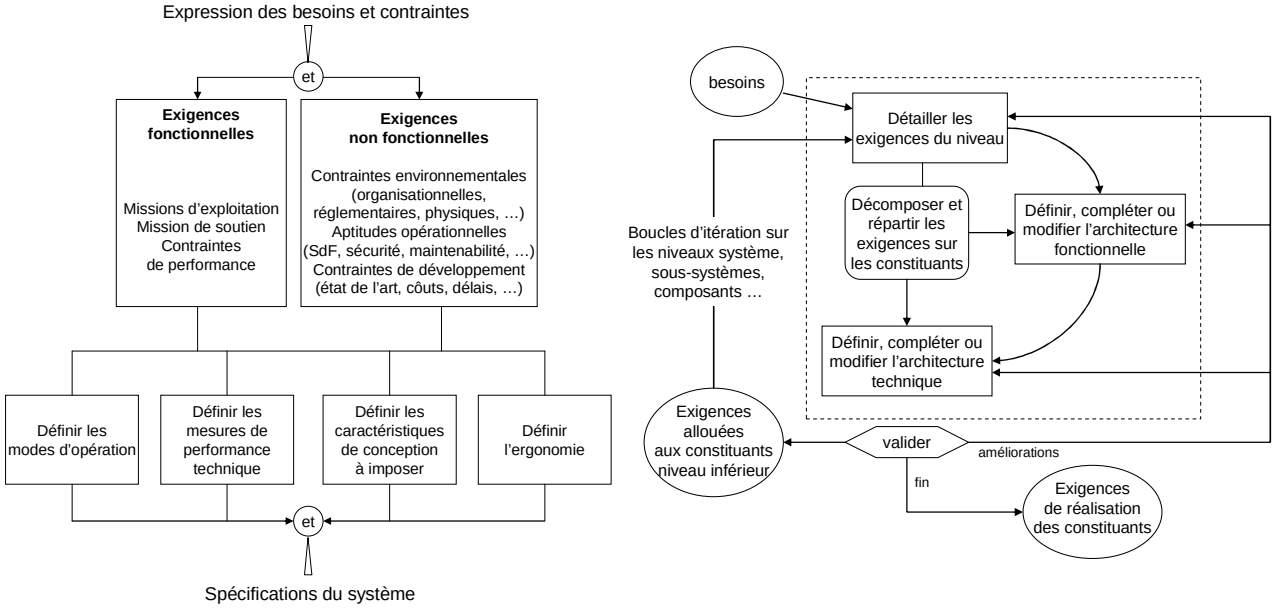
Une exigence est un énoncé qui prescrit une fonction, une aptitude ou une caractéristique auxquelles doit satisfaire un produit ou un système dans un contexte donné. L'ensemble cohérent et complet des exigences est appelé spécification notamment lorsqu'il est validé et agréé par les parties prenantes. (IEEE 1220)

Le processus de définition et d'analyse des exigences repose sur l'expression, sous forme d'un cahier des charges, des besoins et contraintes des parties intéressées (utilisatrices et exploitantes). Ils s'expriment en termes d'objectifs, de missions ou encore de performances, assignés au système à développer au travers de scénarios d'utilisation, de fonctions ou services attendus ou encore de cas d'utilisation.

Afin de garantir l'adéquation de la solution développée à l'ensemble des besoins et contraintes, le processus de définition et d'analyse des exigences a pour objectif de produire des modèles prescriptifs raffinant l'ensemble des besoins définis dans le cahier des charges en vérifiant leur faisabilité, en les hiérarchisant et en ajoutant les exigences des parties prenantes concernées par le développement de la solution. Ces modèles peuvent être formalisés sous la forme d'exigences fonctionnelles traduisant le besoin (ce que doit faire le système) et d'exigences non fonctionnelles traduisant des contraintes imposées au système (Figure 8a).

L'intérêt de distinguer ces différents types d'exigences est de permettre une structuration progressive du processus d'analyse et de réduction du risque de manière duale au processus d'identification des exigences fonctionnelles. Toutefois, il est à noter que, dans le domaine particulier de la machine, l'imbrication des fonctions de commande relatives à leur exploitation et de celles relatives à leur sécurité rend plus délicate cette distinction.

La spécification des exigences est considérée comme un processus continu et itératif dans la mesure où les choix de conception peuvent générer de nouvelles exigences globales à prendre en compte au niveau « système » ou détaillées, résultant de l'allocation (ou de la répartition) des exigences « système » sur les sous fonctions et constituants (Figure 8b). Ceci impose donc de définir et d'analyser l'ensemble des exigences de sécurité tant au niveau « système » qu'au niveau de l'ensemble de ses constituants.



a) typologie des exigences (IEEE 1220) b) boucle de spécification des exigences (Ménadier 2002)

Figure 8 Spécification des exigences

Le processus de définition des exigences regroupe trois sous processus (Figure 9) : l'analyse des exigences des parties prenantes, spécification des exigences « système » et enfin validation/vérification.

L'analyse des exigences des parties prenantes consiste à explorer le problème à résoudre en analysant les besoins et les contraintes des parties prenantes.

Le deuxième sous processus du processus de définition des exigences consiste à transformer les exigences des parties prenantes en un ensemble cohérent d'exigences techniques non ambiguës, faisables et vérifiables. Enfin le dernier, le processus de vérification et validation consiste à montrer que les exigences techniques spécifiées correspondent bien au besoin exprimé par les parties prenantes (activité de validation), qu'elles sont cohérentes, non contradictoire et qu'elles ont été établies dans les règles de l'art (activité de vérification).

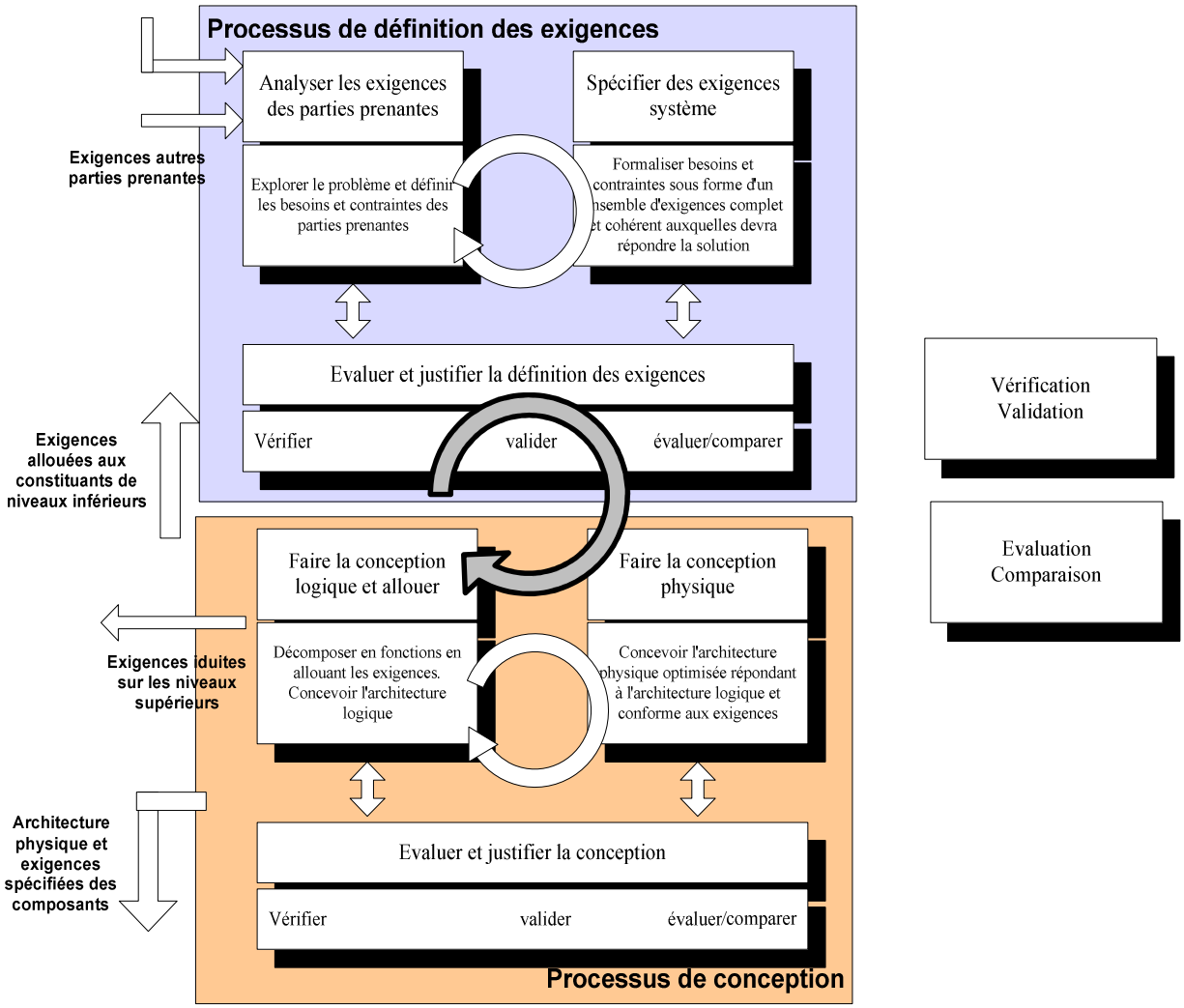


Figure 9 Boucle d'ingénierie (Meinadier 2002)

Le processus de conception regroupe également trois sous processus. Dans un premier temps la conception logique qui crée une architecture logique pour le système grâce à des méthodes d'analyse fonctionnelle. Les exigences fonctionnelles issues du processus d'identification des exigences sont satisfaites par des fonctions particulières. Ensuite la conception physique du système consiste à réaliser une architecture de constituants aptes à réaliser les fonctions de l'architecture logique tout en étant conformes au référentiel d'exigences. Enfin le processus d'évaluation et de justification de la conception permet de montrer que la solution physique vers laquelle on converge est réalisée sans erreurs (activité de vérification), notamment entre les différents passages dans la boucle d'ingénierie, qu'elle correspond au besoin (validation) et qu'elle est optimisée (évaluation comparaison).

III.2.1 Prescriptions normatives

Les exigences de sécurité fonctionnelle sont issues du processus d'appréciation et de réduction du risque défini par la norme ISO12100. A l'intérieur du processus les 3 activités de réduction du risque permettent de définir des exigences pour la prévention du risque, et la protection et l'information des personnes.

En affinant les définitions de la norme ISO 12100, ces exigences peuvent être classées en plusieurs types :

- Les exigences de prévention lors de la conception. Ces exigences visent à réduire ou à supprimer le risque à la source. Elles portent sur les parties matérielles du système et ont vocation à conduire le concepteur vers des choix de composants sûrs, comme l'utilisation d'une électrovanne double corps (redondance matérielle) pour la commande d'un mouvement dangereux ou l'utilisation d'un APIdS, l'utilisation de matériaux peu nocifs etc.
- Les exigences de protection simples. Ces exigences imposent la mise en place de dispositifs de protection pour assurer la sécurité des personnes. Ces dispositifs de protection n'ont aucun impact sur le comportement du système. Il s'agit en règle générale de carters, ou de protecteurs fixes.
- Les exigences de protection comportementales. Ces exigences imposent la mise en place de dispositifs de protection reliés au système de commande. Ces dispositifs sont associés à un comportement spécifique au système. On trouve par exemple les protecteurs inter-verrouillés avec un mouvement dangereux (le protecteur ne peut être ouvert que lorsque le mouvement dangereux est arrêté).
- Les exigences de prévention lors de l'utilisation. Ces exigences de prévention associées à un comportement spécifique du système, ont pour but de prévenir le risque d'accident. On peut citer notamment l'utilisation de vitesses lentes pour les mouvements dangereux, la mise en place d'informations visuelles et sonores etc.

Les exigences d'information pour l'utilisation. Ces exigences, mises en place en dernier recours, permettent d'informer l'utilisateur sur les comportements dangereux de la machine. Elles imposent la rédaction de notices ou autres textes explicatifs ainsi que la mise en place d'informations visuelles et sonores.

III.2.2 Pratiques Industrielles

Nous avons vu au paragraphe II.2 que les industriels s'appuient essentiellement sur leur expérience et sur les normes de type C pour l'analyse et la réduction du risque. Il en va de même en ce qui concerne la définition du système dans son ensemble, étant donné que ces normes proposent des solutions fonctionnelles pour des types de machine bien définis. Dans le cas des machines spéciales, les fabricants se reposent cette fois sur leur expérience pour définir précisément le système qu'ils vont réaliser.

III.2.3 Synthèse

Les processus définis mettent en œuvre des activités de validation à tous les niveaux permettant de s'assurer en permanence que la définition du système solution ne diverge pas par rapport aux besoins exprimés par les parties prenantes. L'enchaînement des processus de définition des exigences et de conception de la solution dans la boucle d'ingénierie système est théorique, dans la réalité ces deux processus sont très imbriqués du fait des rétroactions des choix des contraintes de conception sur la spécification du problème.

Ces processus débouchent sur la spécification d'un ensemble de composants structurés grâce à un modèle d'architecture. Ces composants doivent être ensuite achetés ou réalisés, suivant s'ils existent ou non sur le marché, avant d'être soumis au processus d'intégration, vérification, validation.

III.3 Processus de réalisation des systèmes de commande

III.3.1 Prescriptions normatives

La norme CEI 61508 est un texte ambitieux qui traite exclusivement des systèmes utilisant des technologies, électriques, électroniques, électroniques programmables et peut donc être utilisée pour les systèmes de commandes relatifs à la sécurité. C'est un texte très complet traitant de tous les aspects du cycle de vie d'un système. Pour la partie conception, la démarche proposée comprend trois étapes (Buchweiller, 2003) : premièrement fixer des objectifs, en terme de performance de sécurité, par rapport à l'estimation de la contribution aux exigences système de sécurité fonctionnelle, deuxièmement concevoir le dispositif en

intégrant et en validant ces objectifs, et troisièmement vérifier que les objectifs fixés ont été effectivement atteints par les dispositifs réalisés. Ce texte définit quatre niveaux de prescription d'intégrité de sécurité (SIL – Safety Integrity Level) qui vont jouer sur les méthodes utilisées dans la conception et la réalisation des différentes parties du système de commande (matérielles ou logicielles). On appelle prescriptions d'intégrité de sécurité, toutes les prescriptions qui visent à augmenter la qualité du développement des systèmes relatifs à la sécurité. Ces prescriptions portent de manière générale sur les méthodes et outils à mettre en œuvre pour la réalisation des exigences de sécurité fonctionnelle.

La norme CEI 61508 est texte très général qui est en réalité décliné dans beaucoup de domaines par une norme sectorielle plus adaptée aux usages du secteur d'application. Pour le secteur manufacturier, elle est déclinée par la norme CEI 62061. Ce texte propose une approche globale pour la réalisation de systèmes de commande électriques, électroniques, électroniques programmables. Il reprend les prescriptions d'intégrité de sécurité de la norme CEI 61508. Parmi ces prescriptions, on trouve à la fois des exigences sur les méthodes et outils à employer mais également des exigences de fiabilité à atteindre. Dans cette étude, nous nous intéressons essentiellement aux exigences portant sur les méthodes de développement du logiciel, le logiciel ne faisant pas l'objet d'analyses fiabilistes.

Au niveau méthodologique, ce texte propose une décomposition des exigences de sécurité fonctionnelle en fonctions qui sont réalisées par un processus d'intégration de composants. Les différents types de fonctions et de composants que l'on rencontre dans un système de commande selon ce texte sont définis ici :

Fonction de commande : *Fonction qui évalue les informations ou signaux d'entrée et génère des activités ou informations de sortie.*

Fonction de sécurité : *Fonction d'une machine dont la défaillance peut provoquer un accroissement immédiat du (des) risque(s).*

Système de commande électrique relatif à la sécurité (SRECS) : S*ystème de commande électrique d'une machine dont la défaillance peut provoquer un accroissement immédiat du (des) risque(s).*

Fonction de commande relative à la sécurité (SRCF) : F*onction de commande avec un niveau d'intégrité de sécurité réalisé par un SRECS, prévue pour maintenir la condition de sécurité de la machine ou empêcher un accroissement immédiat du (des) risque(s).*

Cette norme est avant tout une norme d'intégration, qui vise à donner une démarche de conception des logiciels relatifs à la sécurité basée sur l'intégration de composants logiciels existants. Elle se décompose en trois étapes.

La première étape a pour objectif d'identifier les SRCF et de leur allouer un SRECS. L'identification des SRCF est réalisée à partir des exigences de sécurité fonctionnelle issues du processus d'appréciation et de réduction du risque. Le texte ne donne aucune indication sur la façon de réaliser cette identification. L'idée principale de cette première étape est de poser les bases de la décomposition fonctionnelle du système ainsi que de son architecture de composants. Cette décomposition peut être réalisée en utilisant des modèles d'analyse fonctionnelle tels que SADT (Kneppert, 1986).

La deuxième étape consiste à décomposer les SRCF identifiées en blocs fonctionnels. Les prescriptions d'intégrité de sécurité imposent de détailler les caractéristiques propres à chacun des blocs. En parallèle, l'architecture du système est affinée, les SRECS sont décomposés en sous-systèmes auxquels il va falloir allouer les blocs fonctionnels identifiés. Les sous-systèmes héritent automatiquement des prescriptions liées au niveau d'intégrité de sécurité du bloc fonctionnel. Il faut noter que si un sous système ou plus généralement un SRECS intègre à la fois des fonctions relatives à la sécurité et des fonctions nominales alors le SRECS dans son ensemble doit être considéré relatif à la sécurité sauf s'il peut être prouvé que les SRCF et les fonctions nominales sont suffisamment indépendantes dans leur réalisation (c'est-à-dire que le fonctionnement normal ou la défaillance de n'importe laquelle des autres fonctions n'affecte pas les SRCF). Il est donc intéressant, pour éviter d'avoir à traiter l'ensemble du système de commande comme relatif à la sécurité, de séparer le plus tôt possible les fonctions de sécurité et les fonctions nominales, aussi bien dans leur spécification que dans leur réalisation.

La troisième étape est la réalisation des sous-systèmes identifiés et spécifiés. Ici la norme offre deux possibilités : la réutilisation de dispositifs déjà existants qui devront non seulement montrer leur aptitude à réaliser la fonction allouée mais aussi répondre aux prescriptions de sécurité du sous-système, ou bien la conception d'un nouveau dispositif. Là encore l'aspect composant joue un rôle prépondérant puisque la conception des sous-systèmes est réalisée par un assemblage d'éléments déjà existant ou développés dans ce but. La norme met tout particulièrement l'accent sur la nécessité de bien identifier quelle part de la spécification du sous-système va réaliser chacun des éléments de ce sous-système.

III.3.2 Pratiques industrielles

De l'APIdS réalisé par un constructeur de systèmes programmable jusqu'au système de commande utilisé par l'opérateur de la machine, plusieurs acteurs participent à la conception du logiciel d'application. Chacun de ces acteurs utilise ses propres méthodes et outils pour réaliser sa partie de la conception.

III.3.2.1 Les intervenants dans le processus de réalisation

Dans le domaine manufacturier, la conception du logiciel d'application peut être définie sur 3 niveaux (Neugnot, 2002). Un niveau constructeur qui concerne le constructeur de la machine lui-même, un niveau intégrateur et un niveau utilisateur.

Le logiciel applicatif « constructeur » est réalisé par le constructeur de la machine qui y implémente le fonctionnement de la machine, la maintenance en ligne, l'interface homme machine et la gestion des sécurités (conception de blocs logiciels capables d'interpréter les entrées des organes de sécurité).

Le rôle de l'intégrateur consiste à adapter la machine à son environnement. Il va donc concevoir les modes de marche et d'arrêt, le réglage des outils, la synchronisation des différentes machines d'une chaîne, et la gestion des sécurités. A ce niveau la gestion des sécurités consiste à activer les dispositifs de protection à mettre en place pour chaque mode en utilisant les blocs de sécurité réalisés au niveau précédent.

L'utilisateur final a la possibilité de régler les différents aspects de la production (outils, programme pièce etc.) par l'intermédiaire de l'interface homme machine.

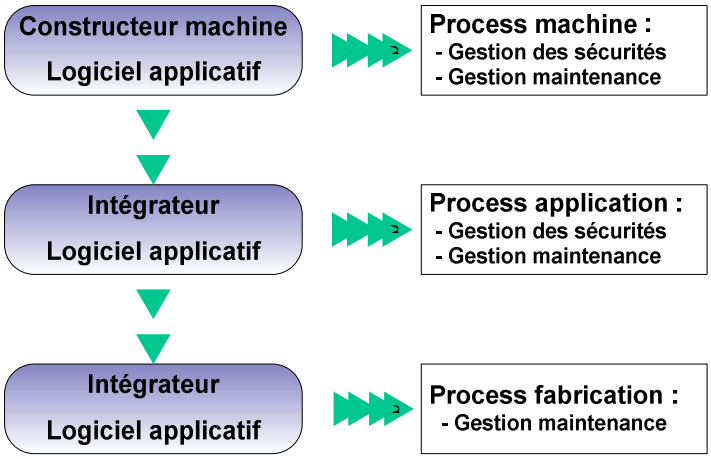


Figure 10 Résumé des différents niveaux de logiciel

Les fonctions de sécurité du logiciel applicatif sont réalisées aux niveaux 1 et 2, leur modification au niveau 3 est protégée par un mot de passe. Souvent, l'intégrateur se trouve être le constructeur, ce qui rend la conception des fonctions de sécurité plus faciles.

III.3.2.2 Intégration de composants pré-certifiés et COTS

Les composants COTS (Commercial Off The Shelf) sont des composants logiciels, qui sont conçus pour répondre à une fonction précise. En sécurité machine, ces composants sont souvent développés par les constructeurs d'automates, qui les intègrent directement dans leurs produits. Certains constructeurs, comme PILZ[3], proposent des automates spécialement dédiés à un type de machines. Ces automates intègrent alors des composants remplissant des fonctions de sécurité propres à la machine (Pilz, 1999). Les composants COTS peuvent être pré-certifiés, ce qui garantie en quelque sorte que le composant réalise effectivement la fonction pour laquelle il est conçu, et satisfait les exigences associées à la fonction.

L'utilisation de COTS justifie le processus de développement prescrit par les normes, et rentre complètement dans le contexte de développement des logiciels, avec un constructeur, un intégrateur, et un utilisateur. Le constructeur de la machine achète l'automate contenant des blocs fonctionnels pré-certifiés, l'intégrateur intègre ces blocs pour obtenir le comportement souhaité par l'utilisateur final.

L'utilisation de COTS est justifiée le plus souvent par deux types de facteurs : le facteur économique et le facteur technologique (Dawking et Riddle, 2000). Du fait de leur distribution commerciale, ces composants sont moins chers, le prix du développement et de la certification étant partagé sur un plus grand nombre d'utilisateurs finaux. D'un point de vue technologique, ils offrent aux constructeurs de machines un gain de temps considérable, car ils sont disponibles immédiatement. De plus ils permettent une mise à jour constante des fonctions de sécurité utilisées sur les machines ce qui permet de suivre d'assez près les différentes innovations technologiques du secteur manufacturier.

Pour toutes ces raisons, les composants COTS sont majoritairement utilisés dans la réalisation de systèmes de commande programmés. Toutefois ces composants proposent des fonctions figées qui ne correspondent pas toujours aux besoins de l'utilisateur, il est donc parfois nécessaire, notamment dans le cas de machines spéciales, de concevoir des composants originaux.

[3] Pilz Gmbh & Co, http://www.pilz.com

III.3.2.3 Conception de nouveaux composants

Le développement d'un composant logiciel doit être réalisé en tenant compte des spécifications du système dans son ensemble, et du comportement particulier de la partie opérative. Dans ce contexte, la conception du logiciel applicatif que ce soit au niveau constructeur ou au niveau intégrateur, fait souvent apparaître un manque de rigueur et de méthode (Neugnot, 2002)(Lamy, 2003). Les analyses fonctionnelles réalisées lors de la phase de spécification laissent généralement apparaître un manque d'exhaustivité. Les concepteurs sont donc souvent amenés à modifier les spécifications après s'être aperçus que certaines fonctions n'étaient pas correctement, voire pas du tout, définies.

A ce manque de méthode s'ajoute la difficulté pour décrire le fonctionnement de tous les composants du système. La machine intègre à la fois des composants mécaniques, des composants électriques/ électromécaniques/ électropneumatiques, et des composants programmables pré-certifiés, dont les comportements sont spécifiés par autant d'outils différents. Il est donc difficile d'obtenir une cohérence globale des spécifications de la machine.

En ce qui concerne les langages utilisés pour la spécification du logiciel, le Grafcet est de plus en plus utilisé, notamment par les jeunes embauchés. En France la programmation des logiciels d'application est majoritairement réalisée en SFC et en Ladder Diagram (EXERA, 2000), langages définis par la norme CEI 61131-3 (CEI 61131-3, 1993).

III.3.3 Synthèse

Les normes de conception, ainsi que les pratiques industrielles mettent en avant une méthode de conception des logiciels de commande basée sur l'intégration de composants. Le logiciel est réalisé sur la base de composants implémentant des fonctions simples. Ces composants doivent si possible être des composants COTS dont le comportement a été validé voire certifié. La structure du logiciel doit séparer les composants réalisant des fonctions relatives à la sécurité de ceux réalisant des fonctions nominales. Un composant réalisant ces deux types de fonctions sera considéré comme relatif à la sécurité.

Au vu du contexte industriel, on s'aperçoit que tant que l'on se cantonne à l'utilisation et l'intégration de composants pré-certifiés, la conception du logiciel de commande reste relativement aisée. Toutefois l'utilisation de ces composants est limitée, notamment par le fait que ni l'intégrateur ni l'utilisateur n'en connaîssent le comportement exact ou même le code

source. Cela signifie, pour l'intégrateur qu'il doit s'appuyer sur des composants dont le comportement n'est pas totalement maîtrisé pour la réalisation du logiciel, et pour l'utilisateur, qu'en cas de défaillance il doit faire appel directement au fournisseur de COTS pour les opérations de maintenance. D'ailleurs globalement la gestion des versions et des mises à jour peut poser des problèmes, la mise à jour de plusieurs composants pouvant faire apparaître des comportements non prévus (Redmill, 2004).

Tant que l'utilisation des COTS est limitée à des fonctions simples, leur intégration et leur maintenance présentent très peu de difficultés, mais, dès lors que ces fonctions se complexifient (gestion d'un mode de fonctionnement complexe, changement de modes etc.), leur utilisation est rendue difficile. Il est alors nécessaire, pour ces fonctions complexes, de mettre en place un processus de développement spécifique intégrant des méthodes et des outils plus robustes que ceux utilisés à l'heure actuelle par les industriels.

III.4 Processus de vérification des systèmes de commande

III.4.1 Définitions

Le processus de validation et de vérification a pour objectif de s'assurer de l'adéquation des modèles et du système développé vis-à-vis des besoins exprimés par les utilisateurs et de la spécification des exigences.

La validation est la confirmation que par examen et apport de preuves tangibles que les exigences particulières pour un usage spécifique sont satisfaites. Elle répond à la question « construisons-nous le bon modèle ? ». (ISO 8402, 1994)

La vérification est la confirmation par examen et apport de preuves tangibles que les exigences spécifiées ont été satisfaites. Elle répond à la question « construisons-nous correctement le modèle ?» (ISO 8402, 1994).

En d'autres termes, la validation cherche à s'assurer que les productions de chacun des processus de spécification, de conception ou d'implémentation (modèles et système réalisé) sont conformes aux besoins exprimés par les utilisateurs alors que la vérification cherche à s'assurer que les modèles élaborés au cours des processus de spécification et de conception sont conformes à l'ensemble des exigences spécifiées (Figure 11).

Les besoins étant le plus souvent exprimés par les utilisateurs de manière non formelle (cahier des charges textuel), les techniques de validation reposent souvent sur l'exécution symbolique des modèles (simulation et/ou émulation) et sur l'exécution de scénarios de tests sur le système réel. La vérification repose plutôt sur l'utilisation de techniques d'analyse formelle dans la mesure où les exigences spécifiées ont été formalisées.

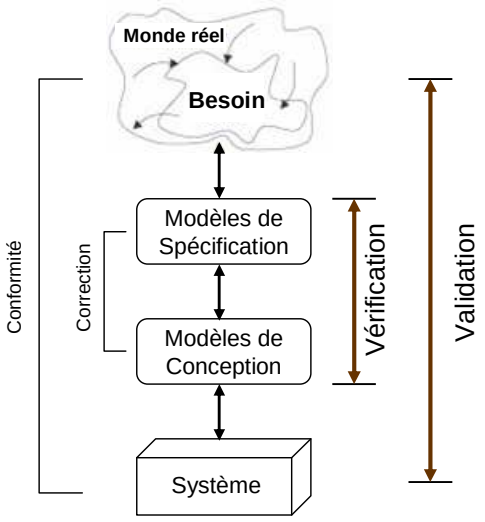


Figure 11 Processus de validation & vérification adapté de (Kamsu Foguem, 2004)

III.4.2 Prescriptions normatives

La norme CEI 62061 propose plusieurs niveaux de vérification dans le développement des systèmes de commande relatifs à la sécurité intégrant des technologies programmables. Chacune de ces activités correspond à un niveau de modularité du système de commande, on retrouve ainsi 4 phases.

Une phase de vérification de chaque module logiciel. Des outils de simulation de partie opérative peuvent être utilisés pour vérifier que le module logiciel remplit bien toutes ses spécifications. Si le module a déjà été validé dans le cadre d'une autre application, les tests peuvent être fortement réduits.

Une phase de vérification de l'intégration des modules logiciels. Les tests restent la méthode préconisée. Si ces tests révèlent des défaillances liées à l'intégration des modules, une action corrective doit être entreprise. Si cette action vise à modifier les spécifications, elle doit alors faire l'objet d'une analyse d'impact, afin d'identifier les modules logiciels, les

SRECS et les SRCF touchés et de s'assurer que l'action corrective ne contredit pas les spécifications de sécurité du système.

Une phase de vérification de l'intégration des sous-systèmes du SRECS. Là encore c'est par l'intermédiaire de tests que l'on s'assure que les sous-systèmes interagissent bien entre eux pour réaliser les fonctions pour lesquels ils sont spécifiés. La norme précise que l'utilisation lors de la conception de méthodes semi-formelles pour la structuration du SRECS peut faciliter ces tests. A ce niveau on effectue également des tests pour vérifier que les prescriptions d'intégrité de sécurité SIL sont atteintes lors de l'intégration du SRECS.

Finalement une phase de vérification du SRECS qui consiste à montrer par test ou par analyse que le SRECS réalise correctement la SRCF à laquelle il est alloué.

L'activité de vérification est donc principalement basée sur l'utilisation de test. Ces tests sont réalisés par les industriels suivant différentes méthodes.

III.4.3 Pratiques industrielles

A l'heure actuelle, les outils de simulation de partie opérative restent le meilleur moyen de tester le logiciel avant son implantation et sa validation sur site. Cependant ces outils ont eu du mal à s'imposer dans l'industrie manufacturière. En réalité c'est le contexte économique qui entraîne l'entreprise industrielle vers la simulation. Dans les grands groupes, on a assisté progressivement à la mise en place de plates-formes de simulation impliquant tous les métiers participant à la réalisation du système : automaticiens, électroniciens, mécaniciens etc. Pour de plus petites entreprises, il s'agissait le plus souvent de l'engagement d'un ingénieur qui apparaissait alors plutôt comme un visionnaire (Lebrun, 2003), ou alors d'une contrainte imposée par un grand groupe client. C'est par exemple le cas de PSA (Mauguy L., 2005), qui a choisi depuis de nombreuses années d'utiliser des outils de simulation (SIMAC de Schneider[4], Controlbuild de Geensys[5]) et qui dorénavant impose à ses sous-traitants l'utilisation de ces outils. Les outils de simulation sont donc de plus en plus présents dans l'industrie permettant ainsi de gagner du temps sur les tests, puisque le test est d'abord réalisé en plate-forme puis sur site. C'est d'ailleurs grâce à des avantages comme le gain de temps, la possibilité d'effectuer des tests plus complets et à la simulation de défaillances, qu'ils ont réussi à s'imposer.

[4] Schneider Electric, www.schneider-electric.com
[5] Tni-software, www.tni-software.org

III.4.4 Synthèse

Dans le domaine manufacturier, la vérification des logiciels de commande relatifs à la sécurité est essentiellement basée sur le test, réel ou simulé. En fonction des résultats des analyses d'impact sont effectuées afin d'identifier les modifications à effectuer sur les composants.

Ce dernier point met en avant l'importance de préserver la cohérence de l'ensemble des modèles utilisés depuis l'identification des exigences de sécurité fonctionnelles issues de l'analyse de risque jusqu'aux composants spécifiés Pour cela, il est essentiel de maîtriser l'évolution de ces exigences au travers des fonctions de sécurité et des composants qui les implémentent. L'utilisation de méthodes semi-formelles lors de la conception peut faciliter l'obtention des « traces ».

Des mesures complémentaires peuvent toutefois être utilisées. La norme 61508-3 va ainsi plus loin, en apportant énormément de méthodes et d'outils supplémentaires comme par exemple les méthodes de vérification formelles. Toutefois ces méthodes et outils ne peuvent être prescrits qu'à la condition qu'ils puissent s'intégrer aux méthodes de conception spécifiques à ce secteur.

IV - Conclusion

La réalisation de systèmes de commande programmables relatifs à la sécurité est encadrée à tous les niveaux par des textes normatifs. De manière générale, les normes traitant du développement de la partie logicielle présentent des prescriptions relatives au processus de développement. Les préconisations importantes mettent l'accent sur :

– la nécessité de tenir compte de l'environnement du logiciel, à savoir les interactions avec les parties matérielles de la machine ainsi que les interactions avec l'opérateur,

– l'utilisation de composants logiciels permettant d'augmenter la lisibilité de l'application et de faciliter leur validation, en particulier pour les composants supportant une fonction de sécurité,

– l'utilisation de méthodes formelles de développement notamment pour les applications de niveau SIL 4.

L'ensemble de ces points définit la trame d'un processus de développement sûr suffisant pour la réalisation de systèmes intégrant des fonctions de sécurité simples, réalisées en logique programmée. La question que nous nous posons, est « qu'en est-il pour les fonctions de sécurité complexes ?». Ce type de processus de développement est-il suffisant ?

De notre point de vue, la principale difficulté liée au processus de développement, ne réside pas tant dans le respect de la trame, que dans la difficulté à combler ses discontinuités. En entrée des processus préconisés par les normes, se trouvent l'appréciation et la réduction du risque. Cette étape permet d'analyser et d'évaluer le risque afin d'identifier des mesures visant à l'éliminer ou à le réduire. Toujours dans les processus préconisés par les normes, l'étape suivante est relative à la conception du système de commande incluant des fonctions de sécurité. L'identification des fonctions de sécurité assurées par le système de commande à partir des besoins identifiés par l'analyse de risque reste délicate. Elle est rendue d'autant plus difficile lorsque le système de commande intègre à la fois des composants logiciels, des composants électriques, des composants électromécaniques etc. Ce problème d'identification du rôle de chaque composant dans la réalisation des exigences systèmes, n'est pas spécifique aux systèmes de commandes programmables relatifs à la sécurité. Il est seulement accentué par la différence de langage utilisé dans la phase de spécification inhérente à l'utilisation de logiciels couplés à du matériel électrique, et est nécessairement amplifié lors du développement de fonctions de sécurité complexes.

Dans ce contexte, les composants logiciels d'un système complexe doivent être vérifiés localement afin de s'assurer qu'ils remplissent bien leurs fonctions (pour peu qu'elles aient été correctement identifiées) mais aussi globalement afin de vérifier que leurs interactions potentielles n'engendrent pas de comportements dangereux. Ce dernier point reste vrai dans le cas de composants de bibliothèque pré-certifiés dont il faut vérifier les conditions d'utilisation et d'intégration.

Si l'on se réfère à ces prescriptions normatives, il est donc nécessaire de mettre en œuvre un processus de développement intégrant :

- des modèles orientés « système » pour transformer le besoin identifié par les analyses de risque en exigences de sécurité relatives à chacun des constituants du système. Ces modèles doivent donc permettre de considérer le logiciel de commande dans son environnement en incluant les comportements induits par les parties mécaniques ou électriques de la machine afin d'appréhender le fonctionnement global de la machine et de ses dispositifs de sécurité,
- des modèles de conception orientés « métier » permettant la simulation ou la preuve des propriétés de chacun des composants logiciels supportant une fonction de sécurité afin de garantir que leurs propriétés intrinsèques contribuent efficacement à satisfaire les propriétés de sécurité identifiées au niveau « système ».

La principale difficulté réside dans la projection (et la traçabilité) de propriétés de sécurité exprimées, à un niveau « système », dans des formalismes semi-formels, sur des propriétés élémentaires relatives à chacun des composants logiciels pouvant être vérifiées par l'utilisation de modèles formels autorisant la simulation ou la preuve.

Chapitre 2 : Méthodes et modèles pour un processus sûr de développement

I - Introduction

Ce chapitre a pour objectif de préciser le positionnement scientifique de notre travail. Nous avons souligné au chapitre précédent que le développement de machines industrielles utilisant des fonctions de sécurité complexes réalisées en logique programmée n'est possible que par la mise en place d'un processus de développement sûr basé sur une approche système pour l'identification, la formalisation et la traçabilité des exigences, et sur des méthodes formelles pour leur vérification.

Au travers de l'étude de différents modèles orientés « système » couvrant l'analyse et la formalisation d'exigences de sécurité et de différentes techniques de vérification formelle (*model-checking, theorem-proving*), nous mettons en évidence :

- la complémentarité de ces deux approches, l'une facilitant l'identification et la structuration des exigences au niveau système mais souffrant d'un manque de formalisation excluant toute vérification, l'autre offrant des mécanismes de preuve efficaces dès lors que l'on est capable d'identifier les propriétés que l'on souhaite prouver, ce qui s'avère souvent difficile.
- la difficulté de combiner ces approches, difficulté issue d'une part de la différence de niveau d'abstraction (niveau système / niveau constituant) et d'autre part de la différence de formalisme (langage semi formel / langage formel).

II - Méthodes & modèles pour la définition du système

II.1 Outils pour l'analyse du risque

II.1.1 Définitions

L'occurrence d'une défaillance est définie par rapport à la fonction d'un système et non par rapport à sa spécification. En effet, si un comportement inacceptable est généralement identifié comme une défaillance en raison d'une déviation de la conformité à la spécification, il est également possible qu'il satisfasse la spécification tout en étant inacceptable pour les utilisateurs du système, révélant ainsi une faute de spécification. (Laprie, 1995)

Une erreur est définie comme étant susceptible de provoquer une défaillance. (Laprie, 1995)

La faute est une cause adjugée ou supposée d'une erreur. (Laprie, 1995)

Dans le cadre de la sûreté de fonctionnement dans son ensemble, nous nous intéressons essentiellement au problème de la sécurité-innocuité, c'est à dire la non-occurrence de conséquences catastrophiques pour l'environnement.

Les machines manufacturières rentrent pour la plupart dans la classe d'applications pour lesquelles l'inactivité est considérée comme une position naturellement sûre. Les exigences de sécurité fonctionnelle de ces machines ont donc comme objectif l'arrêt sur défaillance.

Les défaillances ont comme origine des fautes. Sur une machine manufacturière, on retrouve essentiellement trois types de fautes, des fautes dues à un mauvais comportement de l'opérateur lorsque la machine est en production, des fautes dues à la dégradation d'un élément physique lorsque la machine est en production, ou des fautes de conception. L'objectif de sécurité est alors d'empêcher que ces fautes ne se transforment en défaillances, ou dans le cas où une défaillance arrive tout de même, d'arrêter le système.

Les méthodes d'estimation et de réduction du risque mises en œuvre dans le processus de développement d'une machine doivent donc se baser sur les moyens utilisés dans le cadre de la sûreté de fonctionnement : prévision des fautes, élimination des fautes, prévention des fautes, et tolérance aux fautes. La tolérance aux fautes qui consiste à permettre à un système

d'effectuer sa fonction en dépit des fautes pouvant affecter ses composants, n'est pas utilisée pour les machines manufacturières puisque dès qu'une faute affecte un composant, la machine est stoppée.

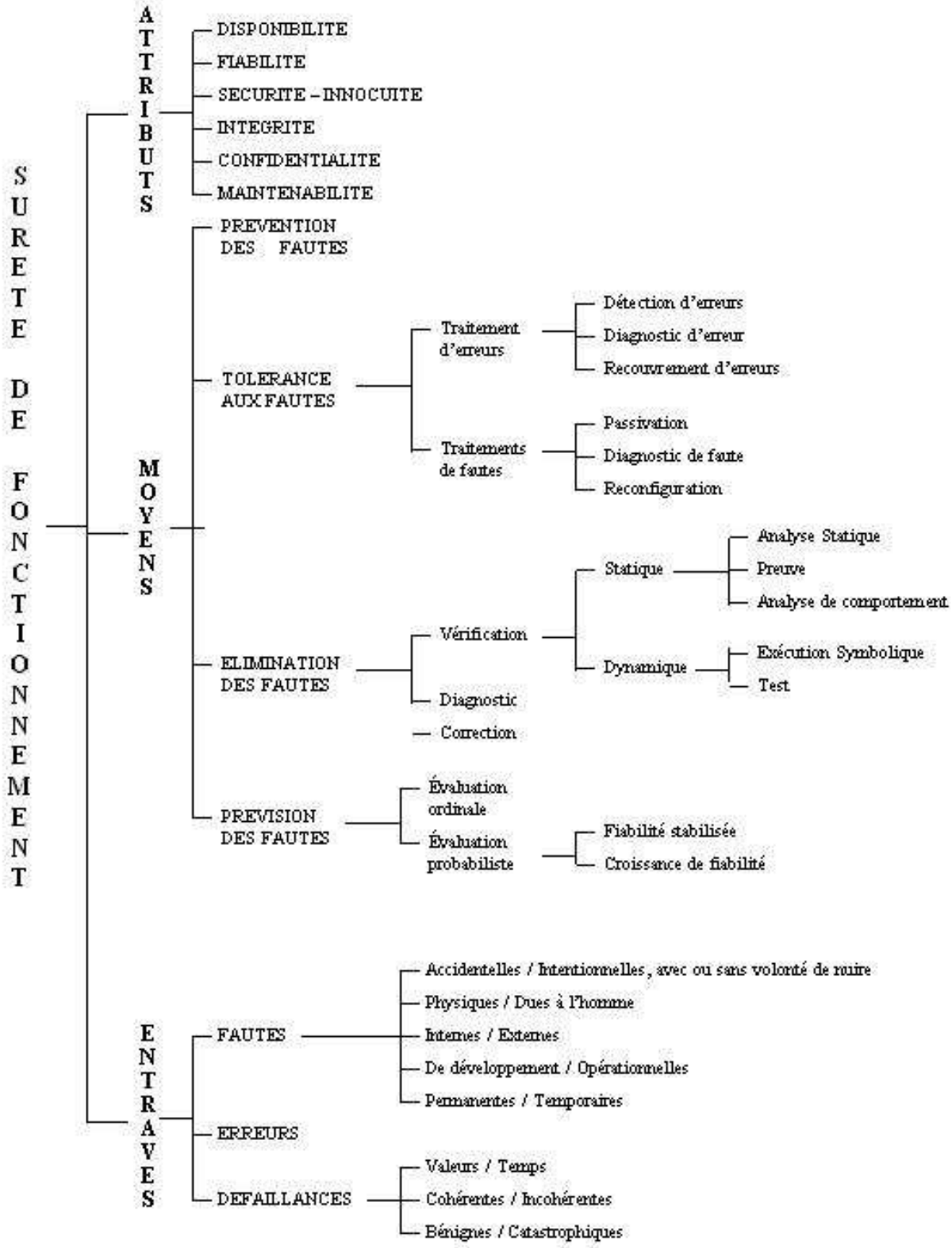


Figure 12 Arbre détaillé de la sûreté de fonctionnement (Laprie, 1995)

II.1.2 Prévision des fautes

La prévision des fautes vise à estimer la présence, la création et les conséquences des fautes ou des erreurs sur la sûreté de fonctionnement permettant ainsi leur élimination. Dans le cadre des machines de l'industrie manufacturière, les fautes que nous chercherons à identifier sont celles ayant un impact sur la sécurité-innocuité.

Pour identifier et prévoir les conséquences d'une défaillance, il existe deux grandes méthodes, l'Analyse des Modes de Défaillance de leurs Effets (et de leur Criticité) (AMDE(C), (Bouti et Kadi, 1994)) et la méthode des Arbres des Fautes (AdF, (Villemeur, 1988), également appelée Arbres de Défaillances ou Arbres des Défauts.

Ces deux méthodes vont permettre d'identifier les défaillances des composants ayant des conséquences portant atteinte à la sécurité des opérateurs de la machine. Le risque induit par ces défaillances doit être ensuite évalué avant de pouvoir être éliminé.

II.1.3 Estimation et réduction du risque

Les normes proposent des méthodes d'estimation et de réduction du risque itératives : estimation du risque initial, réduction du risque initial, estimation du risque résiduel, réduction du risque résiduel, etc. Ce type d'approche repose sur la notion « d'objectif de sécurité » (on parle aussi de performance de sécurité). L'objectif de sécurité est à la fois un résultat du processus d'analyse des risques, et un objectif à atteindre par le processus de conception. Cet objectif est dérivé par un ensemble de prescriptions d'intégrité de sécurité, qui sont associées aux exigences de sécurité fonctionnelle.

On distingue deux types de prescription d'intégrité de sécurité :

- les prescriptions basées sur des critères quantitatifs, tels que les taux de défaillances des composants réalisant l'exigence de sécurité fonctionnelle,
- les prescriptions basées sur des critères qualitatifs, tels que des contraintes architecturales permettant d'assurer l'intégrité du/des composant(s) ou des méthodes robustes pour le développement de cette architecture de composants.

Le processus d'estimation et de réduction du risque induit des conséquences à la fois pour le système à faire et pour le système pour faire. En effet, le système à faire doit répondre à des

exigences de sécurité fonctionnelle ainsi qu'aux prescriptions d'intégrité de sécurité quantitatives et qualitatives associées. Le système pour faire, pour réaliser un système répondant de manière satisfaisante à ces exigences de sécurité fonctionnelle, doit répondre également à des prescriptions d'intégrité de sécurité qualitatives.

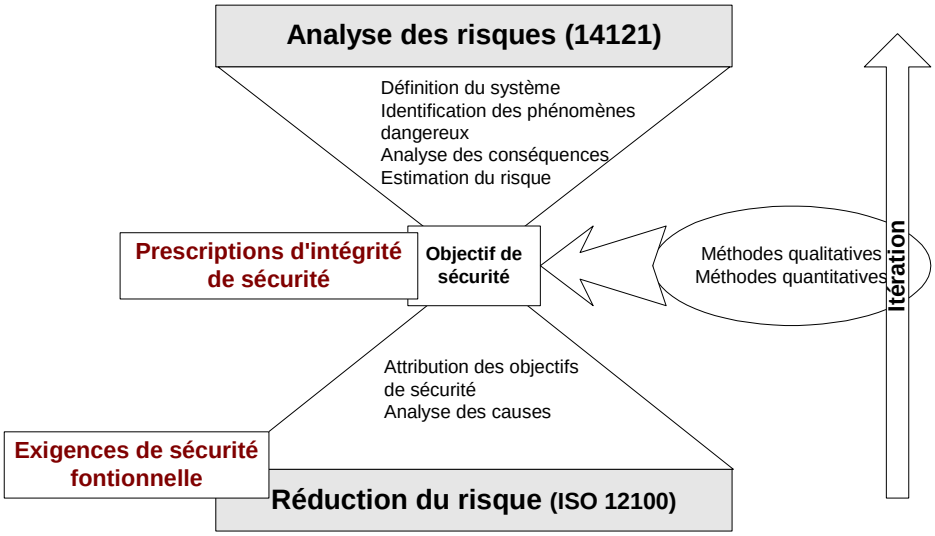


Figure 13 Notion d'objectif de sécurité, adapté de (Beugin *et al*, 2007)

II.1.4 Attribution des objectifs de sécurité

La norme CEI 61508, qui a été déclinée en normes sectorielles, traite des systèmes électriques, électroniques, électroniques programmables. Elle définit les objectifs de sécurité en terme de niveaux d'intégrité de sécurité (SIL : Safety machine Levels). Elle définit 4 niveaux de SILs, le SIL 1 étant le moins contraignant pour le système à faire comme pour le système pour faire.

L'objectif de niveau SIL peut être déterminé pour chacune des défaillances identifiées grâce à une approche qualitative basée sur les graphes de risques. Cette méthode définit le risque à partir de 4 composantes divisées en plusieurs niveaux. La lecture du graphe de risque conduit, en fonction des niveaux choisis pour les différentes composantes, à la détermination d'un niveau SIL. La Figure 14 donne un exemple de graphe des risques (Goble, 1998) (IEC 61508).

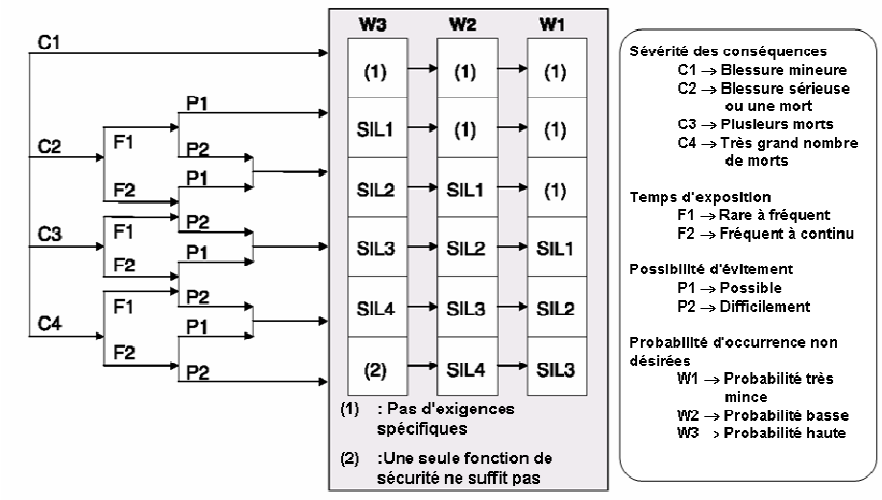


Figure 14 Un exemple de graphe des risques

II.1.5 Prescriptions d'intégrité de sécurité

Une fois que l'objectif de sécurité à atteindre est déterminé pour une exigence de sécurité fonctionnelle donnée, celui ci est décliné en prescriptions d'intégrité de sécurité (qualitatives ou quantitatives).

II.1.5.1 Prescriptions d'intégrité de sécurité quantitatives.

Ces prescriptions (Figure 15) donnent un objectif qualitatif pour caractériser les défaillances du système. Cet objectif est basé sur l'indice PFH (Probability of Failure per Hour). Ces deux indices sont des objectifs à atteindre pour les dispositifs relatifs à la sécurité. Dans le cas d'un système manufacturier, le PFH sera utilisé pour tous les dispositifs de protection, qui par leur nature sont sollicités régulièrement (protecteurs, commande bimanuelle, barrages immatériels etc.).

Niveau d'intégrité de sécurité (SIL)	Sollicitations irrégulières du système de sécurité	Sollicitations régulières du système de sécurité
	Moyenne du PFD (PFD_{avg})	Moyenne du PFH
SIL 4	$10^{-5} \leq PFD_{avg} < 10^{-4}$	$10^{-9} \leq PFH < 10^{-8}$
SIL 3	$10^{-4} \leq PFD_{avg} < 10^{-3}$	$10^{-8} \leq PFH < 10^{-7}$
SIL 2	$10^{-3} \leq PFD_{avg} < 10^{-2}$	$10^{-7} \leq PFH < 10^{-6}$
SIL 1	$10^{-2} \leq PFD_{avg} < 10^{-1}$	$10^{-6} \leq PFH < 10^{-7}$

Figure 15 Définition quantitative des niveaux SILs

Cette approche quantitative est possible pour les défaillances aléatoires (défaillances matérielles) mais ne peut être appliquée à des défaillances systématiques (Beugin *et al*, 2007)

(Brown, 2000). Ces défaillances systématiques sont déterministes par nature et ne peuvent donc être quantifiées. Parmi elles, on va trouver les défaillances liées au logiciel ou à la conception du système.

Toutefois, en ce qui concerne les défaillances du logiciel, certains travaux récents tentent d'apporter des solutions pour la quantification des défaillances d'un logiciel, notamment ceux réalisés par EDF pour la maîtrise de la fiabilité logicielle prévisionnelle (Carer *et al*, 2006). Ces travaux se basent sur les retours d'expérience des logiciels en exploitation, pour évaluer les taux de défaillances des prochains logiciels en fonction d'un certain nombre de métriques (nombre de composants, tailles des composants, complexité des interactions etc.). Ce type d'approche est très intéressant mais les résultats obtenus ne peuvent être appliqués à un autre domaine industriel car ils sont trop dépendants du contexte industriel et des retours d'expérience propres à ce contexte.

II.1.5.2 Prescriptions d'intégrité de sécurité qualitatives sur l'architecture

Les prescriptions portant sur l'architecture du système sont à mettre en relation avec l'approche quantitative du paragraphe précédent. En effet, l'architecture du système, les liaisons séries ou parallèles entre composants ont un impact sur les indices PFD_{avg} et PFS. De nombreuses méthodes portent sur la vérification du niveau SIL pour des systèmes de composants séries et parallèles. Ces études se basent sur différents formalismes tels que les « Reliability Block Diagram » (RBD) (Guo et Yang, 2006) et les chaînes de Markov (Zhang, 2003). Ce ne sont pas les seuls, un comparatif des différentes méthodes utilisées est dressé dans (Rouvroye et Brombacher, 1999)

Encore une fois, ces méthodes fiabilistes s'adressent très peu aux problèmes du logiciel, toutefois, l'architecture du logiciel peu également inclure des redondances au niveau des blocs fonctionnels. Certains critères, permettent tout de même de préconiser une architecture spécifique en fonction du niveau SIL requis. Le tableau ci dessous en est un exemple.

SFF	**Hardware Fault Tolerance**		
	0	1	2
< 60%	SIL1	SIL 2	SIL 3
60% à 90%	SIL 2	SIL 3	SIL 4
90% à 99%	SIL 3	SIL 4	SIL 4
> 99%	SIL 3	SIL 4	SIL 4

Figure 16 Redondance matérielle en fonction du taux de défaillances non dangereuses

Avec ce type d'indicateur, en fonction du SIL requis et du système que l'on veut réaliser, on peut identifier s'il est nécessaire de mettre en œuvre des redondances d'ordre 1 ou 2.

L'utilisation de composants logiciels redondants a, très tôt, fait l'objet de travaux appliqués. L'idée étant de mettre en redondance un certain nombre de composants logiciels réalisant la même fonction, mais développés séparément (Eckhardt *et al*, 1991). Les résultats de ces travaux ont été peu probants, la plupart des défaillances ayant comme facteur commun le cahier des charges.

II.1.6 Synthèse

Les outils de Sûreté de fonctionnement permettent de réaliser une analyse prévisionnelle des risques sur le système. Cette analyse, si elle ne permet pas d'identifier et de formaliser les exigences de sécurité fonctionnelle, contribue à quantifier le risque associé à ces exigences en définissant pour chacune d'entre elles un objectif de sécurité. Cet objectif aura un impact sur la définition des exigences architecturales (redondances matérielles) et sur le choix des méthodes de développement utilisées lors des différentes phases du développement.

Les méthodes d'analyse quantitatives sont peu adaptées aux parties logicielles du système, contrairement aux analyses qualitatives qui elles permettent de définir des objectifs de sécurité à atteindre par le logiciel. Dans ce dernier cas, les prescriptions d'intégrité de sécurité portent exclusivement sur le processus de développement. En ce sens, le Capability Maturity Model (Paulk *et al*, 1995) donne 5 niveaux de maîtrise du processus de développement du logiciel, généralisables aux processus de développement de systèmes incluant du logiciel (Figure 17).

Niveau 5 : Optimisé	Amélioration continue du processus par retour d'expérience quantitatif
Niveau 4 : Observable	Le processus et la qualité du logiciel produit sont observables
Niveau 3 : Défini	Le processus est documenté et peut être partiellement automatisé
Niveau 2 : Reproductible	Le processus est explicite et peut être reconduit d'une application à l'autre
Niveau 1 : Initial	Le processus n'est pas explicite

Figure 17 Capability Maturity Model (CMM)

En marge de l'attribution d'objectifs de sécurité, la formalisation des exigences de sécurité, déduites des processus d'analyse et de réduction du risque, repose, dans le cadre d'une ingénierie « système », sur la définition de propriétés comportementales et structurelles

qui tiennent compte de l'environnement dans lequel va évoluer le logiciel, notamment ses interactions avec les parties matérielles et les interfaces opérateurs.

II.2 Méthodes et Modèles pour l'analyse système

Définition : *L'ingénierie système est une approche coopérative interdisciplinaire pour le développement progressif et la vérification d'une solution pour le système, équilibrée sur l'ensemble de son cycle de vie, satisfaisant aux attentes d'un client et acceptable par tous. (IEEE 1220, 1999)*

II.2.1 Introduction

L'étude du contexte industriel réalisée au premier chapitre, a montré la nécessité de considérer le logiciel de commande dans son environnement en incluant les comportements induits par les parties mécaniques ou électriques de la machine afin d'appréhender le fonctionnement global de la machine et de ses dispositifs de sécurité. Il est notamment nécessaire d'identifier, de formaliser et de structurer les exigences de sécurité fonctionnelle, de les allouer et de les projeter sur les architectures fonctionnelles et matérielles du système.

Ces activités requièrent la mise en œuvre dès les premières phases de spécification de modèles orientés « système» capables de :

- satisfaire un large éventail de besoins de modélisation : comportement, architecture, exigences etc.
- couvrir l'aspect multi-métiers des systèmes : modélisation des systèmes de commande programmés, mais aussi des systèmes de commande câblés, et des parties mécaniques, pneumatiques, hydrauliques du système.

Cette problématique relève du champ disciplinaire de l'ingénierie « système » visant à maîtriser les processus techniques d'ingénierie des systèmes, les processus d'intégration d'un système à partir de ses composants existants ou développés spécifiquement et à maîtriser les relations contractuelles entre les différents acteurs d'un projet.

II.2.2 Modélisation des exigences

La modélisation des exigences est le résultat d'un processus complexe et itératif qui raffine un ensemble de besoins abstraits exprimés par les utilisateurs pour prendre progressivement en compte un ensemble de contraintes fonctionnelles et/ou techniques

émanant des développeurs. Dans le contexte de la sécurité des machines industrielles ce processus repose en partie sur le processus d'appréciation (estimation, évaluation) et de réduction (prévention, protection, …) du risque.

Afin de structurer la modélisation de ces différents types d'exigences (dans notre cas, exigences fonctionnelles, exigences de sécurité, exigences techniques) à différents niveaux d'abstraction, des approches orientées objet permettent la modélisation des exigences comme des objets caractérisés par un ensemble d'attributs (source, type, moyen de vérification, expression, etc.) et de relations de dépendance, d'abstraction/raffinement, de composition, etc.

Parmi ces approches, citons le diagramme d'objectif de KAOS (Heaven & Finkelstein 2004) ou dans le diagramme des exigences de SysML[6] (*Requirements diagram*) (Annexe 2). Ce dernier propose notamment de formaliser les liens entre exigences au travers:

- de relations de composition qui traduisent le raffinement d'une exigence « mère » en un ensemble d'exigences « filles » et permettent ainsi d'aborder le problème de la modélisation des exigences à différents niveaux d'abstraction similaire à la notion de sous objectifs dans KAOS,
- de relations de dérivation qui représentent un lien de dépendance (implication) entre plusieurs exigences.

II.2.3 Traçabilité des exigences

Dans le cadre du processus de définition du système, il est nécessaire d'établir une relation entre les exigences identifiées et les fonctions et/ou composants du système (Figure 18).

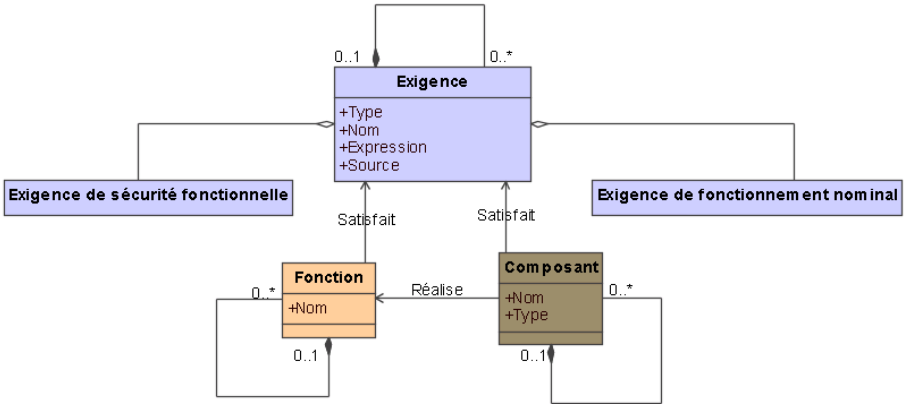


Figure 18 Modèle du système à faire, inspiré du modèle de l'AFIS (AFIS, 2005)

[6] SysML est un profil d'UML2 avec des extensions adaptant UML aux besoins en ingénierie système. (source : http://www.omgsysml.org/)

Ces modèles de traçabilité liant les exigences aux composants du système permettent de réaliser des analyses d'impacts dans le cas d'évolutions du système d'exigences. On est capable ainsi d'évaluer les conséquences de la modification d'une exigence sur la sécurité du système à partir du réseau tissé entre exigences, fonctions et composants.

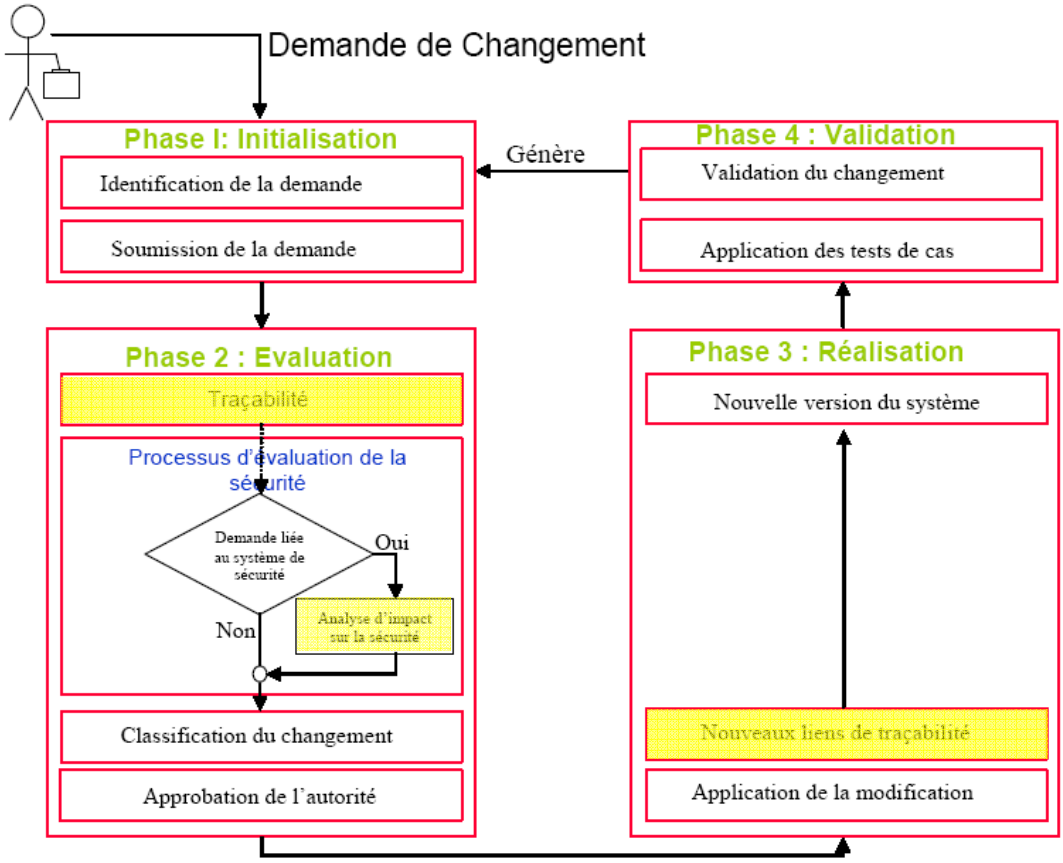


Figure 19 Boucle d'évolution d'une exigence tirée de (Eljamal, 2006)

Les travaux de M Eljamal (2006) ont montré que maîtriser les liens reliant exigences, et composants du système permet de réduire les erreurs et les coûts résultants d'une modification d'exigence ou d'une exigence retardée (mauvaise explicitation de l'exigence lors du premier processus de définition). Il est donc nécessaire que dès la conception du système des outils soient mis en place pour modéliser ces interactions.

Pour ce faire, les langages utilisés doivent pouvoir couvrir, à un niveau système, la représentation du fonctionnement global du système, sa structure, son comportement, … En ce sens, les modèles issus des domaines du Génie Informatique, tels que UML[7] ou ses extensions RT-UML (Selic, 1998) ou UMsdL, et de l'Ingénierie Système, tels que SysML (OMG, 2006)(Willard, 2007) (Annexe 2), les modèles proposés par Sagace (Penalva ,1997)

[7] UML : Unified Modelling Language, http://www.uml.org/

ou ceux proposés par la méthode KAOS (Heaven & Finkelstein, 2004), proposent un ensemble de formalismes (diagrammes statiques, dynamiques, fonctionnels, structurels, informationnels, etc., permettant de représenter les différents types d'objets du système : exigences, composants, fonctions et se révèlent plus complets que les outils d'ingénierie des exigences largement utilisés dans l'industrie (DOORS® [8])

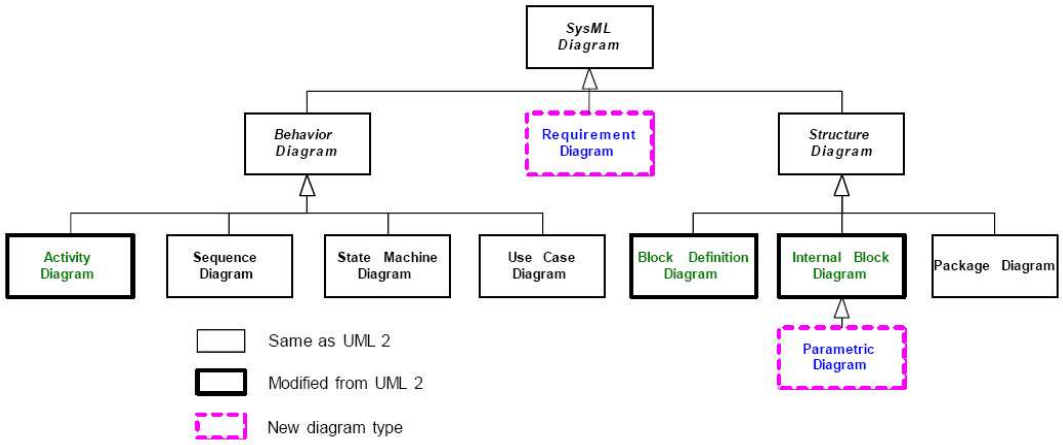


Figure 20 Correspondances entre les diagrammes SysML et UML2

Les exigences peuvent ainsi être projetées sur des constituants du système (fonctions ou composants) qui sont modélisés sous la forme d'objets (Figure 21).

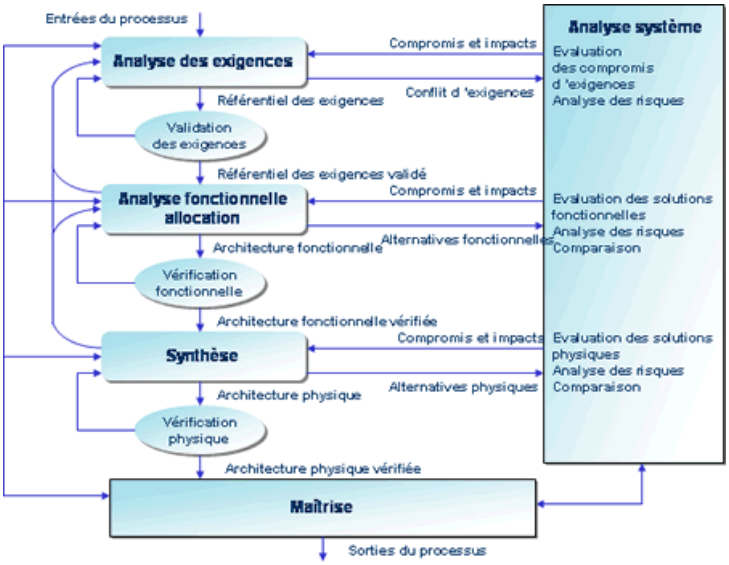


Figure 21 Traçabilité des exigences selon la norme IEEE 1220

En particulier, SysML propose un diagramme de définition de « blocks » qui permet de modéliser la structure physique du système, mais également des diagrammes de cas d'utilisations et d'activités permettant de définir son architecture fonctionnelle. Les blocks

[8] Telelogic DOORS®. http://www.telelogic.com/Products/doors/doors/index.cfm

sont vus comme des objets, des liens de composition permettant de spécifier la structure du système en termes de hiérarchie et d'instances de blocks. L'« Internal Block Diagram» définit les relations entre les entrées et les sorties des différentes instances. Ces modèles architecturaux peuvent être réalisés de manière indépendante des modèles d'exigences, ce qui permet une projection *a posteriori* des exigences sur l'architecture de composants.

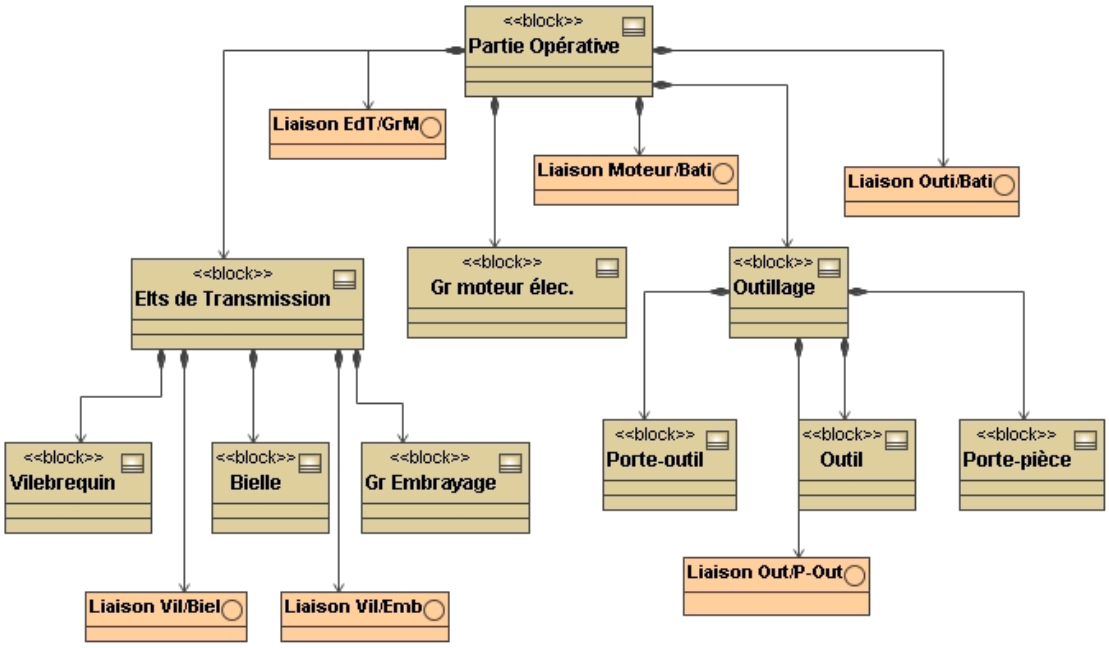


Figure 22 Architecture d'une presse mécanique représentée par un diagramme de block

La méthode KAOS prend une optique différente en ne proposant qu'un seul type de constituants, les agents, qui vont avoir pour mission de prendre en charge les objectifs, qui une fois placés sous leur responsabilité deviennent des exigences. La définition des agents est donc complètement dépendante de l'existence d'objectifs. Le comportement des agents et des blocks est ensuite décrit à l'aide de modèles comportementaux basés sur des formalismes à état. Toutefois l'utilisation de ces modèles orientés « métier » est retardée afin de permettre aux différents acteurs de la conception du système de communiquer sur des modèles systèmes plus génériques et intuitifs.

En terme d'application industrielle, ces deux approches reposent sur un ensemble d'outils informatiques proposés par les fournisseurs de solutions en ingénierie système et basés sur l'utilisation de SysML : Rapsody de I-Logix[9], Artisan Studio de Artisan Software[10] etc., tandis que la méthode KAOS est développé dans l'outil Objectiver.

[9] I-Logix : www.I-Logix.com
[10] Artisan Software Tools, Inc : www.artisansw.com

Les activités de vérification et de validation propres au processus de définition du système sont difficiles à mettre en œuvre sur ce type de modèle du fait de leur manque de formalisation. En particulier les liens de raffinement entre des exigences relatives à des niveaux d'abstractions différents, s'ils existent, ne peuvent être vérifiés. Comment alors garantir la cohérence entre une spécification non formelle des exigences et les différents modèles élaborés (multi points de vue) pour développer un système répondant à ces attentes ?

Ce point apparaît pourtant comme essentiel si l'on considère que la pratique d'une automatisation à échelle industrielle requiert l'utilisation de multiples formalismes permettant d'appréhender le fonctionnement global d'un système à concevoir. Ce problème rejoint celui abordé par l'Ingénierie Dirigée par les Modèles (Favre et al. 2006) ayant pour objectif d'offrir un cadre unificateur de modélisation pour les systèmes à logiciel prépondérant. Si cette approche est basée sur l'utilisation de modèles exprimés dans des formalismes différents pour couvrir les différentes étapes de développement, les différents niveaux d'abstraction ou encore les différents points de vue considérés, son originalité provient surtout de l'utilisation systématique de méta-modèles décrivant les formalismes utilisés et des transformations automatiques ou interactives entre modèles permettant le passage d'un domaine technique à un autre.

Dans le domaine de l'automatisation, plusieurs approches ont été proposées en ce sens. Elles consistent principalement à :

- intégrer dans un formalisme de modélisation des relations explicites vers d'autres modèles ; nous pouvons citer, à ce titre, le diagramme des exigences proposé par SysML qui permet de faire référence à des objets (composants, fonctions, acteurs,etc) décrits dans d'autres diagrammes SysML ou encore les approches mixant des diagrammes d'objets UML et une description comportementale sous forme de modèles Statecharts, de réseaux de Petri (Bouabana-Tebibel et Belmesk, 2007) ou de blocs fonctionnels de la norme IEC 61499 (Tranoris & Thramboulidis 2006, Auinger et al. 2005, Ferrarini et al. 2005, Zhang et al. 2005),
- définir des règles de transformation de modèles fondées sur leur méta-modèle, afin d'autoriser l'utilisation, dans un formalisme donné, d'un ensemble d'informations contenues dans d'autres modèles de l'étude, (Bon-Bierrel 1998, Pietrac 1999, Berruet 2006),

- enfin, formaliser le processus de modélisation lui-même sous la forme d'un référentiel commun, tel que le modèle de données de l'AFIS ou les matrices de traçabilité des exigences, permettant d'assurer la cohérence entre les différents concepts et objets manipulés au cours du processus d'automatisation.

Ces approches contribuent à la cohérence des modèles élaborés au cours du processus d'automatisation en définissant des cadres de modélisation (Sowa & Zachman, 1992) offrant une vision structurée et plus ou moins unifiée des différents concepts et objets de modélisation. En revanche, elles ne permettent pas d'obtenir de certitudes (sous forme de preuves ou de démonstrations) quant à la cohérence des informations contenues dans les différents modèles. En d'autres termes, elles permettent d'augmenter le niveau de qualité du processus de modélisation et présument donc de la qualité des résultats produits sans toutefois pouvoir la garantir.

C'est pourquoi un certain nombre de travaux (Shell, 2001 ; Johnson, 2007) considèrent que le passage vers une véritable spécification formelle est nécessaire pour établir un modèle complet, pertinent, cohérent et correct de ce que le système doit faire et pour partager une compréhension commune et cohérente des services attendus d'un système automatisé.

II.2.4 Modèles formels pour l'ingénierie système

Une telle spécification formelle doit reposer sur un langage formel unifié dont le niveau d'abstraction permet de satisfaire un large éventail de besoins de modélisation : comportement, structure, information, communication, etc. D'autre part, nous avons montré que le processus de définition du système repose sur la modélisation progressive, suivant une approche collaborative et interdisciplinaire, des différents objets d'étude. Le langage formel unifié, support à la spécification formelle, doit donc autoriser une description incrémentale des modèles tout en garantissant, dans le cadre d'une ingénierie dirigée par les modèles, la cohérence entre des spécifications établies à des niveaux différents d'abstraction.

Les premiers travaux en ce sens ont donné naissance à des langages de spécification formels comme le langage VDM (Jones, 1990), le langage Z (Spivey, 1989), ou la méthode B (Abrial, 1996). La méthode B est une méthode formelle utilisant la preuve de propriétés permettant de spécifier, concevoir et implémenter un logiciel applicatif. Elle est basée sur :

- la théorie des ensembles et les fonctions, relations et opérateurs associés,

- la logique du premier ordre avec les opérateurs classiques ($\neg$ P, P$\vee$ Q, P $\wedge$Q, P $\Rightarrow$ Q, P $\Leftrightarrow$ Q) ($\forall$X • p, $\exists$ X • p),
- la notation en machine abstraite.

Une machine B est définie par la structure ci dessous :

MACHINE *m*
SETS *s*
CONSTANTS *c*
PROPERTIES *p*
VARIABLES *x*
INVARIANT *I(x)*
INITIALISATION *init(x)*
EVENTS
O1= select P1(x) then S1(x)
…
On = select Pn(x) then Sn(x)
END

Figure 23 Machine B

Une machine a un nom, la section SETS contient la définition des ensembles relatifs au problème, la section CONSTANTS permet au concepteur d'introduire des informations au sujet de la structure mathématique du problème et la section PROPERTIES contient la définition effective des constantes. La deuxième partie du modèle définit les aspects dynamiques des variables d'états et les propriétés des variables utilisées dans l'invariant. Les évènements *O1…On* décrivent comment les variables d'état sont modifiées grâce aux substitutions généralisées, *Si (x)* contenant la nouvelle valeur de la variable *x* après l'occurrence de l'événement *Oi*. La substitution d'une variable n'est réalisable qu'à la condition que sa garde soit vraie. L'invariant *I(x)* définit un ensemble de conditions qui doivent être respectées par la variable x lors de l'initialisation et qui doivent être conservées par tous les évènements définis dans le modèle. Les conditions à vérifier, appelées « obligation de preuves » sont générées à partir du texte du modèle et elles expriment les hypothèses requises pour la préservation des propriétés de l'invariant :

(INV1) $Init(x) \Rightarrow I(x)$

(INV2) $I(x) \wedge P(x) \Rightarrow I(S(x))$

(INV1) exprime le fait que les conditions initiales doivent établir l'invariant et (INV2) le fait que partant d'une situation où l'invariant est respecté, la transformation de variable doit déboucher sur une situation où l'invariant est préservé. (INV2) doit être vérifié pour chacun

des événements *Oi* du modèle. Le mécanisme de raffinement est utilisé pour décrire les modèles sur plusieurs niveaux d'abstraction en enrichissant au fur et à mesure la description des variables et des événements tout en préservant les invariants déjà prouvés. On peut résumer cette approche par :

(M1,G1) raffiné par (M2,G2) raffiné par … (MN,Gn)

Mi est la i^{ème} itération du modèle et satisfait l'objectif *Gi* ainsi que les objectifs des itérations précédentes. La relation *raffiné par* assure la préservation des objectifs à la condition qu'elle soit prouvée. Cela signifie que si un nouveau modèle est dérivé de *(Mn, Gn)* il faudra prouver que ce nouveau modèle raffine bien l'ancien modèle tout en garantissant les nouvelles propriétés. Le mécanisme de preuve inclus dans la méthode B est basé sur un moteur d'inférence supporté par l'Atelier B[11]. Il s'agit en fait d'un assistant de preuves, qui ne permet donc pas de réaliser automatiquement toutes les preuves. Le langage B a prouvé son efficacité pour développer des systèmes logiciels grâce à son mécanisme de raffinement qui supporte le processus de conception complet, depuis la phase de spécification jusqu'à la phase d'implantation. Plusieurs travaux se sont portés sur l'application de la méthode B à des problèmes d'automatisation de système industriels (Behm *et al*, 1999).

L'intérêt de langages formels tels que la méthode B est multiple : la notation très abstraite permet de spécifier n'importe quel type d'objet, et le caractère formel permet de réaliser des preuves mathématiques par *theorem-proving* sur la validité de ces spécifications. Toutefois ces langages sont relativement complexes et ne possèdent aucune représentation graphique, ce qui rend la modélisation de systèmes complexes difficile (Zimmerman *et al,* 2000), et de plus le processus de preuve nécessaire pour les activités de vérification est un travail d'expert. C'est pourquoi ces dernières années, les ingénieurs système se sont tournés vers des langages semi formels tels que UML (Unified Modeling Language) (OMG, 2003).

II.2.5 Synthèse

Nous avons souligné dans ce paragraphe le besoin en termes de modèles offrant une couverture suffisamment large pour appréhender les applications à un niveau système, pour capturer les exigences systèmes à différents niveaux d'abstraction, et pour les projeter sur une architecture de composants. Les méthodes et modèles, issus du champ disciplinaire de l'Ingénierie Système, répondent effectivement à ce besoin mais n'offrent que peu de

[11] www. atelierb.eu réalisé par Clearsy System Engineering.

possibilités de vérification (SysML, KAOS, …). Des méthodes plus formelles issues du Génie Informatique, telles que la méthode B, offrent ces possibilités mais le niveau d'abstraction de leur formalisme est peu adapté au passage à l'échelle en milieu industriel.

Pour couvrir les aspects « système », l'outil de modélisation retenu est le langage SysML. Ce choix est notamment justifié par :

- la couverture en termes de modélisation offerte par SysML au travers des diagrammes statiques et dynamiques d'UML2 auxquels il convient notamment d'ajouter le diagramme des exigences supportant la définition et la structuration des exigences sous la forme de diagrammes à objets ;
- la modélisation de la sémantique des diagrammes sous la forme de méta-modèles, ce qui permet d'envisager leur intégration avec les modèles de conception et de vérification formelle plus sereinement qu'avec des approches de modélisation dédiées « système » telles que SAGACE ou dédiées « ingénierie des exigences » telles que KAOS,
- l'outillage disponible sous la forme de plug-ins intégrables aux outils de modélisation UML qui constitue un critère important dans le contexte à vocation industrielle de ces travaux.

Néanmoins, les possibilités limitées en termes de vérification nous ont poussés à compléter les aspects système couverts par SysML par des descriptions plus détaillées permettant la modélisation comportementale des composants et la vérification formelles de leurs propriétés de sûreté.

III - Méthodes et modèles pour la vérification

Le prédicat de l'automatisation énoncé par Fusaoka (Fusaoka *et al*, 1983) (Pétin *et al*, 2006) et qui postule qu'automatiser un système consiste à définir les règles qui vont piloter sa partie opérative en partant des objectifs énoncés pour le système. Il peut être écrit de la manière suivante :

Modèle de conception de la commande $\wedge$ *Modèle de conception de la partie opérative* $\supset$ *Modèles des exigences du système*

La boucle de conception doit alors proposer des modèles de conception pour la commande et la partie opérative et vérifier que ces modèles respectent le modèle des exigences. Les méthodes et outils participant à ces activités se répartissent en trois familles (Pétin, 2007):

- Les formalismes de modélisation de chacun des trois termes du prédicat sont formels et homogènes auquel cas il est possible de vérifier formellement que les modèles de conception respectent bien le modèle des exigences.
- Les formalismes de modélisation de chacun des trois termes sont propres aux différents domaines d'activités, certains d'entre eux sont non formels (modèle des exigences) ce qui ne permet pas de vérifier le respect du prédicat autrement que par simulation.
- Les formalismes de modélisation de chacun des trois termes du prédicat sont formels homogènes et permettent d'envisager la génération automatique d'un des termes du prédicat (synthèse de commande).

Les algorithmes de synthèse reposent sur la théorie de la supervision (Supervisory Control Theory, SCT) (Ramadge et Wonham, 1987) qui fournit un cadre formel de modélisation de SED. Cette technique a montré son efficacité pour la synthèse de la commande dans le cadre d'études académiques (Niel *et al*, 2001), mais le passage à l'échelle industrielle se heurte à plusieurs difficultés (Pétin, 2007). Dans le cadre de notre étude, nous nous sommes donc volontairement limités aux deux premières familles avec comme solutions : vérification formelle et simulation.

Quelles que soient les techniques de vérification utilisée, elles reposent toutes sur une formalisation des propriétés attendues du système. Ceci suppose donc de pouvoir traduire les exigences de sécurité identifiées par le processus de définition du système, en propriétés de

sûreté (propriétés invariantes) exprimées sous la forme de prédicats logiques. Se pose alors le problème des règles de traduction des exigences en propriétés tout en respectant leur structuration. En ce sens, des approches de modélisation (Chapurlat 2007) ont été développées notamment afin de faciliter leur réutilisation dans des algorithmes de vérification (*model-checking*, *theorem-proving* ou simulation).

III.1 Modèle de propriétés

III.1.1 Propriétés

Définition : *La propriété est une qualité propre, une caractéristique intrinsèque (fonctionnelle, comportementale, structurelle ou organique dépendante du temps ou non) que doit posséder une entité. Toute propriété traduit une attente, une exigence, une finalité à laquelle l'entité doit répondre* (Chapurlat, 2007).

L'architecture d'un système automatisé est relativement complexe car constituée d'un ensemble de composants basés sur des technologies et des domaines de connaissances différents. Ces propriétés portant sur le système dans son ensemble ont des projections sur chacun des composants du système. Ainsi une propriété exprimée au niveau système peut être constituée des propriétés exprimées au niveau du composant, la véracité de l'ensemble des propriétés des composants impliquant alors la véracité de la propriété système.

Toutefois, du fait de la complexité des interactions entre les composants d'un système automatisé ainsi que de par le grand nombre d'exigences système de sécurité fonctionnelle desquelles sont issues les propriétés du système, cette projection est difficile à réaliser. Elle doit nécessairement être encadrée, notamment, les relations entre propriétés, composants et exigences doivent être clairement établies.

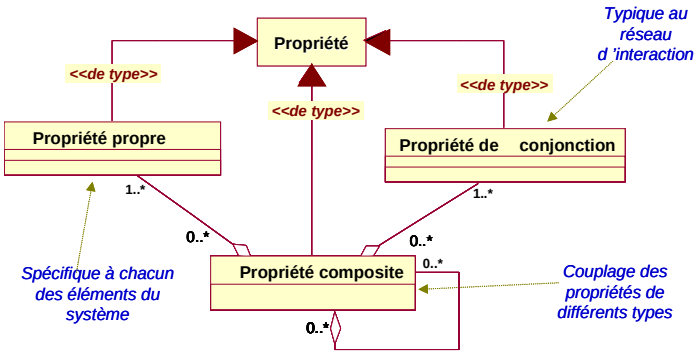


Figure 24 Typologie des propriétés d'un système (Chapurlat, 2007)

Les propriétés de sécurité qui nous intéressent plus particulièrement traduisent les exigences de sécurité fonctionnelle. Ces propriétés exprimées au niveau du système formalisent le fait qu'une situation donnée ne doit pas se produire à un instant donné. Elles sont qualifiées d'invariantes si leur véracité est indépendante du temps ou de l'état du système, de temporelles si leur véracité varie en fonction du temps ou évènementielles si elles caractérisent la réponse du système a des stimuli.

III.1.2 Modèle CREDI

Le modèle CREDI proposé par Lamine (Lamine, 2001) permet de décrire de manière formelle toutes les propriétés d'un système. Ce modèle se base sur la notion de granularité. Cette notion permet de définir les différents niveaux d'abstraction des propriétés du système.

Le modèle CREDI est un quintuplet P : = (C_D , $E_{D'}$, $R_{DD'}$, D'', $I_{D''}$) , avec :

- C_D l'ensemble éventuellement vide de faits (variables de modélisation, paramètres de modélisation, prédicats, et autres propriétés) appelés *causes* et appartenant à un degré de granularité D ;
- E_D est l'ensemble nécessairement non vide de faits appelés *effets* et appartenant à un degré de granularité D'. Les causes et les effets réfèrent un degré de détail donné à l'intérieur d'une même granularité qui est choisie par le modélisateur. D, D' et D'' sont des degrés de granularité qui respectent les règles suivantes : les faits (cause + effets) appartiennent à un niveau de granularité $D \in G$, $D' \in G$ avec $D' \in (D ; D+1; D-1)$, et pour toute propriété P, P est de degré D'' tel que D'' = max (D, D') et $D'' \in G$;
- R est la relation de causalité R typée ;
- L'ensemble I contient les faits pouvant servir d'indicateurs dont l'évolution permet de détecter une possible modification de la véracité de la propriété.

Ce modèle met particulièrement en avant la nécessité de relier une propriété à un niveau d'abstraction, cela impliquant que cette propriété est dépendante de propriétés vérifiées à un niveau d'abstraction inférieur (propriété émergente), ou de propriétés vérifiées à des niveaux supérieurs (propriété immergente). Il renforce ainsi l'intérêt du raffinement prouvé supporté par la méthode B.

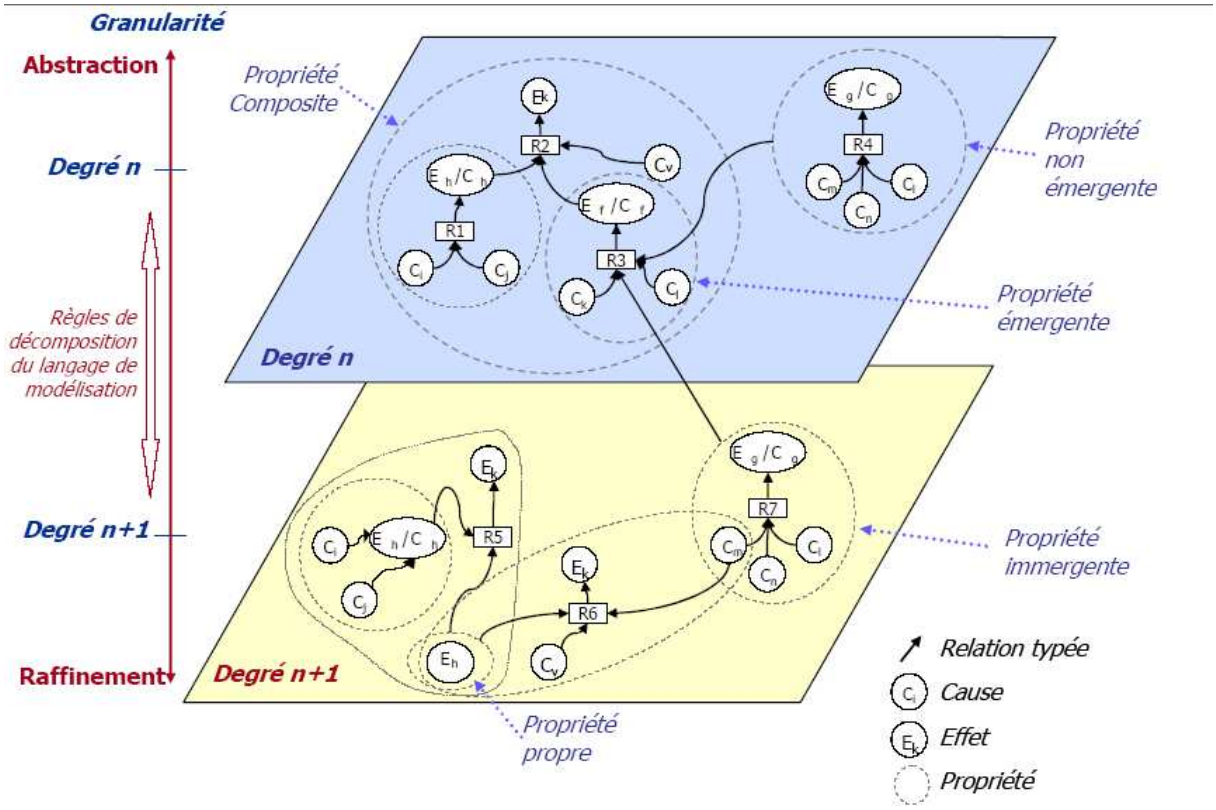


Figure 25 Types de propriétés définis par le modèle CREDI (Chapurlat, 2007)

Ce modèle se révèle très efficace pour la structuration des propriétés mais ne dispense pas pour autant d'une phase d'identification des exigences principalement basée sur les modèles orientés système, présentés au paragraphe II.

III.2 Vérification par simulation

La vérification par simulation des modèles de commande et des systèmes implantés repose respectivement sur l'exécution symbolique des modèles (simulation) et sur l'exécution de tests de conformité visant à déterminer la satisfaction des besoins exprimés par l'utilisateur.

Les techniques de simulation sont utilisées pour la validation des logiciels de commande incluant des fonctions de sécurité, car elles permettent de valider le comportement du système complet (matériel & logiciel) (Brombacher et van Beurden, 1999).

Selon la nature des interactions entre objets simulés et objets réels, (Isermann *et al*, 1999) distingue trois approches complémentaires (Figure 26) : la simulation d'un modèle de commande couplé à des éléments réels non modélisés, la simulation en boucle fermée des

modèles de commande et des processus physiques, et enfin l'émulation des systèmes physiques afin de valider, en plate-forme, le système de commande réel.

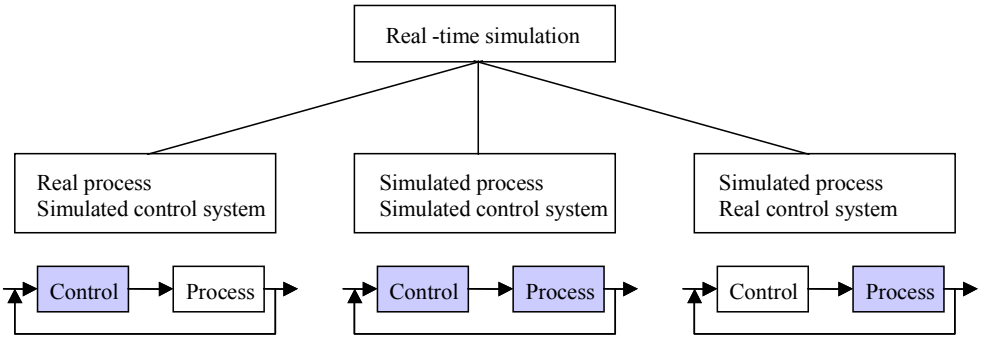


Figure 26 Techniques de simulation (Isermann *et al*, 1999)

L'exécution symbolique des modèles nécessite la définition d'une sémantique opérationnelle permettant d'évaluer, de manière déterministe, le comportement décrit par un modèle en réaction à des stimuli d'entrées. Les modèles et langages à la base des techniques et outils de simulation utilisés dans le domaine manufacturier sont généralement :

- des modèles dynamiques des Systèmes à Evènements Discrets (S.E.D.), dont la (ou les) sémantique(s) opérationnelle(s) ont été formalisée(s) : sans être exhaustif, nous pouvons citer les modèles Statecharts (Harel, 1987), différentes classes de Réseaux de Petri (Jensen *et al*, 2007), le Grafcet (Lhoste *et al*, 1997) ou encore les langages synchrones.
- des langages de programmation, tels que ceux proposés par la norme CEI 611316-3 et des approches (et outils) d'automatisation orientées objets telles que celle proposée par Thramboulidis (Thramboulidis et Tranoris, 2004).

Cette approche pose le problème de l'exhaustivité des scénarios utilisés pour la vérification. Ces scénarios ne parcourent qu'un sous-ensemble du modèle à vérifier. Ces techniques ne donnent donc qu'une présomption du bon comportement du logiciel de commande au sens ou elles permettent de détecter certaines erreurs du modèle, mais ne peuvent garantir l'absence d'erreurs.

Dans ce contexte, la confiance que l'on peut accorder aux résultats de simulation repose essentiellement sur l'expérience et l'expertise des utilisateurs. En effet c'est eux qui devront apprécier quantitativement et qualitativement les différents scénarios de simulation au regard des besoins exprimés (choix des scénarios, taux de couverture, interprétation des résultats, définition d'intervalles de confiance, etc.). Certains logiciels proposent d'ailleurs, en

complément des scénarios, des simulations de Monte-Carlo, pour tester la réponse du modèle à des stimuli aléatoires. La réponse est alors comparée à une post condition qui doit être vraie dans tous les états du logiciel.

Si les techniques de simulation ont fait preuve de leur efficacité, notamment pour la validation d'applications complexes ou de grande taille en milieu industriel, elles ne permettent toutefois d'apporter qu'une présomption de conformité qui peut se révéler insuffisante pour le développement de systèmes soumis à de fortes contraintes de sécurité ou de sûreté de fonctionnement. C'est pourquoi les prescriptions de sécurité associées aux niveaux de SIL (3 et 4) recommandent l'utilisation de méthodes de vérification formelles.

III.3 Méthodes formelles pour la vérification

Les techniques de vérification formelle ont été développées dans les années 80 afin de s'assurer de l'adéquation entre :

- un modèle formel du système de commande qui peut être soit un modèle de conception soit un modèle d'exécution du code implanté dans un automate programmable,
- un modèle formel des propriétés à vérifier conformément aux besoins informels (exprimés par les utilisateurs). L'obtention de ce modèle formel requiert une phase d'identification des exigences de sécurité fonctionnelle et une phase de formalisation pour obtenir les propriétés à respecter.

Il existe deux grandes familles d'outils de vérification formelle : les *model-checker* et les *theorem prover* qui présentent chacune des avantages pour la vérification des systèmes à évènements discrets (Evrot et al, 2006a).

III.3.1 *Theorem-proving*

Avec les méthodes de déduction automatique (Hoare, 1969), le problème de la vérification est vu comme un théorème que l'on cherche à prouver à partir d'un ensemble d'axiomes. Il est nécessaire de pouvoir transformer le programme et la propriété que l'on souhaite vérifier en objets mathématiques. Les méthodes de déduction automatique sont implémentées au sein *d'assistants de preuve (theorem prover)* comme PVS (Owre *et al*, 1996), Coq (Huet *et al*,

1995), HOL (Melham *and* Gordon, 1993) et Isabelle (Paulson, 1994). Ils fonctionnent en finalisant une preuve suggérée par l'utilisateur.

Les techniques de preuve sont basées sur des démonstrations mathématiques de la compatibilité des modèles. Elles sont capables d'inférer directement des conclusions à partir d'une description d'évènements ou d'opérations permettant de faire évoluer le système. En général, le nombre d'opérations autorisées sur un système est petit en comparaison de la représentation explicite des états auxquels l'application de ces opérations permet d'aboutir. L'utilisation des techniques de preuve est cependant rendue difficile par le fait que, même en utilisant des outils informatiques de démonstration tels que les *theorem prover*, leur automatisation complète est rarement possible. En effet, l'utilisateur est souvent obligé de guider la preuve en interagissant avec l'assistant. De plus, la préparation de la preuve exige souvent de la part de l'utilisateur la spécification d'éléments permettant de mener cette preuve, mais qui sortent du cadre direct de l'expression des besoins décrite dans le cahier des charges. L'utilisation de ce type d'outils requiert donc une grande expertise de l'utilisateur pour la conduite des preuves, c'est-à-dire qu'il est généralement nécessaire de sous-traiter les preuves à des experts.

Dans le domaine de la commande des SED, des travaux ont été menés sur l'utilisation de techniques de preuves pour vérifier les propriétés satisfaites par l'ensemble automate plus logiciel (Roussel et Denis, 2002).

Dans le domaine de la sécurité, ces outils sont utilisés pour vérifier les modèles de spécification du logiciel. Cette utilisation introduit une difficulté supplémentaire, à savoir la formalisation des spécifications du logiciel dans un langage compréhensible par l'assistant de preuve ainsi que la formalisation des exigences système de sécurité fonctionnelle sous forme de propriétés (Kim *et al*, 2005)(Son et Seong, 2003).

A l'opposé, les méthodes de *model-checking* ont l'avantage de reposer sur une construction explicite d'un modèle de comportement du type états/transitions proche des formalismes utilisés dans le domaine manufacturier pour la spécification des logiciels de commande.

III.3.2 *Model-checking*

A l'origine les techniques de *model-checking* consistaient en une exploration exhaustive de l'espace des états atteignables du système et portaient sur la vérification de propriétés

fixées telle que l'absence d'inter blocage. Désormais, ces techniques ont été étendues de diverses manières : prise en compte de spécifications et de propriétés formalisées dans une logique temporelle, utilisation de structures de données plus efficaces pour représenter l'espace des états. (Clarke *et al*, 1999), (Bérard *et al*, 2001) constituent deux ouvrages de référence relatifs aux techniques de *model-checking*.

A partir d'un modèle de programme et d'un ensemble de propriétés formelles, le principe du *model-checking* est de parcourir l'ensemble des états atteignables et de s'assurer qu'aucun de ces états ne viole les propriétés à vérifier Figure 27. La plupart des algorithmes de *model-checking* proposent un contre exemple, c'est-à-dire une séquence permettant d'arriver à un état du système dans lequel la propriété n'est pas respectée. Trois *model-checker* souvent utilisés sont PRISM développé à l'Université de Birmingham par l'équipe de Marta Kwiatkowska, SMV développé par les laboratoires Cadence de Berkeley, et UPPAAL développé par une collaboration entre le département d'Information Technology de l'université d'Uppsala (Suède) et le département de Computer Science de l'université D'Aalborg au Danemark.

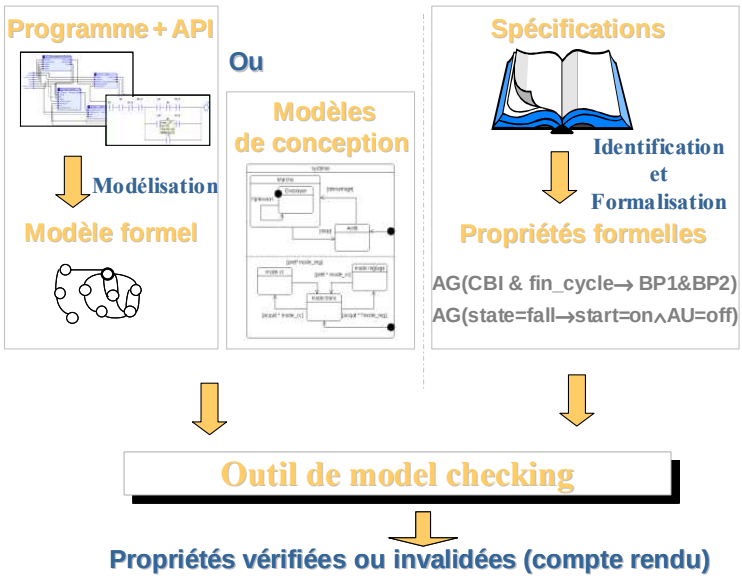


Figure 27 Vérification formelle par *model-checking*

Pour vérifier des programmes implantés sur des API à l'aide d'un *model-checker*, plusieurs étapes sont nécessaires (Gourcuff *et al*, 2006). Dans un premier temps, il s'agit de modéliser le programme que l'on veut vérifier et de formaliser les spécifications auxquelles on veut le confronter ; cette formalisation se fait la plupart du temps en logique temporelle ou alors à l'aide d'un observateur. Modéliser le logiciel consiste alors à exprimer le code du

logiciel dans un langage formel et à donner un modèle du moteur d'exécution du logiciel. Par exemple, Figure 28, la partie haute représente le cycle automate, décomposé en 4 étape :

- L'initialisation (située entre les 2 premiers états sur la figure) qui permet de d'initialiser les variables interne et les entrées-sorties.
- L'acquisition des entrées (située entre le 2eme et le 3eme état sur la figure), durant laquelle un message est envoyé (acq_input) aux modèles d'évolution des entrées (voir partie basse de la figure) afin qu'ils évoluent de manière aléatoire
- Le traitement des lignes de code (située entre le 3eme et le 4eme état sur la figure) durant laquelle l'automate calcul ses variables internes et les sorties à partir des nouvelles valeurs des entrées.
- L'émission des sorties (située entre le 4eme et le 2eme état sur la figure) qui permet d'envoyer le message emission_sorties. La vérification des propriétés n'a lieu qu'au moment où ce message est envoyé, c'est-à-dire à la fin du cycle automate.

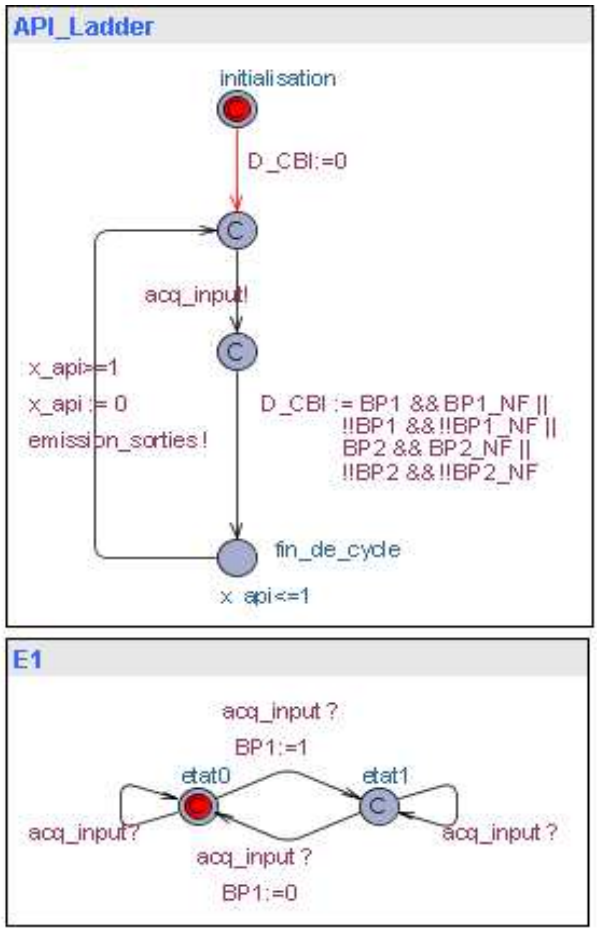


Figure 28 Modélisation du logiciel, de son exécution et des variables d'entrées

Le *model-checking* est une technique qui permet de vérifier de manière exhaustive que le système de commande respecte ses propriétés de sécurité. Il est utilisé dans la vérification de systèmes réels (Ortmeier *et al*, 2003), et la possibilité d'identifier les scénarios de défaillance par l'intermédiaire des contre exemples fournis permet d'identifier plus facilement l'origine de la défaillance (Rushby, 2002).

Le processus de preuve par *model-checking* est complètement automatisé, à partir du moment où l'utilisateur fournit les modèles formels des propriétés et du système à vérifier. Toutefois, son utilisation à une échelle industrielle est rendue difficile par plusieurs verrous.

Le phénomène bien connu d'explosion combinatoire restreint la vérification à des systèmes de petite taille. La taille des structures de données croît très rapidement lorsque les paramètres du système augmentent. Ainsi, même si l'algorithme de vérification est efficace, il est limité par la puissance de calcul des calculateurs disponibles actuellement. Ce phénomène peut être contourné en utilisant un raisonnement compositionnel (Berezin *et al*, 1997). Cette méthode consiste à vérifier chaque composant d'un système de manière indépendante, puis de conclure sur la correction de la composition des composants entre eux. Ainsi, on ne vérifie plus le code dans son ensemble, mais composant par composant. Cette méthode suppose d'être capable d'exprimer les propriétés locales que les composants doivent respecter et, d'autre part d'être capable de montrer que la conformité des composants aux propriétés locales implique la conformité de l'ensemble du logiciel aux propriétés systèmes.

III.4 Synthèse

Les modèles et méthodes pour la vérification ont démontré leur aptitude à prouver certaines propriétés des systèmes, en particulier les propriétés de sûreté exprimées sous forme d'invariance et qui permettent de garantir la sécurité des systèmes. Il est à noter que nous n'avons pas cherché à vérifier les exigences fonctionnelles des systèmes, ce qui aurait justifié la mise en œuvre de techniques permettant la vérification de propriétés telles que la vivacité ou l'équité. Les principales limites de ces techniques, en particulier dans un cadre industriel, sont relatives :

- au phénomène d'explosion combinatoire présent dans les algorithmes de *model-checking* qui limite la vérification à des composants élémentaires ou de taille réduite.

- à la formalisation des modèles de propriétés, au niveau système mais également au niveau des composants. Une mauvaise écriture de la propriété peut conduire à un échec de preuve qui sera alors difficile à interpréter.
- l'élaboration des modèles de parties opérative pour la simulation nécessite d'avoir dans un premier temps spécifié et appréhendé le comportement global du système pour ensuite raffiner ces spécifications et obtenir les modèles comportementaux recherchés.

IV - Conclusion

Dans ce chapitre, nous avons montré d'une part que les modèles orientés système sont efficaces pour capturer les exigences de sécurité à un niveau système puis pour les détailler, les préciser, les structurer et les raffiner et en définitive les mettre en correspondance avec l'architecture de composants du système. Toutefois ces modèles souffrent souvent d'un manque de formalisation limitant les possibilités de vérification des exigences spécifiées.

D'autre part, les modèles issus de l'Automatique des Systèmes à Evénements Discrets ou du Génie Informatique permettent quant à eux efficaces de vérifier des propriétés formelles mais souffrent du phénomène d'explosion combinatoire qui limite les possibilités de vérification aux seuls composants élémentaires des systèmes et de la difficulté à identifier et formaliser les propriétés à vérifier.

Ceci met en évidence la complémentarité entre une approche système permettant l'identification d'exigences « composant » à partir d'exigences « système », et des outils formels de vérification permettant de s'assurer de la conformité des composants (propriétés intrinsèques) et de leur assemblage (contribuant aux propriétés « système »). La relation entre ces deux types d'approches sera assurée par les notions pivots d'exigences et de propriétés dont la modélisation révèle de nombreuses similitudes (diagramme d'objectifs de KAOS, diagramme d'exigences de SysML, modèles de propriétés CREDI).

Chapitre 3 :

Processus de conception et traçabilité des exigences

I - Introduction

Ce chapitre présente notre contribution méthodologique à la vérification des exigences de sécurité. Cette contribution s'appuie sur :

- des modèles SysML permettant d'une part de formaliser, structurer, et tracer les exigences de sécurité exprimées au niveau système et d'autre part de représenter l'architecture du système,
- des outils de vérification formels permettant de prouver que des modèles formels des composants du système de commande respectent bien un ensemble de propriétés.

Les modèles d'exigences exprimés dans le formalisme de SysML servent de base à la formalisation de propriétés invariantes, sous la forme d'axiomes en logique temporelle ou de post-conditions, que l'on cherchera à vérifier pour chacun des constituants à l'aide respectivement de techniques de *model-checking* ou de simulation. Le passage d'un modèle d'exigences « système » à un modèle de propriétés locales est assuré par des mécanismes de raffinement, d'allocation et de projection. La cohérence entre les exigences énoncées à un niveau d'abstraction et leur raffinement en termes de propriétés locales aux constituants est, quant à elle, vérifiée à l'aide de techniques de *theorem-proving*.

II - Modèle de données et traçabilité des exigences

Définition

La traçabilité est l'aptitude à identifier les relations entre les différents artefacts du processus de développement. (EIA 632)

Cette aptitude à établir des liens entre les différentes composantes du système (exigences, composants, propriétés etc.) permet de parcourir celui-ci à la faveur du réseau de relations ainsi tissé, pour pouvoir réaliser des analyses d'impact, que ce soit des exigences vers les composants réalisés, ou des composants vers les exigences prescrites. En se basant sur les traces obtenues, on peut mesurer l'impact d'une modification des spécifications ou des exigences sur le reste du système (Von Knethen, 2001) (Von Knethen, 2002). Cette analyse d'impact est prescrite par les normes de conception.

C'est également un facteur de qualité du processus de développement, car en permettant « d'appréhender les relations existantes entre les exigences, les spécifications et la réalisation » (Palmer, 1997) elle facilite la compréhension du système dans son ensemble.

Pour cela, la traçabilité doit regrouper toutes les relations entre les exigences système et les autres objets issus du processus de définition du système. La Figure 29 représente le méta modèle de la traçabilité d'un système (Ramesh et Jarke, 2001).

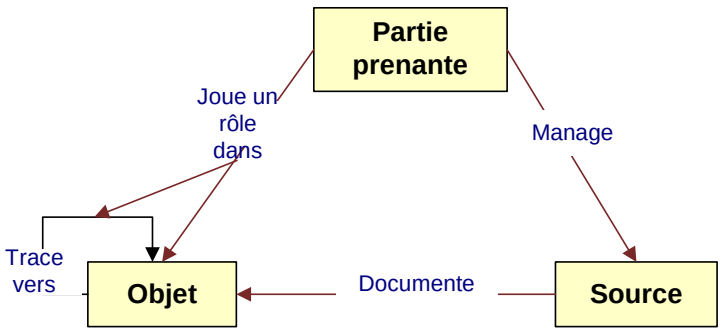


Figure 29 Méta modèle de la traçabilité (Ramesh et Jarke, 2001)

Le terme *objet* regroupe toutes les entités participant à la définition du système (exigences, constituants, scénaris de test, propriétés etc.). Le lien « trace vers » modélise alors tous les types de relations entre ces entités : liens de filiation entre exigences, allocation entre exigences et constituants font parties de ces relations. Les parties prenantes sont impliquées dans l'identification et l'utilisation des objets (l'acquéreur prescrit des exigences, le constructeur réalise et intègre des composants etc.). Elles le sont également dans

l'identification des relations entre les différents objets. Ce rôle est majoritairement celui du maître d'œuvre. Les sources documentent les objets. Elles contiennent les caractéristiques des objets. Les sources peuvent être des modèles graphiques, du texte en langage naturel, du code logiciel etc.

Tous les aspects du modèle de la traçabilité doivent être spécifiés par les différentes parties prenantes, les types de relations, l'origine des objets, le type des sources etc. Dans la pratique on distingue deux types d'utilisation de la traçabilité (Ramesh et Jarke, 2001) :

- Une utilisation « bas niveau », dans laquelle l'utilisateur se contente d'indiquer les relations entre les objets, sans préciser les réflexions, les choix, les calculs qui ont amené cette relation. Dans ce cas, le modèle de traçabilité ne permet que des analyses d'impact, il ne peut servir à justifier le processus d'ingénierie système.
- Une utilisation « haut niveau », dans laquelle les relations entre les objets sont enrichies d'explications, d'éléments de modèles, de feuilles de calcul décrivant le processus qui a débouché sur l'identification de la relation. Dans ce type d'utilisation, le modèle de traçabilité peut servir à justifier le processus d'ingénierie système.

L'utilisation « bas niveau » correspond aux exigences normatives sur le système à faire concernant les analyses d'impact, tandis que les exigences portant sur le système pour faire imposent une utilisation « haut niveau » pour inclure la documentation et la justification des diverses activités du processus de développement.

Le modèle de données basé sur ces prescriptions devra alors inclure à la fois une vue système pour faire et une vue système à faire. Le modèle proposé par l'AFIS (AFIS, 2005) est un exemple de ce que les normes prescrivent (Figure 30).

Cette première vue est enrichie par la vision système pour faire, qui intègre des informations sur les stratégies de vérification et de validation, sur les différentes solutions proposées et les choix retenus etc.

Le modèle que nous proposons pour le développement sûr de systèmes manufacturiers sûrs est adapté au contexte industriel, en prenant en compte notamment les spécificités de chacun des domaines de connaissance tout en gardant comme base la proposition de l'AFIS.

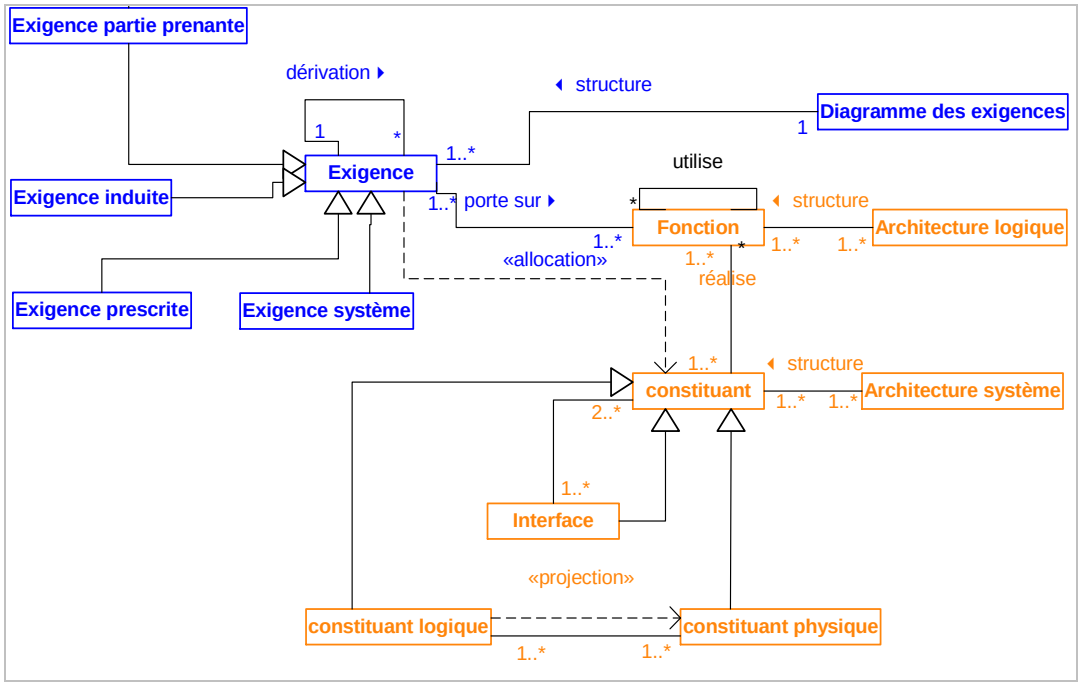


Figure 30 Modèle de données pour les systèmes « à faire » et « pour faire » (AFIS, 2005)

III - Processus proposé

Les principaux problèmes auxquels nous devons répondre, pour pouvoir tracer les exigences système de sécurité fonctionnelle jusqu'aux propriétés locales de sécurité des composants, sont liés au fait que d'une part les exigences que nous voulons satisfaire portent sur plusieurs composants, voire sur le système dans sa totalité, alors que les propriétés doivent uniquement contraindre le comportement d'un seul composant. D'autre part ces exigences sont exprimées en langage naturel alors que nous voulons obtenir des propriétés écrites en langage formel.

III.1 Formalisation du problème

Le problème que nous voulons résoudre peut être exprimé de manière simple, nous voulons montrer que le système, qui est réalisé par un ensemble de composants, satisfait toutes les exigences système de sécurité fonctionnelle. En paraphrasant le prédicat de l'automatisation, on obtient le prédicat suivant :

$$\prod_{i=1}^{n} C_i \Rightarrow \{R_k\}, k \in 0..m \quad \textbf{(1)}$$

Les R_k représentent les exigences de sécurité du système, tandis que les C_i représentent les comportements des composants du système. Il s'agit donc de montrer que le produit des comportements des composants inclus bien l'ensemble des exigences de sécurité du système. Les outils de *model-checking* permettent de prouver que les composants pris individuellement vérifient bien un à un les propriétés de sécurité.

$$C_i \Rightarrow \prod_{k=1}^{k=Xi} P_{k,i} \quad \textbf{(2)}$$

Les $P_{k,i}$ représentent les propriétés de sécurité que le composant i doit satisfaire. La traçabilité des exigences doit nous servir à trouver des propriétés $P_{k,i}$ telles que si tous les composants respectent le prédicat (2), alors le prédicat (1) est respecté.

Problème posé par la vérification formelle de composants : Pour chaque composant k du système, trouver les propriétés de sécurité $P_{k,i}$ telles que :

$$\left(\text{Pour tout i, } C_i \Rightarrow \prod_{k=1}^{k=Xi} P_{k,i}\right) \Rightarrow \left(\prod_{i=1}^{n} C_i \Rightarrow \{R_k\}, k \in 0..m\right) \quad \textbf{(3)}$$

Les chapitres III.2 et III.3 définissent un ensemble de mécanismes du processus de définition du système permettant d'identifier les exigences, les fonctions, et les composants du système ainsi que leurs relations de traçabilité. Le paragraphe III.4 montrent comment ces relations permettent à l'intérieur du processus de réalisation, d'implémenter des solutions répondant aux exigences système.

III.2 Identification des exigences

III.2.1 Analyse de risque et formalisation des exigences de sécurité

La boucle d'ingénierie système (Figure 9) définit les interactions entre le processus d'identification des exigences et le processus de conception ainsi qu'entre les différents passages de cette boucle. Nous allons intégrer le processus d'appréciation et de réduction du risque à cette boucle.

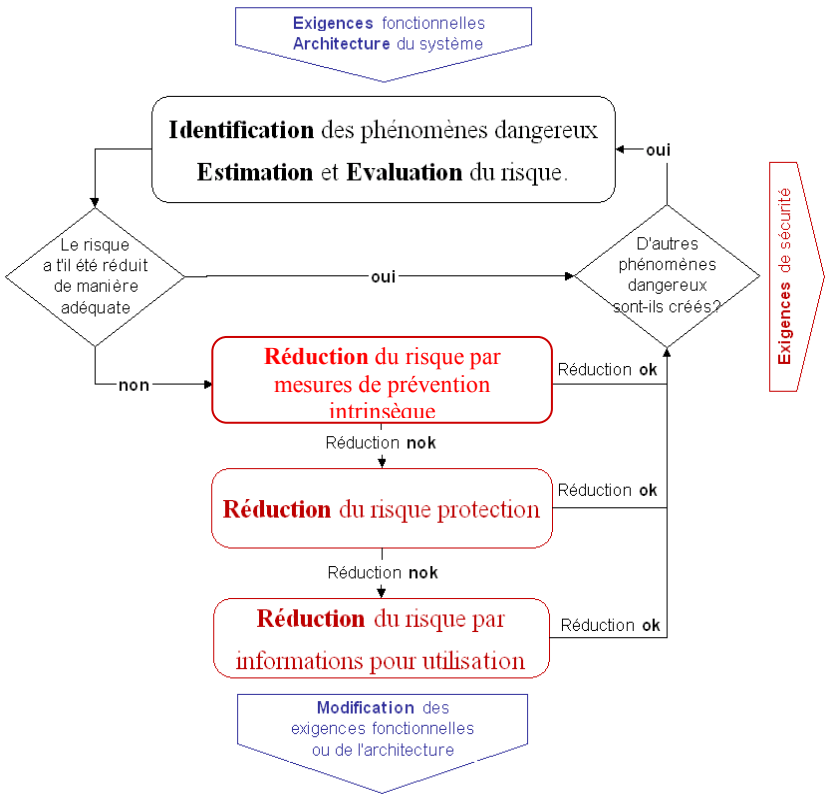


Figure 31 Processus d'appréciation et de réduction du risque simplifié (ISO 12100-1)

Ce processus (Figure 31) est composé de deux grandes activités, une activité d'identification et d'évaluation du risque, qui a pour objectif d'identifier les phénomènes dangereux et d'évaluer le risque associé, et une activité de réduction du risque, qui consiste à

émettre des exigences concernant les mesures de prévention ou les dispositifs de protection à mettre en place pour réduire le risque. Ce processus définit des exigences de sécurité uniquement si les exigences, et les choix de conception retenus pour le système, permettent une réduction adéquate du risque. Si, au contraire, aucune stratégie de réduction du risque ne peut être mise en œuvre du fait de ces exigences et des choix de conception, alors ceux ci doivent être revus. L'intégration du processus de réduction du risque dans la boucle d'ingénierie système doit alors souligner ses interactions avec le processus de définition des exigences et le processus de conception qui sont :

- les exigences fonctionnelles issues du processus d'identification des exigences et les choix d'architecture issus du processus de conception qui sont les données d'entrée de l'activité d'évaluation du risque,
- les exigences de sécurité issues de l'activité de réduction du risque qui doivent faire l'objet d'un processus de conception,
- et les modifications éventuelles des exigences fonctionnelles ou des choix d'architecture issus de l'impossibilité de réduire le risque par des méthodes de prévention, de protection ou d'information.

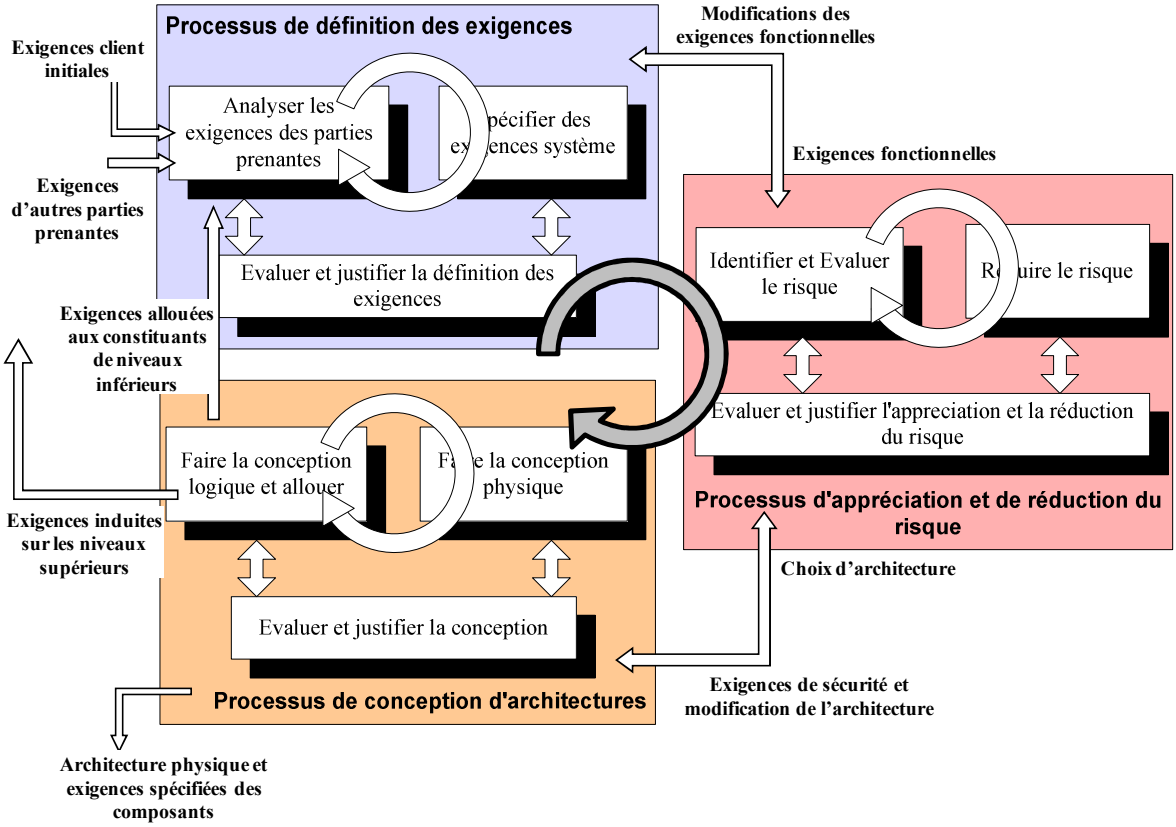


Figure 32 Processus de définition pour les systèmes critiques

Le processus d'appréciation et de réduction du risque s'intègre dans le processus de définition du système, non pas comme une étape obligatoire, mais comme un processus disponible (Figure 32). Il est utilisé dans les cas où la définition des exigences ou les choix de conception sont susceptibles d'induire des risques pour l'utilisateur. Dès lors, ces risques doivent être évalués et réduits.

III.2.2 Identification et raffinement des exigences

Au cours du développement, les exigences, et plus particulièrement les exigences de sécurité, évoluent en fonction des solutions retenues, des risques encourus par l'opérateur, des mesures de prévention/protection à mettre en place. En réalité, et c'est le but d'un développement itératif, les exigences vont être de plus en plus détaillées, à savoir que les exigences de sécurité prescrites au niveau du système vont peu à peu être remplacées par des exigences portant sur la mise en oeuvre et l'intégration des moyens de protection mis en place. Ces exigences sont en quelque sorte raffinées au cours du développement.

Le mécanisme de raffinement tel qu'il existe dans la méthode B par exemple consiste à rendre un modèle de plus en plus concret par ajouts successifs de détails tout en garantissant le respect des propriétés invariantes des modèles précédents. Pour les exigences, il s'agit en fait de substituer aux exigences abstraites (niveau système) des exigences de plus en plus concrètes et détaillées (niveau composant). Cette substitution ne peut être effective qu'à la condition que les exigences concrètes incluent les exigences abstraites, c'est-à-dire, que les exigences $R_{i,1}…R_{i,n}$ sont un raffinement de l'exigence R_i si et seulement si :

$$\{R_{i,k}\}, k \in 0..n \Rightarrow R_i \quad \textbf{(4)}$$

Les exigences $R_{i,k}$ sont le produit des passages successifs par le processus de définition du système, et plus particulièrement du processus d'appréciation et de réduction du risque. Le prédicat (4) signifie que le raffinement d'une exigence peut imposer des contraintes plus importantes que l'exigence initiale, à condition de préserver les contraintes fixées par cette exigence.

III.2.2.1 Mécanismes de modélisation et de raffinement

Dans le formalisme SysML, l'exigence est définie par un stéréotype de classe. La définition d'une exigence est le résultat d'un processus d'identification des exigences pour les exigences fonctionnelles, d'appréciation et de réduction du risque pour les exigences de

sécurité fonctionnelle, ou de conception pour les exigences techniques. Les exigences techniques peuvent représenter le choix du concepteur d'utiliser au maximum des composants logiciels disponibles dans le commerce afin de limiter le développement de ce type de composants dans le processus de développement.

Les exigences sont modélisées dans le diagramme de « Requirements » de SysML. Le mécanisme de raffinement est représenté par le lien de composition qui permet de substituer des exigences abstraites par des exigences plus concrètes (Figure 33). Le système ne doit alors satisfaire que les exigences concrètes, le lien de raffinement signifiant que si un système satisfait aux exigences concrètes, alors il satisfait également l'exigence abstraite.

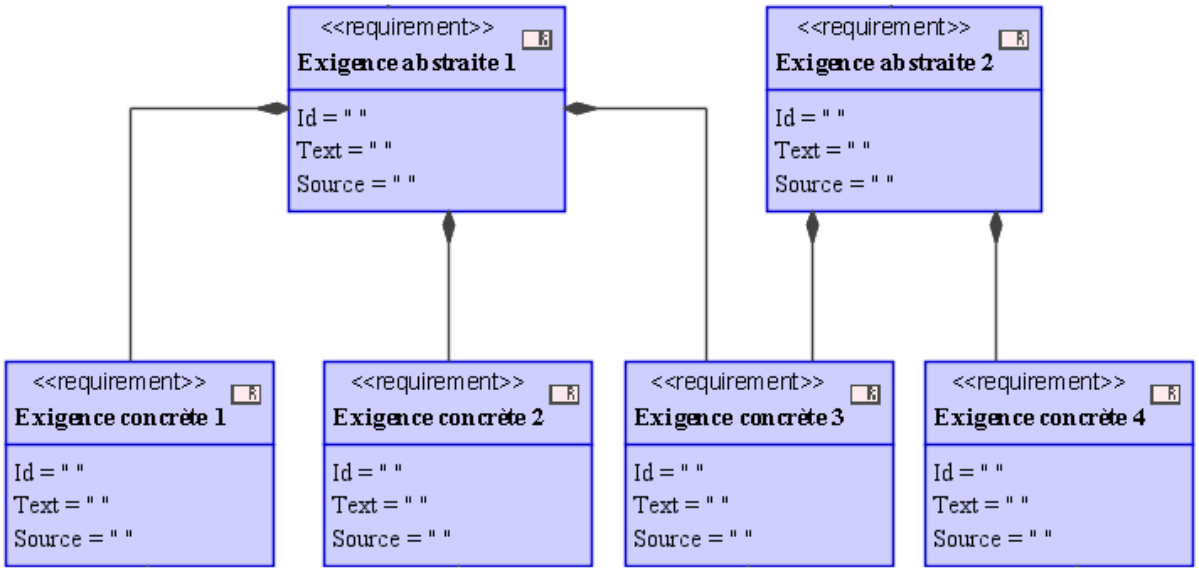


Figure 33 Lien de raffinement entre une exigence abstraite et des exigences concrètes

III.2.2.2 Règles de raffinement des exigences en SysML

La construction du modèle d'exigences et, en particulier, la définition des relations de raffinement doit être encadrée par des règles de bonne pratique. Le bon respect de ces règles de construction est en soi un gage de qualité du modèle :

- Règle de raffinement 1 : Une exigence de sécurité ne peut être raffinée que par des exigences de sécurité.
- Règle de raffinement 2 : Le raffinement des exigences ne doit pas contenir de boucle.

– Règle de raffinement 3 : Il faut éviter qu'une exigence soit raffinée par un trop grand nombre d'exigences. Cela peut être le cas lorsque des étapes ont été sautées dans le processus de développement itératif, ce qui arrive souvent.

– Règle de raffinement 4 : Il faut éviter les exigences raffinées par une seule exigence. C'est souvent la marque d'un manque de précision lors de la spécification initiale de l'exigence.

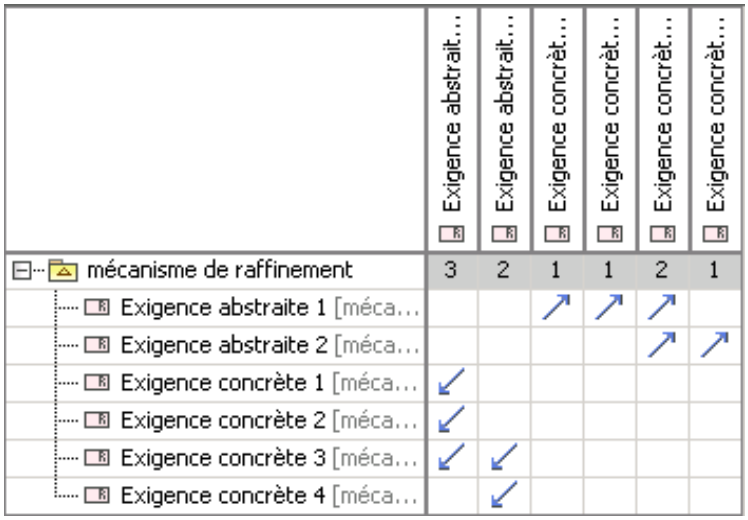


Figure 34 Matrice de dépendance du critère « raffinée par »

La Figure 34 est une matrice de dépendance issue du logiciel Magicdraw. Celle-ci représente les relations « raffinée par » qui existent entre les exigences de la Figure 33. Cette matrice peut être exportée vers un tableur pour vérifier les 4 règles définies plus haut.

Ces règles vérifient seulement la cohérence et la qualité du modèle d'exigences. Elles n'apportent pas la preuve qu'une exigence est bien raffinée par les exigences qui sont liées à elle par le lien de raffinement. Outre la vérification formelle par preuve de propriété que nous allons développer plus bas, il est possible de montrer de manière qualitative que ce lien de raffinement existe. Par exemple, il est possible de s'appuyer sur le fait que l'exigence abstraite est prescrite par une norme générale de type A faisant référence à des articles de normes plus précises de type C, articles qui sont à l'origine d'exigences concrètes. Ce type de justification doit alors apparaître sur le lien de raffinement.

III.2.2.3 Relations exigences / propriétés

Pour pouvoir employer des outils de vérification formelle, les exigences de sécurité, qu'elles soient exprimées au niveau du système ou au niveau composant, doivent être formalisées par des propriétés.

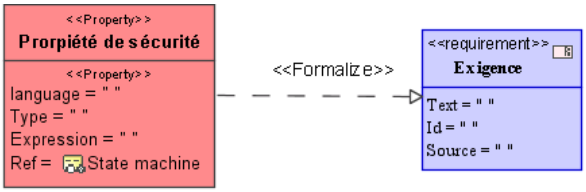


Figure 35 Modèle de propriété en SysML

Afin de modéliser les propriétés et leurs relations avec les exigences, nous avons introduit un nouveau stéréotype de classe, possédant plusieurs attributs :

- le langage dans lequel est exprimée la propriété. Celui ci doit être formel, on peut citer la logique temporelle, la logique des prédicats, le langage B etc. La préférence allant à la logique temporelle qui est utilisée par les *model-checker* ;
- Le type qui permet de préciser si la propriété exprime une contrainte combinatoire, séquentielle ou temporelle.
- L'expression définit la propriété elle-même, tandis que la référence permet de lier la propriété à une machine à état permettant de modéliser les propriétés impliquant des séquences complexes par des observateurs représentés sous forme de modèles à état.

La propriété est ensuite reliée à une exigence par un lien « formalize » que nous avons également ajouté au modèle SysML. Une exigence peut être formalisée par une ou plusieurs propriétés, mais une propriété ne peut formaliser qu'une exigence.

Les logiciels basés sur le formalisme SysML offrent des outils d'analyse des modèles, notamment des matrices de dépendance permettant de visualiser de manière matricielle les liaisons entre les objets du modèle. En utilisant ces matrices de dépendance il est possible de vérifier certains critères de qualité sur la relation de formalisation.

- Règle de formalisation 1 : chaque exigence de sécurité doit être formalisée par une propriété.
- Règle de formalisation 2 : aucune propriété ne peut formaliser plus d'une exigence.

Ces deux règles sont des critères qualitatifs sur le modèle global, et permettent d'assurer une certaine qualité de la formalisation des exigences. Maintenant que les exigences sont formalisées par des propriétés, nous allons pouvoir mettre en place des méthodes formelles de vérification pour le mécanisme de raffinement.

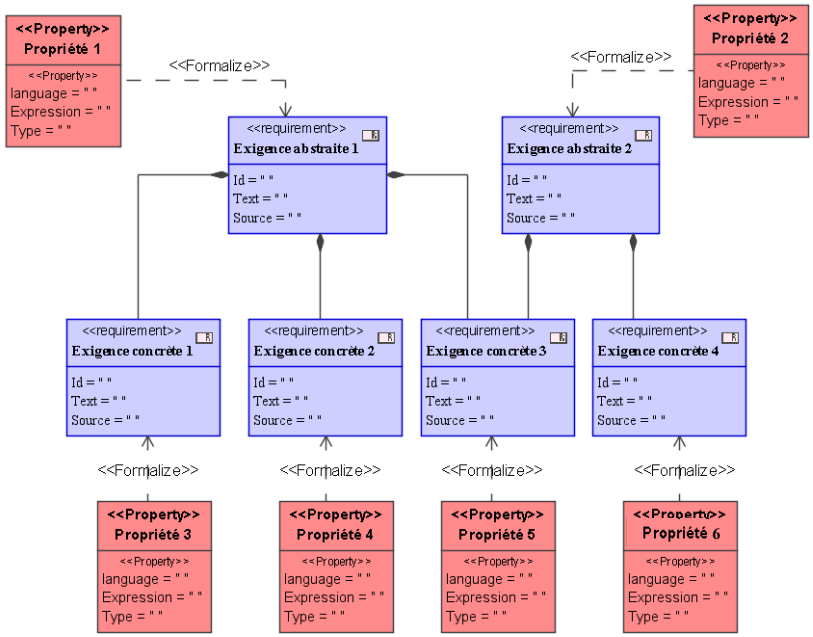


Figure 36 Lien de raffinement entre propriétés

Le lien de raffinement relie les exigences, mais il a également un impact sur les propriétés, puisque si les exigences concrètes 3 et 4 raffinent bien l'exigence abstraite 2, alors les propriétés 2, 5 et 6 doivent respecter le prédicat (4) associé au mécanisme de raffinement. On peut noter qu'une exigence (exigence concrète 3) peut faire partie du raffinement de plusieurs exigences (exigences abstraites 1 et 2).

Il est bien entendu utopique d'imaginer que chaque processus d'appréciation et de réduction du risque se contente de raffiner les exigences prescrites par les itérations précédentes. Il peut également prescrire de nouvelles exigences qui ne se substituent pas aux précédentes mais les complètent, en s'ajoutant au modèle d'exigences. Dans ce cas, un lien de dérivation permet de relier l'exigence additionnelle aux exigences existantes. Un exemple type d'utilisation de ce lien de dérivation est le cas d'un système pour lequel on désire plusieurs modes de fonctionnement différents. Un ensemble d'exigences défini le fonctionnement de ces modes, auxquelles on ajoute une exigence définissant le protocole de changement de mode qui « dérive » des autres exigences. Dans ce cas là, le système doit satisfaire les exigences définissant les modes ainsi que l'exigence de changement de mode.

Grâce à ces deux liens on est capable de construire un modèle d'exigences pour le système, en partant d'un ensemble d'exigences système pour obtenir un ensemble d'exigences détaillées.

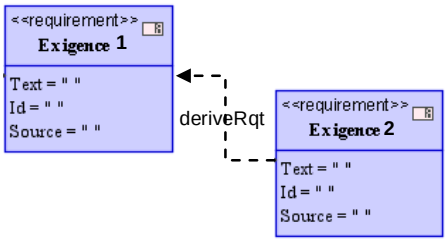


Figure 37 Lien de dérivation

III.2.2.4 Vérification formelle *a posteriori* des relations de raffinement

Pour vérifier de manière formelle les relations de raffinement, nous nous sommes appuyés sur l'assistant de preuve COQ (Huet *et al*, 1995). La vérification est effectuée automatiquement par l'assistant, l'utilisateur choisissant les tactiques que l'assistant doit utiliser pour réussir la preuve.

La modélisation des propriétés, ainsi que la réalisation de la preuve nécessite un certain savoir-faire, mais surtout cette preuve doit être réalisée en étroite collaboration avec les personnes ayant réalisé l'analyse de risque. En effet une propriété évidente pour la personne ayant réalisé l'analyse de risque, et qui par conséquence ne serait pas exprimée dans les exigences, peut bloquer le mécanisme de preuve.

Par exemple, prenons le cas d'une exigence de sécurité indiquant qu'il faut empêcher les dommages liés à un mouvement de la machine, qu'on appellera mouvement dangereux. Cette exigence est formalisée par la propriété P1 : (dommage_mvt = false).

L'analyse des normes et des solutions existantes conduit à raffiner cette exigence par deux nouvelles exigences, l'une prescrivant l'utilisation d'un protecteur mobile pour empêcher l'accès à la zone dangereuse, et l'autre prescrivant que ce protecteur soit inter verrouillé avec le mouvement dangereux. Ces deux exigences sont formalisées par les propriétés P2 : (Protecteur = true $\Rightarrow$ Zone_dangereuse = false) et P3 (Protecteur = false $\Rightarrow$ Mvt_dangereux = false). La preuve de raffinement doit alors montrer que P2 $\wedge$ P3 $\Rightarrow$ P1. Cela peut paraître trivial, pourtant l'assistant ne peut pas résoudre cette preuve car une hypothèse n'a pas été donnée : il ne peut y avoir un dommage que si l'accès à la zone dangereuse est libre et que le mouvement dangereux existe. Cette hypothèse H (dommage_mvt = true $\Rightarrow$ Mvt_dangereux = true $\wedge$ Zone_dangereuse = true) permet de réaliser la preuve : H $\wedge$ P2 $\wedge$ P3 $\Rightarrow$ P1.

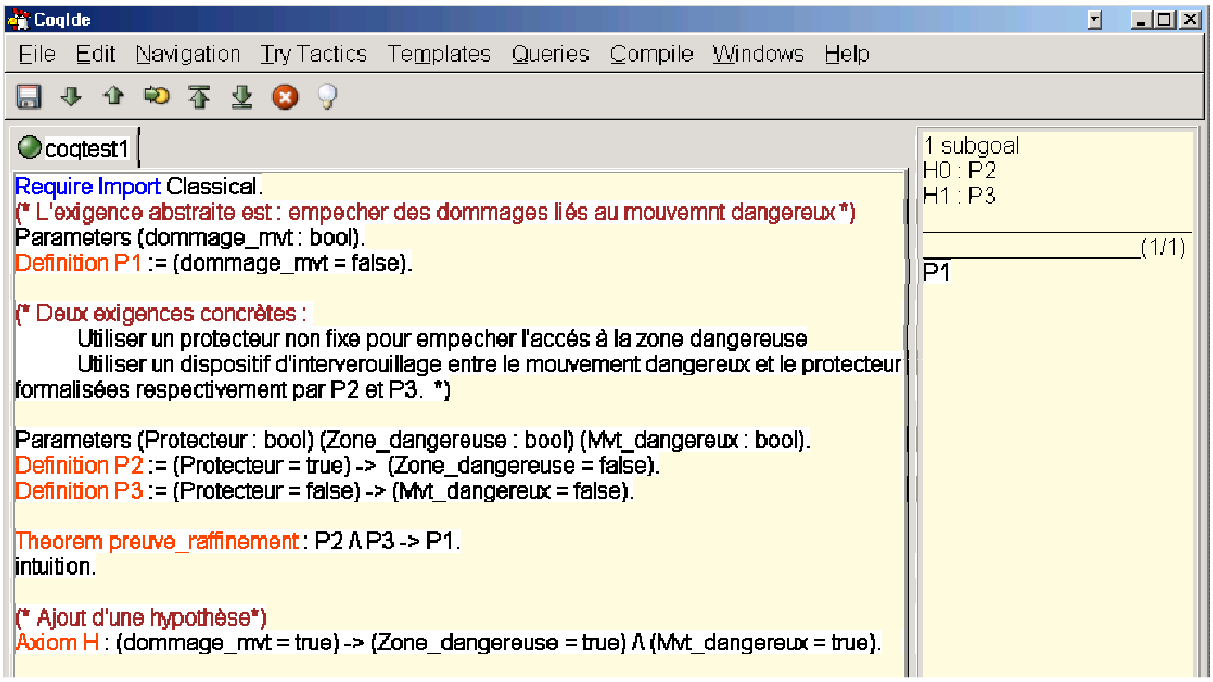


Figure 38 Preuve réalisée avec l'assistant de preuve COQ

Le lien de raffinement peut donc être vérifié de manière formelle par l'intermédiaire d'un assistant de preuve tel que le logiciel COQ, mais cette activité de vérification doit inclure obligatoirement les personnes ayant participé à l'élaboration des exigences.

III.3 Allocation et projection des exigences

En parallèle de la construction des modèles d'exigences, nous construisons les architectures fonctionnelles (fonctions activités) et physiques (composants matériels et logiciels) du système. On peut ainsi allouer de manière progressive les exigences identifiées à un ensemble de fonctions et de composants.

III.3.1 Allocation et projection des exigences : principes

Le mécanisme de raffinement nous assure d'obtenir des exigences détaillées préservant les exigences système de départ. Il s'agit maintenant, à partir de ces exigences détaillées, d'obtenir des exigences ne portant que sur un et un seul composant.

Pour arriver à cela, il faut dans un premier temps identifier les composants participant à la réalisation d'une exigence particulière. Cette identification est basée sur les résultats du processus de conception, qui permet de relier les exigences aux composants, soit de manière directe, soit de manière détournée en passant par une analyse fonctionnelle. L'exigence est

alors allouée aux composants identifiés. Cette allocation signifie que le sous-ensemble de composants auquel est allouée l'exigence est suffisant pour satisfaire cette dernière. C'est-à-dire que si l'exigence R_i est allouée aux composants 1 à j, alors :

$$(\prod_{k=1}^{k=n} C_k \Rightarrow R_i) \Leftrightarrow (\prod_{k=1}^{k=j} C_k \Rightarrow R_i) \quad \textbf{(5)}$$

Pour vérifier que le système dans sa totalité satisfait l'exigence R_i, il suffit de vérifier que les composants 1 à j la satisfont. Toutefois, notre problème n'est pas encore résolu, car une exigence est allouée à plusieurs composants, or nous voulons des propriétés qui ne sont satisfaites que par un composant. La dernière étape est donc de projeter les exigences sur les composants. Cette projection doit permettre de révéler le rôle de chacun des composants dans la réalisation de l'exigence de sécurité. L'ensemble des exigences projetées doit être équivalent égal à l'exigence initiale ; ainsi si ces exigences projetées sont satisfaites par les différents composants, on aura la preuve que l'exigence initiale l'est aussi. C'est à dire que si une exigence R_i est allouée aux composants 1 à j, alors les exigences projetées $R_{i,k}$ doivent vérifier :

$$\{R_{i,k}\}, k \in 0..j \Leftrightarrow R_i \textit{ où } R_{i,k} \textit{ est la projection de } R_i \textit{ sur le composant } k \quad \textbf{(6)}$$

La projection des exigences nous permet alors d'obtenir des exigences satisfaites par un et un seul composant.

III.3.2 Modélisation SysML

L'architecture fonctionnelle est définie en utilisant les activités et les cas d'utilisation de SysML. Une activité est définie par des opérations appelées, des informations échangées, et peut être associée à une vue architecturale utilisant le diagramme d'activité de SysML. Cette vue permet de définir de manière précise les informations échangées entre les différentes activités du système.

Les opérations appelées par les activités sont réalisées par les composants du système. Ces composants sont modélisés par des « blocks » qui sont également des stéréotypes de classe. Ces blocks sont donc des objets « instanciables » à volonté et réalisant des opérations appelées dans les différentes activités du système. Ils sont définis par les opérations qu'ils réalisent ainsi que par leurs interfaces appelées « ports ». Les blocks sont modélisés par deux diagrammes, le diagramme de définition de blocks, qui permet de définir l'architecture du block ainsi que les opérations réalisées, et le diagramme de définition interne de blocks qui

permet de définir les ports de chacun des blocks ainsi que les variables échangées entre les différents blocks via ces ports. Il s'agit en fait d'un diagramme d'instance.

L'allocation des exigences a donc pour but de définir le sous-ensemble de composants suffisant pour la réalisation d'une exigence. Cette identification doit s'appuyer sur les liens existant entre les exigences, les activités, les opérations et les composants ; liens issus des divers sous-processus du processus de définition. La relation d'allocation est seulement une représentation plus directe des relations existantes entre composants et exigences. Une exigence sera donc allouée à un composant si :

- il existe un lien de satisfaction entre l'exigence et le composant,
- une des opérations réalisée par le composant est appelée par une activité ayant un lien de satisfaction avec l'exigence.

L'allocation des exigences au composant est préservée par la relation de composition entre composants. C'est-à-dire que si une exigence est allouée à un composant, elle est également allouée à ses sous composants. Cela signifie que la propriété doit être projetée non pas sur le composant principal, mais sur ses sous composants.

Une fois que le lien d'allocation existe, il est possible de passer au mécanisme de projection. Les propriétés résultantes du mécanisme de projection sont reliées à la propriété initiale par un lien de composition. Chacune est reliée par un lien de vérification au composant sur lequel elle a été projetée (Figure 39).

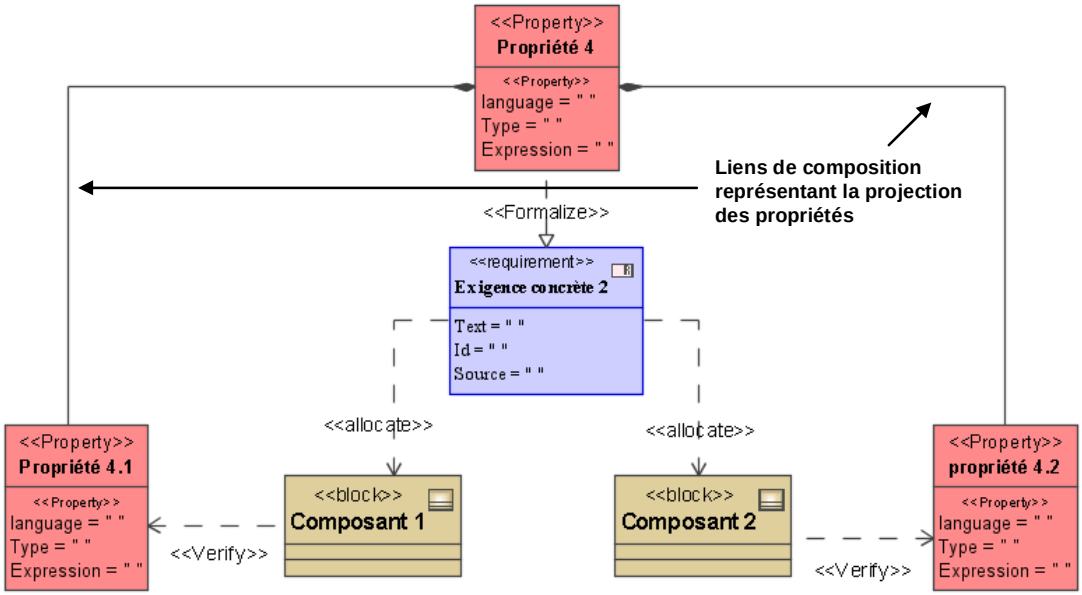


Figure 39 Relations entre allocation, projection et vérification en SysML.

III.3.3 Règles d'allocation et de projection des exigences

Comme nous l'avons dit plus haut, la relation d'allocation ne fait que préciser des relations existantes. Il faut donc s'assurer qu'elle est bien en accord avec ces relations. Pour cela il suffit que les règles suivantes soient respectées :

- Règle d'allocation 1 : toute exigence ayant un lien de satisfaction avec un composant doit être alloué à ce dernier.
- Règle d'allocation 2 : toute exigence ayant un lien de satisfaction avec une activité doit être allouée aux composants réalisant les opérations appelées par la dite activité.
- Règle d'allocation 3 : tous les liens d'allocation doivent relever de la règle 1 ou de la règle 2.

Ces règles suffisent à elles seules pour vérifier que les relations d'allocation sont correctes, il n'est pas nécessaire d'utiliser des moyens plus formels.

Le mécanisme de projection est plus délicat, il est en effet possible de préciser certaines règles permettant de s'assurer que les propriétés locales sont cohérentes avec le modèle, mais le respect de ces règles ne prouvera pas que les propriétés résultantes du mécanisme correspondent bien à la projection de la propriété principale sur les composants. Les règles de projection sont les suivantes :

- Règle de projection 1 : Chacune des propriétés composant la propriété projetée ne doit être liée qu'à un et un seul composant via un lien de vérification.
- Règle de projection 2 : Une propriété vérifiée par un composant ne doit se baser que sur les variables d'interface de ce composant.
- Règle de projection 3 : Une propriété formalisant une exigence allouée à un macro composant doit être projetée sur tous les sous composants.

Ces règles nous permettent d'avoir l'assurance que la propriété a été projetée sur les bons composants, que les propriétés locales obtenues ne doivent être vérifiées que par un et un seul composant.

III.3.4 Vérification formelle *a posteriori* de la projection

La projection des propriétés doit permettre d'obtenir les propriétés locales des composants, propriétés qui doivent respecter le prédicat (6) associé au mécanisme de projection.

$$\prod_{k=0}^{k=j} P_{i,k} \Leftrightarrow \mathrm{P_i} \text{ où } P_{i,,k} \text{ est la projection de } P_i \text{ sur le composant } k \quad \textbf{(6)}$$

Le mécanisme de projection diffère du mécanisme de raffinement au sens où avec le mécanisme de raffinement, on cherche à montrer que des exigences créées par un processus d'identification raffinent des exigences existantes, alors qu'avec le mécanisme de projection, on cherche à identifier un ensemble de propriétés locales vérifiant la propriété principale.

L'emploi d'une méthode de vérification formelle à posteriori telle que celle développée pour le mécanisme de raffinement, permet de montrer que les propriétés locales identifiées vérifient bien la propriété principale. L'emploi d'un assistant de preuve tel que COQ apporte donc un plus par rapport aux règles de projection décrite plus haut.

Toutefois cette preuve d'équivalence entre les propriétés locales et la propriété principale n'est pas suffisante pour prouver de manière formelle que les propriétés locales sont bien la projection de la propriété principale sur les composants qui doivent la vérifier. Il existe en effet une infinité de décompositions de la propriété P_0. Dans le cas de notre étude, une seule décomposition nous intéresse : celle ne comportant que des propriétés devant être respectées par un et un seul composant.

La preuve d'équivalence montre que les propriétés locales sont une décomposition de la propriété principale. Toutefois, il existe un nombre infini de décompositions de la propriété principale en propriétés locales. Or nous cherchons uniquement celle pour laquelle il y a adéquation entre les propriétés locales et les composants. En effet, pour les autres décompositions, il nous sera impossible de prouver qu'un composant satisfait une propriété locale. Nous avons donc développé une méthode basée sur la méthode B permettant l'identification formelle des propriétés locales à partie d'une propriété principale et d'une architecture de composants.

III.3.5 Vérification formelle a priori de la projection

Pour obtenir ces propriétés, nous nous basons sur la génération automatique d'obligation de preuve incluse dans la méthode B (Evot et al, 2006b) (Figure 40).

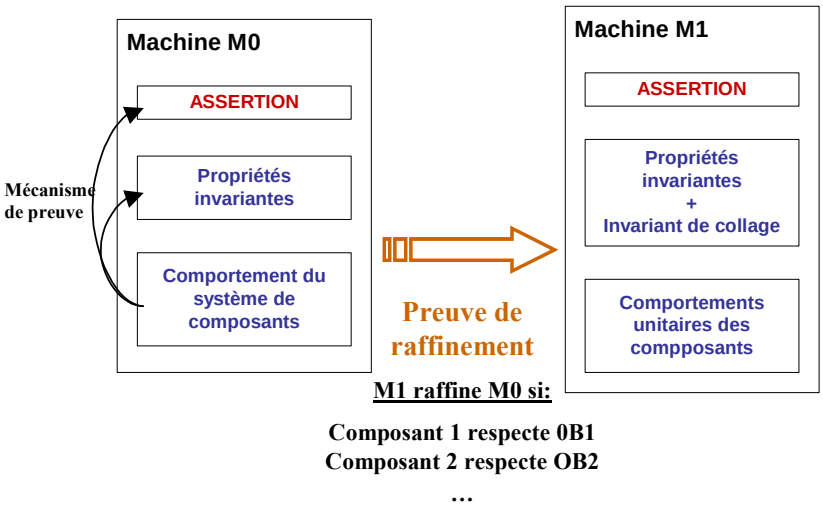


Figure 40 Utilisation de la méthode B pour la projection de propriétés

La machine M0 comporte une assertion, qui est en réalité la propriété P_0 que l'on veut projeter ; des propriétés invariantes, qui fixent notamment les types des variables et leur limites ; et d'opérations qui décrivent le comportement global attendu par l'ensemble des composants *k*. Ce comportement est décrit de manière très abstraite, ne donnant que les états principaux du système de composant.

Un mécanisme de preuve prouve de manière automatique que le modèle de comportement préserve bien à la fois les propriétés invariantes et l'assertion. Ce mécanisme peut être réalisé automatiquement, mais uniquement dans le cas de systèmes simples, c'est pourquoi il est important que nous appliquions le mécanisme de projection que sur des composants de bas niveaux, et à partir d'exigences détaillées et raffinées. Dans le cas contraire, un certain nombre de preuves doivent être réalisées à la main, ce nombre augmentant avec la complexité du modèle.

La machine M1 reprend l'assertion définie par la machine M0 ainsi que les propriétés invariantes, en y ajoutant les invariants de collage permettant de faire le lien entre les variables des deux machines. La modification principale est liée à la définition des comportements unitaires des composants. La partie comportementale est divisée en k ensembles d'opérations qui représentent le comportement des composants. Encore une fois cette représentation reste abstraite, elle ne montre que les évolutions des variables d'entrées/sorties dans les différents états du système. Ces modèles de comportement doivent s'appuyer sur les comportements attendus des composants. Leur élaboration nécessite des échanges avec les différents acteurs du processus de développement.

Finalement, la preuve de raffinement entre M1 et M0 va nous permettre d'obtenir les propriétés projetées sur les composants. En effet la génération automatique d'obligation de preuve va identifier des obligations OB_i telles que M1 raffine M0 si et seulement si M1 respecte les OB_i. La dernière étape est donc d'identifier parmi les OB_i celles correspondant aux divers composants.

Cette identification est réalisée de manière assez simple à l'aide d'un système de drapeau mis en place dans le modèle de comportement. En effet, dans un modèle B, l'assertion et les propriétés invariantes doivent être préservées par toutes les opérations du modèle. Or dans le cas d'un modèle multi-composants, elles ne pourront être préservées que par un cycle d'évènements incluant chacun des composants. Pour que cette notion de cycle d'exécution des évènements apparaissent dans notre modèle nous avons du incorporer un moteur d'exécution des évènements. Il a pour but d'obliger à ce que dans chaque cycle d'exécution, il n'y ait qu'un seul événement exécuté par composant. Ce moteur d'exécution repose sur un système de drapeaux qui va permettre d'ordonnancer l'exécution des évènements. La variable *Drapeau* peut prendre les valeurs D_{ij}, où *i* correspond à l'ordre d'exécution du composant, et *j* à l'ordre d'exécution de l'événement dans le composant. La condition $Drapeau = D_{ij}$ est insérée dans la garde (select) de l'événement *ij*, la variable drapeau est actualisée par les substitutions des évènements.

Event ij =
Select Drapeau = D_{ij}
Then If G_{ij} Then S_{ij} || Drapeau := $D_{(i+1)1}$
Else Drapeau := $F_{i(j+1)}$
End;

Où G_{ij} et F_{ij} représentent respectivement la garde et la substitution de l'événement ij

L'assertion et les propriétés invariantes ne doivent être vérifiées que lorsque le cycle d'événements a bien été exécuté, c'est-à-dire que *Drapeau* est égale D_{11}. Parmi les obligations de preuve générées celles qui sont projetées sur le composant k sont celles qui contiennent les variables D_{ki}.

III.3.6 Conclusion sur la vérification formelle de la projection

L'utilisation de la méthode B permet d'identifier de manière formelle les propriétés projetées à partir de la propriété principale et de l'architecture de composants. Toutefois, l'utilisation de cette méthode reste relativement difficile en raison de la nécessité et de la difficulté de manipuler le langage B, de la nécessité de connaître et de modéliser de manière

formelle le comportement des composants, et de la nécessité de manipuler les obligations de preuves générées afin d'identifier celles correspondant à un composant particulier.

Pour ces raisons, nous préférons vérifier à l'aide d'un assistant de preuve que des propriétés $P_{0,k,}$ identifiées de manière non formelle, sont bien équivalentes à la propriété principale. Cette première solution combinée aux métriques de projection définies plus haut, permet d'assurer que le mécanisme de projection a été réalisé de manière correcte. De cette manière, si les propriétés locales sont vérifiées par les composants, alors il est sûr que la propriété principale est vérifiée. En cas d'échec de preuve des propriétés locales, alors il sera peut être nécessaire de revoir la projection.

III.3.7 Composants de sécurité et composants fonctionnels

Lors de l'étude du contexte normatif, nous avons mis en avant le fait que si un composant réalise à la fois des fonctions de sécurité et des fonctions nominales, alors le composant dans son ensemble devra être considéré comme relatif à la sécurité sauf s'il peut être prouvé que les fonctions relatives à la sécurité et les fonctions nominales sont suffisamment indépendantes dans leur réalisation. Il est donc intéressant de minimiser au maximum le nombre de composants alloué à la fois à des exigences de sécurité et à des exigences fonctionnelles. On obtient ainsi des architectures de composants avec de manière séparées des composants fonctionnels et des composants de sécurité. L'objectif étant que les composants de sécurité servent de filtre aux composant fonctionnels (Marangé et al, 2007) de telle manière que quelque soient le comportement des composants implantés derrière ces filtres, les propriétés de sécurité restent vraies.

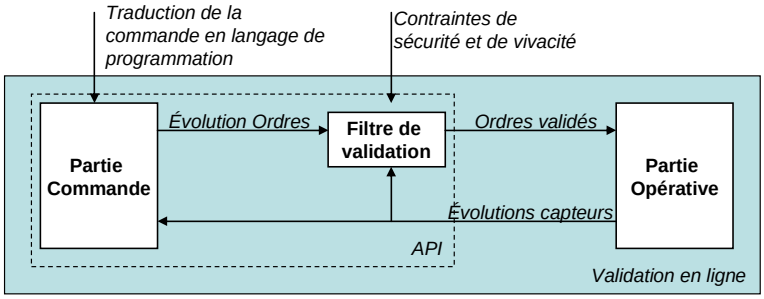


Figure 41 Filtre de sécurité, schéma de principe (Marangé et al, 2007)

L'activité de vérification est donc séparée en deux avec des outils de simulation pour valider le comportement des spécifications, des outils d'émulation pour vérifier le

comportement réel des composants fonctionnels, et des outils de *model-checking* pour la vérification des composants de sécurité implémentés.

III.4 Processus de réalisation

III.4.1 Définition

Le processus de réalisation contient 3 activités, la spécification comportementale des composants, l'implémentation des composants, et la vérification de ces composants.

La spécification comportementale des composants, consiste à spécifier les opérations définies par le processus de définition du système. Cette activité est identique qu'il s'agisse d'un composant de sécurité ou d'un composant nominal.

L'implémentation consiste à coder le comportement spécifié dans un des langages utilisés par l'automate. Cette phase peut être automatisée en fonction du langage utilisé pour la spécification. En effet la traduction des langages à état tels que le Grafcet vers des langages de programmation de la norme ISO 61131-3 est complètement maîtrisée, voire automatisée. Dans le cas de composants de sécurité, il est nécessaire de préciser à l'automate, via le logiciel constructeur fourni, que ces composants participent à la sécurité du système afin qu'ils soient exécutés de manière redondante par les deux voies de l'APIdS.

Enfin, l'activité de vérification et de validation doit permettre de montrer que les composants spécifiés et implémentés respectent bien les exigences spécifiées par le processus de définition du système.

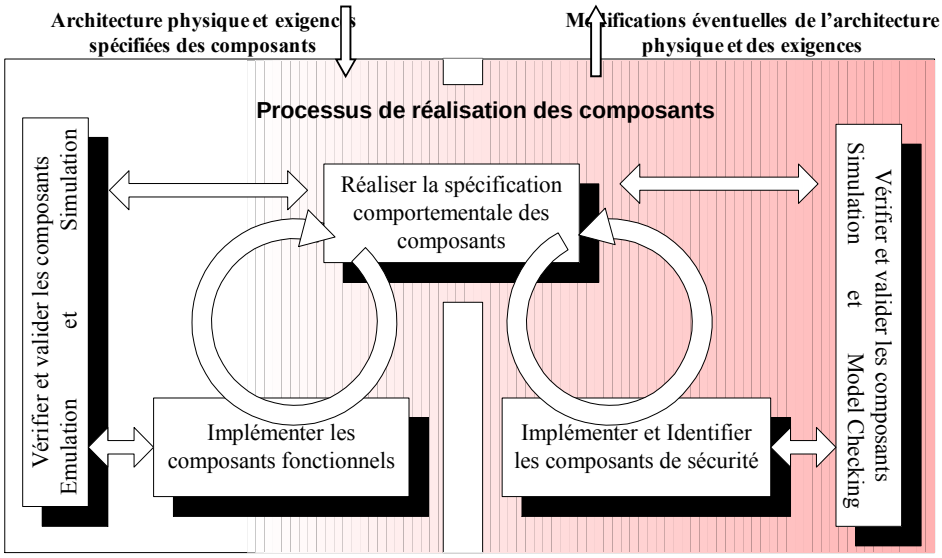


Figure 42 Processus de réalisation des composants

Le processus de réalisation des composants (Figure 42) permettra de réaliser des composants de sécurité et des composants fonctionnels. Il se base sur l'architecture de composants définie par le processus de conception ainsi que sur les exigences spécifiées des composants. Ces deux éléments peuvent d'ailleurs être remis en cause si lors de la réalisation, on s'aperçoit que l'architecture est inappropriée, ou que les exigences spécifiées ne sont pas correctes. Ce processus de réalisation des composants, associé au processus de conception d'architecture compose la boucle de conception/réalisation.

III.4.2 Modèles utilisés.

La spécification comportementale des composants requiert d'utiliser des modèles à états pour les composants les plus complexes et de simples modèles (équations) combinatoires pour les composants les plus simples. Une fois encore SysML permet de spécifier le comportement attendu d'un composant à l'aide des machines à états (proche des statecharts). Toutefois ces modèles, s'ils peuvent être exécutés par certains outils basés sur SysML afin de valider leur comportement, ne permettent pas de simuler le comportement de la partie mécanique du système. Une autre solution est d'utiliser un atelier de conception d'automatismes, permettant à la fois la simulation et l'émulation, et intégrant des outils de traduction automatique de modèles de spécification vers le code automate.

En fonction de la solution retenue, la rupture entre les outils utilisés est positionnée entre le processus de conception d'architecture et le processus de réalisation des composants, ou à l'intérieur même du processus de réalisation des composants.

La vérification par *model-checking* impose d'avoir un modèle états-transitions du code automate associé à un modèle d'exécution de ce code. Les propriétés que les composants doivent vérifier sont modélisées en logique temporelle.

III.5 Vérification unitaire des composants par simulation

Certains ateliers d'automatisation, tels que Controlbuild, offrent la possibilité de faire tourner des simulations de Monte-Carlo sur les composants. Le simulateur doit alors vérifier qu'aucune des séquences tirées aléatoirement ne permet d'enfreindre des conditions propres aux composants. Il en existe deux types, les post condition qui doivent être vraies à la fin de l'exécution du composant et les pré conditions qui doivent être vraies avant l'exécution du composant.

Dans cette optique, les propriétés locales des composants identifiées par l'analyse système peuvent être importées directement dans ces ateliers sous forme de post conditions, permettant ainsi une vérification non formelle (donc moins coûteuse) des composants. Dans ce cas les propriétés ne sont pas modélisées en logique temporelle.

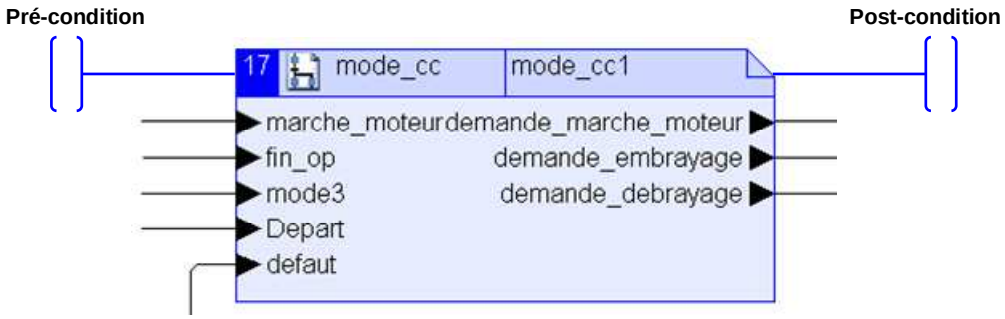


Figure 43 Pré et Post conditions sur ControlBuild

La phase de réalisation des composants logiciels commence donc avec l'importation de l'architecture de composants identifiée par l'analyse système, mais aussi des propriétés locales de composants.

III.6 Vérification unitaire des composants par *model-checking*

La phase de vérification par *model-checking* nécessite quand à elle d'avoir un modèle formel du code automate. Il est donc nécessaire d'importer dans l'outil de *model-checking*, à

la fois les propriétés identifiées par l'analyse système, mais également un modèle d'exécution du code réalisée dans l'atelier d'automatisation.

Les propriétés identifiées et modélisées en SysML doivent donc être formalisées en logique temporelle pour permettre leur vérification par le *model-checker*. Les propriétés de sécurité sont de type « il est toujours vrai que… », leur formalisation en logique temporelle commence donc par l'opérateur AG signifiant « il est toujours vrai que ». On distingue 3 types de propriétés nécessitant des formalisations différentes :

- Les propriétés combinatoires, ou séquentielles avec séquence courte. Ces propriétés peuvent être formalisées avec les opérateurs classiques de logique temporelle. Leur formalisation est de la forme *AG(P)* (il est toujours vrai que P, P étant la propriété de sécurité)
- Les propriétés séquentielles avec séquence longue. Ces propriétés peuvent difficilement être formalisées avec les seuls opérateurs de logique temporelle. Il est donc nécessaire d'introduire un observateur sous la forme d'un automate à état. L'automate décrit la séquence menant à l'état interdit, la propriété associée consiste donc à dire que cet état ne doit jamais être activé *AG(Etat_interdit = 0)*.

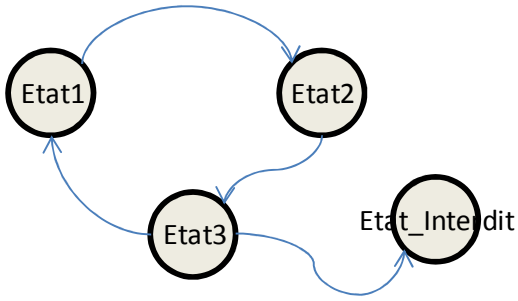


Figure 44 Observateur

- Les propriétés temporelles. Ces propriétés nécessitent la modélisation du temps dans le modèle d'exécution du code. Pour cela on précise une durée de cycle automate dans le modèle d'exécution, la variable temps étant de la durée incrémentée à chaque cycle automate. La formalisation de la propriété s'appuie alors sur cette variable temps *AG(P(temps))*.

IV - Synthèse

IV.1 Processus : vue d'ensemble

Le processus de développement que nous proposons repose donc sur des processus de définition du système et de réalisation (Evrot et al, 2007). Le processus de définition du système est supporté par le langage SysML qui permet l'identification des exigences grâce au diagramme d'exigences, l'identification de l'architecture fonctionnelle grâce au diagramme d'activités et de cas d'utilisation, la définition de l'architecture de composants et des relations entre composants à l'aide des diagrammes de block (*block définition diagram* et *internal block diagram)*.

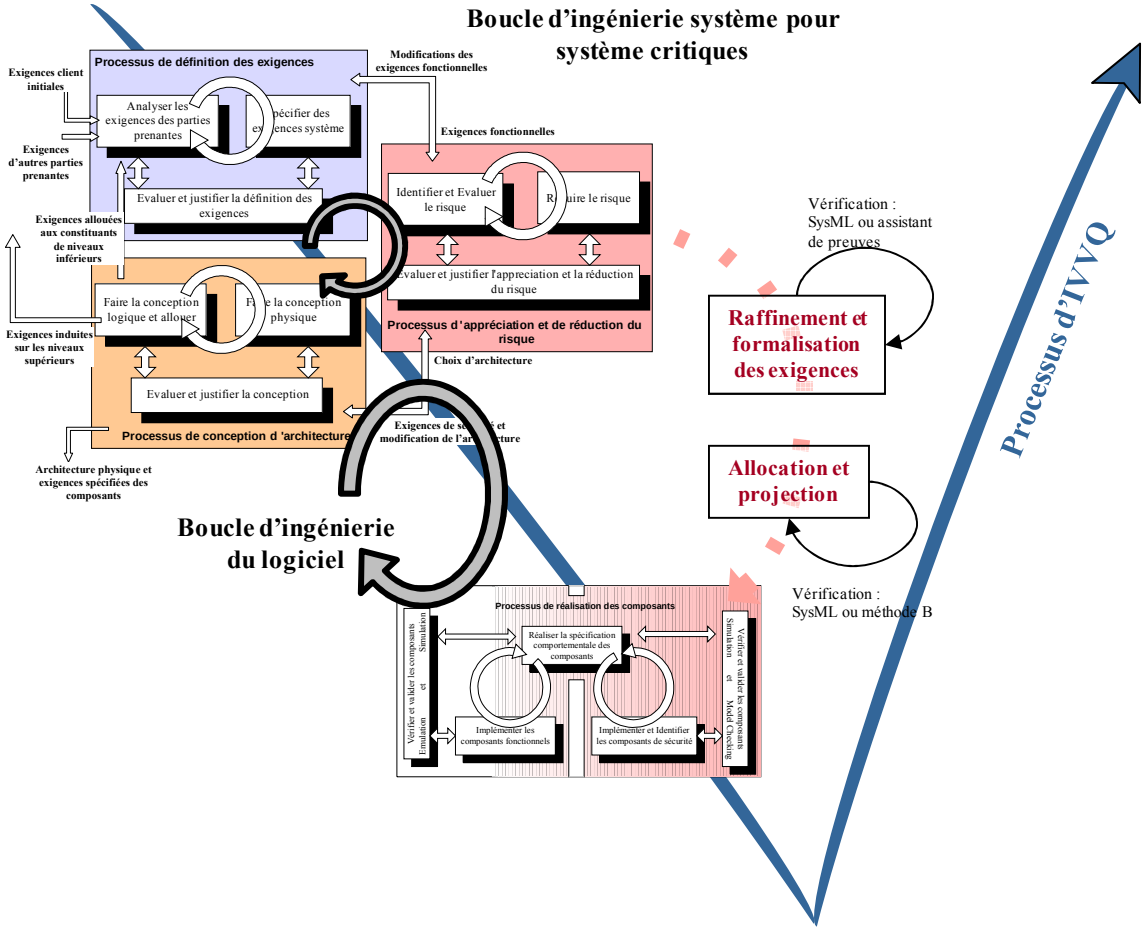


Figure 45 Processus proposé basé sur l'utilisation de SysML et de mécanismes formels.

Le processus de réalisation est supporté par des outils d'automaticiens basés sur une approche composants, et permettant la vérification par simulation. Les composants peuvent également être spécifiés à l'aide des statecharts présents dans UML.

L'idée principale du processus de développement proposé est de se baser sur les capacités de modélisation de SysML pour obtenir une spécification cohérente de toutes les composantes du système : mécanique, électrique, logicielle, matérielle, etc. L'introduction des exigences directement dans les modèles d'architectures fonctionnelles et matérielles permet de relier tout au long du développement la solution spécifiée avec les exigences de départ. Du point de vue de la sécurité du système, cette liaison exigences de sécurité / composants spécifiés, facilite le travail d'identification et de séparation des composants participant à la sécurité du système.

Toutefois la vérification par *model-checking* des composants de sécurité ainsi identifiés requiert d'avoir préalablement identifié et formalisé les propriétés de sécurité que ces composants doivent vérifier. Ces propriétés de sécurité, comme nous l'avons déjà évoqué, doivent représenter le rôle des composants dans la réalisation des exigences système de sécurité fonctionnelle. Or ces exigences sont modifiées, réécrites, détaillées, spécifiées par les différents processus mis en place. Il a donc été nécessaire de mettre en place un certain nombre de mécanismes permettant d'assurer la traçabilité des exigences tout au long de ce cycle de développement.

IV.2 Synthèse des mécanismes

Le problème posé était de prouver le prédicat (3). Le mécanisme de raffinement permet de substituer aux exigences système des exigences plus détaillées.

C'est-à-dire que les exigences raffinées successivement, jusqu'à obtenir finalement les exigences Rf_k, incluent les exigences « système », tout en étant plus concrètes et plus proche des composants du système. On peut alors les allouer à un sous-ensemble de composants suffisant pour les réaliser. On obtient alors pour chacune des exigences Rf_k le prédicat (5) définissant la relation d'allocation. Or si pour tout k il existe un ensemble de composants C_0 à C_j tels que$(\prod_{i=1}^{i=j} C_i \Rightarrow Rf_k)$, alors on peut dire que l'ensemble des composants du système respectent l'ensemble des exigences de bas niveau. $(\prod_{k=1}^{k=n} C_k \Rightarrow \{Rf_k\}, k \in 0..m_f)$ **(7)**

Soit, grâce au raffinement : $(\prod_{i=1}^{i=j} C_i \Rightarrow \mathrm{Rf_k}) \Rightarrow (\prod_{k=1}^{k=n} C_k \Rightarrow \{R_k\}, k \in 0..m\ )$ **(8)**

Finalement l'opération de projection des exigences sur les composants permet de décomposer les exigences $\mathrm{Rf_k}$, en exigences allouées à un et un seul des composants Ci. C'est-à-dire que pour tout i,

$$((\mathrm{C_i} \Rightarrow \{Rf_{k,i}\}) \text{ avec } (\{Rf_{k,i}\} \Leftrightarrow \mathrm{Rf_k})) \Rightarrow (\prod_{i=1}^{i=j} C_i \Rightarrow \mathrm{Rf_k}) \quad \textbf{(9)}$$

C'est-à-dire, en associant (8) et (9), pour tout i,

$$(\mathrm{C_i} \Rightarrow \{Rf_{k,i}\}) \Rightarrow (\prod_{k=1}^{k=n} C_k \Rightarrow \{R_k\}, k \in 0..m) \quad \textbf{(10)}$$

L'ensemble de ces mécanismes nous permet d'obtenir des exigences qui ne doivent être satisfaites que par un et un seul composant. Si toutes ces exigences locales sont satisfaites par les composants auxquels elles sont allouées alors les exigences système sont respectées par le système. Le dernier mécanisme, celui de la formalisation de ces exigences, va nous permettre d'obtenir les propriétés de sécurité locale des composants à partir des exigences raffinées, puis projetées. Il nous permettra également de prouver de manière formelle les prédicats associés aux mécanismes de raffinement et de projection.

Notre problème d'identification et de formalisation des propriétés locales des composants est solutionné par la combinaison des mécanismes de raffinement, allocation, projection et formalisation.

V - Conclusion

Nous proposons une contribution méthodologique à la vérification d'exigences de sécurité basée sur des mécanismes de raffinement, de projection et d'allocation des exigences sur une architecture de composants matériels et/ou logiciels. Ces modèles, exprimés dans le formalisme de SysML offrent une grande cohérence entre les différentes vues du système (exigence, architecture, de comportement) et une description intuitive et pragmatique du système.

La formalisation des exigences par des propriétés invariantes exprimées sous forme d'axiomes permet d'encadrer la construction des modèles SysML par des mécanismes de raffinement, de projection et d'allocation qu'il est possible de vérifier formellement. Ces mécanismes permettent d'identifier de manière incrémentale les propriétés locales que doivent vérifier les composants du système pour que le système dans son ensemble vérifie les exigences système. La vérification unitaire des composants par *model-checking* apporte donc la preuve que les exigences système sont satisfaites.

Le paragraphe suivant présente la mise en œuvre de cette approche sur un cas d'étude.

Chapitre 4 :

Application sur un cas

I - Introduction

Ce chapitre illustre le processus proposé au chapitre précédent sur un cas d'étude fourni par l'INRS relatif à l'automatisation en logique programmée des fonctions de sécurité d'une presse mécanique. Afin de faciliter la prise en compte des propriétés de sécurité identifiées lors de l'analyse SysML par les outils de simulation et de *model-checking*, nous avons développé un démonstrateur intégrant, autour d'une structure de données XML, les outils MagicDraw (SysML), ControlBuild (Simulation de Systèmes à Evénements Discrets) et UPPAAL (*model-checking*).

II - Démonstrateur

II.1 Outils utilisés

Le démonstrateur que nous avons mis en place a pour objectif de supporter le processus de développement proposé au chapitre précédent. Il intègre 4 outils :

- Un outil de modélisation en SysML. De plus en plus d'éditeurs informatiques développent ce type d'outils, ainsi les suites logicielles telles que Rational d'IBM, Artisan Studio d'Artisan Software, ou même Rhapsody d'I-Logix proposent des profils SysML. Pour notre part, nous avons utilisé le logiciel Magicdraw UML, de la société NoMagic, qui intègre également un « plug-in » SysML.
- Un outil de conception d'automatisme. Pour notre processus, il nous fallait un outil proposant à la fois une conception orientée composant, et la possibilité d'effectuer des tests par simulation de partie opérative. Nous avons opté pour l'outil ControlBuild de la société Geensys.
- Un outil de vérification formelle par *Model-checking*. L'outil que nous avons choisi est UPPAAL, qui nous permet de modéliser facilement le comportement d'un automate, grâce à son langage à état, et qui permet également de vérifier des propriétés temporelles.
- Un outil de vérification formelle par *theorem-proving*. Nous avons choisi l'outil COQ présenté au chapitre précédent.

Ces outils présentent tous la caractéristique d'exporter leurs données au format XML. Nous avons donc choisi de concevoir un démonstrateur basé sur la transformation de structures XML pour permettre l'interopérabilité entre les 4 logiciels proposés avec comme contrainte, de faciliter l'intégration dans le futur d'autres applications.

II.2 Principe

Pour réaliser ce démonstrateur, nous avons opté pour une solution en étoile qui repose sur une structure XML centrale, qui stockera toutes les informations développées par les 4

logiciels. Le passage d'une structure propre à un logiciel vers le la structure XML centrale est réalisée par des transformations XSLT.

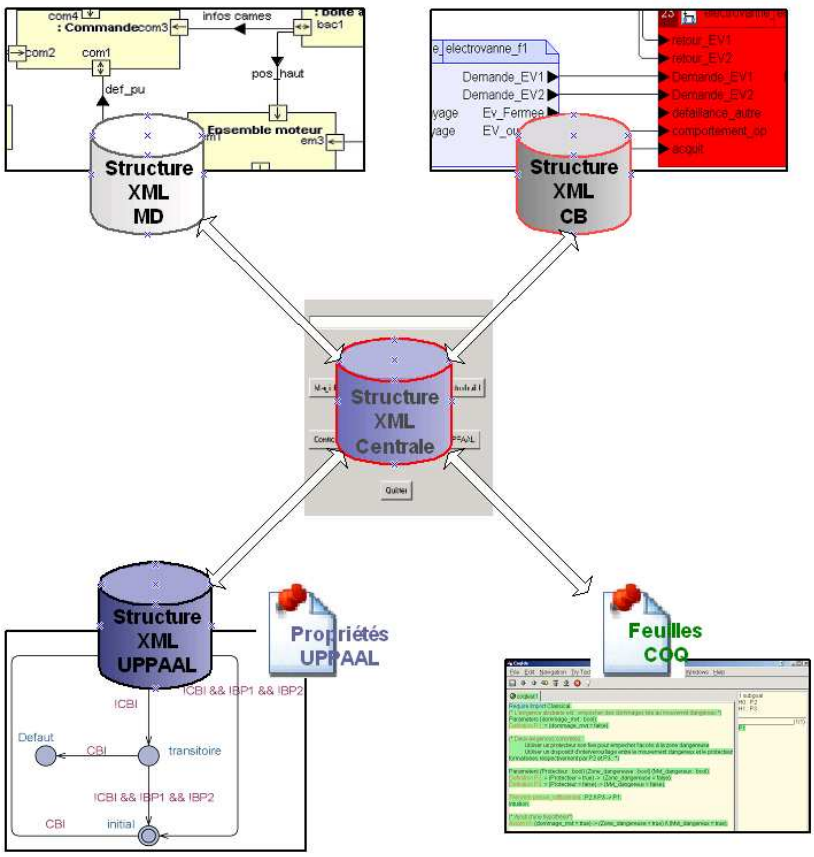


Figure 46 Structure du démonstrateur.

La structure en étoile a l'avantage de faciliter l'intégration de nouveaux logiciels, ainsi que le remplacement d'un logiciel existant. Il suffit en effet de concevoir deux nouvelles opérations de transformation pour pouvoir intégrer une nouvelle application dans le démonstrateur. Ces nouvelles transformations ne modifient en rien la structure et les transformations existantes.

II.3 Réalisation

La structure XML centrale est basée sur le modèle de données défini au chapitre précédent, qui reprend les interactions entre les différents objets du système. Cette structure est basée sur une vue composant du système. Il s'agit donc, composant par composant de définir les variables utilisées, les fonctions réalisées, les exigences satisfaites, le code implémenté etc.

Pour concevoir cette structure, nous nous sommes appuyés à la fois sur la structure définie par la norme PLCOpen et à la fois sur la structure XMI définie par l'OMG. La structure proposée est définie Figure 47. Cette structure est basée sur une vue composant.

Un composant est défini comme un objet qui possède des variables, des opérations, des exigences satisfaites, du comportement et des propriétés à vérifier. Le comportement du composant peut être défini directement par du code, si celui ci est de bas niveau, ou par des instances de composants et des relations entre ces instances.

Les fonctions réalisées sont définies par un nom, une ou plusieurs exigences satisfaites et les opérations appelées. Les exigences sont définies par leurs attributs SysML. La balise « raffine » permet de définir les raffinements successifs des exigences.

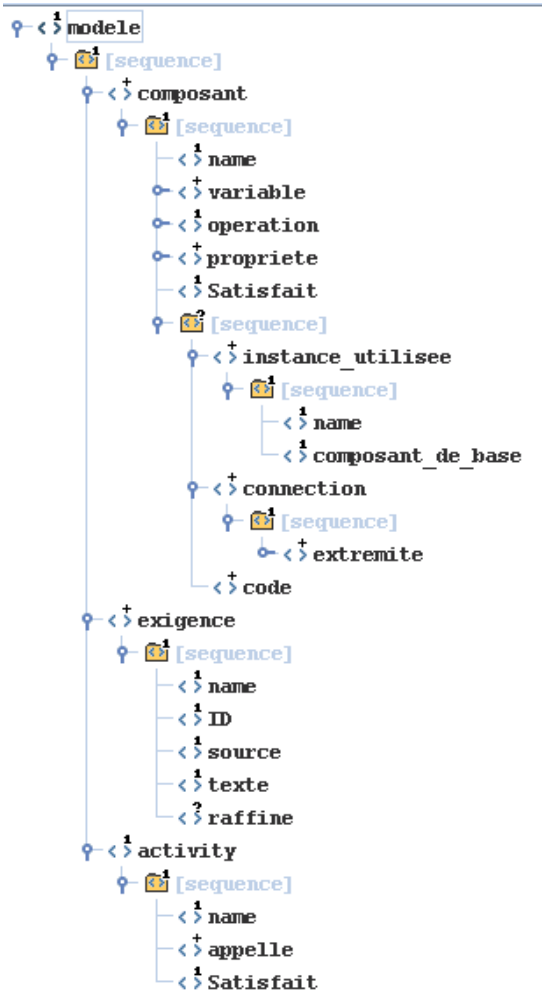


Figure 47 Définition de la structure de données

Une fois que cette structure est définie, il ne reste plus qu'à écrire les règles de transformation entre les structures XML, dans le langage XSLT. La figure suivante donne la nature des informations échangées entre les structures.

III - Application : processus de définition du système

Ce paragraphe a pour but d'illustrer les différentes étapes mises en œuvre pour développer un logiciel applicatif pour une presse mécanique. Il ne décrit pas de manière exhaustive le processus, mais en donne les grandes étapes et les résultats.

III.1 Première itération : Définition du globale du système

III.1.1 Processus de définition des exigences

Le premier modèle doit formaliser les besoins du client par des exigences système. Pour la presse mécanique, le client désire une presse capable de s'intégrer dans son atelier, et d'emboutir des pièces d'une certaine dimension. Les exigences du client sont représentées sur la Figure 48.

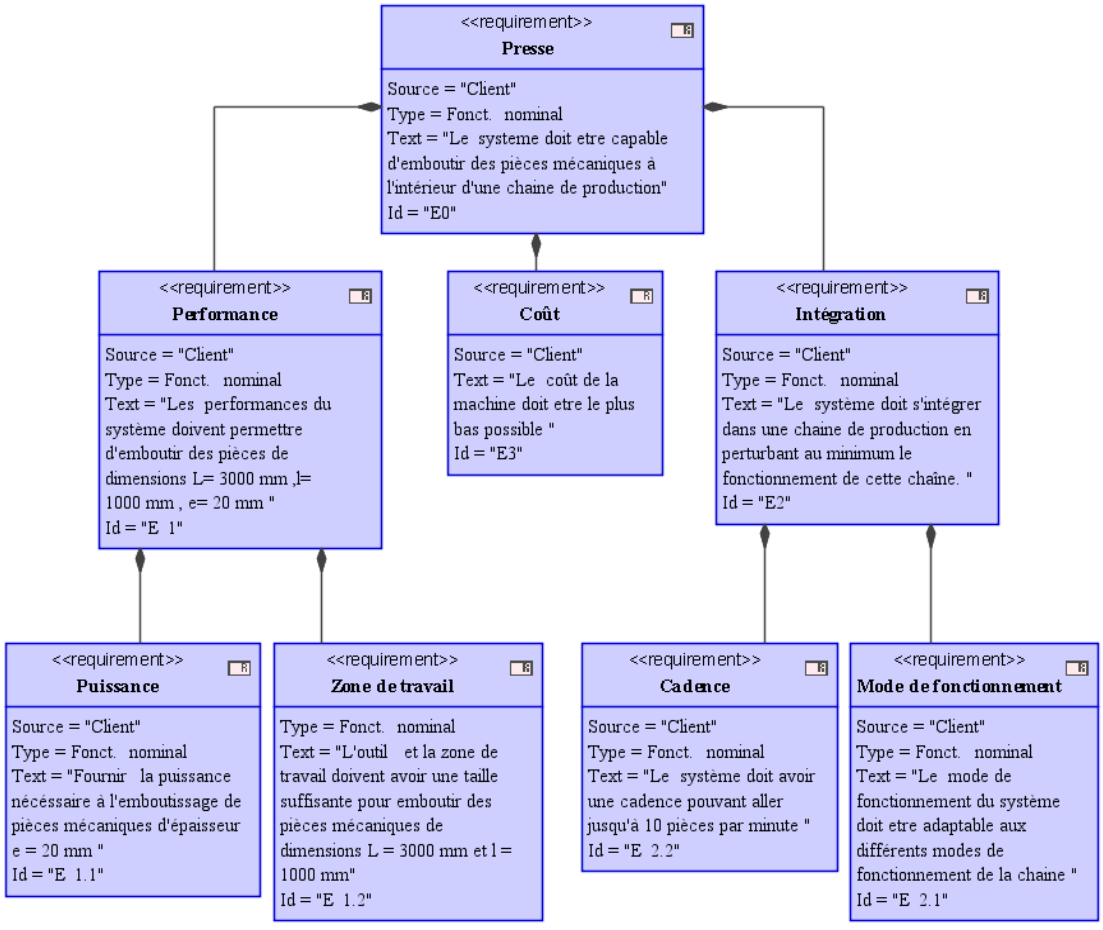


Figure 48 Exigences client

On peut voir que les premières exigences sont définies déjà sur 3 niveaux de raffinement mais restent très abstraites. Cela signifie que l'on a déjà réalisé, à l'intérieur du processus de définition des exigences, trois boucles d'analyse et de spécification des exigences.

Ainsi l'exigence « performance » indiquant les dimensions des pièces que la presse doit être capable d'emboutir est raffinée en une exigence de puissance précisant la puissance d'emboutissage de la presse et une exigence de zone de travail, précisant les dimensions de l'espace de travail. De même l'exigence « intégration » est raffinée par l'exigence « mode de fonctionnement » et l'exigence « cadence ».

On peut également noter l'attribut « type », qui permet de typer les exigences. Ainsi on pourra différencier dans le modèle les exigences de fonctionnement nominal (type fonct. nominal), les exigences de sécurité fonctionnelle (type secu. fonctionnelle), etc.

III.1.2 Processus de conception

La définition des exigences est suivie par un processus de conception qui va nous permettre de définir les premiers composants et fonctions du système. Une première analyse fonctionnelle nous a permis d'identifier deux fonctions. La première regroupera les opérations de transformation de mouvement et d'énergie permettant de créer le mouvement d'emboutissage et la deuxième regroupera toutes les fonctions permettant de commander ce mouvement. Les fonctions « Réaliser l'emboutissage » et « Commander l'emboutissage » sont allouées au système presse (Figure 49).

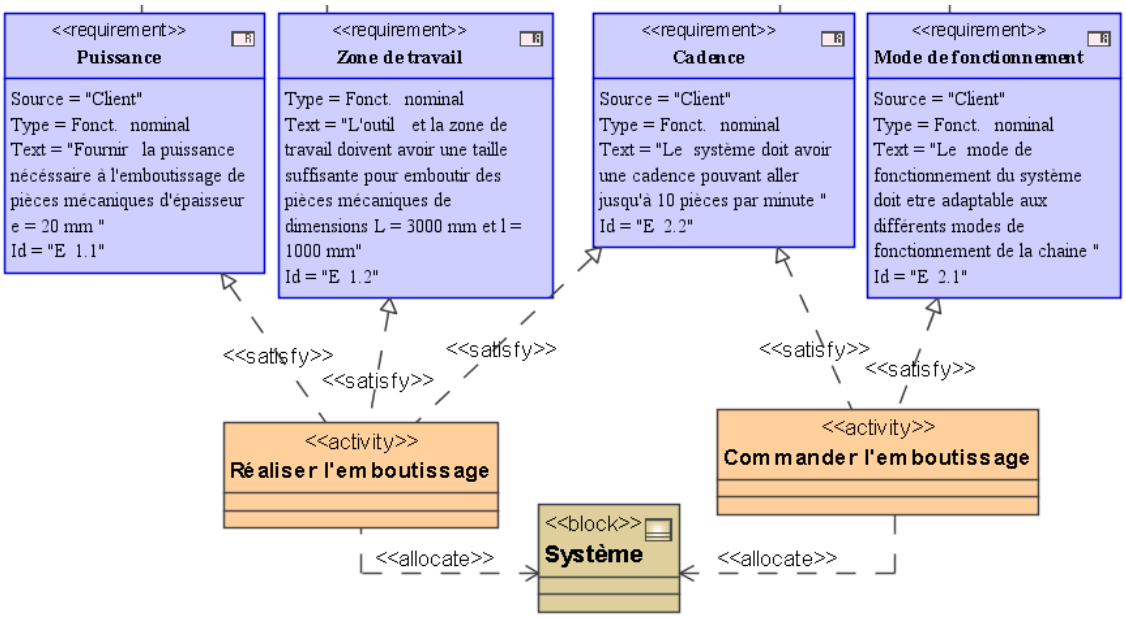


Figure 49 Allocation des exigences

Cette première boucle d'ingénierie système a permis jusqu'à maintenant de poser les bases du référentiel d'exigences, de l'architecture fonctionnelle et de l'architecture de composants du système du point de vue des besoins du client. Avant de conclure cette première boucle d'ingénierie système, nous allons intégrer au référentiel d'exigences les premières exigences système de sécurité fonctionnelle.

III.1.3 Processus d'appréciation et réduction du risque

Pour assurer la sécurité de l'opérateur, il faut dans un premier temps identifier les phénomènes dangereux susceptibles de le blesser. La liste des phénomènes dangereux définis dans la norme ISO 12100 est longue, en voici une partie :

- Phénomènes dangereux mécaniques : écrasement, cisaillement, coupure ou sectionnement, happement, enroulement, entraînement ou emprisonnement, choc, perforation ou piqûre, frottement ou abrasion, injection de fluide sous haute pression (phénomène dangereux d'éjection).
- Phénomènes dangereux électriques …
- Phénomènes dangereux thermiques etc.

Dans notre cas, au vu de la machine développée, les principaux phénomènes dangereux existants sont d'une part l'écrasement dû au mouvement d'emboutissage et d'autre part le risque d'éjection de la pièce. Les exigences ES1 et ES2 visent à réduire ces risques. Pour la clarté de l'étude, nous ne développerons que l'exigence ES1.

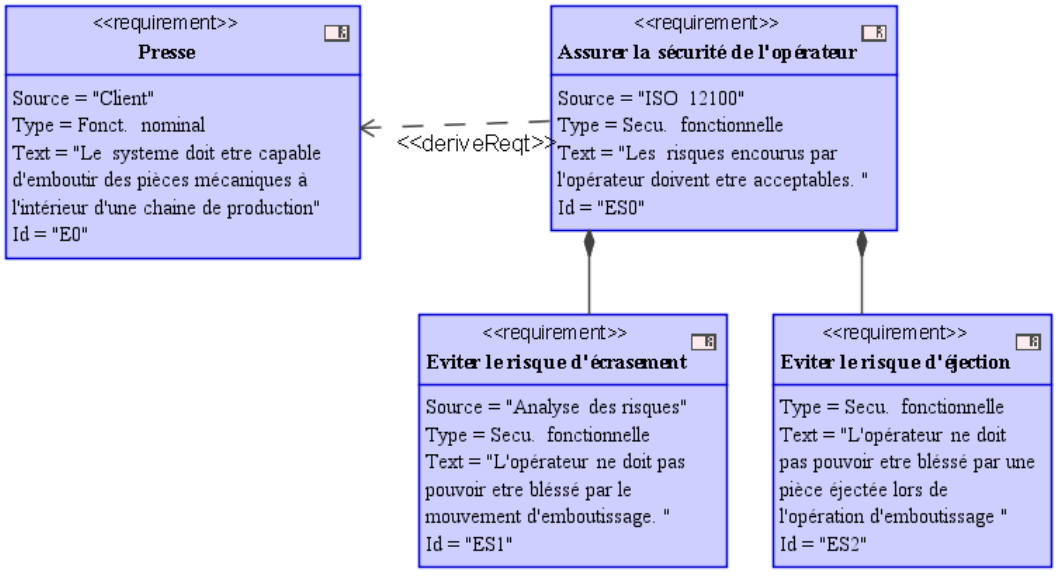


Figure 50 Exigences système de sécurité fonctionnelle

Ces exigences, associées aux exigences de la Figure 48 forment un référentiel complet des exigences système formalisant les besoins des différentes parties prenantes (client, normes). Ce référentiel défini l'attendu du système, c'est par rapport à lui qu'il sera nécessaire de valider le système final. Les prochaines étapes vont permettre de raffiner ces exigences et de concevoir des solutions permettant de les réaliser.

III.2 Deuxième et troisième itérations : modes de fonctionnement

Afin de raffiner ces premiers modèles, il est nécessaire de mettre en œuvre un certain nombre de passages dans la boucle d'ingénierie système que nous avons définie.

III.2.1 Processus d'identification des exigences et processus de conception

Le processus d'identification des exigences s'est intéressé aux modes de fonctionnement. Nous avons d'abord raffiné l'exigence « mode de fonctionnement », en précisant que la presse, pour pouvoir s'adapter au fonctionnement de l'atelier, doit fonctionner en mode coup par coup et en mode continu, et qu'elle doit pouvoir être réglée afin de s'adapter aux différentes productions.

A partir de cet ensemble d'exigences, le processus de conception a permis de créer un composant de gestion des modes et un composant « alimentation » pour gérer les modes d'alimentation automatique et manuelle de la presse. Une architecture physique de type mécanique a également été choisie pour la partie opérative de la presse. L'analyse des exigences de puissance, de cadence, et de coûts, a montré qu'une presse mécanique avec un embrayage-frein, un moteur et un système bielle-vilebrequin était une solution suffisante pour répondre à ces exigences.

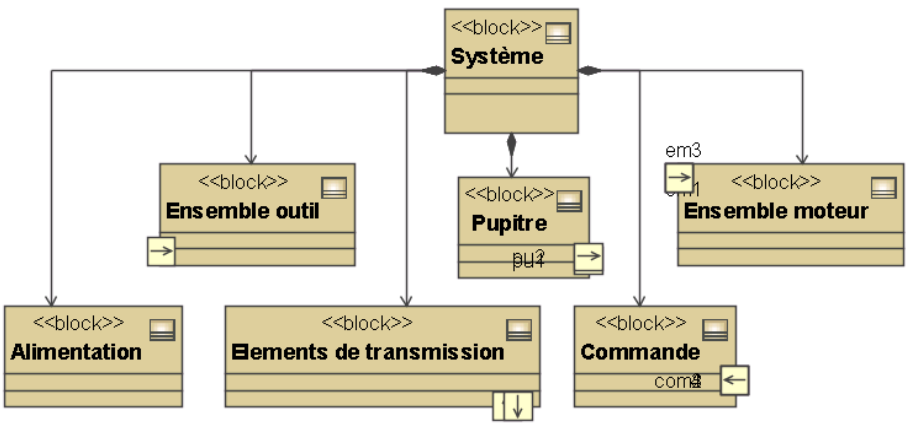


Figure 51 Architecture de composant de la presse mécanique

Un nouveau passage dans la boucle d'ingénierie système a permis au final d'identifier toutes les exigences relatives à la gestion des modes, notamment l'exigence « changement de mode », de créer une architecture fonctionnelle répondant à ces exigences, et une architecture de composants, basée sur l'architecture physique d'une presse mécanique, réalisant l'architecture fonctionnelle. La Figure 52 est le résultat des deux boucles d'ingénierie système focalisées sur les modes de fonctionnement.

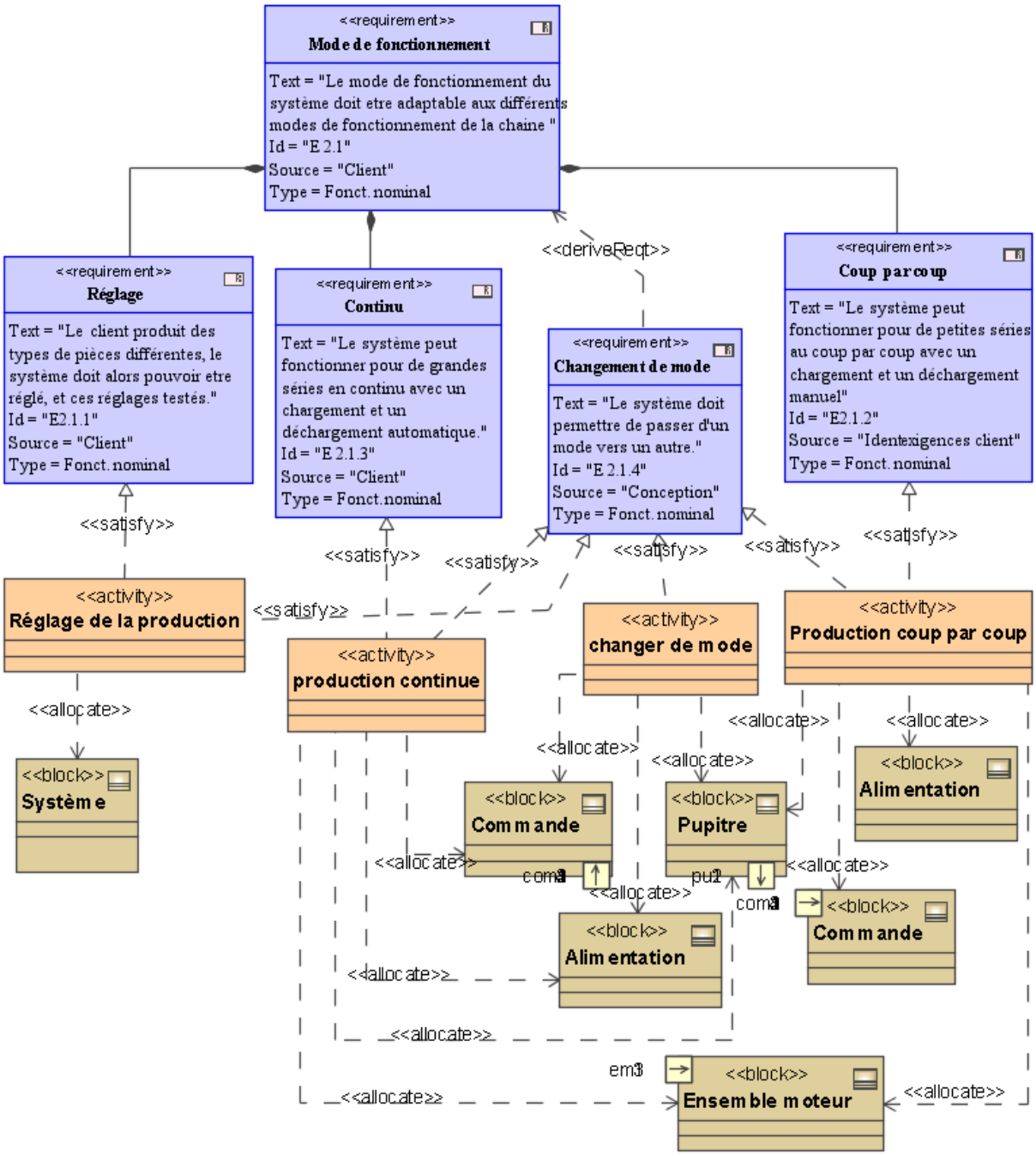


Figure 52 Réalisation des exigences relatives aux modes de fonctionnement

III.2.2 Premières conclusions

Nous avons, pour l'instant, mis en œuvre trois passages dans la boucle d'ingénierie système. Le premier a permis de construire un référentiel d'exigences système formalisant les besoins des parties prenantes (besoins fonctionnels et réduction du risque). Les deuxième et troisième passages se sont concentrés sur la définition des modes de fonctionnement en identifiant un certain nombre d'exigences relatives aux modes de fonctionnement de la presse, et en construisant une architecture fonctionnelle répondant à ces exigences et une architecture de composants réalisant ces fonctions.

L'approche par processus a permis de construire de manière incrémentale les modèles d'exigences, de fonctions et de composants. Cela permet d'appréhender le système morceau par morceau, facilitant ainsi la construction de ces modèles. On a pu ainsi construire un premier diagramme de composant (internal block diagram) du système.

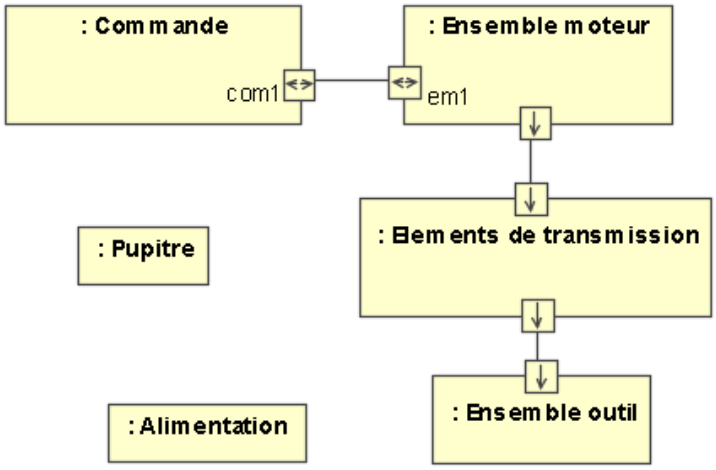


Figure 53 Diagramme de composants du système

Ce diagramme nous montre les interactions existantes entre les différents composants du système. Ainsi on voit que la commande devra commander l'ensemble moteur qui fournit un mouvement de rotation aux éléments de transmission qui eux même fournissent un mouvement de translation à l'outil.

Le formalisme SysML nous permet donc de développer effectivement toutes les vues nécessaires au développement d'un système, toutefois les modèles développés n'ont pas été validés, les liens créés entre les différents objets des modèles n'existent que de manière informelle. Cette validation doit être apportée par les mécanismes définis au chapitre précédent.

Dans la suite du paragraphe, nous allons nous focaliser sur le développement du mode coup par coup, et plus précisément sur la mise en œuvre des mécanismes de raffinement, formalisation, projection et allocation.

III.3 Quatrième itération : protection du mode coup par coup

Le développement du mode de fonctionnement coup par coup est réalisé par une boucle d'ingénierie système. Cette boucle est décrite dans les paragraphes suivants.

III.3.1 Processus de définition des exigences en fonctionnement nominal

Pour identifier les exigences de fonctionnement nominal du mode coup par coup, le processus de définition des exigences doit se baser d'une part sur le besoin exprimé par le client pour le fonctionnement du mode continu, et d'autre part sur l'architecture physique retenue. Ainsi nous avons identifié 3 exigences concernant : le démarrage du mouvement, l'arrêt du mouvement, le chargement des pièces. Ces exigences définissent le fonctionnement du mode coup par coup (Figure 54).

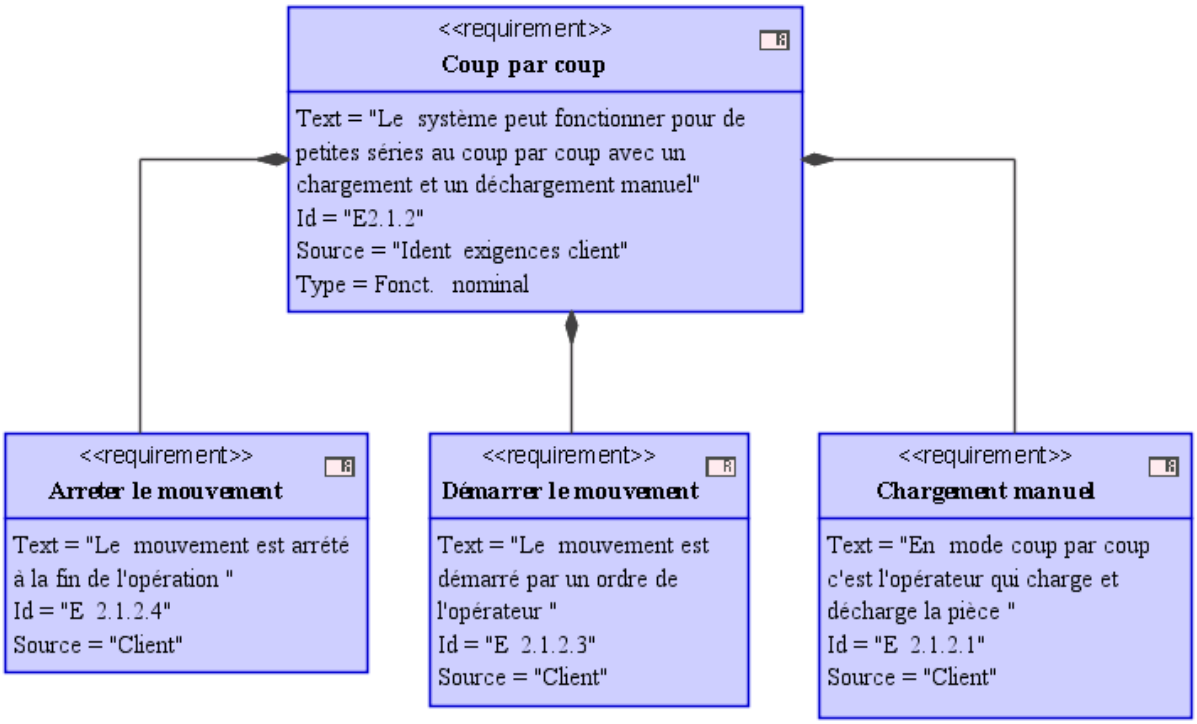


Figure 54 Exigence de fonctionnement nominal du mode coup par coup

A partir de cette définition des exigences fonctionnelles, il est possible d'initier le processus d'analyse et de réduction du risque.

III.3.2 Processus d'appréciation et de réduction du risque

A partir du phénomène dangereux identifié par l'exigence de sécurité fonctionnelle ES1 (Ecrasement), nous allons dérouler un processus d'analyse et de réduction du risque en tenant compte de l'architecture physique retenue. Pour cela il nous faut tenir compte des différentes situations possibles :

- La machine accomplit la fonction prévue (fonctionnement normal);
- La machine n'accomplit pas la fonction prévue (dysfonctionnement)
- Comportement involontaire de l'opérateur ou mauvais usage raisonnablement prévisible de la machine.

Une analyse de risque (AMDE et Arbre des Fautes) montre que sur une presse mécanique le risque d'écrasement n'existe que si l'outil est en mouvement ou s'il y a une défaillance sur un des composants assurant le maintien à l'arrêt du coulisseau.

Une fois le risque défini, il faut chercher à réduire ce risque. La norme ISO 12100-2 nous offre pour cela 3 possibilités : tout d'abord des mesures de prévention intrinsèques, ensuite des mesures de protection permettant d'isoler le risque, ou en dernier recours des informations pour utilisation. Aucune mesure de prévention intrinsèque ne pouvant être mise en place (l'utilisation d'outils en mousse élimine effectivement le risque d'écrasement, mais reste assez limitée d'un point de vue fonctionnel), nous optons pour des mesures de protection.

L'analyse de risque nous dit que les mesures de protection doivent avoir pour but d'empêcher l'opérateur de pénétrer la zone dangereuse durant le mouvement d'emboutissage. On fait donc le choix d'utiliser des dispositifs de protection pour limiter l'accès à la zone dangereuse du système. On se réfère alors à l'article 5 de la norme ISO 12100-2. Elle donne l'algorithme suivant pour le choix des protecteurs. Le phénomène dangereux identifié est engendré par des éléments mobiles qui contribuent au travail. Le besoin d'accessibilité à ces éléments pendant le travail dépend du type d'accès :

- Accès frontal : L'opérateur doit avoir accès à la zone dangereuse par une ouverture frontale pour changer la pièce pendant le mouvement de remontée. Il faut donc choisir des protecteurs parmi les suivants : protecteur avec dispositif de verrouillage ou d'inter verrouillage, commande bimanuelle, etc.
- Accès latéraux : pas nécessaires durant le mouvement de la presse, il faut donc utiliser des protecteurs fixes, mobiles ou réglables.

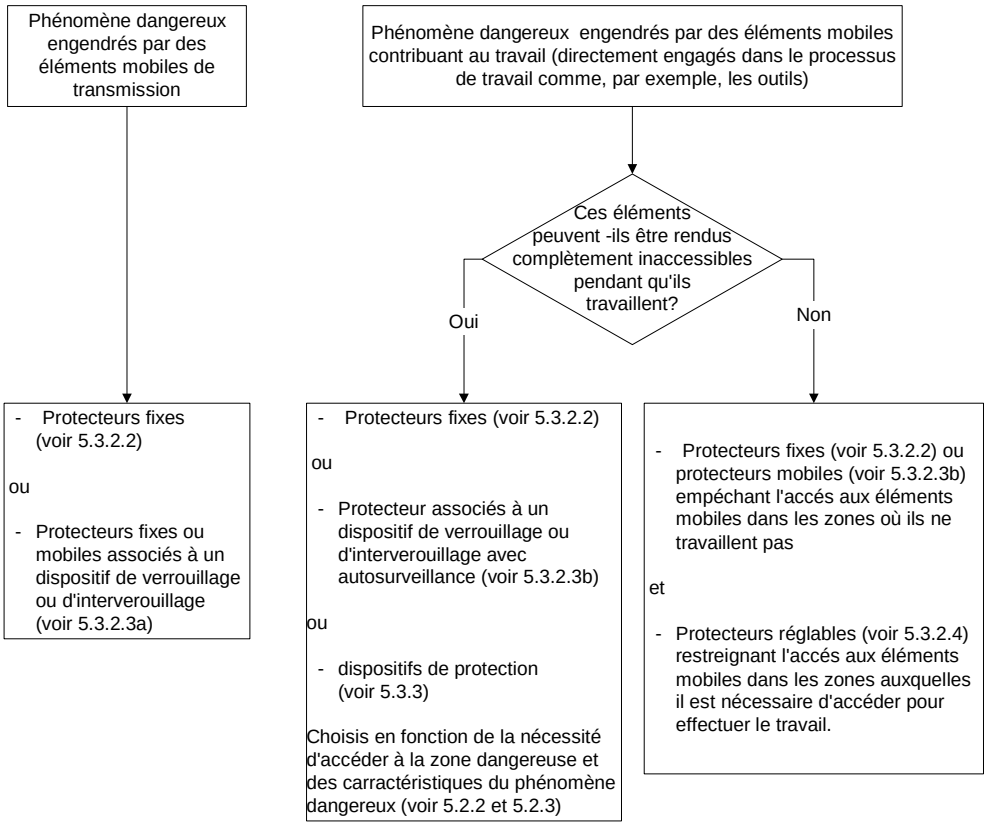


Figure 55 Algorithme de choix des dispositifs de protection (norme ISO 12100-2)

Finalement, nous avons opté pour une commande bimanuelle pour empêcher l'opérateur d'accéder à la zone dangereuse via l'accès frontal et des protecteurs mobiles pour empêcher l'opérateur d'accéder à la zone dangereuse via les accès latéraux (Figure 56). L'analyse de risque estime que l'accès à la zone dangereuse par l'accès frontal durant la remontée de l'outil ne comporte pas de risque d'écrasement.

III.3.3 Vérification des liens de raffinement créés

Si on s'intéresse aux règles de construction définies pour le mécanisme de raffinement, on s'aperçoit que dans ce cas particulier, elles sont toutes respectées. Les exigences sont toutes de type sécurité fonctionnelle, il y a un nombre suffisant d'exigences, et on ne constate pas de boucles dans les liens de raffinement.

Pour pouvoir vérifier de manière formelle les liens de raffinement, il faut commencer par formaliser les exigences en propriétés. Les exigences ES1, ES3, ES4, ES5 sont formalisées respectivement par les propriétés P1, P3, P4, P5.

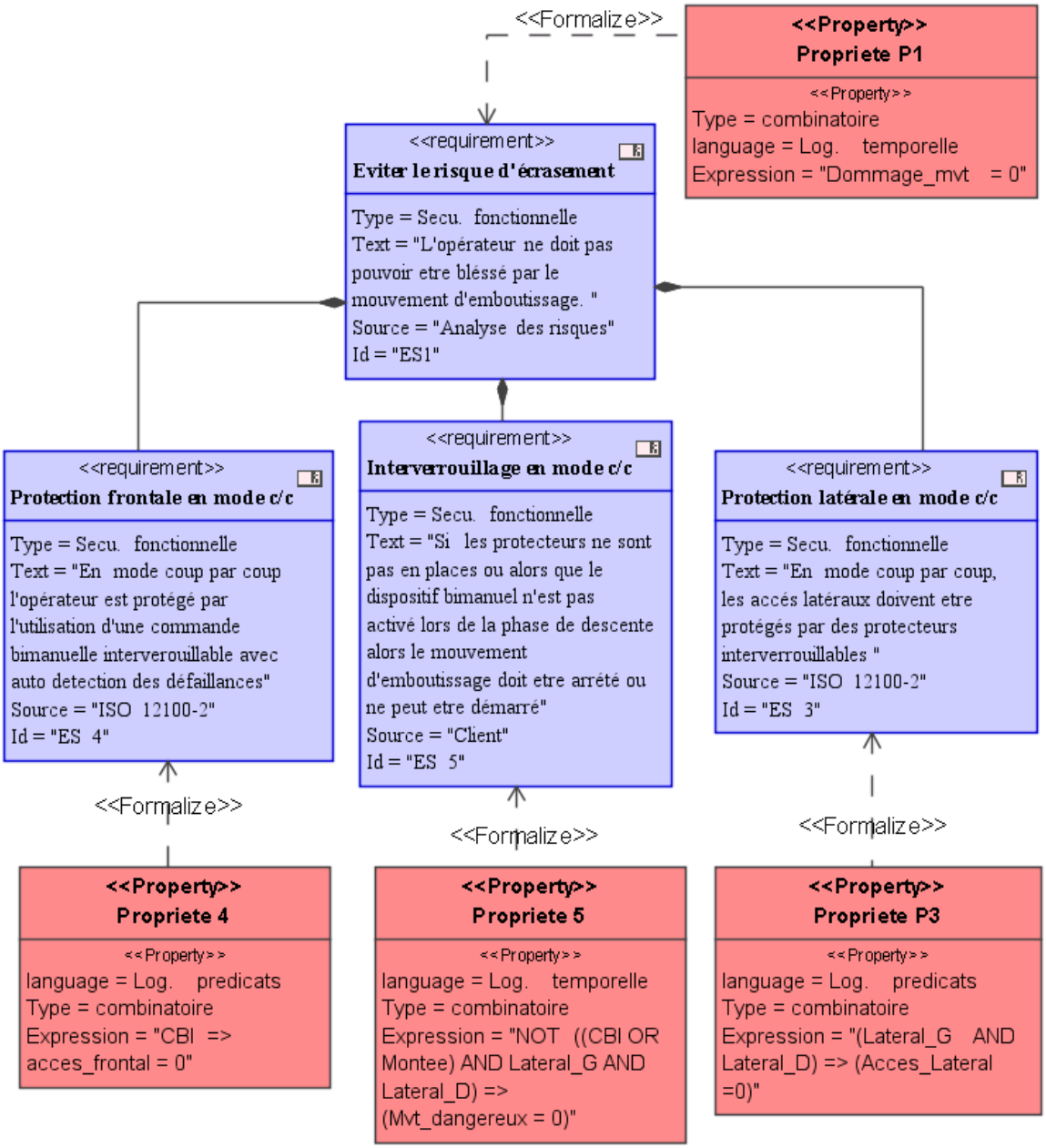


Figure 56 Choix de protecteurs en mode coup par coup et formalisation des exigences

Afin de pouvoir prouver de manière formelle que (P3 $\wedge$ P4 $\wedge$ P5 $\Rightarrow$ P1), il est nécessaire d'ajouter dans l'assistant de preuves les hypothèses issues de l'analyse des risques (Figure 57):

H1 : Dommage_mvt $\Rightarrow$ Acces_Lateral $\wedge$ Mvt_dangereux $\vee$ Acces_Frontal $\wedge$ Mvt_dangereux $\wedge$ $\neg$ Montee.

Une fois cette preuve réalisée en COQ, nous pouvons passer au processus de conception.

```
(*Preuve de Raffinement*)

Parameter (Lateral_G : bool) (Lateral_D : bool) ( CBI : bool)(Zone_dangereuse : bool).
Parameter (dommage_mvt : bool)(Mvt_dangereux : bool)(Montee : bool)(Acces_Lateral : bool) (Acces_frontal : bool).

Theorem Raffinement :

                              (*Propriété 4, protection frontale en mode cc*)
((CBI = true -> Acces_frontal = false)

                              (*Propriété 3, protection latérale en mode cc*)
/\((Lateral_G = true /\ Lateral_D = true) -> (Acces_Lateral = false))

                              (*Propriété 5, Interverrouillage en mode cc*)
/\(Mvt_dangereux = true-> (CBI = true \/ Montee = true) /\ (Lateral_G = true /\ Lateral_D = true) )

                              (*Hypothèse sur dommage mouvement*)
/\(dommage_mvt = true -> ( (Acces_Lateral = true /\ Mvt_dangereux = true)
                              \/ (Acces_frontal = true /\ Montee = false /\ Mvt_dangereux = true))) (**)

                              (*Propriété 1, Eviter le risque d'écrasement*)
->(dommage_mvt = false)).
```

Figure 57 Preuve de raffinement en COQ

III.3.4 Processus de conception du mode coup par coup

Lors du précédent processus de conception, l'exigence principale du mode coup par coup a été allouée indirectement aux composants « commande », « alimentation », « ensemble moteur » et « pupitre ». Le résultat du processus de conception devra donc allouer les exigences du mode coup par coup à ces composants, ou leurs sous-composants.

Tout d'abord, l'exigence « chargement manuel », est satisfaite par la fonction « charger manuellement », qui est une sous fonction de la fonction « production coup par coup ». Cette fonction est allouée au bloc fonctionnel «Alimentation ». Comme il s'agit d'une fonction manuelle, ce composant sera utilisé par l'opérateur. A ce stade du développement la nature de ce composant n'est pas définie.

Ensuite les exigences « démarrer le mouvement » et « arrêter le mouvement » sont satisfaites par la fonction mode coup par coup qui est également une sous fonction de la fonction « production coup par coup ». Cette fonction est réalisée par les composants « ensemble moteur » et « pupitre », qui n'ont pas évolués, et par le composant « Mode c/c » qui est un sous composant du composant « commande ». Derrière ce composant, on retrouvera l'ensemble des composants logiciels de la commande participant à la gestion fonctionnement du mode coup par coup. On peut noter qu'étant donné que l'exigence « démarrer le mouvement » fait référence à un ordre de l'opérateur, un lien d'utilisation a été tracé entre le pupitre et l'opérateur.

La même démarche a été appliquée aux exigences de sécurité fonctionnelle, pour obtenir le résultat décrit Figure 58. La prochaine étape du développement va consister à raffiner le modèle de la commande bimanuelle.

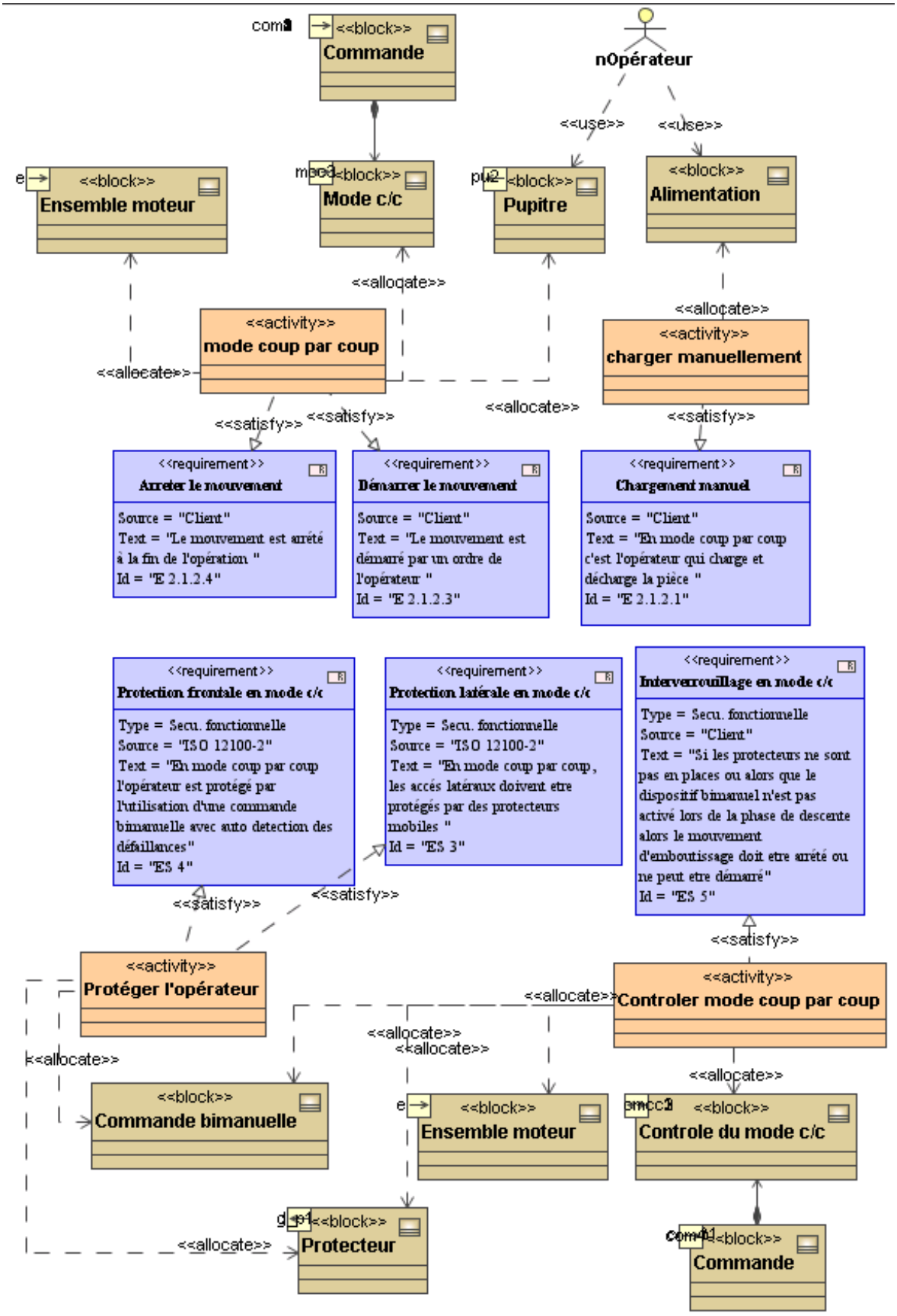


Figure 58 Résultat du processus de conception du mode coup par coup

III.4 Cinquième itération : la commande bimanuelle

Nous allons maintenant détailler une nouvelle boucle d'ingénierie système visant spécifiquement le développement de la commande bimanuelle.

Pour répondre à l'exigence de sécurité ES4 portant sur la commande bimanuelle, d'après les normes dédiées aux commandes bimanuelles, certains points doivent être précisés :

- l'ergonomie de la commande bimanuelle, pour assurer que l'opérateur soit obligé d'utiliser ses deux mains pour actionner les deux boutons,
- la détection des défauts,
- l'activation du signal CBI,
- la désactivation du signal CBI.

Les exigences ES4. (1 à 5) sont le résultat du mécanisme de raffinement (Figure 59). Elles doivent être toutes satisfaites par le composant « commande bimanuelle » étant donné qu'il doit dors et déjà satisfaire l'exigence ES 4.

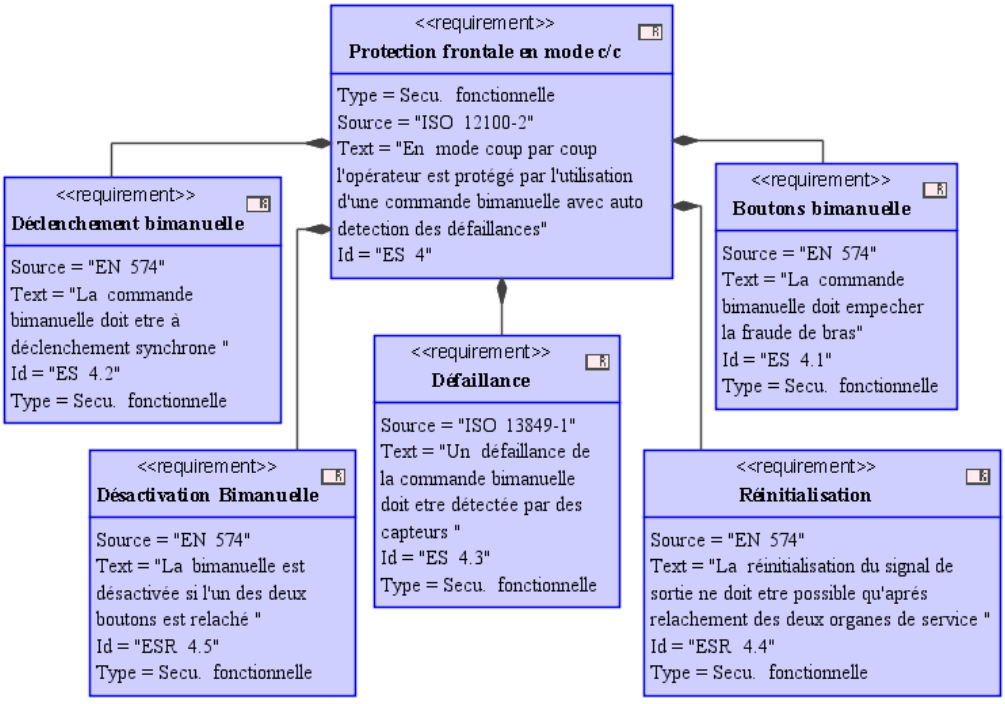


Figure 59 SysML : Raffinement des exigences

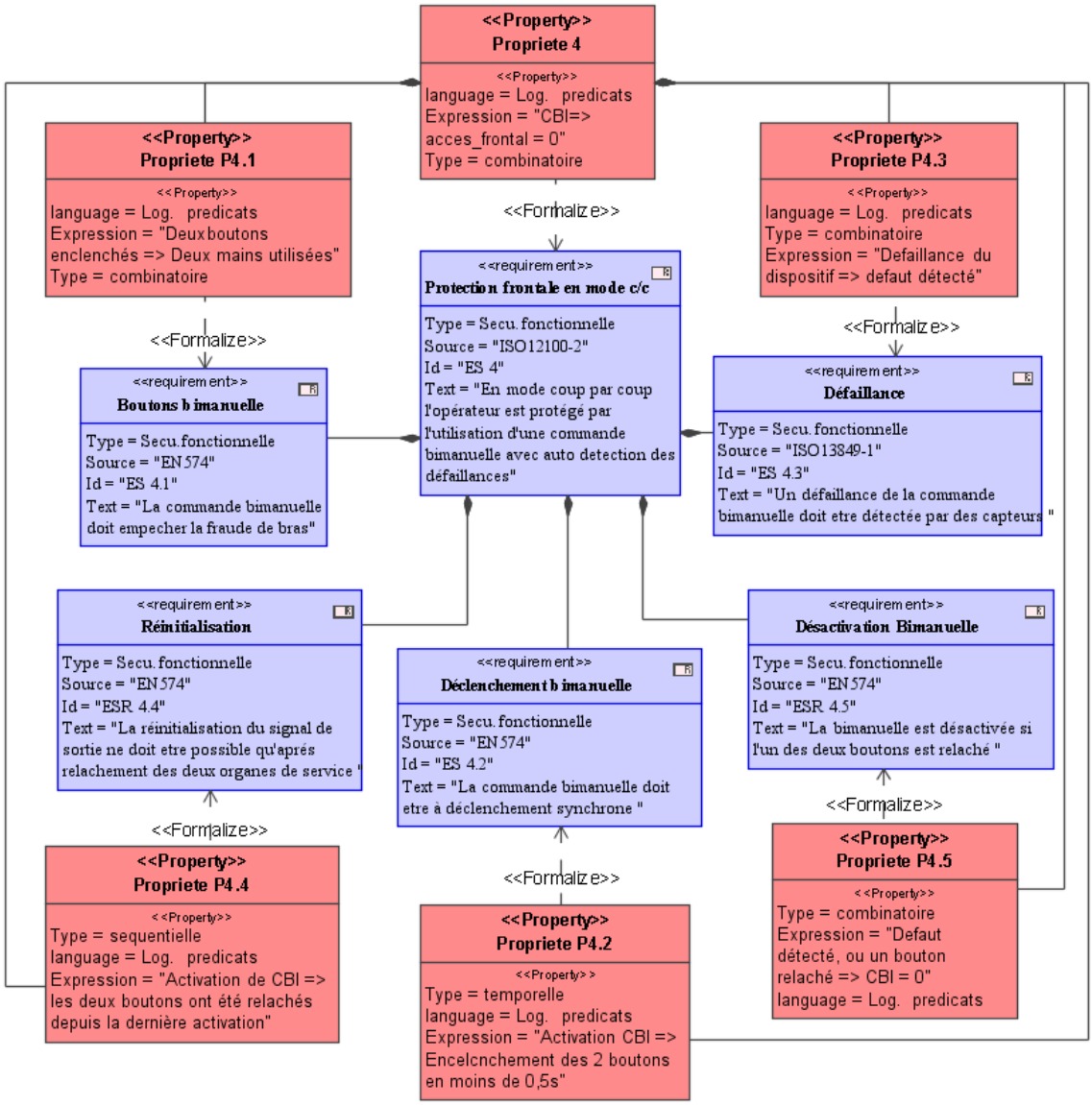


Figure 60 Formalisation des exigences de sécurité.

III.4.1 Processus de conception : dispositif bimanuel

Nous avons établi les exigences auxquelles doit répondre le composant « commande bimanuelle ». Reste maintenant à définir la structure de ce dernier. Pour cela nous allons mettre en place un processus de conception (conception logique et conception physique). Les parties matérielles des commandes bimanuelles sont réalisées par des équipements de sécurité standards. De la même façon il existe des blocs logiciels pré-certifiés qui permettent de réaliser la partie commande de ces équipements. Nous nous plaçons dans le cas d'un intégrateur qui achète la partie matérielle pour l'intégrer sur la machine et qui développe lui-même la partie logicielle.

La première décomposition est donc entre partie matérielle et partie logicielle. Il faut noter que nous entrons dans le processus de conception en réalisant dans un premier temps une conception matérielle. C'est souvent le cas pour des composants standard. Dans ces cas là il s'agit uniquement de choisir le composant standard qui répond le mieux aux exigences spécifiées.

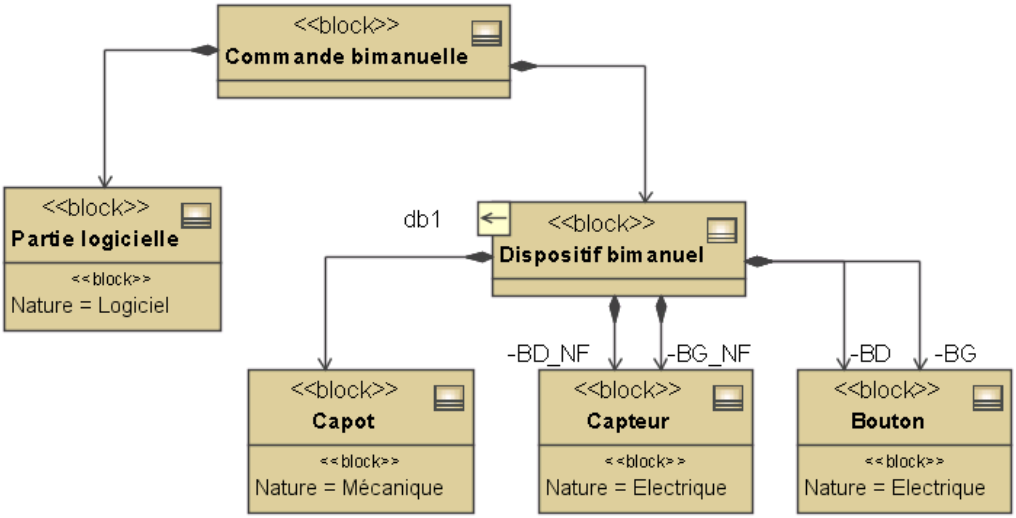


Figure 61 Décomposition du dispositif bimanuel

Pour la partie matérielle, nous choisissons une commande bimanuelle avec un capot pour éviter la fraude de bras de l'opérateur, deux boutons (BD et BG) associés à deux capteurs normalement fermés (BD_NF et BG_NF). Les exigences seront donc allouées en partie à ces composants.

L'exigence ES 4.1 qui prescrit l'empêchement de la fraude de bras peut être allouée directement à la partie matérielle, et plus particulièrement au capot. En effet, seule la forme du capot pourra empêcher la fraude de bras. Toutes les autres exigences dépendent à la fois de la partie matérielle et de la partie logicielle (Figure 62).

- ES 4. 2 et 4 et 5 : Ces exigences sollicitent à la fois les boutons (BD et BG) et la partie logicielle.
- ES 4.3 : Cette exigence sollicite à la fois l'état des capteurs, l'état des boutons et la partie logicielle.

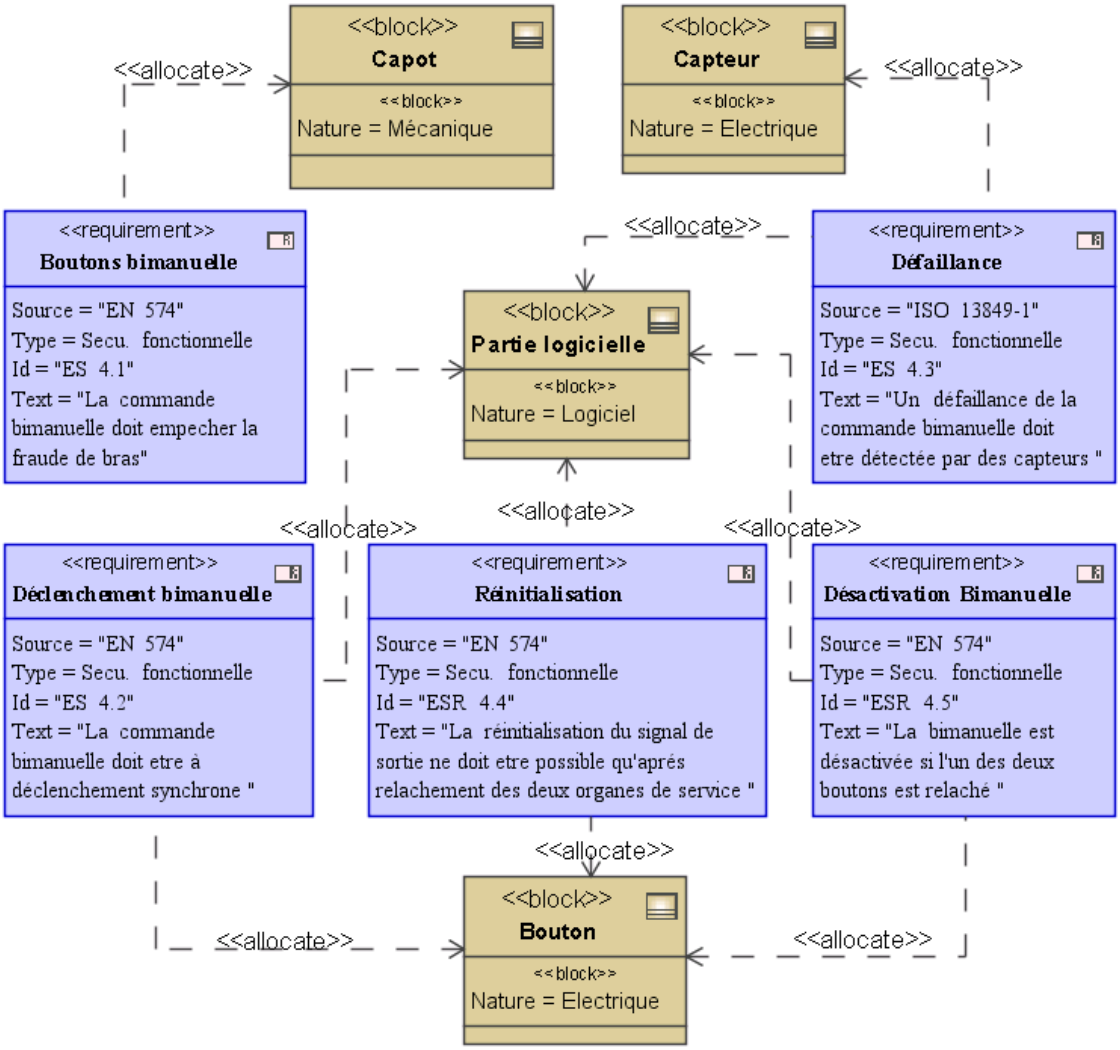


Figure 62 Allocation des exigences de la commande bimanuelle

Maintenant que la partie matérielle est définie, il nous faut décomposer la partie logicielle en composants. Pour cela, toujours à l'intérieur du processus de conception, nous allons réaliser la conception logique du logiciel.

A la lecture des exigences, on peut dégager deux fonctions dans l'architecture logique de la partie logicielle. La première va permettre d'activer et de désactiver la variable CBI (CBI à 1 signifiant que la bimanuelle est activée) en fonction des états successifs des boutons et de l'état de la variable D_CBI (D_CBI à 1 signifiant que la bimanuelle est en défaut). La deuxième activera et désactivera justement la variable D_CBI, en fonction des états des boutons et des capteurs de la partie matérielle. Au final on obtient le diagramme d'activités suivant (Figure 63) :

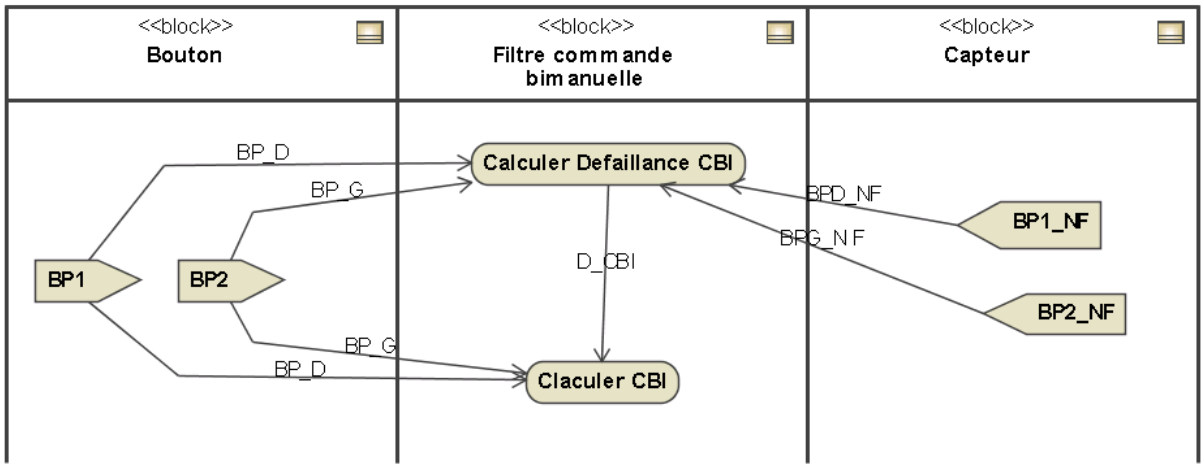


Figure 63 Diagramme d'activité pour la commande bimanuelle.

Il reste maintenant à faire la conception des parties matérielles et logicielles, toujours dans le processus de conception. Pour réaliser la partie logicielle, plusieurs choix sont possibles, on peut imaginer par exemple de mettre en place deux composants, un par fonction, voir même de réaliser ce composant non pas en logique programmée mais en logique câblée. Nous prendrons l'option de ne réaliser la partie logicielle qu'à l'aide d'un composant. Nous appellerons ce block : « filtre bimanuelle » (Figure 65).

A l'aide du diagramme de définition des blocks, et du diagramme d'activité, nous créons le diagramme d'instance (internal block diagram) correspondant à la commande bimanuelle. Celui-ci pose les instances de blocks utilisées ainsi que les signaux échangés (Figure 64).

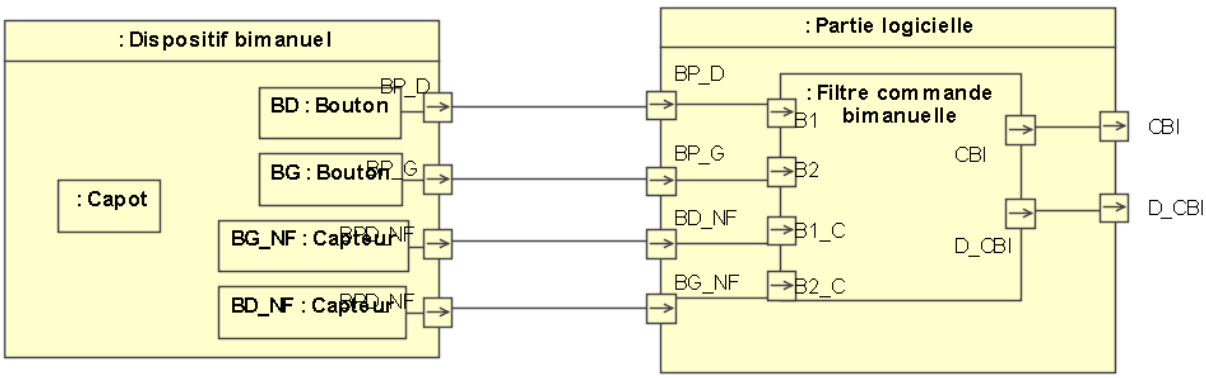


Figure 64 Internal block diagram du dispositif bimanuel

Ce diagramme de composants peut ensuite être exporté vers des ateliers de conception d'automatisme sur lesquels des techniciens pourront réaliser les fonctions à implanter dans les blocs définis par le processus de conception. Les fonctions peuvent également être réalisées en langage à état (Statechart) directement dans l'atelier de spécification SysML. Il faudra alors générer le code automate directement à partir des statecharts.

Dans les deux cas, les fonctions réalisées devront être soumises à un processus de validation pour s'assurer qu'elles respectent bien les exigences auxquelles elles sont allouées.

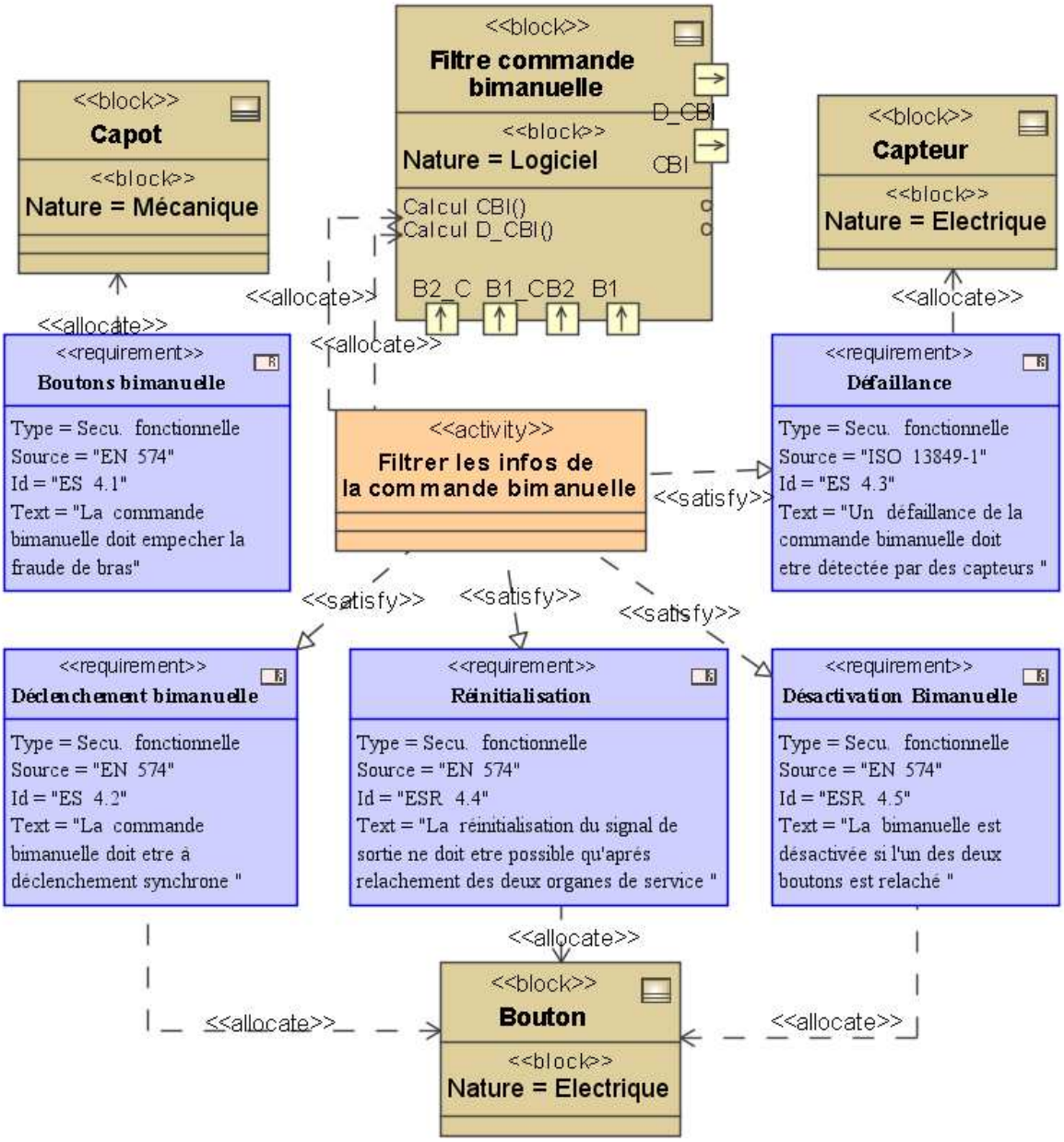


Figure 65 Allocation finale des exigences.

III.4.2 Opération de projection

La plupart de ces exigences sont réalisées par plusieurs composants du système. Il est donc nécessaire de les projeter pour obtenir des exigences ne portant que sur un bloc fonctionnel, ceci afin de faciliter la vérification.

La projection consiste pour chacun des blocs vérifiés à exprimer en fonction de ses entrées/sortie et du rôle qu'il doit jouer dans les exigences qui lui sont allouées. Pour la

projection, nous nous servons des propriétés. Nous allons donc projeter les propriétés P4.2 à P4.5. (La propriété P4.1 n'est allouée qu'au seul composant capot et ne sera donc pas projetée).

P4.2 : activation CBI ⇒ Enclenchement des deux boutons en moins de 0,5s

D'après la Figure 65, cette propriété est assurée par les composants boutons et filtre bimanuelle. Le composant bouton a comme sorties BP1 et BP2. Le composant filtre bimanuelle a comme entrées BP1 et BP2 et comme sortie CBI.

On obtient la projection suivante (Figure 66):

P4.2 ⇔ P4.2.a ∧ P4.2.b avec

P4.2.a = ↑CBI ⇒ moins de 0.5s depuis ↑BP1 ∧ moins de 0.5s depuis ↑BP2 *(projection sur filtre bimanuelle)*

P4.2.b = (↑BP1 ⇒ enclenchement de bouton droit) ∧ (↑BP2 ⇒ enclenchement de bouton gauche) *(projection sur bouton)*

En appliquant ce principe aux autres propriétés, on obtient les projections suivantes :

P4.4 ⇔ P4.4.a ∧ P4.4.b avec

P4.4.a = ↑CBI ⇒ (↓BP1 ∧ NOT BP2) ∨ (↓BP2 ∧ NOT BP1) depuis dernier ↓CBI *(projection sur filtre bimanuelle)*

P4.4.b = (↓BP1 ⇒ relâchement de bouton droit) ∧ (↓BP2 ⇒ relâchement de bouton gauche) *(projection sur bouton)*

P4.5 ⇔ P4.5.a ∧ P4.4.b avec

P4.5.a = not BP1 ∨ not BP2 ∨ D_CBI ⇒ not CBI (*projection sur filtre bimanuelle)*

P4.3 ⇔ P4.3.a ∧ P4.3.b∧ P4.4.b∧ P4.2.b avec

P4.3.a = Not (BP1 XOR BP1_C) ∨ NOT(BP2 XOR BP2_C) ⇒ D_CBI *(projection sur filtre bimanuelle)*

P4.3.b = Défaillance du dispositif ⇒ (BP1 XOR BP1_C) ∨ NOT(BP2 XOR BP2_C) *(projection sur bouton et capteurs)*

Ces projections doivent ensuite si possible être vérifiées formellement à l'aide d'un assistant de preuves. Le principe est le même que la preuve de raffinement (Figure 57)

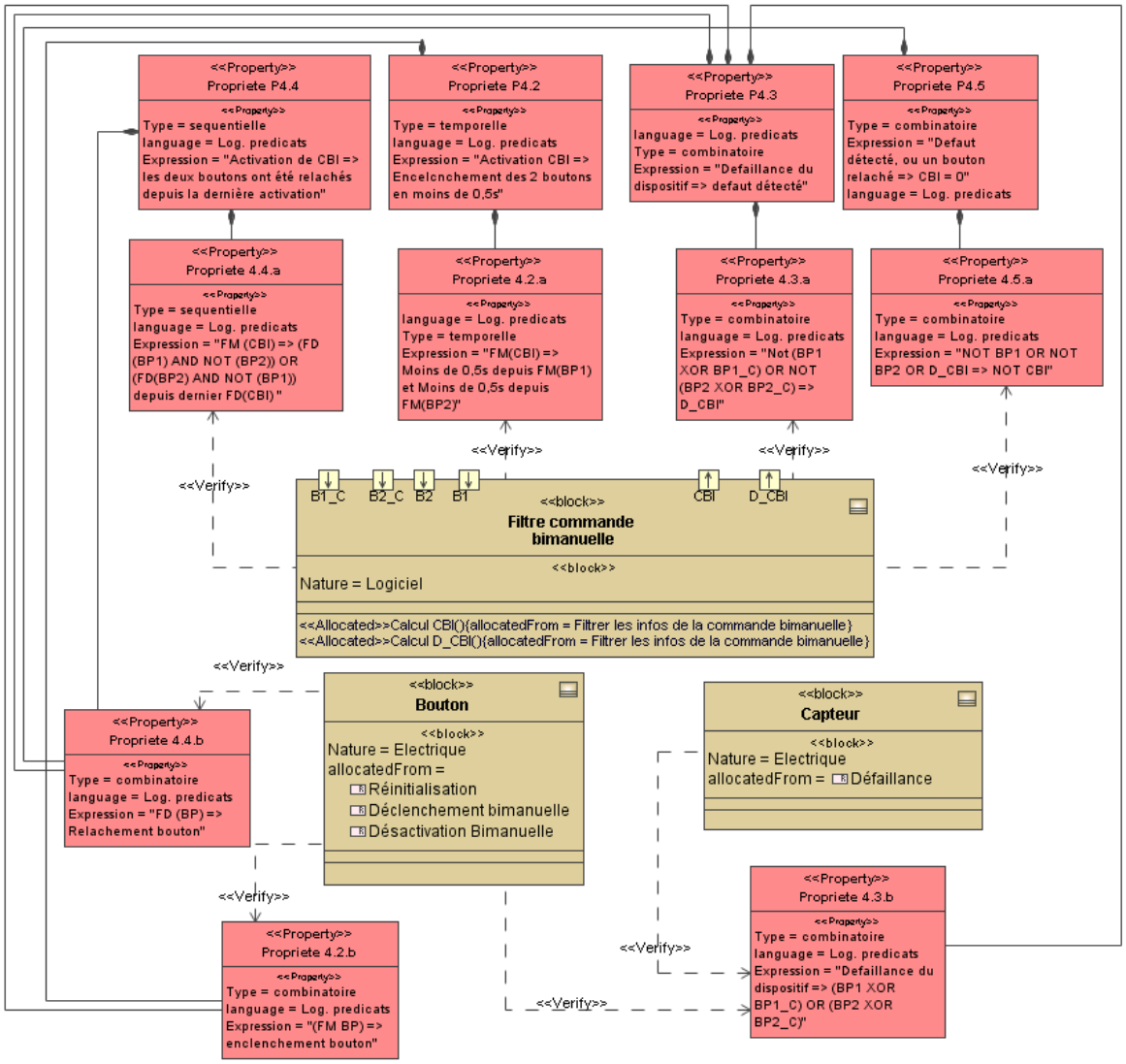


Figure 66 Projection des propriétés sur les composants de la commande bimanuelle

III.5 Conclusion : Modèle global

Le dernier modèle SysML est celui qui doit permettre la réalisation des composants. Il doit donc définir, de manière complète, l'architecture de composant du système, le type de composant (logiciel/matériel), les exigences projetées allouées à chacun des composants, et les échanges de variables entre ces composants.

La Figure 67 représente l'architecture du système ; on y trouve les différents composants système ainsi que les variables, ou groupes de variables, échangés par ces composants via les ports. On peut ensuite rentrer dans le détail des composants.

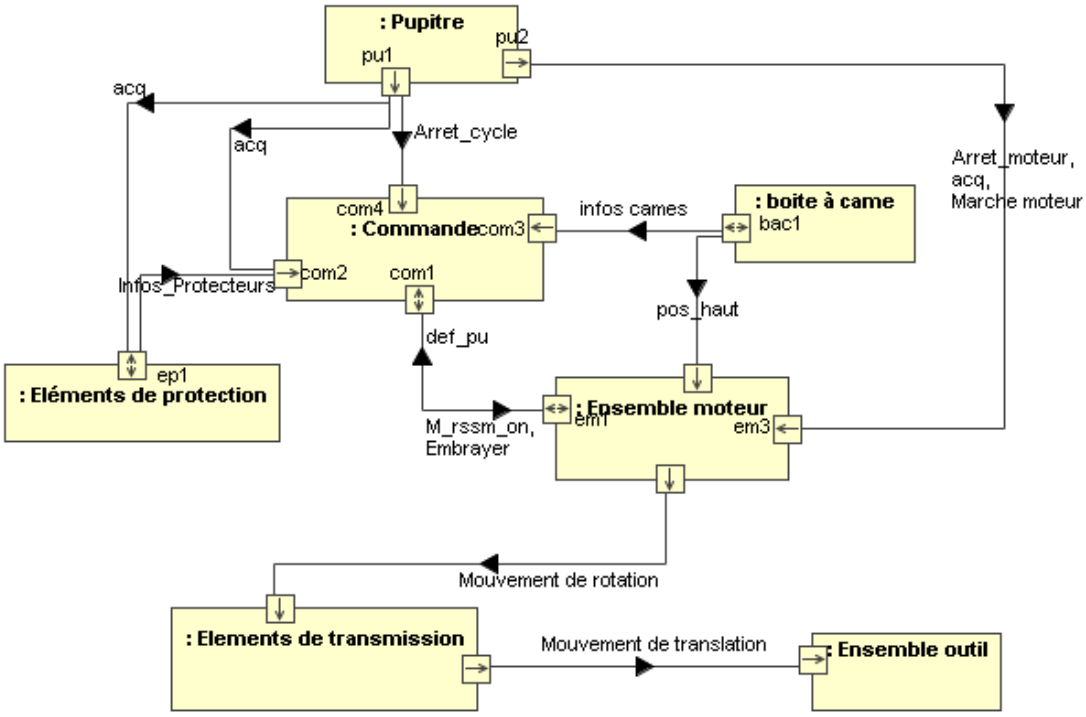


Figure 67 Architecture de composant du système

Le système complet est composé de 50 composants qui participent à la réalisation de 141 exigences. Parmi ces composants, on trouve 16 composants logiciels de bas niveau, dont 10 devant vérifier des propriétés de sécurité.

Au final la vérification par *model-checking* lors de la boucle de d'ingénierie du logiciel, portera sur 10 composants devant respecter localement 36 propriétés de sécurité.

IV - Processus de réalisation de la commande

IV.1 Réalisation des composants

Les modèles de composants réalisés en SysML sont ensuite exportés dans l'atelier d'automatisation Controlbuild, à l'aide du démonstrateur présenté au début de ce chapitre. La structure de composants ainsi générée, comporte non seulement les composants logiciels, mais aussi tous les autres composants du système.

Le comportement des composants logiciels de bas niveau est ensuite spécifié en Grafcet, avant d'être traduit dans un des langages de programmation de la norme CEI 61131-3.

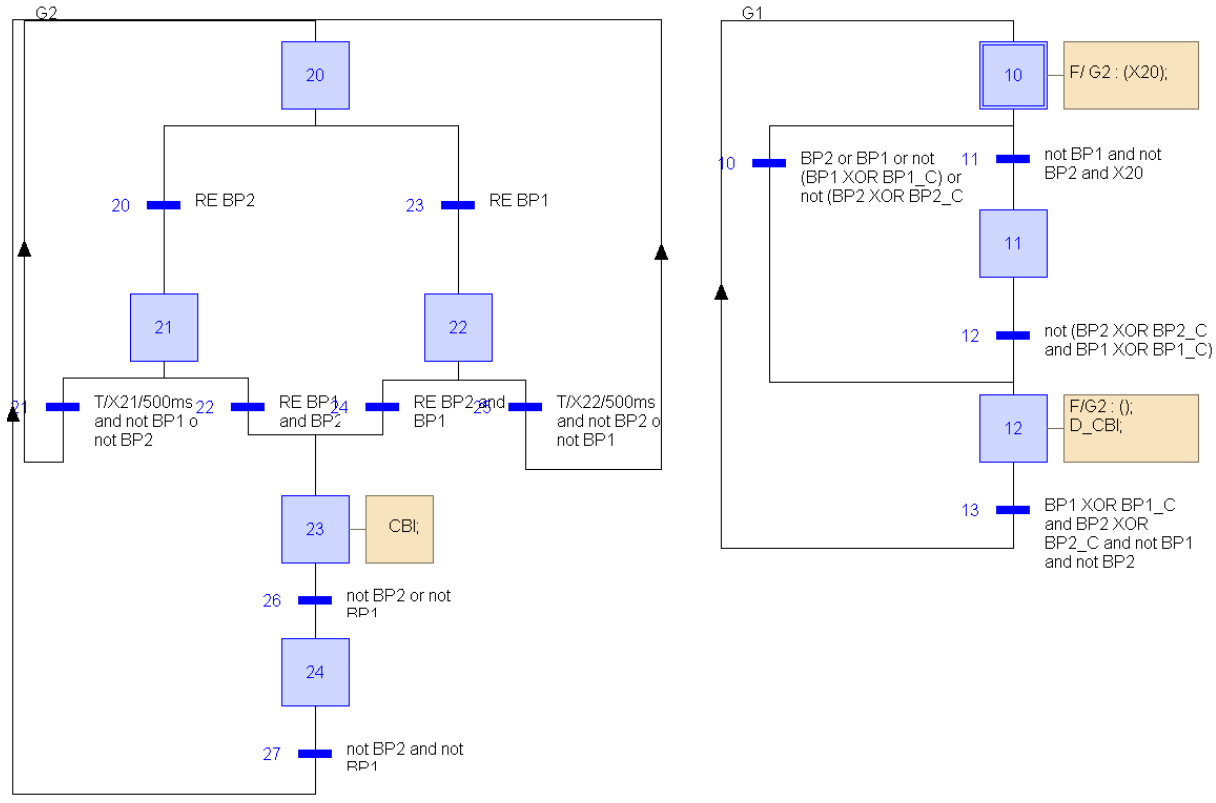


Figure 68 Spécification Grafcet du composant filtre_bimanuelle.

IV.2 Vérification des composants par simulation

On peut ensuite créer des modèles de comportement pour les composants physiques, afin de pouvoir simuler le comportement du système. Les scénarios de simulation se basent alors

sur l'analyse de risque (arbre des fautes, AMDE) pour établir les séquences pouvant mener à la violation d'une propriété locale.

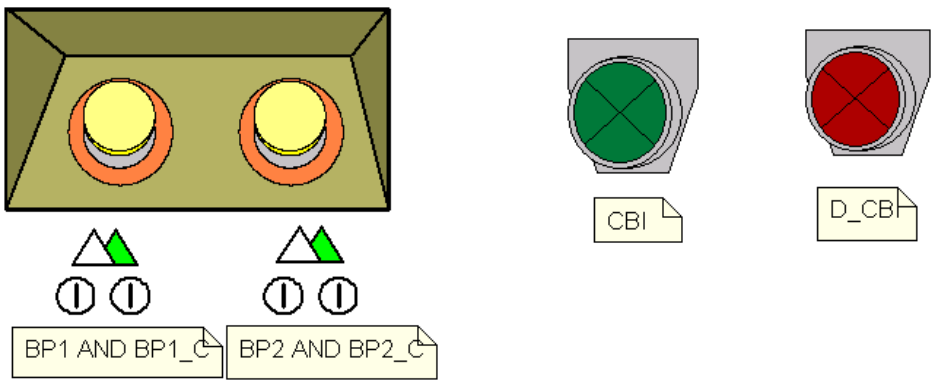


Figure 69 Vue de simulation pour le composant filtre_bimanuelle

Les propriétés locales des composants sont directement écrites dans les « post-conditions » de Controlbuild. Cela permet de s'assurer lors des simulations que chacun des composants respecte ses propriétés locales.

IV.3 Vérification des composants par *model-checking*

Comme on l'a vu au chapitre 2, les outils de *model-checking* sont bien adaptés à la vérification formelle de composants logiciels. C'est la technique que nous utilisons pour la phase de vérification des composants participant à la réalisation d'exigences de sécurité. En ce qui concerne la vérification des opérations de raffinement, le formalisme SysML n'étant pas formel, les seuls moyens disponibles pour la vérification des opérations de raffinement à un haut niveau d'abstraction, sont des systèmes de revues par cycle auteur lecteur. Pour les modèles plus concrets, la formalisation des exigences à l'aide de propriétés peut permettre une vérification plus formelle de l'opération de raffinement. La Figure 70 illustre les propriétés que doit vérifier le composant «filtre bimanuelle ». On retrouve des propriétés combinatoires (P4.5.a ; P4.3.a), temporelles (P4.2.a) et séquentielles (P4.4.a).

La formalisation de ces propriétés en logique temporelle, qui est le langage utilisé par les outils de *model-checking* pour l'expression des propriétés, est une étape qui se révèle plus ou moins compliquée en fonction du type de propriété. Les propriétés de type combinatoire sont les plus simples, elles sont de type « il est toujours vrai que ».Ainsi les propriétés P4.5.a et P4.3.a sont formalisées de la manière suivante :

P4.5.a : AG (NOT(BP1) OR NOT BP2 OR D_CBI ⇒ NOT (CBI))

P4.3.a : AG (NOT (BP1 XOR BP1_C) OR NOT(BP2 XOR BP2_C) ⇒ D_CBI)

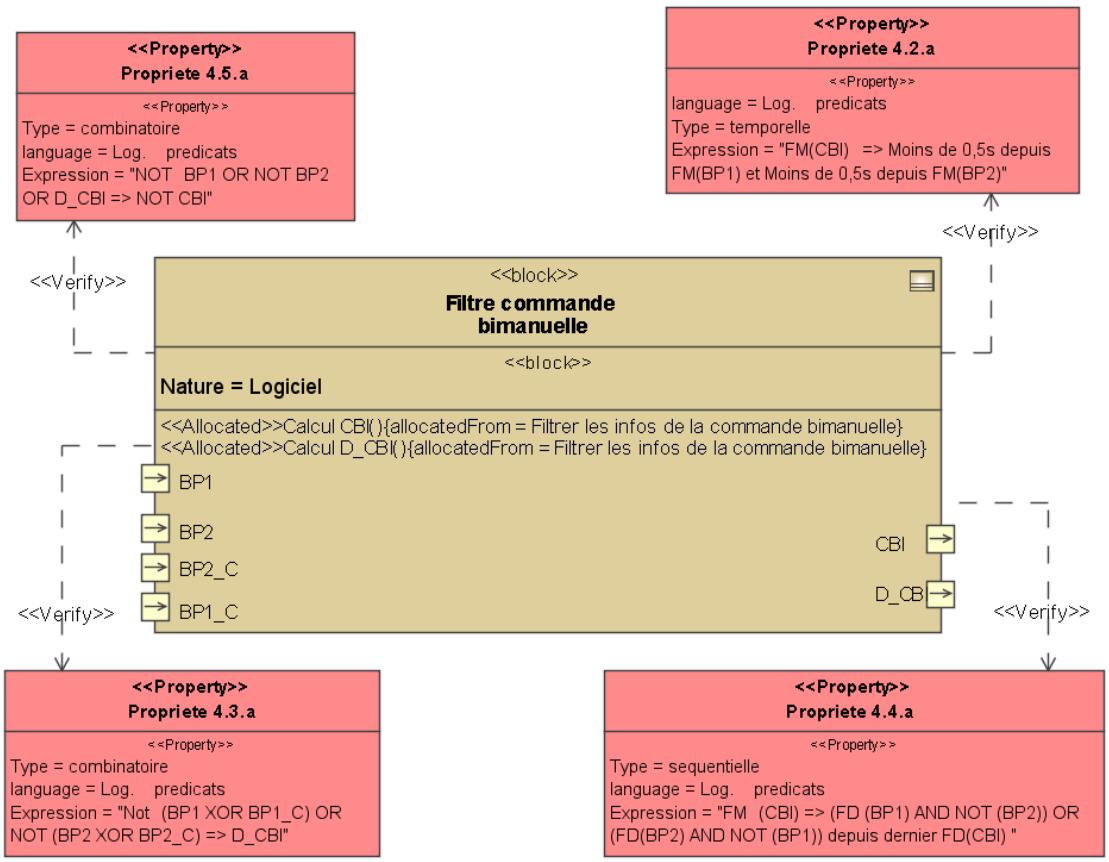


Figure 70 Propriétés du composant « filtre bimanuelle ».

Les propriétés de type séquentiel sont plus compliquées à écrire. En effet, même s'il existe un certain nombre d'opérateurs de logique temporelle permettant d'exprimer des propriétés sur des séquences d'états, leurs possibilités restent limitées. La propriété P1.1.4 par exemple ne peut être exprimée par ce type d'opérateur. Il faut alors avoir recours à un observateur. Pour des propriétés temporelles, aucun opérateur ne permet de les exprimer, on a alors systématiquement recourt à des observateurs temporisés.

Les observateurs sont des machines à état qui évoluent en fonction des changements d'état du système. Si ces changements d'états conduisent le système à un état non désiré, alors l'observateur rentre dans un état « Défaut ». Prouver la propriété consiste alors à prouver que l'observateur ne peut jamais atteindre l'état défaut. La propriété P4.4.a, suit une séquence à 4 étapes, une première étape où la bimanuelle est activée et les boutons enclenchés, une deuxième étape où la bimanuelle est désactivée, une troisième étape où la bimanuelle est désactivée avec aucun des boutons enclenchés, et une dernière étape qui est la copie conforme

de la première. Si cette séquence n'est pas respectée, que l'étape 3 est sautée, alors la propriété n'est pas respectée, et on rentre dans un état de défaut.

La Figure 71 illustre l'observateur à état représentant cette séquence. Les étapes 1 et 4 sont représentées par l'état « activée », l'étape 2 par l'état « transitoire », et l'étape 3 par l'état initial. L'état défaut n'est accessible qu'à la condition que la propriété ne soit pas respectée.

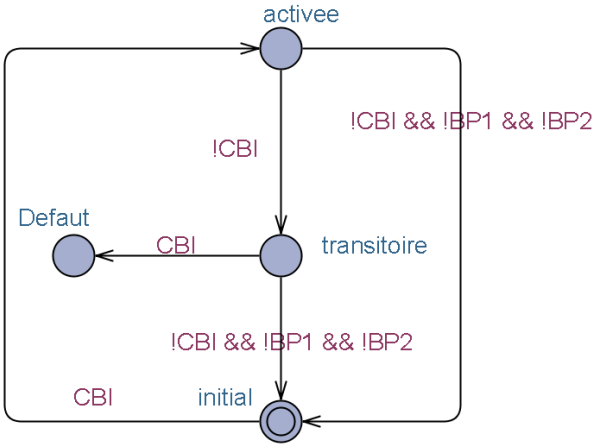


Figure 71 Observateur pour la propriété P4.4.a

La propriété P4.4.a s'exprime alors :

P4.4.a : AG (Defaut = 0)

V - Traces des exigences

Dans notre processus de développement le problème de la traçabilité est central. Il est adressé par trois facteurs :

- Les exigences d'intégrité de sécurité qui obligent le concepteur à connaître l'origine de tous les composants, fonctions et exigences identifiées pendant le processus de développement et à pouvoir justifier leur existence.
- Les analyses d'impact recommandées par les normes imposent de pouvoir parcourir le système des composants jusqu'aux exigences système de sécurité fonctionnelle, et inversement.
- La rupture entre les modèles de la boucle d'ingénierie système et ceux de la boucle d'ingénierie du logiciel nous oblige à maîtriser complètement les liens entre les exigences et l'architecture de composant et les propriétés de sécurité.

Dès lors que l'on veut tracer à la fois les processus de développement et les objets issus de ces processus, c'est que l'on se dirige vers une utilisation « haut niveau » de la traçabilité. En nous appuyant sur l'exemple développé dans le paragraphe précédent, nous avons montré comment construire un réseau de relations entre les différents objets du système (composants, exigences, fonctions, propriétés) afin de pouvoir réaliser les analyses d'impact (descendantes et ascendantes) prescrites par les normes de conception. Ce premier modèle de traçabilité sera enrichi en ajoutant les liens entre les processus mis en place et les objets conçus.

V.1 Traçabilité sur le système à faire

La traçabilité sur le système à faire permet de mener l'analyse d'impacts. Au fil de la construction du modèle SysML du système, des relations sont créées entre les objets du système.

Ces liens servent de base à toutes les analyses d'impact menées sur le système. Il est possible de remonter des composants vers les exigences système au travers des liens d'allocation, satisfaction et raffinement.

Toutefois dans le cas où les processus de vérification mettraient à jour des erreurs de modélisation, ces liens restent insuffisants pour détecter la source de l'erreur, car ils ne renseignent pas la provenance des objets du système.

V.2 Traçabilité sur le système pour faire

Il est donc nécessaire de tracer l'évolution du modèle au travers des processus utilisés pour sa construction, afin de connaître justement la provenance des fonctions, exigences et autres composants qui constituent le système. La Figure 72 représente les différentes relations entre les processus.

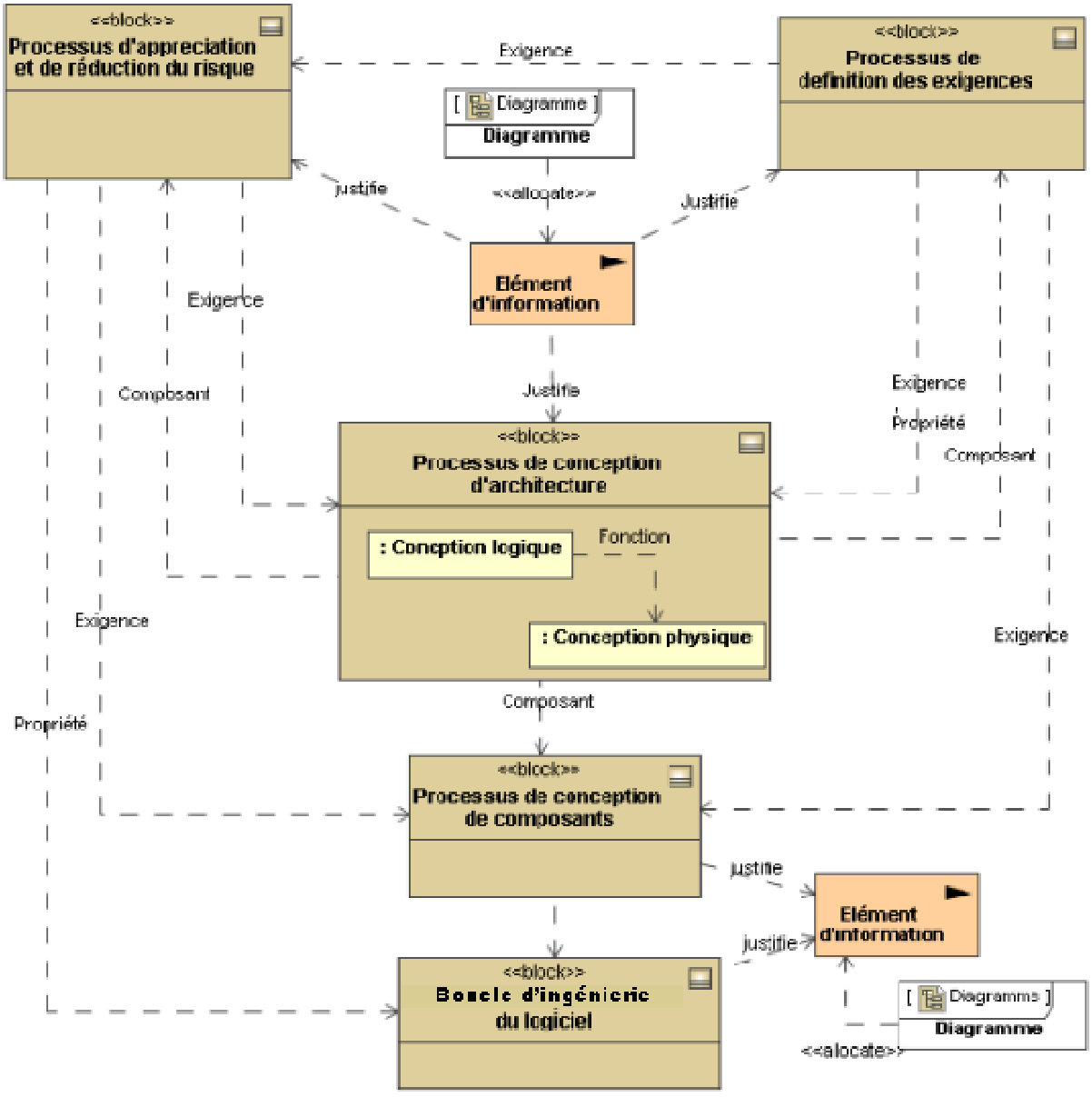


Figure 72 Modèle pour la traçabilité objets-processus.

Les processus s'échangent des objets du système, notamment la création des composants et des fonctions est attribuée au processus de conception, composants qui servent d'entrées aux processus de définition des exigences et d'analyse et de réduction du risque. Ces processus sont documentés par des éléments d'informations, qui peuvent être des diagrammes SysML, des calculs de dimensionnement, des analyses de risque etc. Ces éléments d'informations justifient les choix effectués dans le déroulement des différents processus.

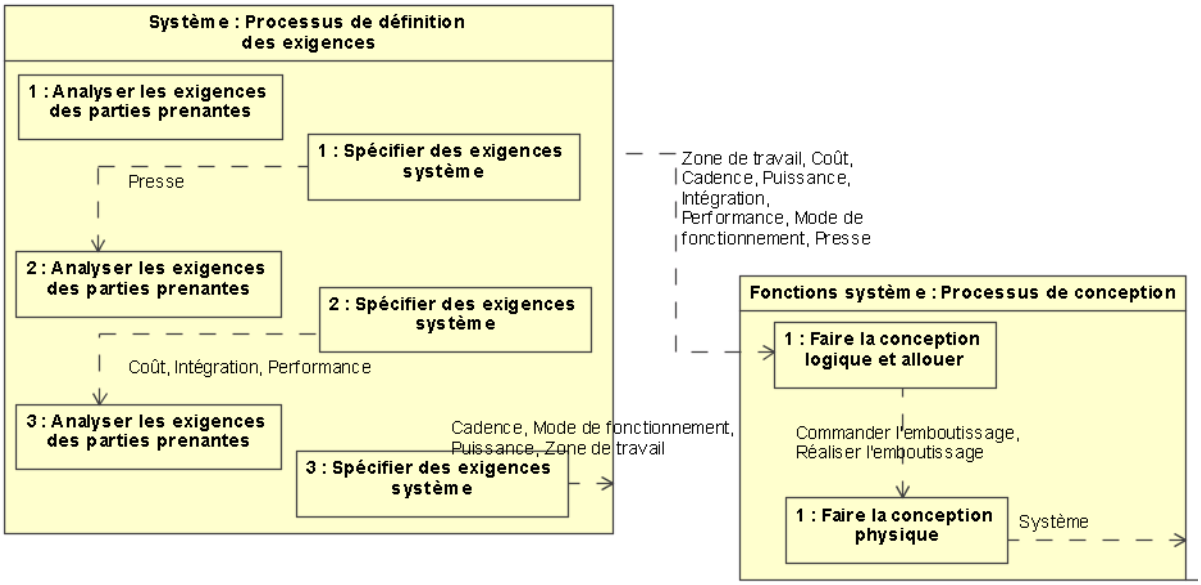


Figure 73 Traçabilité objets-processus pour la première boucle d'ingénierie système

VI - Conclusion

VI.1 Bilan

Le développement de la presse a mis en œuvre un certain nombre de processus :

- 9 processus de définition des exigences, qui ont permis de définir et de préciser dans l'ordre : les exigences « système », les modes de fonctionnement, les changements de modes, les exigences sur les positions de la presse, les conditions initiales des modes, le fonctionnement des organes de puissance, l'acquittement des modes.
- 7 processus d'appréciation et de réduction du risque qui ont permis de définir : les exigences système de sécurité fonctionnelle, les éléments de protections, les conditions de changement de mode, les exigences de sécurité sur les capteurs de position, les exigences de sécurité sur les organes de puissance, l'acquittement des défauts.
- 12 processus de conception, qui ont permis de définir des solutions aux exigences définies par les processus cités au-dessus.

VI.1.1.1 Réalisation du logiciel de commande

Le démonstrateur réalisé permet d'exporter automatiquement le modèle de composants ainsi que les propriétés des composants vers un atelier de conception de systèmes automatisés. Afin d'être sûrs que les modèles SysML des composants étaient suffisants pour la réalisation du logiciel de commande, la conception des composants logiciels, ainsi que leur vérification par simulation ont été réalisées par Quentin JANOT, étudiant en deuxième année à l'IUT de Saint-Dié des Vosges, et stagiaire à l'INRS.

Durant ce stage, nous nous sommes aperçus que les informations exportées à l'heure actuelle par le démonstrateur sont insuffisantes pour permettre à une personne novice sur ce type de système de développer une application complète. En effet, le démonstrateur n'exporte vers l'atelier d'automatismes que l'architecture de composant ainsi que les propriétés locales des composants et les exigences de bas niveau.

Or si cela suffit pour définir le comportement attendu d'un composant, c'est insuffisant pour comprendre le rôle du composant dans le système, voire même le comportement attendu du système. Il est donc nécessaire que la personne réalisant les composants logiciels ait connaissance de l'ensemble des exigences du système afin de comprendre le rôle des composants logiciels dans le système.

VI.1.1.2 Vérification

Le démonstrateur permet d'exporter les propriétés vers l'assistant de preuves COQ, permettant ainsi de vérifier les mécanismes de raffinement et de projection, mais également le code « Structured Text » des composants logiciels de bas niveau et les propriétés locales de ces composants vers le *model-checker* UPPAAL. Ainsi pour chacune des preuves (de raffinement, de projection, de comportement) à réaliser sur le système un fichier de base est généré automatiquement pour permettre la réalisation de ces preuves.

Cela permet dans un premier temps d'identifier automatiquement les preuves à réaliser sur le modèle, et dans un deuxième temps d'éviter des erreurs de recopie entre le modèle SysML, IEC 61131-3, et les modèles formels pour la vérification.

Toutefois la preuve n'est pas automatique, et quelques ajustements sont encore nécessaires :

- Pour la preuve par *model-checking*, il s'agit d'écrire la propriété qui est souvent écrite sous forme d'un prédicat en logique temporelle, voire même de construire un observateur dans le cas où celui ci n'aurait pas été construit dans le modèle SysML.
- Pour la preuve par *theorem-proving*, il est nécessaire comme on l'a vu plus haut d'introduire des propriétés souvent évidentes mais nécessaires à la réalisation de la preuve. La conduite de la preuve peut également nécessiter l'emploi de stratégies particulières ce qui nécessite un travail d'expert.

Conclusions et Perspectives

Lors de ce travail de thèse, nous avons essayé de répondre au problème posé par l'inadéquation entre des modèles d'exigences élaborés dans des processus d'ingénierie système, et des modèles de composants représentant la solution technique. Cette inadéquation est due en majeure partie aux différences de formalismes et de structures entre ces modèles :

- Depuis des années, les modèles de composants sont exprimés dans des formalismes de plus en plus formels pouvant être simulés, ou même dans le cas de systèmes à fortes contraintes de sécurité, pouvant supporter un processus de preuve. Dans le même temps les modèles d'exigences s'appuient sur des formalismes moins formels permettant d'appréhender un système dans son ensemble, en faisant abstraction des solutions technologiques retenues.
- S'ajoute à cela un problème d'ordre structurel, à savoir qu'il n'y a pas bijection entre d'un coté une architecture de composants et de l'autre une architecture d'exigences. Les exigences sont réalisées par des composants qui participent à la réalisation de plusieurs exigences.

Nous avons donc mis en place un ensemble de processus permettant dans un premier temps de combler l'écart entre les deux modèles, en formalisant les exigences sous forme de propriété, et dans un deuxième temps de projeter l'architecture de propriétés ainsi obtenue sur l'architecture de composants. Nous assurons ainsi la traçabilité des exigences du niveau système au niveau composant.

Dans le cadre du contrat de thèse passé avec l'INRS, nous avons cherché des solutions au contexte spécifique de l'industrie manufacturière. En effet, face à l'accroissement programmé de la complexité des fonctions de sécurité implantées dans les systèmes de commande des

machines de l'industrie manufacturière, et aux manques de continuité des processus de développement proposés par les normes de ce secteur, nous avons essayé de mettre en place un processus de développement sûr basé sur l'emploi de méthodes robustes et sur la traçabilité des exigences au travers des différents modèles.

En ce qui concerne les méthodes robustes, nous nous sommes concentrés dans un premier temps sur les méthodes de spécification et de vérification formelle. Toutefois, devant les difficultés rencontrées avec des méthodes de spécification formelle (méthode B) pour appréhender des systèmes complexes, nous avons opté pour un processus basé sur le formalisme SysML et sur des méthodes de vérification formelle (*Model-checking*). Ces deux formalismes associés aux processus que nous avons développés répondent effectivement aux attentes identifiées dans l'étude du contexte industriel :

- Le formalisme SysML permet l'expression des exigences en langage naturel et leur association avec des composants et des fonctions, le tout de manière graphique. Cela permet une modélisation hiérarchique et multi métier du système.
- La réalisation de preuves formelles sur les composants de sécurité du système rempli les objectifs fixés en terme de vérification.
- Enfin les processus de raffinement, allocation et projection mis en place permettent de tracer les exigences du niveau système au niveau composant, permettant ainsi des analyses d'impact lors d'échec de preuve en phase de vérification, ou lors de modification des exigences en phase de re-conception.

L'application de ce processus à la conception d'une presse mécanique a permis de valider dans le cadre du contrat de thèse l'applicabilité du processus proposé :

- le formalisme SysML est compréhensible par les ingénieurs de l'industrie manufacturière sans formation préalable. Ainsi nous avons pu faire valider les exigences de sécurité de la presse par le personnel de l'INRS chargé de la certification des presses mécaniques.
- il est également compréhensible par les techniciens. Nous avons demandé à un stagiaire de DUT de réaliser le programme de commande de la presse à partir des spécifications SysML de bas niveau.
- La vérification formelle par model-checking des composants individuels ne pose plus de problème d'explosion combinatoire.

Ce processus initialement pensé pour le secteur manufacturier sera d'autant plus « rentable » à mettre en place que le système sera complexe. C'est pourquoi nous avons dors et déjà prospecté, notamment par le biais du groupe résilience de l'AFIS, dans d'autres secteurs industriels tels que les systèmes de commande embarqués dans les transports où la complexité des systèmes développés et les contraintes de sûreté de fonctionnement auxquelles ils sont soumis justifient notre approche. En particulier, notre travail constitue un des éléments à la base d'une proposition de projet ANR dans le cadre du programme ARPEGE portant sur la définition d'une approche d'Ingénierie Système formelle pour la développement de système de commande à base de COTS dans le domaine du ferroviaire.

Références et Annexes

Tables des Références

Abrial J.R., 1996. *The B Book Assigning Programs to Meanings.* Cambridge Univ. Press

Abrial, J.R. (2006), Practical System Modelling. Example: a Mechanical Press Controller.,5

AFIS, 2005. Modèle de données version 2.0. Document AFIS 20/04/2005.

Auinger F., Brennan R., Christensen J., Lastra L.M., Vyatkin V. (2005). Requirements And Solutions To Software Encapsulation And Engineering In Next Generation Manufacturing Systems: OOONEIDA Approach. International Journal of Computer Integrated Manufacturing, Vol. 18(7), p572-585.,57

Behm P., Benoit P., Faivre A., Meynadier J.-M., 1999. Météor: a succesful application of B in large project. FM'99, Toulouse, France.

Bérard B., Bidoit M., Finkel A., Laroussinie F., Petit A., Petrucci L., Schnoebelen P. and McKenzie P., 2001. Systems and software verification. *Springer Verlag*.

Berezin S., Campos S., Clarke E.M, 1997. Compositional reasoning in model checking. Proc. COMPOS,pp81-102.

Beugin J., Renaux D., Cauffriez L., 2007. A SIL quantification approach based on an operating situation model for safety evaluation in complex guided transportation systems. *Reliability Engineering and System Safety* (à paraitre)

Blaise J.C., Lhoste P., Ciccotelli P., (2003). Formalisation of normative knowledge for safe design. Safety Science, Volume 41, Issues 2-3,March 2003, Pages 241-261.,15

Boehm B. W., 1986. A spiral model of software development. *ACM, SIGSOFT, vol 11, n°4.*

Boehm B., Port D., 2001. *Balancing discipline and flexibility with the spiral model and MBASE.* Cross Talk decembre 2001, p 23-28

Bon-Bierel E. (1998). Contribution à l'intégration des modèles de systèmes de production manufacturière par méta-modélisation, Thèse de l'Université de Nancy I.,57

Bouabana-Tebibel T., Belmesk M. (2007). An object-oriented approach to formally analyze the UML 2.0 activity partitions, *Information and Software technology*, Vol. 49, pp 999–1016.

Bouti A., Kadi D. A., 1994. A State-Of-The-Art Review of FMEA/FMECA. *International Journal of Reliability, Quality and Safety Eng.,* vol. 1, n°4, pp 515-543, 1994.

Brombacher A. C., van Beurden I. W. R. J., 1999. RIFT, analysing hardware *and* software in safeguarding systems. *Reliability Engineering and System Safety,* vol 66, p149-156.

Brown S., 2000. Overview of IEC 61508 Design of electrical/electronic/programmable electronic safety-related systems. *Computing and Control Engineering Journal.* Fevrier 2007.

Buchweiller J.P., Quelques repères pour s'orienter dans la normalisation traitant des circuits de commande relatifs à la sécurité. *Cahier de notes documentaires - Hygiène et sécurité du travail - N°192, 3ème trimestre 2003.*

Carer P., Leclercq P., Humbert S., 2006. Etudes pour la maîtrise de la fiabilité logicielle des nouveaux systèmes numériques installés sur les réseaux moyenne tension et basse tension d'EDF. *15eme Colloque Lambda MU-IMdR,* octobre 2006, Lille.

CEI 61131-3, Automates programmables - Partie 3, Langages de programmation. *1993, 411p.*

CEI 61508, Sécurité fonctionnelle des systèmes électriques/ électroniques/ électroniques programmables relatifs à la sécurité . *2002 , 7parties.*

CEI 62061, Sécurité des machines. Sécurité de fonctionnement des systèmes de commande électriques/ électroniques/ électroniques programmables pour les machines. *2005, 93p*.

Chapurlat V. (2007). *Vérification et validation de modèles de systèmes complexes*: *application à la modélisation d'entreprise*. Habilitation à Diriger des Recherches, Université de Montpellier II.

Charpentier et Ciccotelli, Equipements de travail, sécurité des systèmes programmés. *Techniques de l'ingénieur, art S8 270, 2006, 18p.*

Clarke E., Grumberg O., and Peled D., 1999. Model Checking. *MIT Press*.

Dawking, S.K., Riddle S., 2000. Managing and supporting the use of COTS. *Proceedings of the 8th Safety CRitical Systems Sympsium, southampton, UK*. Springer Verlag, London.

Directive 98/37/CE du 22 juin 1998 concernant le rapprochement des legislations des états membres relatives aux machines. *Journal Offciel des Communautés Européennes, n°L.207 du 23 jullet 1998, 46p.*

Eckhardt, D. E., Caglayan, A. K., Knight, J. C., Lee, L. D., McAllister, D. F., Vouk, M. A., and Kelly, J. J. 1991. An Experimental Evaluation of Software Redundancy as a Strategy for Improving Reliability. IEEE Trans. Softw. Eng. 17, 7 (Jul. 1991), 692-702.

EIA 632. Processes for Engineering a System,23

Eljamal M. H., 2006. Contribution à 'évolution des exigences et son impact sur la sécurité. *Thèse de doctorat de l'université Paul Sabatier de Toulouse*, soutenue publiquement le 12/12/2006.,53

Evrot D., Lamy P., Petin J-F., 2006a. Un comparatif d'outils formels. *Actes du congré Lambda Mu 15*, Lille, Octobre 2006.,67

Evrot D., Pétin J-F., Mery D., Formal specification of safe manufacturing machines using the B method: application to a mechanical press, *in proc. 12th IFAC INCOM*, vol1 pp 277-282, St-Etienne, France.,94

Evrot D., Pétin J-F., Morel G., Lamy P., 2007. Using SysML for identification and refinement of machinery safety properties. *Actes du congrés Dependable Control of Discretes Systems*, Cachan, 2007.,102

EXERA, Compte rendu du CT "groupe d'évaluation des langages d'automates". *Réunion du 24/05/200, INRS doc interne n° IET-S/00CRR-041, 2000, 3p.*

Fadier E., De la Garza C., Safety Design: Towards a new philosophy. *Safety Science N°44 2006, p55-73.*

Favre J.M., Estublier J., Blay-Fornarino M. (2006). L'ingénierie dirigée par les modèles: au-delà du MDA. Informatique et Systèmes d'Informations. Hermes, Lavoisier 2006, ISBN 2-7462-1213-7.,56

Figarol D., Chatty S., Schlienger C., 2002. *La méthode Dphi de conception de systèmes interactifs*. IntuiLab Rapport TECH02-16

Forsberg K., Cotterman H., Mooz H., 2005. Vizualizing Project Management: Models and frameworks for mastering complex systems. Relié - 23 semptemebre 2005.

Forsberg K., Mooz H., 1991. *The relationship of system engineering to the project cycle.* Proceedings of the 1991 INCOSE Symposium.

Fusaoka *et al*, 1982

Goble, W. M. 1998 Control System Safety Evaluation and Reliability. 2nd. ISA.

Gourcuff V., De Smet O., Faure J. M., 2006. Efficient representation for formal verification of PLC programs. *8th international Workshop On Discrete Event Systems (WODES '06)*, Ann Arbor (USA), p182-187.

Guo H., Yang X., 2006. A simple reliability block diagram method for safety integrity verification. *Reliability Engineering and System Safety*, n°92, p1267-1273.

Harel D., 1987. Statecharts, A Visual Formalism for Complex Systems, *Science Computer Programming*, n°8, p231-274

Heaven W. and Finkelstein A. A UML Profile to Support Requirements Engineering with KAOS. *IEEE Proceedings - Software*, Vol. 151, pp. 10-27.

Hoare C.A.R., An axiomatic basis for computer programming. *Communications of the ACM*, 12: 576-580, 583, 1969.

Huet G., Khan G. and Paulin-Mohring C, 1995. The Coq proof assistant – a tutorial. *Technical Report 178*, Inria.

IEEE 1220. Standard for application and management of the systems engineering process, janvier 1999.

Isermann R., Schaffnit J., Sinsel S., 1999. Hardware-in-the-loop simulation for the design and testing of engine-control systems. *Control Engineering Practice*, Vol 7, p 643-653.

ISO 12100, sécurité de machines. Notions fondamentales. Principes généraux de conception. *2003.*

ISO 13849. *2006, 88p*.

ISO 14121, Sécurité des machines. Principes pour l'appréciation du risque. *1999, 28p.*

ISO 15288. Systems Engineering – System Life-Cycle Processes,23

ISO 8402 (1995). Management de la qualité et assurance de la qualité - Vocabulaire

ISO/CEI Guide 51, Aspects liés à la sécurité. Principes directeurs pour les inclure dans les normes. *1999, 11p.*

Jensen K., Kristensen L. M., and Wells L., 2007. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. Int. J. Softw. Tools Technol. Transf. 9, 3 (May. 2007), 213-254.

Johnson T. L. (2007). Improving automation software dependability: a role for formal methods ? *Control Engineering Practice, Volume 15, Issue 11, November 2007, Pages 1403-1415*.

Jones, C.B., 1990. *Systematic Software Development using VDM.* Prentice Hall 1990.

Kamsu Foguem B. (2004), *Modélisation et vérification des propriétés de systèmes complexes*: *application aux processus d'entreprise*, Thèse de doctorat de l'Université de Montpellier II.

Kim T., Stringer-Calvert D., Cha S., 2005. Formal verification of functionnal properties of a SCR-style software requirements specification using PVS. *Reliability Engineering and System Safety,* vol 87, p351-363.

Kneppert M. et Dei-Svaldi D., Les APIdS. *Jautomatise n°22, juillet-Aout 2003.*

Kneppert M., 1986. SADT. Cahier de note INRS n° 123, 1986, P175-192.,31

Lamine E (2001). *Définition d'un modèle de propriété et proposition d'un langage de spécifiction associé*: *LUSP*, Thèse de doctorat de l'université de Montpelier II.

Lamy P., 2003. Experience feedback concerning develoment methodology for programmable electronic system application software. *Actes du du 3ème congrés international Safety of Industrial Automated Systems*, Octobre, Nancy.

Lebrun M., Simulation et CAO en mécanique et mécatronique. *Techniques de l'ingénieur, art S7 260 2003, 15p.*

Lhoste P., Faure J. M., Lesage J. J., Zaytoon J., 1987. Comportement temporel du Grafcet. *JESA-AFCET/CNRS,* editions Hermes, Vol 31-N°4, p713-740.

Marangé P., Gellot F., Carré-Ménétrier V., Riera B., 2007. Validation de commande des systèmes à évènements discrets. Actes de MSR 07,p86-102.,97

Mauguy L., Historique et attentes de PSA sur les otuils numériques. *Journée* "méthodes et outils pour l'automatisation" *du club automation. Mars 2005.*

Melham T.F. and Gordon M.J.C., 1993. Introduction to HOL: A theorem proving Environment for higher-order logic. Cambridge University Press.

Ménadier J.P. (2002). *Le metier d'intégration de systèmes*. Hermès Science Publications, Lavoisier, ISBN 2-7462-0596-3.

Neugnot C., Kneppert M., Logiciels applicatifs relatifs à la sécurité. Etude des problèmes liés à leur exploitation. Cahiers de notes documentaires - Hygiènes et sécurité du travail - N° 187, 2ème trimestre 2002

Niel E., Pietrac L., Regimbal L. (2001). Advantages and drawbacks of the logic programm synthesis using supervisory control theory.10th *IFAC/INCOM'01 Symposium*, Vienna, Austria.

OMG, 2003. *UML Superstructure, version 2.0*, OMG document ad/03-08-02.

OMG, 2006. *System Modeling Language version 1.0*, OMG document

Ortmeier F., Schellhorn G., Thums A., Reif W., Hering B., Trppschuh H., 2003. Safety analysis of the height control system for the Elbtunnel. *Reliability Engineering and System Safety,* N°81, p259-268.

Owre S., Rajan S., Rushby J., Shankar N. and Srivas M.K., 1996. PVS: combining specification, proof checking and model checking. *Computer Aided Verification 1996*, LNCS 1102, pp 411-414. Springer-Verlag.

Palmer J. D., 1997. Traceability, In *Software Requirements Engineering,* Thayer R.H. and Dorfman M. (Eds), IEEE Computer Society Press, pp 364-374.

Paulk M. C., Weber C. V., Curtis B., Chrissis M. B., 1995. The Capability Maturity Model. Guidelines for Improving the Software Process. Addison-Wesley Longman Publishing Co., Inc.

Paulson L. C. 1994. Isabelle: A Generic Theorem Prover. Lecture Notes in Computer Science, vol. 828. Springer-Verlag, New York, NY.

Pénalva J-M. (1997). La représentation par les systèmes en situation complexe. Doctorat de l'Université d'Orsay.

Petin J. F., 2007. *Méthodes et Modèles pour un processus sur d'automatisation*, Habilitation à diriger les recherches, Université de Nancy. Soutenue publiquement en décembre 2007)

Pétin J-F., Morel G., Panetto H. (2006) Formal specification method for systems automation. European Journal of Control, vol. 01, ISSN 0947-3580.

Piétrac L. (1999). Apport de la méta-modélisation formelle pour la conception des systèmes automatisés de production, Thèse de l'Ecole Normale Supérieure de Cachan.,57

Pilz, Manuel Utilisateur. *Ref*: *8-8-012-03/99 Sach-Nr. 80 005* Printed in Germany, 1999

Ramadge P.J., Wonham W.M. (1987). Supervisory control of a class of discrete event processes. *SIAM J. Control and Optimization*, Vol. 25(1).

Ramesh B. Jarke M., 2001. Toward reference models for requirements traceability. *IEEE Transactions on Software Engineering*, vol 27, N°1, janvier 2001, pp 58-93.

Redmill F., 2004. Analysis of the COTS debate. *Safety Science N°42, p335-367.*

Roussel J.M., Denis D., Safety properties verification of ladder diagram programs, Européen des Systèmes Automatisés, 36(7), pp. 905-917, 2002,68

Rouvroye J., Brombracher A., 1999. New quantitative safety standards: different techniques, differents results? *Reliability Engineering and System Safety,* n°66, p121-126.

Royce W. W., 1970. *Managing the development of large software systems*. IEEE WESCON.

Rushby J, 2002. Using model checking to help discover mode confusions and other automation surprises. *Reliability Engineering and System Safety*, N°75, p167-177.

Sheard S. (2006). Complex Systems Science and its Effects on Systems Engineering. *European Systems Engineering Conference*, 18-20 September 2006, Edinburgh, UK.

Shell T. (2001). Systems functions implementation and behavioural modelling: system theoretic approach, *International Journal of Systems Engineering*, Vol. 4(1)

Son S. H., Seong P. H., 2001. Development of a safety critical software requirements verification method with combined CPN and PVS - A nuclear power plant protection system application. *Reliability Engineering and System Safety,* N°80, p19-32.

Sowa J.F., Zachman J.A. (1992). Extending and formalizing the framework for Information system architecture, IBM Systems Journal, Vol. 31(3), pp 590-616.,57

Spivey J. M., 1989. *The Z notation*: *a reference manual.* Prentice Hall, Inc.

Statistiques Financières et technologiques des accidents du travail. Edité par la CNAM. ISBN 0761-327X. 2000

Thramboulidis K., Tranoris C., 2007. Developping a CASE tolls for distributed control applications. *Journal of advanced manufacturing technology,* vol 24 N° 1-2, p 24-31, springer verlag.

Tranoris C., Thramboulidis K. (2006). A tool supported engineering process for developing control applications, Computers in Industry, Vol. 57(5), June 2006, Pages 462-472.,57

Villemeur A., 1988. *Sûreté de fonctionnement des systèmes industriels - Fiabilité, facteurs humains, informatisation.* Collection de la direction des études et recherche d'EdF, Paris, Eyrolles, 1988.

Von Knethen A., 2001. A trace model for system requirements changes on embedded systems. *Proceedings of 4th International Workshop on principles of software evolution,* september 2001.,77

von Knethen A., 2002. Change-Oriented requirements traceability. Support for evolution of embedded systems. *Proceedings of International conference on software Maintenance,* october 2002, pp 482-485.

Willard B., 2006. UML for System Engineering. *Computer Standards & Interfaces* n°29, pp 69-81, 2007.

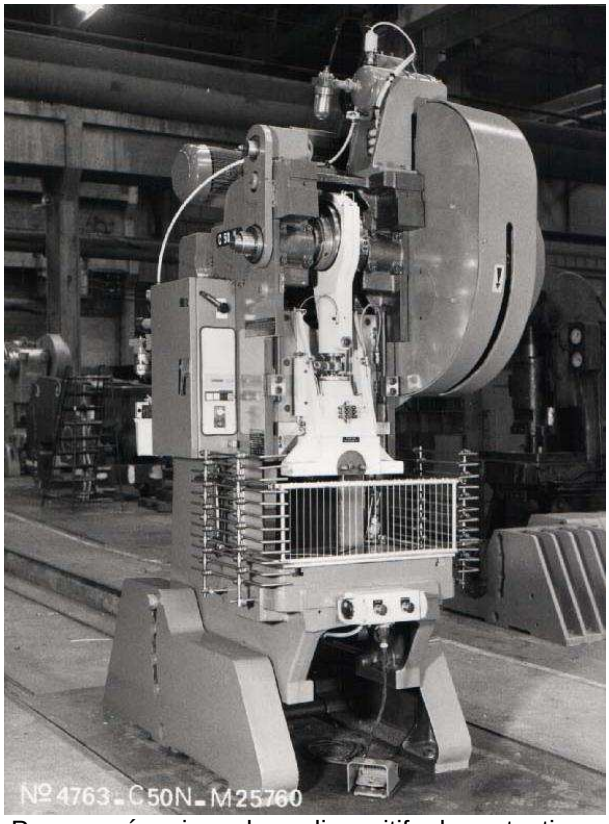
Zhang T., Long W., Sato Y., 2003. Availability of systems with self diagnostics components-applying Markov model to IEC 61508-6. *Reliability Engineering and System Safety,* n°80, p133-141.

Zhang W., Diedrich Ch., Halang W.A. (2005). Specification of Function Block Applications with UML. Proc. IEEE Intl. Conf. on Robotics and Automation, Barcelona, Spain.,57

Zimmerman M., Rodriguez M., Ingram B., Katahira M., de Villepin M., Levesson N., 2000. Making formal method practical. *Proceedings of 19th Digital avionics systems conference*, philadelphia, 2000.

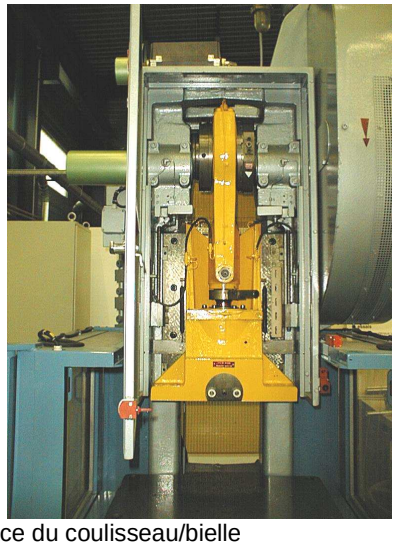
Annexe 1 : Cahier des charges de la presse

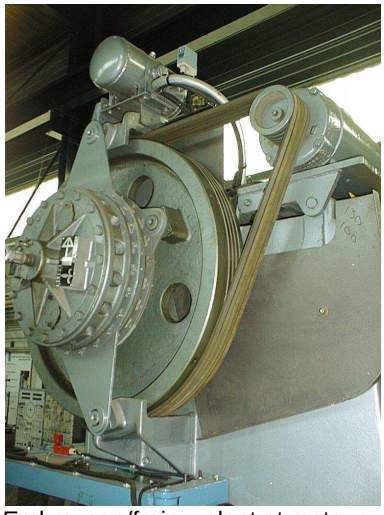
Description de la machine



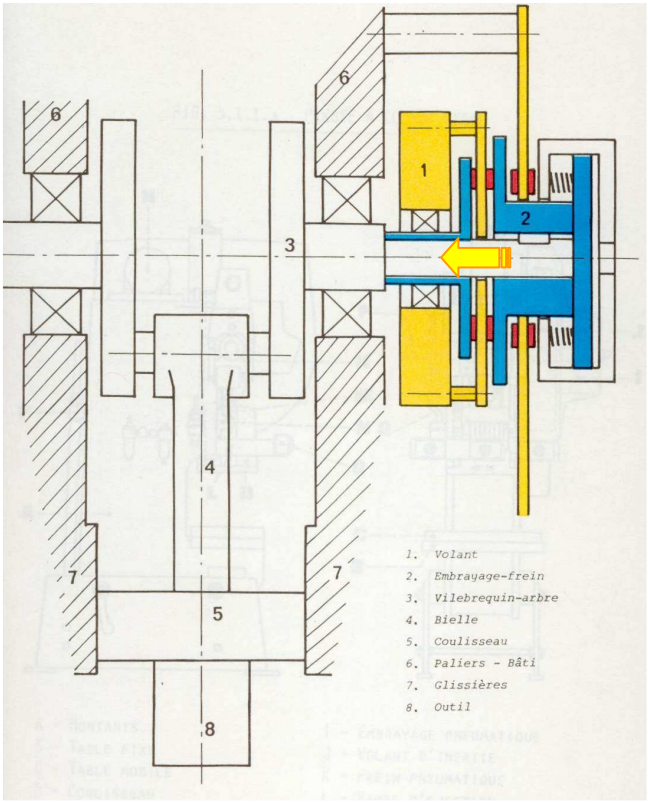
Presse mécanique, hors dispositifs de protections

Presse mécanique à embrayage/frein combiné à friction à bâti col de cygne.


Vue de face du coulisseau/bielle


Embrayage/frein, volant et moteur

Analyse cinématique



La figure ci-dessus illustre la cinématique d'une presse mécanique, en position "embrayée". Le mouvement du coulisseau est obtenu par rotation du vilebrequin. Celle ci est obtenue lorsque le volant n°1 est en rotation (entraîné par un moteur électrique non représenté sur cette figure) et que embrayage est en position embrayé grâce à l'injection d'air comprimé à l'endroit indiqué par une flèche, via une électrovanne électro pneumatique non représentée sur cette figure. L'arrêt est obtenu par manque d'alimentation en air comprimé. Les ressorts rappellent la partie représentée n°2 vers une partie fixe n°6 liée au bâti de la presse.

Analyse des modes de fonctionnement

La sélection des cycles de fonctionnement s'effectue par un sélecteur verrouillable à clef. Pour les modes de marche ou le moteur fonctionne, le changement du sélecteur ne devra pas entraîner d'arrêt du moteur.

Un armement de la machine est effectué par action sur un bouton poussoir :

- après chaque mise sous tension,
- après chaque changement de mode de marche,

- après le réarmement d'un dépassement de l'angle de freinage,
- après un arrêt d'urgence,
- pour les modes de marche "production", après chaque ouverture/fermeture du carter de bielle,
- pour les modes de marche "production" avec outils ouverts, après chaque ouverture/fermeture des protecteurs latéraux.

Mode de réglage sans moteur

Le moteur et le volant doivent être à l'arrêt. Le démarrage du moteur doit être impossible.

Une action maintenue sur la pédale commande l'embrayage. Le volant peut alors être tourné manuellement à l'aide d'une barre.

Les protecteurs avant, latéraux et le carter de bielle peuvent être ouverts, sans interrompre le cycle.

Mode de réglage en marche avant :

Le moteur principal doit être en marche.

L'embrayage est commandé par une action synchrone sur les deux boutons poussoirs de la commande bimanuelle. Cette action doit être maintenue pendant la totalité du cycle (descente puis montée), jusqu'à l'arrêt automatique point mort haut. Une fonction de non-répétition interdit de faire plus d'un cycle si la commande bimanuelle n'a pas été relâchée puis actionnée de nouveau lors de l'arrêt au point mort haut.

La protection de l'opérateur est assurée par la commande bimanuelle. Tout relâchement en n'importe quel point du cycle d'au moins un bouton poussoir de la commande bimanuelle provoque l'arrêt du coulisseau. Les protecteurs avant, latéraux et le carter de bielle peuvent être ouverts, sans interrompre le cycle.

Mode de marche coup par coup commande bimanuelle :

Le moteur principal doit être en marche.

Le cycle est commandé par une action synchrone sur les deux boutons poussoirs de la commande bimanuelle. Cette action doit être maintenue jusqu'au point mort bas, le retour du coulisseau au point mort haut étant ensuite automatique. Une fonction de non-répétition

interdit de faire plus d'un cycle si la commande bimanuelle n'a pas été relâchée puis actionnée de nouveau lors de l'arrêt au point mort haut.

La protection de l'opérateur est assurée par la commande bimanuelle. Tout relâchement d'au moins un bouton poussoir de la commande bimanuelle pendant la phase de descente provoque l'arrêt du coulisseau. Les protecteurs latéraux et le carter de bielle doivent être obligatoirement fermés. Leur ouverture commande l'arrêt du mouvement du coulisseau lors de la descente et de la montée.

Mode de marche continue

Le moteur principal doit être en marche.

Le cycle "continu" est commandé par une impulsion synchrone sur les deux boutons poussoirs de la commande bimanuelle. Une impulsion sur le bouton d'arrêt spécifique pour ce mode de marche, en n'importe quel point du cycle, commande l'arrêt automatique en point mort haut.

La protection de l'opérateur en face avant est assurée par un protecteur mobile. Les protecteurs latéraux et le carter de bielle doivent être obligatoirement fermés. L'ouverture d'un protecteur, avant, latéral ou du carter de bielle commande l'arrêt du mouvement du coulisseau à la descente et à la montée.

Fonctionnement du moteur principal :

La vitesse de rotation est variable. Le démarrage "direct" du moteur est commandé par action sur le bouton poussoir "Marche Moteur". Son arrêt par action sur le bouton poussoir "Arrêt".

Annexe 2 : Diagrammes SysML

Différences entre SysML et UML2.0

La figure ci-dessous présente les ajout et évolutions existants entre SysML et UML2.0. Dans notre travail nous nous sommes essentiellement appuyés sur le diagramme des exigences, le diagramme de blocs et le digramme interne de blocs.

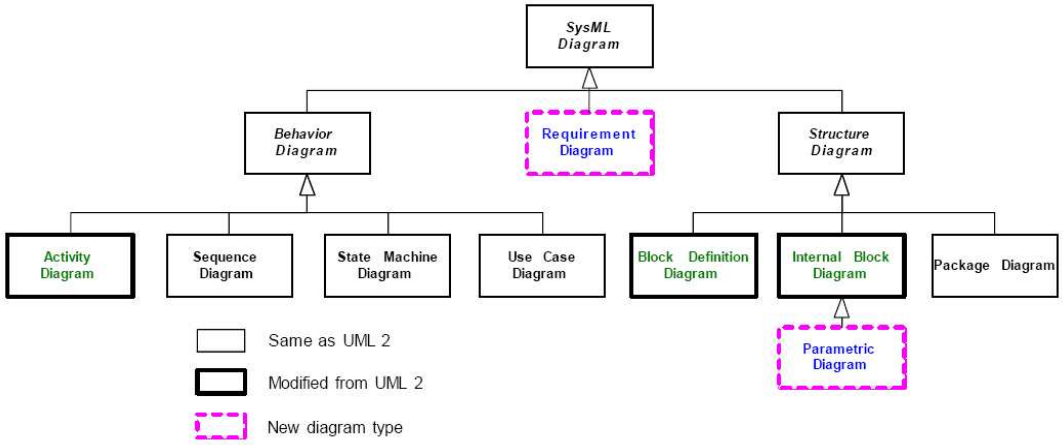
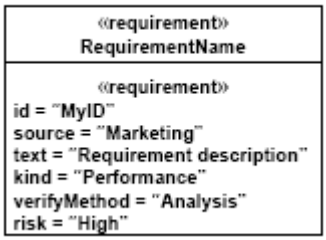


Diagramme des exigences (requirement diagram)

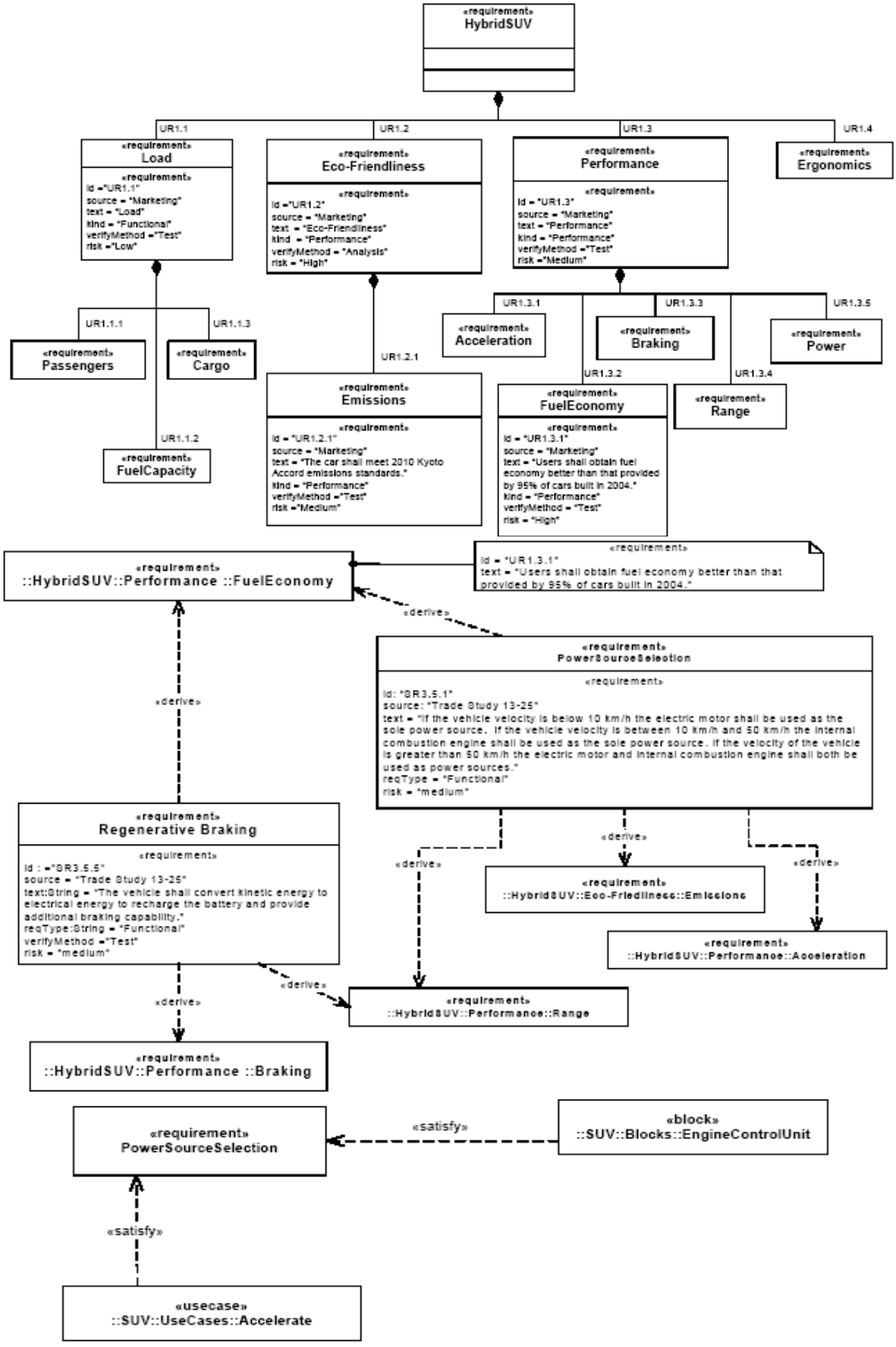


Une exigence est définie comme stéréotype de classe avec comme attributs, une identité (« *id* »), une origine (« *source* », une description textuelle (« *text* »), un type (« *kind* »), une méthode de vérification (« verify method ») etc. La liste des attributs est ouverte, et peut être complémentée par toutes les informations nécessaires aux parties prenantes pour la définition d'une exigence.

Une exigence peut être décomposée en plusieurs exigences par des liens de composition. Un lien de dérivation peut également exister entre deux exigences. Une exigence peut être satisfaite par des composants du système (blocks) ou des ativités. Une exigence peut également être vérifiée au moyen d'un TestCase.

PATH TYPE	*CONCRETE SYNTAX*
Composition	
Derive	«derive»
Satisfy	«satisfy»
Verify	«verify»

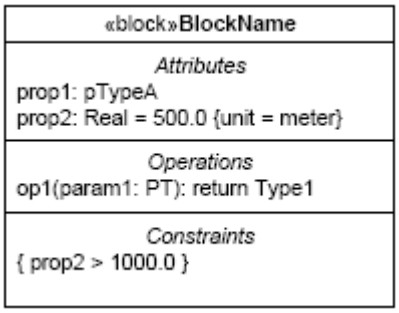
A noter que dans nos modèles une exigence est satisfaite par une fonction et est allouée à un composant, ceci afin de bien différencier ces deux types de construction.



Exemple de diagramme des exigences avec liens de satisfaction, composition et dérivation.

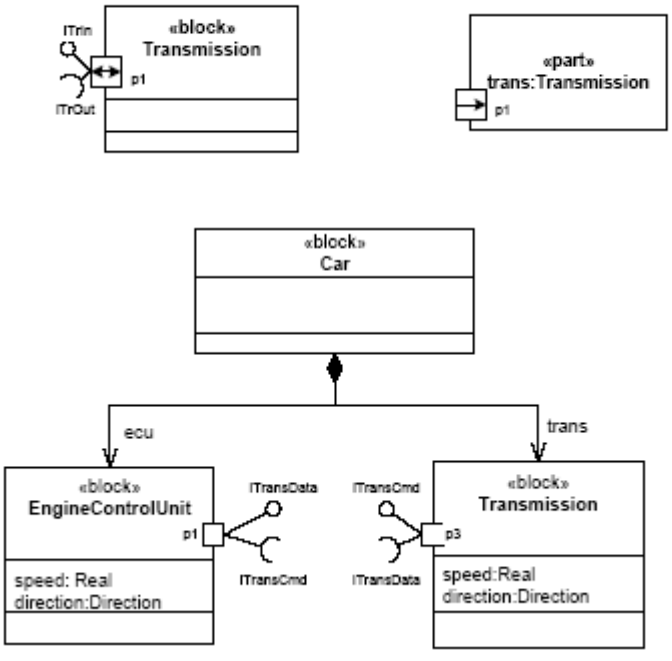
Diagrammes de blocs (block definition diagram et internal block diagrams)

Le diagramme de définition des blocs permet de définir les composants (blocs) comme des stéréotypes de classe. Ils possèdent des attributs, des opérations et des contraintes. Ils peuvent être reliés entre eux par des liens de composition, agrégation, containment etc. Ce diagramme permet de donner l'architecture des composants.

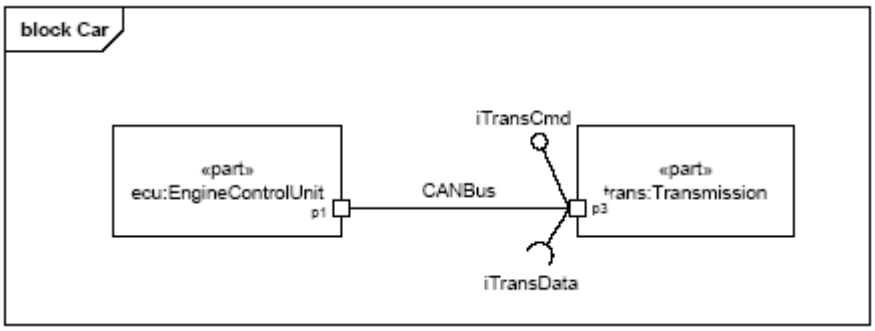


PATH NAME	CONCRETE SYNTAX
Aggregation	
Association	
Composition	
Connector	connectorName: associationName
Containment	

L'Internal block diagram permet lui de définir les informations échangées par les instances des blocs (part) via des ports (flow ports)



Exemple de Block Definition Diagram



Exemple d'Internal Block Diagram