Ontology centric design process : Sharing a conceptualization

Ofaina Taofifenua

CONSERVATOIRE NATIONAL DES ARTS ET MÉTIERS

le cnam

École Doctorale EDITE

Laboratoire CEDRIC

THÈSE DE DOCTORAT

présentée par : **Ofaina TAOFIFENUA**

soutenue le : **10 Juillet 2012**

pour obtenir le grade de : **Docteur du Conservatoire National des Arts et Métiers**

Discipline / Spécialité : **Informatique**

ONTOLOGY CENTRIC DESIGN PROCESS

Sharing a Conceptualization

THÈSE DIRIGÉE PAR

Mme LÉVY Nicole	*CNAM, CEDRIC*

RAPPORTEURS

M. AIT-AMEUR Yamine	*ENSEEIHT, IRIT*
M. KAANICHE Mohamed	*LAAS-CNRS*

EXAMINATEURS

Mme DUBOIS Catherine	*ENSIIE, CEDRIC*
M. BELMONTE Fabien	*ALSTOM*
M. BOULANGER Jean-Louis	*CERTIFER*
M. GAUDRÉ Thierry	*RENAULT*

"New knowledge is the most valuable commodity on earth. The more truth we have to work with, the richer we become."
Kurt Vonnegut, Breakfast of Champions

Acknowledgements

The work at the origin of this thesis have been realized jointly at Renault's DELTA ("Direction de l'Électronique et des Technologies Avancées") in the system and software department within the team SAEE ("Sytème et Architecture Électrique Électronique) and at CNAM's SITI (École Sciences industrielles & technologies de l'information) in team CPR ("systèmes sûrs: Conception et Programmation Raisonnées") of the CEDRIC ("Centre d'Étude et De Recherche en Informatique et Communications") laboratory. So I take this opportunity to thank Pr Bastard and Pr Himbert respective directors of DELTA and SITI. I give my thanks to my department head and successive team managers at Renault, Bruno Font, Olivier Guetta, Hugo Chalé, Philippe Quéré and Christophe Dang. Similarly I give my thanks to CEDRIC's director Pr Crucianu. I also want to thank CPR's team manager Pr Dubois. To all these people, a big thank you for welcoming me in your organization.

I would also like to express my deepest thanks to Nicole Lévy who kindly agreed to supervise this PhD thesis, Thierry Gaudré my Renault's supervisor and Jean-Louis Boulanger who kindly agreed to share his expertise in this work. Without their availability, patience, support, guidance, words of encouragement, provided throughout these three years, this thesis could probably not have been possible. I will never thank them enough for the trust they put in my person by taking me as a PhD candidate. Please find here the testimony of my gratitude and kind regards.

I would like to thank Mohamed Kaâniche and Yamine Ait-Ameur who accepted the burden of rapporteur. They also agreed to be part of my jury. I give my thanks to Catherine Dubois and Fabien Belmonte in addition to my rapporteurs and supervisors for making me the honor to examine my work. Their presence in the jury is my pleasure.

The staff of the department, by their kindness and friendliness which they have demonstrated during my presence, have made my stay very pleasant among them. I want to thank the department's team of engineers: Atef, Boubker, Cosmin, Emmanuel, François, Frédéric, Hoang, Hugo, Joris, Ludovic, Patrick, Pascal (the two of them) Paul-Éric, Paulo, Philippe, Régis, Rémy, Saidou, Sann, Sophie, Sylvain, Thierry, Xavier and Youssef. They were able to make warm and fruitful these past few years among them by their human qualities and skills. I have fond memories of the many discussions, professional and non-professional, which have enriched me personally. In particular, I would like to address my acknowledgment to Hugo and Thierry who took great interest in this work and gave much support both in time and expertise. I also thank the department's administrative staff, Catherine, Christine and the two Dominique, for their effectiveness and all services provided.

I would like to address my acknowledgments to the staff of the LISV ("Laboratoire d'Ingénierie des Systémes de Versailles") who welcomed me in their laboratory. I want to express my gratitude to Pr Amar who provided unconditional support. I don't forget my comrades who shared and shares similar experience that helped to put in perspective my personal one. A big thank you to Hassan, Rima, Sébastien, Mona, Maya, Mata and Sylvain. I also give my thanks to LISV administrative staff, Catherine and Dominique for they availability and all services provided.

I think it is appropriate to give my acknowledgments to all my teachers and professors encountered during my academic. Thank you for giving me the desire to continue in studies. In particular, I want to address my thanks to Alain Griffault, David Powell and Jérémie Giochet who introduced me to dependability which is the domain I deeply connected with.

A big thank you to all my friends who supported me for so many years: Albane, Céline, Enzo, Fa Yuan, Frédérique, Jonathan, Julianna, Landry, Moana, Raimana (the two of them), Rémy, Sara and Vincent. Thank you for putting up with my person during difficult times and trying to make me relax. I don't forget my friends outside of metropolitan France: Brandon, Carole, Diego, Heimanarii, Heimata, Jake, Julianna, Maui, Moana, Namoiata, Nicolas, Raihei, Teiki, Titaina, Vétéa, Yannick and Yvonnick. Thank you for making me live in your mind, you live in mine too.

My deepest gratitude goes to my wonderful family which is my personal blessing. Thank you all for your unconditional support and love. A special mention goes to my loving mother and her husband. Thank you for believing in me in difficult times and giving me all the means to continue my studies. I address my thanks to my brother and sister for giving me additional hardships. I do not love you less. Finally, my thoughts goes to my late father to whom this thesis is dedicated.

Abstract

For the last several years, car manufacturers have had to face an always-increasing list of stakes and challenges. In the strongly competitive worldwide market of today, a car manufacturer has to offer to its customers relevant, innovative, reliable, environment friendly and safe services of the highest quality. All this must be done at very competitive costs while complying with more and more stringent regulations and tighter deadlines. This work addresses these challenges and aims at improving the design process for automotive safety critical mechatronics systems. In addition, the introduction of systems engineering at Renault and the emergence of international standard ISO 26262 (published in November 2011) that addresses functional safety in the automotive are central considerations.

At Renault, systems engineering introduction leans towards model-based systems engineering where different models with respective viewpoints about one system are developed during the design phase. The first implementations of the design process were mainly document-centric and depended largely on testing and simulation. Although these first attempts yielded quite satisfactory results, the creation of the different objects of the process was somewhat troublesome and relatively time-consuming. The reason for this is that the objects were modeled by means of transformations of ad-hoc data and information contained in the different documents that were transmitted from one process step to the other. The main difficulty in implementing the process consisted in the lack of semantic consistency among the different modeled objects.

Functional safety studies are carried out following a second process in parallel to the precedent design process. These two processes are interfaced to develop safe systems. These two processes manipulate some concepts that are similar, however with a different conceptualization. These different conceptualizations are translated in loss of knowledge

at the processes interfaces which imped communication between the two domains as they can be incompatible and can potentially result in undetectable inconsistencies.

This need for a better formalization is further stressed by the fact that car manufacturers rely heavily on third parties that also have their own conceptualizations. Better formalization of processes and of the process objects would certainly contribute to avoid confusion and misinterpretations in the development of systems.

All this led us to the conclusion that the use of formal and informal models can commit to a common semantic model, *i.e.*, a system and safety ontology, that enables to ensure the consistency of the whole design process and compliance with ISO 26262. The improved design process is called ontology centric design process. The concepts in this work have been applied on a regenerative hybrid braking system integrated into a full electrical vehicle. It demonstrated that the realized ontology enables to record the information produced during design and that using ontologies effectively enables to detect semantic inconsistencies which improves design information quality, promotes reuse and ensures ISO 26262 compliance.

Résumé

Les constructeurs automobiles doivent faire face à une liste toujours croissante d'enjeux et défis. Dans le marché mondial fortement concurrentiel actuel, un constructeur doit offrir à ses clients des services de la plus haute qualité. Ces derniers doivent être pertinents, novateurs, fiables, respectueux de l'environnement et sûrs. Tout ceci doit se faire à des coûts très compétitifs tout en respectant des réglementations de plus en plus strictes et des délais de plus en plus courts. Les travaux présentés dans ce mémoire de thèse répondent à ces attentes et visent à améliorer le processus de conception des systèmes mécatroniques critiques automobile. En particulier, nous avons cherché à évaluer la contribution des techniques formelles, en continuité ou en rupture, sur le processus de développement des systèmes mécatroniques, qui doit être conforme à l'ISO 26262, d'un constructeur automobile tel que Renault. Nous cherchons notamment à répondre aux questions de recherche suivantes :

Q1 Comment les connaissances du domaine (automobile) peuvent être formalisées ?

Q2 Comment les connaissances d'experts sur le processus de développement peuvent être formalisées ?

Q3 Comment le respect de la norme ISO 26262 peut être vérifié ?

Chapitre 1. L'objectif du premier chapitre est de présenter les domaines contextuels ainsi que les bases théoriques sur lesquelles se fonde notre travail. Dans un premier temps, nous présentons le processus de développement produit, par la discipline qui s'est développée autour de l'élément central appelé système : l'ingénierie des systèmes. Cette discipline s'attache à définir des processus (activités), supportés par des méthodes (techniques), elles mêmes supportées par des outils. La représentation usuelle du processus de développe-

ment système est le cycle en V. Cette représentation définit trois phases générales. La branche descendante du V correspond à la conception, la base adresse la réalisation et la branche ascendante représente l'intégration du système. Dans nos travaux nous ne considérons que la conception. Cette phase peut être résumée à l'ingénierie des exigences (où des exigences sont définies et décrivent le système en devenir) et à la conception des architectures (fonctionnelle et physique) systèmes. Une autre spécialité s'est également développée pour prendre en compte les systèmes dits critiques (à enjeux de sécurité, économiques et environnementaux) : la sûreté de fonctionnement. Quand les conséquences d'un mauvais fonctionnement sont catastrophiques pour l'utilisateur, on parle de sécurité-innocuité et de sécurité fonctionnelle. Les activités de la sécurité fonctionnelle se représentent dans un deuxième cycle en V exécuté en parallèle du processus de développement usuel afin de démontrer l'absence de risques inacceptables. Pour finir, les principales méthodes utilisées en ingénierie des systèmes sont présentées avec un bref focus sur MOF, le standard de l'OMG, qui permet de représenter et manipuler des métas-modèles. En se basant sur MOF, il est possible d'unifier toutes les étapes du développement mais il reste cependant un problème d'intégration (sémantique des outils).

Le chapitre poursuit sur les ontologies et met l'accent sur les ontologies formelles qui nous intéressent. Ces dernières permettent de formaliser un domaine en définissant les éléments suivants ainsi que des propriétés sur ces éléments : individus, classes, propriétés et attributs. Ces définitions sont l'objet d'axiomes qui définissent des assertions sur un domaine particulier qui peuvent également contenir la théorie dérivée des formules axiomatiques génératives. De plus, les ontologies contiennent également des axiomes qui rendent les hypothèses d'un domaine explicites pour les humains et les systèmes informatiques. Cette structure mathématique permet le raisonnement mathématique spécifique en fonction de la logique et de la fonction d'interprétation utilisées. La fonction d'interprétation associe les valeurs habituelles vrai ou faux aux assertions du domaine quand l'hypothèse du monde fermé est prise, *i.e.*, ce qui n'est pas connu pour être vrai est faux. Une autre hypothèse peut cependant être prise rajoutant un individu au domaine d'interprétation : l'hypothèse du monde ouvert, *i.e.*, ce qui n'est pas connu pour être vrai est inconnu. Ces bases mathématiques font que les ontologies formelles ont une propriété de cohérence

mathématique, *i.e.*, présence ou non de contradiction. Au final, les ontologies formelles sont l'artefact idéal pour conceptualiser une compréhension commune de disciplines multiples afin de définir le terrain d'entente sur lequel différents acteurs peuvent s'entendre et répondre au problème d'intégration ou de cohérence sémantique identifié plus loin. Nous finissons sur les ontologies en présentant les technologies du web sémantique que nous avons choisi d'utiliser dans nos travaux. OWL, le langage ontologique du web, permet de définir des classes (concepts), des propriétés (propriétés et attributs) et des instances (individus). SWRL, le langage de règles du web sémantique, ajoute la capacité de définir des règles pour des ontologies en OWL. Et SQWRL, le langage de requêtes du web sémantique, permet d'extraire des informations des ontologies en OWL.

Le chapitre 1 conclut en présentant les travaux connexes, notamment les travaux sur l'intégration sémantique (d'outils) dont nous reprenons les idées pour les amener plus loin vers la cohérence sémantique du processus de conception système.

Chapitre 2. Ce chapitre présente les contributions apportées. Nous commençons par les ontologies de domaine sur l'ingénierie des systèmes puis sur la sécurité fonctionnelle. Bien que nous ayons choisi d'utiliser les technologies du web sémantique, les ontologies sont présentées en utilisant la logique du premier ordre afin de faciliter la lecture (en particulier, le lecteur peut prendre l'hypothèse plus courante du monde fermé).

En ingénierie des systèmes nous avons pris en compte les activités les plus générales d'analyse du besoin, d'ingénierie des exigences et de conception d'architectures. Le vocabulaire utilisé lors de ces activités a été formalisé (défini, structuré et contraint) en OWL en s'inspirant des définitions de l'INCOSE (l'organisation de référence dédiée à l'avancement de l'ingénierie des systèmes, de l'AFIS (la branche française de l'INCOSE) et du vocabulaire utilisé à Renault. La formalisation est relativement simple pour l'analyse des besoins. Le point à retenir est que les besoins doivent êtres pris en compte par des exigences ce qui est formalisé avec une propriété reliant les besoins aux exigences. Pour l'ingénierie des exigences, nous avons fait les choix importants qui suivent. Les exigences considérées correspondent à la conception. Elles sont structurées en graphe acyclique orienté (une hiérarchie permettant à un élément d'avoir plusieurs parents). Les exigences sur le processus ne sont

par exemple pas prises en compte. Les exigences sont naturellement typées fonctionnelle et non fonctionnelle. Les exigences sont précisément définies avec les fonctionnelles donnant lieux à des fonctions, les non fonctionnelles au niveau système sont allouées au système et les non fonctionnelles au niveau d'abstraction inférieur sont allouées aux fonctions ou aux composants. Pour la conception des architectures nous décrivons l'organisation des fonctions et des composants. Les fonctions sont également structurées en graphe acyclique orienté et les composants sont organisés en hiérarchie sans qu'un composant puisse avoir plusieurs parents. L'architecture fonctionnelle décrit l'organisation des fonctions à travers les flux consommés et produits. Puis les fonctions sont allouées aux composants qui sont également organisés à travers les flux consommés et produits par les fonctions qu'ils réalisent décrivant l'architecture physique du système. La prise en compte des exigences fonctionnelles est donc précisément définie avec leur matérialisation en fonction dans l'architecture fonctionnelle puis la prise en compte de ces fonctions par des composants (nous avons également donné une sémantique précise à la déclinaison des exigences fonctionnelles en fonctions et à l'allocation des fonctions aux composants). Concernant les exigences non fonctionnelles, nous avons donné des éléments de réponse pour leur formalisation (en vue de leurs prises en compte). Les types d'exigences non fonctionnelles sont nombreux et pour chaque type il faut définir les éléments permettant de les concrétiser dans l'architecture fonctionnelle et / ou physique. Par exemple, une exigence non fonctionnelle de poids allouée au système est décomposée en exigences non fonctionnelles de poids allouées aux différents composants du système. Elles peuvent être concrétisées par un attribut entier des composants et du système avec une sémantique à choisir ce qui par exemple permettrait, une fois cet attribut renseigné pour les composants, de calculer le poids du système automatiquement et ainsi de vérifier que toutes les exigences de poids soient bien satisfaites.

En sécurité fonctionnelle, nous avons considéré le processus actuel de Renault et les parties de la norme ISO 26262 au niveau système. De manière générale, la sécurité fonctionnelle est prise en compte à travers un processus de gestion du risque contenant les activités d'analyse du risque et d'évaluation du risque qui sont considérées dans ces travaux. Nous avons donc formalisé tout le vocabulaire utilisé en s'inspirant des standards ISO/IEC guide 51, IEC 61508 et ISO 26262 ainsi que du vocabulaire Renault. Nous avons choisi de démar-

rer l'analyse du risque en partant des exigences fonctionnelles. Les notions d'exigences, de fonctions et de composants présentes en ingénierie des systèmes sont également présentes en sécurité fonctionnelle et reformalisées (quoique partiellement, l'idée étant de réutiliser les définitions faites dans l'ontologie d'ingénierie des systèmes). En analyse du risque, les exigences fonctionnelles sont analysées de manière plus systématique à travers l'utilisation d'un modèle de défaillances à appliquer pour chaque exigence afin de déterminer des événements redoutés. A Renault, ces derniers sont soit des événements redoutés système ou des événements redoutés clients. Les événements redoutés clients sont particulièrement intéressants car la sécurité-innocuité est une propriété observable au niveau client ou véhicule, or la notion de système correspond culturellement à un sous-système du véhicule à Renault. Ils permettent donc de traiter la sécurité fonctionnelle au niveau d'abstraction nécessaire. L'analyse du risque se poursuit en analysant ces événements redoutés afin de quantifier qualitativement leur probabilité d'occurrence (E), leur contrôlabilité (C) et la sévérité de leur conséquence (S). L'évaluation du risque se fait automatiquement en suivant la norme ISO 26262 qui fait correspondre un ASIL (un niveau d'intégrité) pour chaque triplet (E,C,S). Ces activités doivent ensuite donner lieu à un concept de sécurité (ensemble d'exigences de sécurité). Pour ce faire, l'ASIL le plus contraignant évalué pour un événement redouté est assigné à cet événement et un objectif de sécurité (type d'exigence) est systématiquement défini comme la négation de chaque événement redouté client avec son ASIL. L'ASIL est également défini comme un attribut du système, des exigences, des fonctions et des composants. Pour finir, la décomposition des ASIL définie dans la norme ISO 26262 est également formalisée dans l'ontologie ce qui permet de réduire l'ASIL des exigences. Le reste de la norme est soumise à interprétation en fonction des spécificités de l'entreprise ce qui est fait dans la formalisation globale du domaine d'étude (l'ingénierie des systèmes et la sécurité fonctionnelle).

La formalisation globale du domaine se fait en intégrant les deux ontologies ingénierie des systèmes et sécurité fonctionnelle. En particulier nous avons vu que des concepts similaires pouvaient être conceptualisés différemment dans chaque ontologie. L'intégration doit donc permettre de s'assurer que ces deux conceptualisations ne sont pas incompatibles. Pour se faire, les deux ontologies sont importées dans une seule ontologie système et

sécurité et nous définissons les concepts des deux ontologies par rapport aux autres (par exemple, les classes équivalentes sont définis comme telles, ou certaines classes sont des sous classes de classes appartenant à l'autre ontologie, *etc.*). En particulier, nous avons clarifié les notions d'objectif de sécurité comme des exigences de parties prenantes, les exigences fonctionnelles de sécurité comme des exigences fonctionnelles système et les exigences techniques de sécurité comme des exigences non fonctionnelles sur les éléments du système. La vérification automatique de la cohérence de l'ontologie globale nous permet de vérifier que les deux précédentes ontologies ne contenaient pas de concepts incompatibles. Ces précisions formelles nous ont permis de définir comment l'ASIL de l'évaluation du risque pouvait être propagé sur tous les éléments de la conception en se basant sur la traçabilité descendante présentée dans la figure suivante.

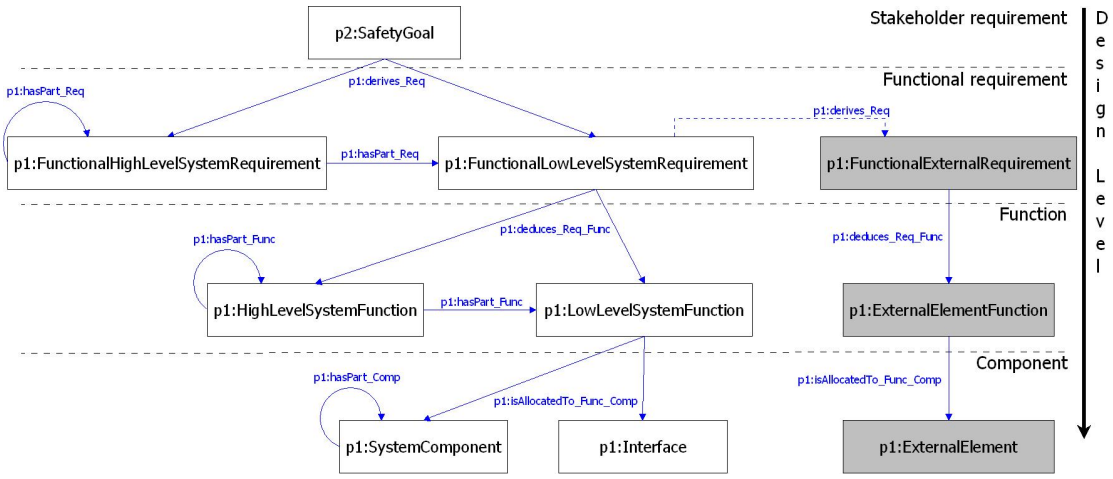

Figure 1: Traceability for functional safety

Cette propagation se fait semi-automatiquement, des objectifs de sécurité (et des événements redoutés) vers les exigences puis vers les fonctions et finit sur les composants en renseignant l'attribut ASIL. La particularité de cette propagation est qu'elle prend en compte la décomposition des ASIL faite sur les exigences fonctionnelles ce qui permet d'avoir des ASIL différents de l'ASIL du système sur les fonctions et des composants ayant des fonctions d'ASIL également différents. De ce fait il est possible très tôt dans la conception d'obtenir des contraintes en terme d'ASIL qui vont guider la réalisation des architectures précédant les ASIL plus concrets obtenus ultérieurement et également de détecter les dif-

férences entre ces ASIL et ceux obtenus par propagation come défini dans l'ontologie. Les synergies possibles entre plusieurs domaines peuvent donc être concrétisées par l'intégration de domaines comme le révèle cette application de propagation d'ASIL et justifie en partie l'effort à fournir pour plus de formalisation.

Les implémentations précédentes du processus de conception système ont révélé des difficultés concernant la production des différents objets du processus basée sur des transformations implicites des informations contenues dans les documents transmis à chaque étape qui se ramène à un manque de cohérence sémantique entre ces objets. Notre réponse à l'amélioration du processus de conception tente donc de résoudre ce problème et se concrétise à travers les ontologies précédentes. Nous proposons une nouvelle approche de conception basée sur une ontologie. Cette solution place l'ontologie système et sécurité au centre du processus de conception. L'ontologie est considérée comme LE modèle de référence global du système auquel tous les documents et modèles de conception doivent être conformes afin d'assurer la cohérence de l'ensemble du processus de conception. Pour se faire, nous nous replaçons dans le cadre de MOF afin de définir des correspondances (mappings) entre l'ontologie et chaque méta-modèle des documents (méta-modèle implicite) et modèles du processus de conception. Ces correspondances permettent d'enrichir l'ontologie au niveau instance, de réutiliser les connaissances (instances) contenues dans l'ontologie dans les modèles (et documents), de vérifier la cohérence (syntaxique ET sémantique) d'un modèle (ou document) en vérifiant que l'ontologie est cohérente une fois que les informations du modèle ont été enregistrées dans l'ontologie, de vérifier que deux modèles (ou documents) sont cohérents en vérifiant que l'ontologie est cohérente après que les informations des modèles ont été enregistrées dans l'ontologie et par extension de vérifier que tous les modèles sont cohérents et donc que l'ensemble du processus est cohérent. Un point important concerne la nature hétérogène des modèles du processus de conception due à la collaboration de divers acteurs ayant des conceptualisations différentes, notamment quand des acteurs externes à Renault sont impliqués. Des transformations (ou interprétations) sont faites implicitement ce qui peut mener à des incohérences. Nous identifions ces transformations comme des transformations dans MOF afin de montrer que les correspondances définies entre deux méta-modèles permettent de vérifier automatiquement la cohérence de

la transformation à travers la cohérence de l'ontologie.

Le chapitre poursuit sur la présentation du processus de conception basé sur une ontologie. Nous prévoyons que notre approche améliore le processus actuel de plusieurs façons. Premièrement, l'écriture des exigences et l'établissement de leur traçabilité se fait avec l'interprétation sémantique définie dans l'ontologie ce qui devrait réduire la distance entre le monde informel du discours et le monde plus formel des modèles. Deuxièmement, rendre les connaissances implicites explicites permet de commencer les activités au plus tôt, tout en laissant aux ingénieurs la capacité de faire des choix justifiés, et facilite l'identification des possibilités d'automatisation en exploitant les principes de la conception à base de modèles. De manière plus générale, aller vers plus de formalisation (au niveau sémantique) et le partage d'une conceptualisation formalisée facilite la réutilisation. Troisièmement, se conformer à l'ontologie permet de travailler au niveau interdisciplinaire, la justification étant que le système est bien le point commun de différentes professions. Par exemple l'analyse de la cohérence multi-modèles permet de vérifier que c'est bien le même système qui est en train d'être développé. Dernièrement, les concepts formalisés de l'ISO 26262 participent à la démonstration de la conformité au standard. Dans ce processus de conception, nous représentons les rôles d'ingénieur système (pour l'ingénierie des systèmes) et d'ingénieur sécurité fonctionnelle. Les particularités de l'approche sont d'une part une plus grande collaboration entre les ingénieurs système et sécurité fonctionnelle et d'autre part l'ajout d'un ingénieur ontologique garant de la cohérence de la conception. L'ingénieur sécurité fonctionnelle exprime des exigences (objectifs de sécurité) considérées comme des exigences de parties prenantes vers l'ingénieur système. L'ingénieur sécurité fonctionnelle est également responsable de la déclinaison de ces objectifs de sécurité en exigences de sécurité fonctionnelles puis techniques. Les objectifs de sécurité sont pris en compte de la même manière que les autres exigences de parties prenantes par le processus de conception (dont la responsabilité incombe à l'ingénieur système), cependant, afin d'assurer leur prise en compte efficace et éviter les itérations, il convient que l'ingénieur sécurité fonctionnelle fasse part de son expertise à l'ingénieur système de manière plus active de façon collaborative plutôt qu'en parallèle. L'ingénieur ontologique s'interface quant à lui à toutes les étapes du processus pour récupérer les informations produites et les rentrer dans

l'ontologie. Ceci permet de vérifier la cohérence de ces informations (même par rapport aux informations des autres domaines), d'effectuer d'autres analyses possibles seulement due à l'intégration de plusieurs domaines (par exemple la propagation des ASIL) et ainsi de vérifier s'il est normal qu'une incohérence ait été détectée.

Chapitre 3. L'objectif du chapitre 3 est de valider l'approche proposée en l'appliquant sur un cas d'étude. Dans l'automobile, le freinage et la direction ont été les dernières fonctionnalités véhicule à rester purement mécaniques en raison de fortes réglementations visant à réduire les accidents de la route, et ce malgré l'avènement de la mécatronique qui répond aux attentes d'innovations en intégrant des métiers qui étaient historiquement traités séparément. Nous avons sélectionné un système de freinage régénératif hybride intégré dans un véhicule complètement électrique. Ce système est bien entendu un système mécatronique critique, systèmes visés par nos travaux. Le chapitre revient premièrement sur l'historique des projets de freinage électrique à Renault. Posant les bases du freinage électrique, ces projets ont en particulier révélé que le développement de systèmes critiques nécessite le positionnement du constructeur automobile en développeur de système (au sens véhicule) plutôt qu'en intégrateur de solutions commerciales (sous-systèmes du véhicule). Puis est présenté le système de freinage régénératif hybride dont les principales spécificités sont un freinage régénératif exploitant le moteur électrique afin de régénérer de l'énergie électrique durant les phases de décélération et le choix architectural d'équiper les roues avants de freins électromécaniques et les roues arrières de freins à tambour. La solution est élégante. D'une part, la réglementation est clairement respectée avec trois types de freinage qui sont aussi précisément représentés que dans un système de freinage classique (le freinage d'urgence et le freinage résiduel sont pris en compte par les freins à tambour, et le freinage de service est assuré par les freins électriques, les freins à tambours et le freinage électrique). D'autre part, le freinage fourni par les freins à tambour simplifie l'analyse du système (il n'est par exemple plus nécessaire de simuler la résistance de la pédale de freinage, résistance fournie par le circuit hydraulique des freins à tambour).

Le chapitre 3 poursuit sur l'application de notre approche. Toutes les étapes de conception sont présentées et nous observons les résultats qui suivent. Premièrement, l'intégration

de l'ingénierie des systèmes et de la sécurité fonctionnelle (dans l'ontologie) donne une définition précise unique de notions similaires préalablement incompatibles. Les informations contenues dans l'ontologie ne sont pas sujettes à des interprétations implicites, donnent une description du système conforme à l'ontologie, sont de meilleures qualités et de ce fait sont potentiellement plus à même à être réutilisées. Deuxièmement, tous les concepts au niveau système de la norme ISO 26262 (concernant la conception) ont été définis et intégrés avec les concepts de Renault. Cette intégration fournit les bases du processus de conception basé sur une ontologie, ce dernier étant un processus applicable à Renault (qui tient en compte des spécificités de l'entreprise) conforme à l'ISO 26262. Pour finir, le processus de conception basé sur une ontologie répond aux problèmes de cohérence résultant principalement de la perte d'information aux interfaces des processus (de branches parallèles de développement), cette cohérence pouvant maintenant être vérifiée par un ingénieur ontologique.

Nous tirons les conclusions suivantes du chapitre 3. Le processus de conception basé sur une ontologie et cette ontologie centrale au processus sont adaptés aux spécificités de Renault et de la norme ISO 26262. Les informations (disponibles) produites par le projet freinage régénératif hybride ont été enregistrées sans difficulté quand elles étaient précises, les difficultés révélant un manque d'information ou des différences de conceptualisation. L'application du processus de conception démontre concrètement son alignement quand exécuté et est un exemple de capitalisation de projet encore plus favorable à la réutilisation que des documents et modèles (intégrés seulement implicitement). Au final, cette application permet de valider notre approche. Cette dernière améliore le processus de développement de Renault en apportant la rigueur nécessaire au développement de systèmes critiques et définit tous les principes permettant d'améliorer la qualité et la cohérence des modèles à travers une ontologie.

Nos travaux ont investigué les questions de recherches énoncées plus haut et ont apporté les réponses suivantes. Les ontologies permettent de formaliser une conceptualisation d'un domaine de la façon la plus précise et répondent naturellement aux questions Q1 et Q2 traitant de la formalisation de connaissance. La vérification de la propriété de cohérence définie dans les langages formels est une analyse automatique centrale à ces travaux. Ayant

formalisé la norme ISO 26262 dans une ontologie formelle, la question Q3 (vérifier le respect de la norme ISO 26262) trouve une réponse automatique en vérifiant la cohérence de l'ontologie. Au final, les techniques formelles (basées sur des langages formels) permettent d'améliorer le processus de conception des systèmes mécatroniques critiques. Cette thèse a permit d'enrichir la réflexion à Renault comme le montre les travaux futurs envisagés (tels que la formalisation des exigences non fonctionnelles, l'intégrations d'autres domaines ou encore le développement d'outils permettant la synchronisation du processus de conception) ainsi que les nombreux travaux connexes (apport d'une sémantique formelle aux futurs profils SysML et UML Renault développés dans le cadre du projet ingénierie des systèmes et ingénierie logicielle à base de modèles) qui peuvent tous être intégrés en utilisant les bases théoriques établies par nos travaux.

Contents

List of Tables

List of Figures

General Introduction

Industrial context

For the last several years, car manufacturers have had to face an always-increasing list of stakes and challenges. In the strongly competitive worldwide market of today, a car manufacturer has to offer to its customers relevant, innovative, reliable, environment-friendly and safe services of the highest quality. All this must be done at very competitive costs while complying with more and more stringent regulations and tighter deadlines [Chalé Góngora et al. 2009].

We have witnessed a change in attitudes vis-à-vis the automobile product. The automotive industry is intended for mass production to the contrary of other transportation industries. The customers are numerous and different by nature and they have heterogeneous needs. Customer needs drive the development process and serve ultimately to validate the final product. Over the years, customers needs have evolved and new needs are continuously identified. As for now, a vehicle is not anymore only limited to its functional role of transportation but has to propose non-functional services as well (*e.g.*, driving pleasure). In order to succeed (*i.e.*, to bring in revenue), the final product has to either have unanimous adoption or be highly customizable thus answering the heterogeneous customers needs. Unanimous adoption is impossible to obtain considering contradicting requirements (*e.g.*, the color of a vehicle) and customization is therefore adopted with optional services resulting in the corresponding product alternatives explosion. A status quo can be phrased: *car manufacturer survival lies on adaptability.* The product has to correspond to the ever changing customers needs. As such, car manufacturers strive for *innovations* [Bishop 2008] that are optionally integrable and that will answer customers

needs or, going further, that will answer the innovation demanding market, creating new needs that competitors will also have to address. These goals can be envisioned in the Renault brand signature unveiled during the 2009 Frankfurt auto show: *Drive the change.*

The advent of *mechatronics* has dramatically changed the automotive landscape. Compared to traditional mechanical systems, mechatronics systems tightly integrate multiple engineering fields *i.e.*, mainly but not limited to, mechanical, electronic and computer science. Mechatronics makes possible new solutions and opens possibilities for new, as yet unknown products [Tudorache 2006], tackling innovation issues. In the automotive industry, the mechatronics paradigm adoption comes from the quick electronic revolution in miniaturization and cost reduction. These enabled rapid growth of electronic allotment in a vehicle, that adds up on average to more than 40% of the vehicle total price in 2010. Electronics tour de force is termed as the increase of possible services which come without increasing material cost, thus answering customers search of high-tech features at relatively low cost. Software illustrates the best this state of affairs as one only has to develop another piece of software to offer a new functionality. This addresses the benefit / cost ratio issue of new developments as new solutions are created from new definitions of heterogeneous, *i.e.*, cross-domain, components interactions. 90% of automotive innovations come from electronics out of which 80% is actually implemented in software, thus demonstrating the key role of mechatronics in the current automobile product. For instance, the Anti Blocking System (ABS) unblocks a wheel blocked during braking preserving the driver with steering capability. Electronic Stability Program (ESP) interprets and computes driver intended trajectory that is compared to vehicle trajectory; in case of deviation the ESP takes action on the wheels rotation speed in order to follow computed trajectory. With power steering one can steer the wheels effortlessly while at full stop. More recent and one example that illustrates completely the benefits of mechatronics systems, the Renault Active Drive System (ADS) exploits the previous mechatronics systems (*i.e.*, ABS, ESP and power steering) sensors and introduce only one Electronic Controller Unit (ECU), actuators on the rear wheels and control laws. This system allocates the steering function to the four wheels improving greatly steering capability and vehicle flat turns at high speed by the compensation of the centrifugal force for exceptional driving pleasure; that would

not have been possible otherwise. Extensive use of mechatronics and software solutions is therefore the current trend in automotive industry and often the only solution to meet the concurrent challenges of *competitive costs*, *time to market* and *overall quality*. This trend, however, increases *system complexity* and consequently increases the *risks* due to systematic (software process) and random (hardware) failures. These risks are of even more serious consequences when we deal with safety-critical systems.

The emergence of the international standard IS ISO 26262 automotive standard (published in November 2011) which deals with the functional safety of embedded Electric/Electronic systems (E/E systems) within road vehicles, brings along new requirements and constraints with which the systems as well as the processes allowing their development have to comply. Although ISO 26262 is concerned with E/E systems, it provides a framework within which safety-related systems based on other technologies can be considered. This standard is undoubtedly acting as a catalyst for the research of new processes, methods and tools to cope with these new requirements as it will establish a state of the art of the requisites to guarantee functional safety. Even though ISO 26262 is not (yet) mandatory, in case of an incident or accident, a product manufacturer is responsible before the law to prove that its product is not the origin of the accident. This let us foretell automotive dependability culture evolution towards this standard.

The growing complexity of automotive mechatronics systems comes mainly from the integration of more and more elements that are heterogeneous in nature (*e.g.*, software, mechanical, electric, electronic) but have to work altogether in order to perform functionalities that would not be possible without the close cooperation of different fields that were historically treated separately. This is perfectly illustrated with the severance from conventional automobiles to electric vehicles where electric components skyrocket while mechanical and hydraulic systems may still be present. The consequence of the growing complexity of automotive systems is that facing the always-increasing list of stakes and challenges has become very complicated, thus, time-consuming and expensive given the time scales of typical vehicle systems development cycles.

Motivation

From this context we draw the conclusion that new *development processes* supported by adequate *methods* and *tools* and *new methods for analyzing systems* are necessary.

One of the current challenges at Renault consists in preparing its engineering divisions so that they are capable of developing mechatronic safety-critical systems according to ISO 26262 standard. This standard defines a system life cycle and the activities that must be performed in the different phases of this life cycle along with the support processes that are necessary for these activities. It also defines a specific method for automotive hazard analysis that identifies hazards and classifies them using ASIL, that stands for Automotive Safety Integrity Levels. The result of this analysis is the definition of the ASIL of the hazards of the system called Safety Goals. Safety goals are allocated from the system level to its components according to the rules defined by the standard. This leads to the definition of specific safety requirements on the system, on its components and on the associated development processes, depending on the ASIL quotation. The satisfaction of these requirements allows asserting the absence of unacceptable residual risks.

Therefore, the standard raises some problems concerning the demonstration of functional safety and, more generally, concerning the development processes which are currently under-formalized. Indeed, one of the strengths of ISO 26262 is that each requirement in the standard is associated to an ASIL. So, the compliance of the system, of its components (whatever their nature), and of their development processes to the standard can be obtained and verified in a systematic way. This suggests that better formalization can be beneficial to ensure consistency with respect to the standard.

Research Question

In this chapter, we presented the automotive industrial context and identified the peculiar features of automobile systems. The conclusion is the necessity to adapt and improve their development process. We identified two areas for improvement: one, better control over the complexity of automobile mechatronics systems and, two, improvement, in terms of power and sophistication, of the methods of analysis for automobile mechatronics

systems.

The areas of research revolves around the following fundamental question:

> *How can formal techniques contribute, in the continuity or in severance, to the development process of safety-critical mechatronics systems, that have to be compliant to ISO 26262, for a customer-oriented automobile manufacturer such as Renault ?*

In this work, we focus on formalization techniques as the whole development process will benefit greatly by using these techniques. By formalization techniques we are to understand the formalization of knowledge, *i.e.*, domain knowledge and expert knowledge, and the more sophisticated manipulation of this structured information. These two aspects are the subject of more targeted questions:

Q1 How can domain knowledge (*i.e.*, automobile domain knowledge) be formalized ?

Q2 How can expert knowledge about the development process be formalized ?

Q3 How can conformance to standard ISO 26262 be verified ?

Organization of the Thesis

In this work, we chose to focus on the design part of the development process where the system fundamental elements are manipulated. In addition, the automotive industry is confronted to the recently published international standard ISO 26262 on functional safety which adds other activities to the design process. In the end, this work contributes to the improvement of the design process for automotive safety critical mechatronics systems in three ways:

1. By integrating formalization techniques for desired enrichments that are difficult or even impossible with currently used techniques

2. By integrating safety concepts in the current systems engineering metamodel

3. By making a highly formalized attempt in the conformance to ISO 26262 standard

Our proposal is a design process approach similar to Model Driven Engineering. The novelty is the addition of a central ontology that formalizes a shared conceptualization of systems engineering and functional safety. Using ontologies enables to tackle semantic problems intrinsic to usual approaches. The ontology is used as the reference model for the whole design process. It enables to guarantee the design process consistency not only at the syntactic level but also at the semantic level for any environment. It is a general theoretical approach that is applicable in any company and that has been tested at Renault.

As such, the thesis is articulated in the following manner: Chapter 1 presents the state of the art. It goes through systems engineering and functional safety domains; Renault design process and attempts for improvement; the main approaches used to formalize the conceptualization in systems engineering; and ontologies as the most precise way to formalize a conceptualization. Chapter 2 regroups our contributions: the formalization of a conceptualization for systems engineering and functional safety, the central place of this formalization in the design process of safety critical mechatronics systems, and a proposal for a more precise design process named Ontology Centric Design Approach for Safety Critical Automotive Mechatronics Systems. Chapter 3 introduces the case study of a Regenerative Combi-Brake (RCB) system on which the design process is applied. Finally, we give a conclusion to the thesis with a recapitulation on the contents and the future works that are both undergoing and envisioned in reflection to this work.

Chapter 1

State of the Art

Sommaire

1.1 Introduction

This chapter presents the state of the art relative to our final objective: the improvement of the design process of safety critical mechatronics systems at Renault. It is organized in four sections. Section 1.2 presents the product development process. We present the general domain of systems engineering that addresses the development of any system and the dependability domain which specializes on safety critical systems. The main development approaches are brought into the perspective with the conclusion that they are based on particular conceptualizations. Section 1.3 presents the ontology paradigm. The fundamentals are explained and then we focus on the semantic web that is a growing field of activity. Finally, section 1.4 concludes the state of the art on our relatively new approach to implement semantic web technologies in the design process of safety critical mechatronics systems at Renault.

1.2 The Product Development Process

Over the years, the automobile product has turned more complex. The general introduction of this work presents the current context of the automobile product and shows that complexification is not only a trend but will amplify [Green et al. 2001]. To face this growing complexity adequate processes, methods and tools need to be defined.

It is the domain of the science of systems engineering to develop a framework for organizing and conducting complex programs. Systems engineering is the solution for Renault to position itself as a system developer. Systems engineering puts great emphasis on risk management. The International Council On Systems Engineering (INCOSE) identifies two main branches of risk management: the Project Risk Management (PRM) and the Environmental Risk Management (ERM) [INC 2011]. When dealing with safety critical systems we are interested in the ERM branch that incorporates safety.

Section 1.2.1 presents the general systems engineering point of view that addresses the development of any system. The development phase of the product development process is the main point of focus. Section 1.2.2 presents the dependability domain. It is part of systems engineering and it corresponds to additional processes, methods and tools that

specifically deals with systems that are safety critical. Section 1.2.3 briefly present the main development approaches.

1.2.1 Systems Engineering for Mechatronics Systems

"ISO/IEC 15288 establishes a common framework for describing the life cycle of systems created by humans" [ISO 2002a]. This international standard on systems engineering defines six different phases that every system goes through. It starts with the conceptualization of a need for the system and progresses through development, realization, utilization and retirement of the system.

As we said, we only focus on the development phase of the life cycle. We begin by presenting the different elements of systems engineering that we manipulate. The following definitions come from ISO/IEC 15288 [ISO 2002a], the INCOSE [INC 2011] and its french chapter the *"Association Française d'Ingénierie Système"* (AFIS) .

1.2.1.1 Elements of Systems Engineering

The central element of systems engineering is the system.

> System: *An integrated set of elements that accomplish a defined objective. These elements include products (hardware, software, firmware), processes, people, information, techniques, facilities, services, and other support elements.*

More precisely, this integrated set of elements are in interaction and are organized to accomplish a defined objective that would be impossible without one of these elements. A system is easily understood in the light of the idiom *"the whole is greater than the sum of its parts"*. In practice, systems are products or services. The discipline developed around the central element, *i.e.*, the system, is systems engineering.

> Systems engineering: *An interdisciplinary approach and means to enable the realization of successful systems.*

To the approach, we add the collaborative characteristic that is very important (see section 1.3). The discipline can deal with many kinds of systems and in particular with the

automobile mechatronics systems. To do so, three elements are identified. They are the object of figure 1.1.

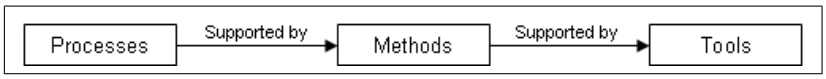

Figure 1.1: Elements of systems engineering

First, the processes.

> Process: *A set of interactive activities that are coordinated to progressively transform entry elements into output elements.*

A process defines the activities that need to be performed, input elements prerequisite to the realization of activities and output elements that are produced by activities. An activity is a structuring element of a process. It is time and resource consuming thus the necessary coordination to achieve good performance and contribute to the overall success of the process. A process is supported by different methods.

> Method: *A set of techniques that are coordinated to progressively realize an activity.*

An activity can be realized by different methods. A method is by itself a set of different techniques that need to be used to contribute to the overall success of an activity. A method is supported by different tools.

> Tool: *Anything used as a mean to help in the implementation of a method.*

In the context of complex systems, the usual example would be the computer tool; however it really can be anything. Let's have a look at the following example to understand how all of it articulates. As a process, we use the risk management process. One activity is the preliminary risk analysis. This activity takes as an entry a specification (set of requirements) and outputs identified hazards and their associated criticality. As this activity is programmed in the development process earliest phases, one technique that can be used is brainstorming. Brainstorming can be supported by different tools such as a blackboard,

post-it, computer spreadsheets, *etc.* In order to perform systems engineering, processes, methods and tools need to be defined.

The V Life Cycle. The usual representation of the system development process is the V-model. This model is the graphical representation of the development life cycle and is largely used in the automotive domain. Figure 1.2 presents a simplified V-model for the development process.

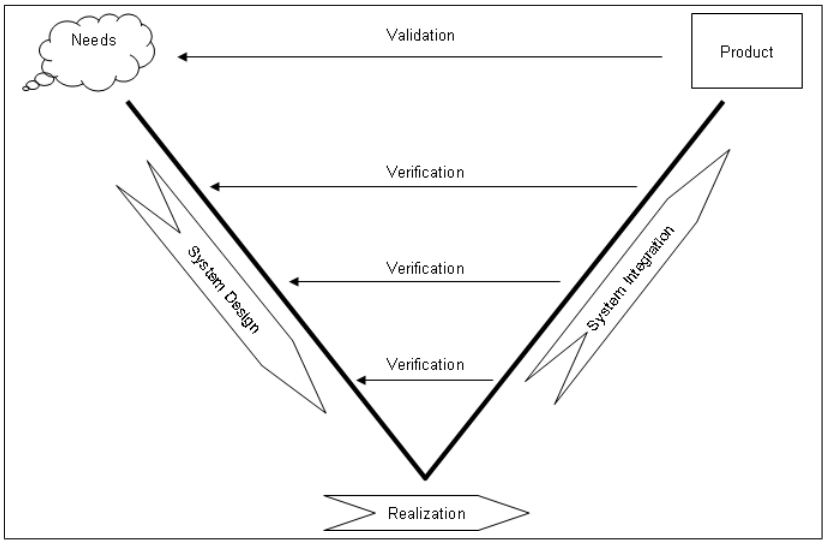

Figure 1.2: V-model for the development process

The development process is iterative by nature. It goes from system level through subsystems levels to components level (*i.e.*, understanding of the system is constructed with more and more details from the most general to the finest grained precision). In figure 1.2, only the development process at system level is presented. The V-model process is a top-down approach that implements fallback. It requires precedent steps to be carried out before going to the next step. The V-model reveals three main phases in the development life cycle. The phases are represented by arrow banners in the figure. The V-model starting point is the *Needs* (of all the stakeholders). From the needs, the first phase that corresponds to the descending branch of the V is carried out to design and analyze the system. The next phase at the base of the V is the realization phase. The last phase corresponds to the ascending branch of the V. It is the integration phase and relate to verification and

validation aspects. When carried out, it results a product that is also the ending point of the V-model. The arrows that go from the system integration phase towards the system design phase correspond to one important aspect of verification and validation: if something is not right in the ascending branch then something at the same level in the descending branch is not right therefore a fallback to the descending branch is undergone.

To reduce these errors and the cost of these errors, the science of requirements engineering can enforce necessary completion of a step before advancing to the next step and identify an error at an earlier time than the integration phase.

Requirements Engineering. The most important element of the science of requirements engineering is the requirement.

> Requirement: *Characteristics that identify the accomplishment levels needed to achieve specific objectives for a given set of conditions. Contractually binding technical requirements are stated in approved specifications.*

Actually it is the most important element in the development process. In figure 1.2, the process starting point are the needs of the system's stakeholders. Those needs are often not clearly defined, they can be constrained by factors outside the control of the stakeholders, or they may be influenced by other goals which themselves change in the course of time. Requirements engineering is the process that initially transforms stakeholders needs into a form that is suitable (mostly, better structured natural language) for both the development and the communication between developers and stakeholders: the requirements. The process continues throughout the V-model in a top-down manner, the requirements becoming more and more accurate and corresponding to specific system descriptions.

Fig 1.3 describes requirements engineering on different levels.

Good practice in requirements engineering calls for the distinction between the problem domain and the solution domain. In the problem domain, the stakeholders needs are elaborated. In the solution domain, it is the system that will be the solution to the problem that is elaborated. Figure 1.3 illustrates the development process and is read top-down. Once a sound set of stakeholders requirements has been agreed upon it is time to think of

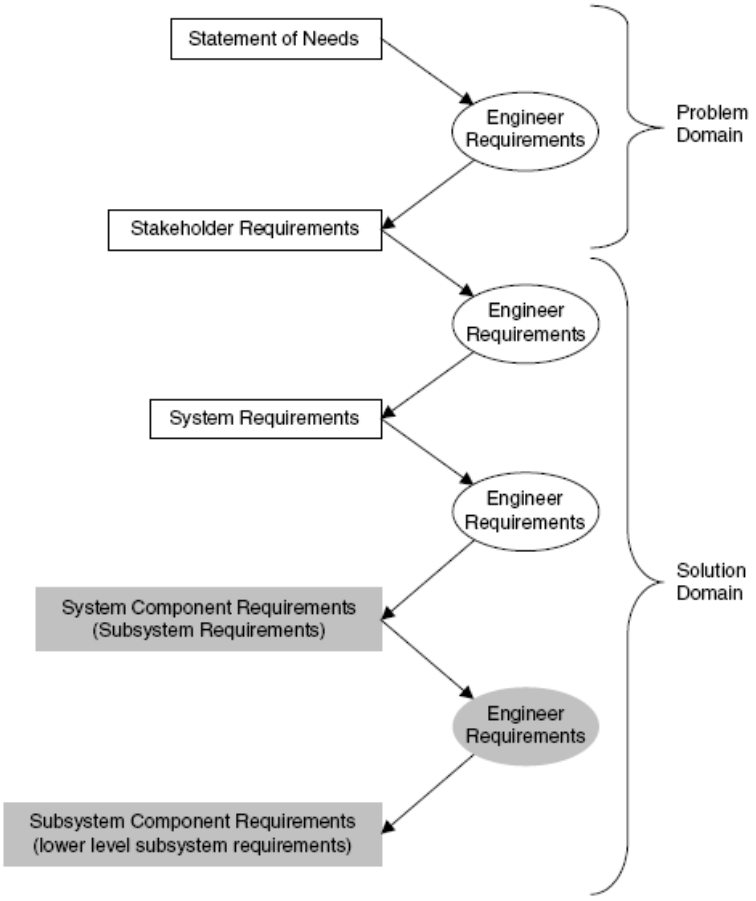

Figure 1.3: Different abstraction levels in requirements engineering Hull et al. [2004]

the solution. The system requirements defines system characteristics and are at a level that wants to be general in the sense that a general solution is described that still leaves place for creativity in order to consider different solutions (as a way to avoid jumping into the development of a solution that will get nowhere, whereas putting some effort into comparing different solutions could have revealed some flaws). From the system requirements, a design architecture (*i.e.*, one solution) is constructed as a set of interacting subsystems that exhibit properties that have to match the system requirements. This architecture defines the functional requirements of the subsystems. The same process is carried out as many time as necessary on the subsystems requirements to obtain subsystems design architectures until the component level is attained where an implementation can be realized. Requirements drive the project activity and are essential for project planning, verification

(checking that the system implements the specification), risk management (risks raised against requirements can be tracked, their impact assessed and the effects of mitigation and fallback understood) and change management. As a basis of every project, it is not surprising that project failure mainly comes from the requirements that can be incomplete, poorly expressed, poorly organized or changing too rapidly [Hull et al. 2004; Stevens et al. 1998].

In systems engineering, the support process *traceability* is essential as it enables to formalize an understanding of how objectives are met.

> Traceability: *The ability to trace (identify and measure) all the stages that led to a particular point in a process that consists of a chain of interrelated events.*

Establishing traceability on the requirements enables to formalize a certain understanding on how a system all fit together. This contributes for instance to assess the impact of change. For example, when changing one component of a system one can retrieve all the requirements that concern the component by tracing back and look at the impacted elements by tracing down from the requirements. In the development process, traceability helps in understanding the system; in the distance, it is one way to capitalize some knowledge about the system for evolution, maintenance or reuse. We develop on this last property as it is essential. The only realistic development approach, for both economical reason and given the complex nature of mechatronics systems, is one based on already existing implementations by reusing elements (*e.g.*, components, architectures, off-the-shelf components, *etc.*) of existing systems. Incremental modifications can then be performed to reach the final solution for the new system. Reuse should therefore be favored in the whole development process (*i.e.*, processes, methods and tools should favor and facilitate reuse).

Architectures Design. The other main elements that do not appear on the previous figures correspond to subprocesses of the design phase. In the design phase, requirements are made more and more precise. Specific artifacts that correspond to some requirements at different levels of abstraction are produced. These artifacts are in the solution domain

of figure 1.3. Two abstraction levels are considered. First, at functional level, the system's functional (or logical) architecture is designed. It is a solution of the system which is independent from implementation choices. This solution describes system's functions and how these functions interact with one another.

> Function: *A task, action, or activity that must be accomplished to achieve a desired outcome.*

Second, at component level, the system's physical architecture is designed to be one particular solution for the functional architecture. It describes how functional architecture's functions can be implemented into physical components and component's interfaces that enable components interaction.

> Component: *A modular part of a system that encapsulates its contents and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces. As such, a component serves as a type, whose conformance is defined by these provided and required interfaces (encompassing both their static as well as dynamic semantics).*

The general notion of flow is introduced as the mean to represent functions and components interconnection.

> Flow: *Non-broken circulation of information, energy or material.*

We do not give more detail on architectures design however it can be noted that these architectures are mainly defined using block diagram notation that uses blocks to represent functions or components and arrows to represent flows. Naturally, traceability is established between requirements and their corresponding architectural elements.

1.2.1.2 Model-Based Development

Mechatronics is an engineering science, in which the functionality of a technical system is realized by a tight interaction of mechanical, electronic and computer science engineering fields.

> Mechatronics: *The synergetic integration of mechanical engineering with electronics and intelligent computer control in the design and manufacturing of industrial products and processes* [Bishop 2008].

The automotive industry develops mechatronics systems. These are particular types of systems that emphasize even more the advantages of system parts collaboration and affirm that fields collaboration is essential. Developing mechatronics systems is complex and can be mastered using a model-based approach in the development process [Struss and Price 2004].

> Model: *Any representation of a function or process, be it mathematical, physical, or descriptive.*

In model-based development, the development effort is centered on a formalized system specification (*i.e.*, a model). Based on requirements engineering, the formalized specification is the set of system requirements. This specification is subject to many modeling activities that produce different models which represent different system views. In the context of automotive mechatronics systems it is particularly fitting as systems are complex and heterogeneous by nature. Systems engineering decomposes the system into smaller elements that are better understood. Model-based development encourages compositionality while providing specialized views that abstract unimportant details, therefore improving readability, focus or productivity on particular points.

1.2.1.3 Conclusion

We do not need to cover more material on fundamentals about systems engineering. The reader only have to retain that systems engineering defines system's descriptions in terms of requirements, functions, flows and components. If interested, the reader is redirected to the following literature. INC [2011] and Stevens et al. [1998] handle systems engineering. Hull et al. [2004] and Stevens et al. [1998] cover requirements engineering in a brilliant way. Wieringa [2004] discusses and makes clear problem and solution domain. Maiden et al. [2008] covers stakeholders requirements definition. Brinkkemper et al. [2008] tries to

improve requirements management. Bishop [2008] presents model-based development for mechatronics systems.

While systems engineering can bring best practices to the development process of a mechatronics system, it remains that the automobile product can involve some dangerous and fatal injuries where survival is compromised. As such it is classified as a safety critical system. Previously, we stressed out the importance of innovation for the automotive industry. This innovation comes mostly from the contribution of electronics, however this innovation brings along new risks. Systems engineering puts great emphasis on risk management. We are interested in safety aspects of risk management and this is the domain of the dependability field.

1.2.2 Dependability

> Dependability: *The ability of a system to deliver service that can justifiably be trusted.*
>
> Dependability: *The ability of a system to avoid service failure that are more frequent and more severe than is acceptable.*

Those two definitions [Avizienis et al. 2004] define dependability first by stressing the need for justification of trust in a service then by giving a criterion for deciding if a service is dependable. Dependability is defined by a set of properties, the main ones are the followings:

> Reliability: *Continuity of correct service.*
>
> Maintainability: *Ability to undergo modifications and repairs.*
>
> Availability: *Readiness for correct service.*
>
> Safety: *Absence of catastrophic consequences on the user(s) and the environment.*

The interested reader can consult [Avizienis et al. 2004] for the definitions relative to dependability domain. We are interested in safety aspects in the development process the objective being the proof of a safe system.

1.2.2.1 V-model and Safety Activities

As can be seen in figure 1.4, dependability studies are carried out in a second V cycle that is executed in parallel to the development process.

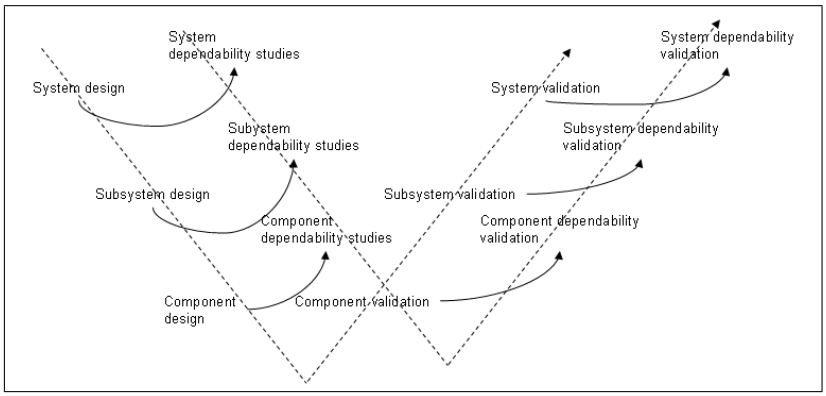

Figure 1.4: Safety in the V-model

This layout results from usual organization that commits dependability studies to a team that is independent of the design team. The safety team has to demonstrate that the system is dependable. Most often, the time lapse between design studies and dependability studies is very, not to say too much, important. For instance, if a problem appears in the dependability studies that are systematically based on design activities, the development team needs to go back on the impaired activities. We advocate that important improvement of the development process lies in a better coordination or combination of both design and dependability teams. This is shown in figure 1.4 as we placed the dependability V-model in parallel to the development process but somewhat earlier by placing it higher. In fact, dependability studies should be performed earlier than the design phase of the development process [Boulanger 2006].

Now that we understand the place of dependability studies in the development process, we define some fundamental elements of the safety field.

1.2.2.2 ISO/IEC Guide 51

The international standard ISO/IEC Guide 51 [ISO 1999] covers safety aspects for their inclusion in standards. We will present the notions of undesired consequences and come back on their causes. Then we will present risk management as a way to improve safety.

Fundamentals. From the notion of harm, it is possible to define risk, safety and all associated concepts such as hazards and hazardous events.

The ultimate unwanted outcome when using a vehicle is harm. This concept is shared in many domains (e.g., nuclear, transportation, and medical technology), and is applicable to every application domain of automobile. Harm is defined as:

> Harm: *Physical injury or damage to the health of people, or damage to property or the environment.*

Three attributes of harm are usually defined: *nature* of the harm, its *severity* and its *probability of occurrence*. In the case of an automobile in physical interaction with humans, the first potential harm generally considered is *damage to the health of people* due to vehicle crashes, but many other potential harms exist (electrocution, stressful conditions for users, *etc.*).

Severity is used to denote the degree of harm and is usually expressed as a qualitative level. Many severity indexes can be used (coming from other domains). These indexes express severity levels in terms of impact forces, kinetic energy, acceleration, speed, *etc.* table 1.1 is used in automotive applications and gives a generic severity scale [AAAM 1998].

A more controversial concept is the notion of probability of occurrence or likelihood of harm. Indeed, this metric was historically used for hardware systems where it is possible to evaluate failure rates (and then, the probability of occurrence of the failure and its outcome) based on test-bed results. Due to the growth of software, this quantitative evaluation is not yet possible for today's systems. It is also noted that some standards (e.g. in the

AIS	Severity	Type of injury
0	None	None
1	Minor	Superficial injury
2	Moderate	Recoverable
3	Serious	Possibly recoverable
4	Severe	Not fully recoverable without care
5	Critical	Not fully recoverable with care
6	Fatal	Not survivable

Table 1.1: The Abbreviated Injury Scale (AIS)

domain of medical devices) state that *"...a good qualitative description is preferable to a quantitative inaccuracy"* [ISO 2000]. Nevertheless, levels can sometimes be associated to different quantitative evaluation in order to give indications to designers. This choice is strongly related to the considered application, and no generic values exist. An example of levels is given in table 1.2. The levels graduate from high risk, intermediate risk, low risk and negligible risk.

Likelihood	Indicative probability (per year)
Frequent	> 1
Probable	$1 - 10^{-1}$
Occasional	$10^{-1} - 10^{-2}$
Rare	$10^{-2} - 10^{-6}$
Impossible	$> 10^{-6}$

Table 1.2: Indicative probability values for likelihood estimation

Risk of harm is defined as:

> Risk: *Combination of the probability of occurrence of harm and the severity of that harm.*

This definition is sometime extended with other metrics such as likelihood of exposure of the user to the system, but most studies define a risk level for each pair of qualitative levels (likelihood, severity). Table 1.3 is an example of possible risk values with a classic graduation.

The main objective of such a definition of risk levels is to identify risks deemed to be tolerable:

Likelihood	**Severity**						
	6 Fatal	5 Critical	4 Severe	3 Serious	2 Moderate	1 Minor	0 None
Frequent	H	H	H	H	H	I	N
Probable	H	H	H	H	I	L	N
Occasional	H	H	H	I	L	N	N
Rare	H	H	I	L	N	N	N
Improbable	I	I	L	N	N	N	N
Impossible	L	L	N	N	N	N	N

H: High, I: Intermediate, L: Low, N: Negligible

Table 1.3: Example of risk level table

Tolerable risk: *Risk which is accepted in a given context based on the current values of society.*

Based on table 1.3, the tolerable risk is defined (which is usually N, but in some cases L is also tolerable).

Safety, previously defined as an absolute property [Leveson 1995], is now also expressed in a relative and probabilistic way:

Safety: *Freedom from unacceptable risk.*

Safety is achieved by reducing risks to a tolerable level. Tolerable risk is determined by the search for an optimal balance between the ideal of absolute safety and factors such as benefit of the system to the user, suitability of the system for the purpose, and cost. These criteria and others have to be considered in the context of the values of the society concerned. It follows that there is a need to continually review the tolerable level as technological developments can lead to technically and economically feasible solutions to allow for safer vehicles.

Taking into account the notions of harm, risk and safety, we can now analyze the causes of harm which are the hazards. Historically, in many standards and studies, hazard was defined in terms of energy transfer. Today the notion is used to express any potential cause of harm.

Hazard: *Potential source of harm.*

A hazard can be a failure, a human error, a variable lighting condition (which is an adverse situation for a driver), *etc.* Often a hazard can indicate the origin or nature of harm when it gives information about the source (*e.g.*, electric shock hazard, crushing hazard, driver focus hazard, *etc.*). In many cases, an accident is the combination of the presence of a hazard and a situation where humans are exposed to this hazard. This concept is defined by the term hazardous situation:

Hazardous situation: *Circumstance in which people, property or the environment are exposed to one or more hazards.*

The term situation integrates the notion of scenario, *i.e.*, the description of environment conditions, system state, and actions performed during the scenario. A hazardous situation does not necessarily lead to an accident. Hence the concepts of harmful event and incident:

Harmful event: *Occurrence in which a hazardous situation results in harm.*

Incident: *Event that does not lead to harm, but which has the potential to create harm in other circumstances.*

From those two definitions the notion of event appears. From a system point of view that is broader than the safety one, the notion of feared event that includes but does not limit to harm is introduced:

Feared Event: *Event that must not occur or that must occur with low probability.*

In order for these events to not occur, or to hardly occur, one has to understand *how* they can occur. Dependability of a complex system can be undermined by three types of events: failures, errors and faults [Avizienis et al. 2004]. Elements of a system are subject to failures that can bring to a hazardous situation.

Failure: *... event that occurs when the delivered service deviates from correct service.*

A failure is an incorrect behavior of the system. The system does a *transition* from correct service to incorrect service. A representation of system states will therefore include incorrect states. A failure is an observable external event caused by an error.

> Error: *... the part of the total state of the system that may lead to its subsequent service failure.*

The cause of an error is called a fault.

> Fault: *The adjudged or hypothesized cause of an error.*

A fault is characterized as either dormant or active. In dormant state, a fault can possibly be never activated. For instance, there can be a fault in a function of a program that is never used. Comes the day when this faulty function is used. The fault generates an error. First the error is not apparent to the user of the service. When it becomes apparent the error is deemed as a failure. An example can be the Automatic Cruise Control (ACC) service that controls the vehicle to be at defined speed. The driver uses the ACC on the highway and arrives to his exit. He decelerates by braking that should deactivate the ACC but a fault has been missed out. The dormant fault activates and becomes an error as the ACC that should be deactivated is not. The driver wants to decelerate, but the ACC mitigates braking effectiveness. This error has become a failure. Figure 1.5 presents the fundamental chain that links the threats of dependability together [Avizienis et al. 2004].

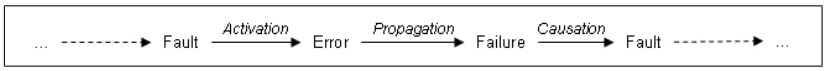

Figure 1.5: The fundamental chain of dependability and security threats

The role of the risk management process when considering safety aspects is to justify the system dependability. It is based on the elements presented before and therefore study the failures, their causes (*i.e.*, the hazards) and their consequences (*i.e.*, the risks).

Risk management. Risk management is the overall process analyzing hazards and their possible outcomes, and deciding which risk reduction strategies are selected.

> Risk management: *The process whereby organizations methodically address the risks attaching to their activities with the goal of achieving sustained benefit within each activity and across the portfolio of all activities* [ISO 2002b].

Risk management is a process to direct and control an organization with regard to risk. It generally includes risk assessment, risk treatment, risk acceptance and risk communication [ISO 2002b]. Figure 1.6 presents the part of the risk management process that considers safety [ISO 1999].

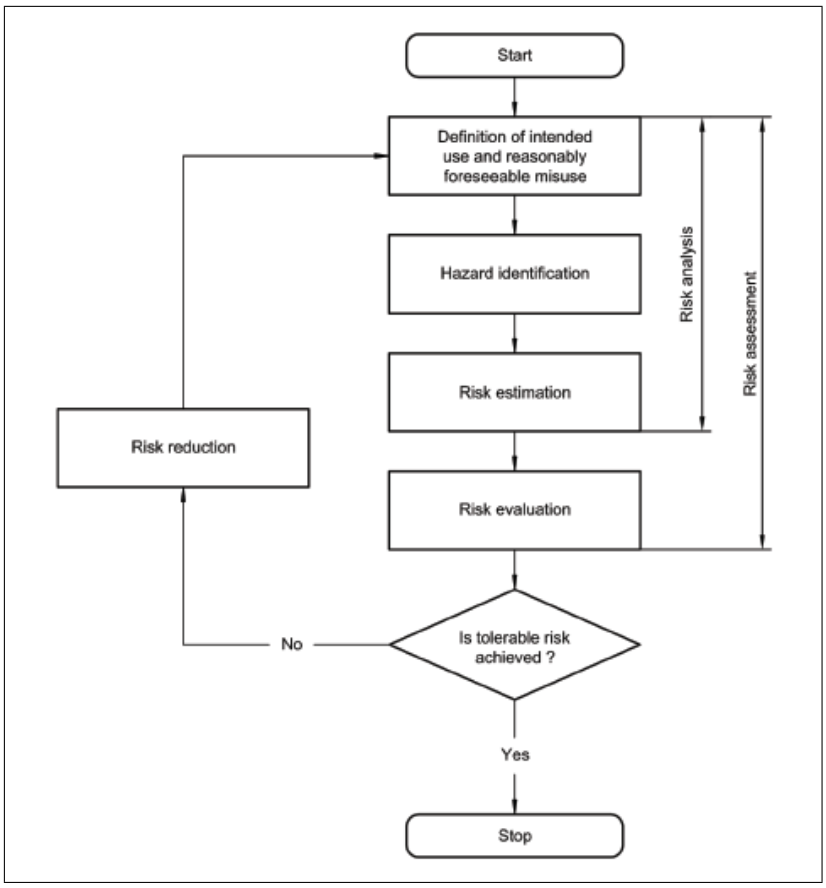

Figure 1.6: Iterative process of risk assessment and risk reduction

Risk analysis is a part of the overall process presented in figure 1.6, and is included in risk assessment. It is defined as the systematic use of information to identify hazards and to estimate the associated risk. The considered information may stem from historical data, theoretical analysis, informed opinions, and the concerns of all stakeholders (designers,

end users, regulatory authority, *etc.*). During risk analysis, various methods can be used to handle functional and technological issues, for example: HAZOP-like techniques (HAZard OPerability), Failure Modes, Effects, and Criticality Analysis (FMECA), and Fault Tree Analysis (FTA). These are the three main techniques, and have been widely used in many domains. They are also recommended in many standards on dependability. During the risk reduction step, actions are taken to reduce the probability and/or the negative consequences associated to a risk. The means to attain dependability can be grouped in four major categories:

- "Fault prevention" means to prevent the occurrence or introduction of faults.
- "Fault tolerance" means to avoid service failures in the presence of faults.
- "Fault removal" means to reduce the number and severity of faults.
- "Fault forecasting" means to estimate the present number, the future incidence, and the likely consequences of faults.

The establishment of dependability therefore requires the selection of suitable means that can justify this dependability. Once all identified risks on the project are controlled, the system can be stated dependable.

1.2.2.3 Standards and Norms

Renault is confronted to three different types of standards. First, the regulations. Regulations are constraints that have to be respected for a vehicle to be commercialized in a country. It is essential for a constructor to consider this type of standard. The next two types of standards are characterized by the external / internal criterion. A constructor is interested in external standards as they establish a state of the art on a specific subject. Keeping an eye out for new, more efficient technologies, processes, methods or tools that will benefit the industry is not only essential but common sense. An external standard can be normative but we class it under regulation type. Non normative standards are categorized under external standards. Finally, an internal standard affects all the actors inside the industry and even the partners, be they associates, suppliers or others.

Those internal standards implement regulations and external standards while at the same time making abstraction of inconsequential elements. Often, they are used for quality department that evaluates application of good practice. Those evaluations are part of the verification activities. We do not distinguish between standards and norms; the two terms are used indifferently throughout this report.

Even though consideration of an external standard is by no mean mandatory, when dealing with safety critical systems, if a catastrophic consequence (on a user of the system or the environment) that comes from any system element happens, this catastrophic event justifies by itself the need to take into account any standard that deals with the domain considered. The consequence is so enormous that if it can be avoided then it has to. In this section we present only the external standards that are in relation to the automotive industry.

IEC 61508. IEC 61508 is an international generic standard for the functional safety of programmable electrical, electronic and programmable electronic (E/E/PE) safety-related systems [IEC 2000]. System safety is achieved by reducing risks to a tolerable level. Tolerable risk is determined by the search for an optimal balance between the ideal of absolute safety and of factors such as: benefit of the system to the user, suitability of the system for the purpose, and cost. These criteria and others have to be considered in the context of the values of the society concerned. It follows that there is a need to continually review the tolerable level as technological developments can lead to technically and economically feasible solutions to allow for safer systems.

The standard defines a *safety function* as a function to be implemented by any technological means which is a risk reduction strategy. The standard IEC 61508 was originally developed for machine systems, where safety of equipment under control (like a machine with a sharp blade) is guaranteed by independent safety systems such as fences or interlocking devices. Such independent systems are called *safety-related systems* in the IEC 61508 standard. In the case of a vehicle it is also possible to apply these concepts because the vehicle itself can be considered as the equipment under control. In that case, safety related systems can be assigned *Safety Integrity Levels* (SIL) using the standard. Safety

integrity is defined as the probability of a safety-related system satisfactorily performing the required safety functions under all the stated conditions within a stated period of time.

In the case of a vehicle, some safety functions are fully integrated in the mechatronics system itself, and even deep within the vehicle control software. Even system functions such as decelerate the vehicle can be considered as a safety function. In fact, in the sense of the standard, every function can be considered as a safety function. By induction, this result in the whole system being considered as a safety-related system, therefore we lose the distinction between safety-related systems and the system. It is then hard in the case of a vehicle system to define independent safety-related systems in the sense of the standard, and it is thus difficult to assign safety integrity levels following the standards definitions.

The notion of SIL can however be interpreted as a level of confidence a user can have in a safety function. The higher the level of safety integrity of the safety-related systems, the lower the probability that the safety-related systems fail to carry out the required safety functions. The *safety integrity requirements specification* is then the specification containing the safety integrity requirements of the safety functions that have to be performed by the safety-related systems. Determination of this level depends on the application domain, and no prescriptive method is proposed in the 61508 standard. One example is given in figure 1.7, that is extracted from IEC 61508, Annex E, page 55 [IEC 2000], where only three severity levels and three levels of probability of occurrence are used .

This matrix illustrates the fact that for an event with a given severity and likelihood, the SIL of each risk reduction facility decreases when the number of facilities increases. For instance, for an event with serious severity and high likelihood, and in case of one safety-related system, the required SIL is SIL3. But if a second safety-related system is implemented, then they both are assigned a SIL2.

Following the SIL assignment, dependability means should be engaged in order to guarantee the required integrity of the safety-related system or safety function. For instance, you may consider using formal specification to do the specification of a SIL3 function. In the 61508 standard, many dependability means (particularly for software requirements) are listed and noted HR (high recommended), R (Recommended), or NR (not recommended)

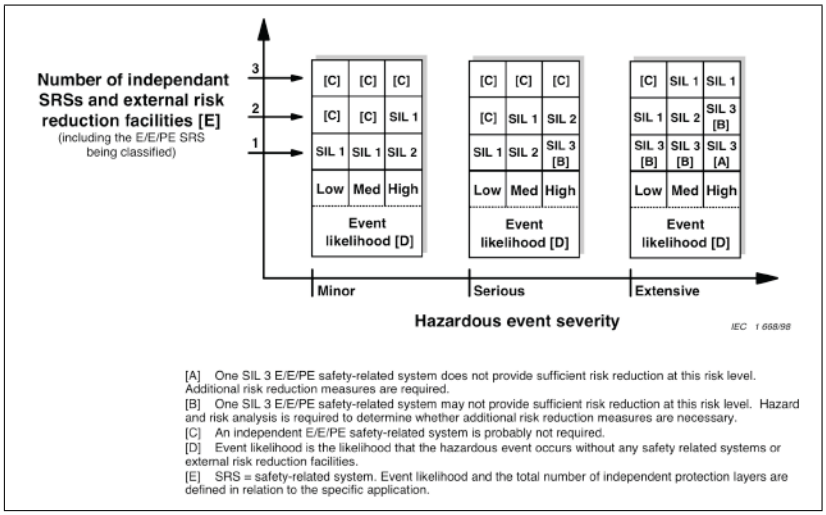

Figure 1.7: Hazardous event severity matrix: example (illustrates general principles only)

depending on the safety integrity level.

Its generic scope has helped IEC 61508 become a reference in all the main industrial sectors and has made it the object of numerous adaptations that take into account the specificities of these different sectors [McDermid 2001]. Those adaptations are based nevertheless on the same principles of the parent standard; they all assign safety integrity levels to the system that are used to specify how dependability should be demonstrated. IEC 61508 is the basis of IEC 61511 for industrial processes, IEC 61513 for the nuclear power sector, IEC 62061 for machines, EN 50126, 50128 and 50129 for the railroad sector and, finally, ISO 26262 for the automotive sector.

ISO 26262. ISO 26262 is the adaptation of IEC 61508 to comply with needs specific to the application sector of E/E systems within road vehicles. This adaptation applies to all activities during the safety life cycle of safety-related systems comprised of electrical, electronic and software elements that provide safety related functions [ISO 2011].

ISO 26262 has been published as an international standard in November 2011. It remains largely in compliance with IEC 61508 in its substance but diverges in its structure. One important evolution consists of the fact that the system main functionalities can be considered *a priori* as safety related functions; all system functionalities are analyzed in

order to determine whether they are safety related, *i.e.*, the functionalities are analyzed to determine if they have the potential to contribute to the violation of a *safety goal* which is a top level safety requirement.

Not surprisingly, we find in ISO 26262 the definition of safety integrity levels, which determine the activities to be performed according to each integrity level in order to justify an acceptable safety level of the system design. However, ISO 26262 ASIL (that stands for Automotive Safety Integrity Level) is now assigned to safety goals. The safety justification consists in the demonstration that the safety goals are satisfied. There are numerous adaptations in ISO 26262, concerning primarily the system life cycle, that deal with the specificities of the automotive domain.

ISO 26262 defines four ASILs: A, B, C and D. QM stands for Quality Management and denotes no safety requirement according to ISO 26262. These levels are determined by combining the following criteria: *severity*, *probability of exposure* and *controllability*. Severity is a qualitative measurement of the consequences of a car accident. Classes of severity S0, S1, S2 and S3 correspond respectively to "no injury", "light and moderate injuries", "severe injuries (survival likelihood)" and "dangerous and fatal injuries (survival compromised)". Probability of exposure is a qualitative measurement of the possibility of the user being in a situation where the occurrence of the accident is conceivable. Classes of probability of exposure E1, E2, E3 and E4 are separated from one another by one order of magnitude and correspond respectively to "very low probability", "low probability", "average probability" and "big probability". Finally, controllability is a qualitative measurement of the capability of the user to avoid a dangerous situation. This criterion is specific to the automotive domain where the user (the driver) can exercise a certain control on a permissive system (the vehicle does not inhibit unforeseen behaviors). Classes of controllability C0, C1, C2 and C3 correspond to "generally controllable", "simply controllable", "normally controllable" and "difficult to control or uncontrollable". These three criteria allow determining in a systematic way the ASIL of a system or of one of its features as shown in table 1.4 below. C0 and E0 class are not represented in this table as they always corresponds to QM regardless of the other two criteria values combined with them.

Severity	**Exposure**	**Controllability**		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Table 1.4: Automotive Safety Integrity Levels

Two other topics of the automotive domain are considered in ISO 26262: the human factor and the relationship between car manufacturers and their suppliers. As previously mentioned, the user can have unexpected or unwanted behaviors (e.g., crossing downtown at 100Mph). This type of risks are specific to the automotive domain and relatively non-existent in the nuclear power, aerospace or railways sectors where systems and procedures authorize only foreseen behaviors in precise contexts. However, the question of how to handle these risks still remains little approached. Concerning the relationship between car manufacturers and suppliers, ISO 26262 defines all the activities to be performed by both parties, but it does not define who should execute this or that activity. The share of responsibilities between the car manufacturer and its suppliers is thus left open; ISO 26262 imposes only to define this share of responsibilities at the beginning of the project.

One important element to note, which is a big strength of ISO 26262 compared to its predecessor, is that every normative part of the standard depends on the safety integrity levels. Hence, the compliance with the standard will be obtained and verified in a systematic way, contrary to IEC 61508 which could lead to different interpretations upon which parts of the standard to use for a given safety integrity level. In other words, ASIL leads to the specification of a necessary set of safety requirements, which, if satisfied, allow asserting the absence of unacceptable risks.

1.2.2.4 Conclusion

Functional safety has been presented as part of the product development process to address critical systems. The interested reader can refer to Avizienis et al. [2004] that is the reference on dependability. ISO [1999] and ISO [2002b] are guidelines to include risk management and safety aspects in standards. Among other, they define a vocabulary for safety. INC [2011] presents the risk management process in the systems engineering development process. Leveson [1995] presents system safety, *i.e.*, safety for systems engineering. Herrmann [1999], McDermid [2001] discuss standards on safety prior to ISO 26262. International standards IEC [2000] and ISO [2011] on functional safety for E/E/PE systems which concern the automobile industry. The emergence of ISO 26262 international standard in the automotive industry can be perceived either as a source of concern and apprehension, or as an opportunity to improve current systems engineering processes and working methods. Either way, this standard is undoubtedly acting as a catalyst for the research of new processes, methods and tools to cope with these new requirements. The new processes, methods and tools that address functional safety mainly innovate on safety studies that correspond to risk analysis step in the overall risk management process. In this thesis, we are literally only interested in the design activity. Therefore, the different analyses that bring information that results in informed design decisions are not supported and not presented. Similarly, design approaches that address functional safety are not relevant and the next section presents a state of the art on the different design approaches which mostly are inconsiderate of functional safety but nonetheless adapted for the design of any system.

1.2.3 Main Design Approaches

As a general definition, a design approach (or method or methodology) refers to a codified conceptualization which helps to systematically create a set of interrelated artifacts that ultimately lead to the sought after system. We will return in section 1.3 on the notion of conceptualization (for interrelated artifacts) which is the fundamental issue that concerns *all* existing approaches that we address in this work. For now this section identifies and briefly describes the main design approaches for systems engineering (that includes

functional safety design). Section 1.2.3.1 presents the main development process of mechatronics systems in the automotive industry. Section 1.2.3.2 briefly presents approaches based on the standards de facto UML and its adaptation to systems engineering, SysML. Finally, section 1.2.3.3 gives the conclusion on the design approaches and the automotive industry context.

1.2.3.1 Simulation Based Development Process

The V-model (see figure 1.2) is currently the standard *de facto* used in the automotive industry for the development of mechatronics systems. It has been adapted to detect design issues before the system integration phase and to automate some activities tackling issues such as short development time constraints and cost of error (detected too late). In this approach, models are introduced to answer the requirements beginning at architecture design phase. At this point, the remaining of the development process is model based. The architecture models are executable (the most widely used simulation tool for mechatronics systems that include software is Matlab/Simulink). Design verification is based on testing and is first done on architecture models before the realization phase of the V in a simulated environment (Model In the Loop or MIL). Some tests are generated automatically from models and the same tests are reused for different abstraction levels of the integration phase. Software code, executed on particular hardware platforms, can also be generated automatically. It is verified in a simulated environment (Software In the Loop or SIL) so that implementation on hardware and integration are initiated on solid basis. Finally, the simulated hardware is replaced with its physical realization and tested in simulation (Hardware In the Loop or HIL).

In this approach requirements management and traceability are done with additional tool support. These activities can be somewhat imprecise. These issues are considered in our approach and resolved at the semantic level (*i.e.*, we give meaning to tracing a requirement to a design element).

1.2.3.2 UML/SysML Based Approach

UML (Unified Modeling Language) [UML 2007], often wrongly considered as an approach, is a graphical notation considered as a semi-formal language which is the basis of many approaches. As UML is mostly better suited with software engineering, the general purpose language SysML (Systems Modeling Language) [Sys 2007] has been standardized to address systems engineering. It is specified as a *profile* (*i.e.*, a dialect) of UML and defines in particular requirements as a conceptual class of the language. SysML based approaches are of the same kinds as UML based ones. We identify three kinds of approaches. As the two standards position themselves as general purpose language, they should be viewed as the common frame of development best practices. Therefore they should be adapted depending on the domain. The first kind of approaches ignores this adaptation. SysML is left aside for these kind of approaches as it is by nature an adaptation to systems engineering. For instance, it is adapted for requirements engineering which is specifically addressed by our approach. Second kind of approaches uses extensibility mechanisms to better suit the specificities of a domain but leaves syntax and semantic informal. The last kind of approaches rise at a higher abstraction level to define their own language with its semantic that remains however informal.

Unified Process. Unified Process or UP [Booch et al. 2005] approaches (*e.g.*, Rational Unified Process or RUP) are often used in very large software projects. Characteristics of UP are iterative, incremental, architecture centric, use case driven and risk focused (any risk type including risk of harm). Concerning design, UP addresses three activities with respective models. The definition of the needs activity corresponds to the elaboration of requirements. Functional requirements are modeled with UML *use case diagrams*. Non functional requirements are not considered. Then, analysis of needs activity produces specifications assumed to provide a correct understanding of the needs. Here, the needs refer to use cases which are specified using UML *sequence* and *interaction overview* diagrams. Finally, design activity gives a thorough understanding on components (read architectural elements) interaction and their internal behavior with UML *class*, *activity* and state machine diagrams for instance.

As mentioned, even though UP approaches should be viewed as the common frame of development best practices and should be adapted depending on the domain, it is often viewed as a universal process and used as is. Here UP is presented as an example for approaches that are imprecise in the sense that important conceptual elements are not considered as such (for example, to the notion of need, one has to understand requirements or use cases). With UML, the requirements are not considered as a conceptual class and therefore requirements traceability is not supported.

OOSEM. OOSEM (Object-Oriented Systems Engineering Method) [Lykins et al. 2000] is promoted by the systems engineering community and is a reference for systems development. OOSEM is a modeling method based on UML. It implements UML extensibility mechanisms that differentiate with the previous kind of UML based approaches. UML extensibility mechanisms are used so that the modeling language can be made more precise, however, *with no additional syntax nor semantic*. OOSEM uses stereotyping (*i.e.*, an extensibility mechanism) of UML modeling elements to represent systems engineering concepts such as the system or logical components for instance. It has four core design activities: analyze needs, define system requirements, define logical architecture and synthetize candidate allocated architectures. Analyze needs results in use cases, scenario descriptions, static system model with external interacting systems. UML *use case*, *sequence* and *class* diagrams are used. The systems requirements are represented with a UML *class* diagram that represents the system as a black box with its inputs, outputs and external collaborators. Another class diagram is used to represent the hierarchy of *logical* components as an aggregation hierarchy of stereotyped classes. Finally, any means of UML can be used to represent the allocation or *realization* components to the logical ones. Even though OOSEM promotes requirements traceability, this is not supported by the formalism it uses, therefore it is supported externally by means of a Requirements and Verification Traceability database (RVT).

Here, we presented OOSEM as it was practiced before the introduction of SysML to illustrate extensibility mechanisms that enable to consider the logical components as a conceptual class for instance. SysML does so by default, however, the point is that by means

of extensibility mechanisms, other concepts can be made more precise in UML/SysML. For instance, mechatronics systems basic components are sensors, calculators and actuators, and it can be advantageous to define specific classes for these types of components. Extensibility mechanisms enable to adapt to the specificities of a particular domain however they are not sufficient to define a language with its syntax and semantic. The syntax and semantic are defined in a general manner for UML/SysML elements (of the notation). Defining a specific UML stereotyped class for sensors, one can want to use UML aggregation between sensors class only. The relation only makes sense in this case but as it is defined in a general manner, it is still possible to aggregate other classes into a sensor. The ability to define new languages with syntax and semantic is presented in the next paragraph. As a final word on OOSEM, it is actually a general approach which principles can be applied to any development process.

Model Driven Architecture. Model Driven Development (MDD) approaches introduces a higher level of abstraction by defining meta-models as first class entities. The idea behind MDD is to create different models of a system at different levels of abstraction and using transformations to produce the system implementation. A model-driven approach requires languages for the specification of models, the definition of transformations, and the description of meta-models. Concerning UML/SysML, the Object Management Group (OMG) proposes Model Driven Architecture (MDA) which uses UML for object oriented modeling, XML Metadata Interchange (XMI) for tools interoperability with documents and models represented in eXtensible Markup Language (XML), Meta-Object Facility (MOF) to define new modeling languages, Object Constraint Language (OCL) to establish rules about any MOF meta-model and Query, Views and Transformations (QVT) for transformation between models defined with a MOF meta-model [Anneke et al. 2003]. MDA suggests building Computational Independent Models (CIM), Platform Independent Models (PIM), and Platform Specific Models (PSM) corresponding respectively to a business, a design, and an implementation viewpoint. The applicability of MDA to systems engineering is currently investigated [INC 2009]. CIMs correspond to need analysis with UML *use case*, *sequence* and *activity* diagrams representing system goals and stakeholders re-

quirements. These models are (partially) transformed into system PIMs that address the derivation of stakeholders requirements into system requirements and the system architecture. Subsystems PIMs can also be developed. Finally, PSMs address specific capabilities.

Compared with the previous UML based approaches, the ability to define new languages and the constraints of such languages enables to completely support any design approach inside the MDA framework *i.e.*, documents and models need to have a representation in XML.

EDONA Method. EDONA ("Environnements de Développement Ouverts aux Normes de l'Automobile") [EDO] is an open platform that supports the development process of embedded software in the automotive industry including ISO 26262 safety critical software. Having identified abstraction capability issues to represent systems in which software is embedded, EDONA proposes a method articulated around, a requirement traceability management tool to ensure their consideration, EAST-ADL2 language for systems modeling [ATT 2008] and AUTOSAR [AUT] for software modeling. The method has first been defined by the MeMVaTEx ("Méthode de Modélisation pour la Validation et la Traçabilité des Exigences") project [MEM]. MeMVaTEx approach [Albinet et al. 2007] includes SysML *requirement* diagram. The requirements represented with *stereotyped classes* are defined in the earliest phases of the design process and traced all along the development using concepts of *refinement*, *composition*, *verification* and *satisfaction* defined in SysML meta-model. The system's architectures are defined using EAST-ADL profile (of UML / SysML) so that EAST-ADL requirements can effectively trace to any EAST-ADL element. The same approach is used in EDONA [Albinet et al. 2010] with EAST-ADL2 replacing its former version for architectures description. In practice, other languages can be used by development teams (*e.g.*, internal and OEM-suppliers teams) both for requirement management (and traceability) and architecture description. The approach addresses this issue with the definition of requirement traceability and analysis along the whole development life cycle. A requirement traceability management tool such as MKS INTEGRITY or TELELOGIC DOORS is used to extract and analyze the relations between requirements expressed in most industrial tools. It enables to manage requirement at the appropriate

granularity throughout the whole development life cycle.

This approach comes close to the approach presented in this thesis however it deals primarily with tools (or languages) interoperability and we address the more general (semantic) integration issue.

1.2.3.3 Conclusion

Some very general design approaches used and defined for the automotive industry have been presented. It is inconsequential to go further on the subject as we actually address an issue at a fundamental level that concerns any approach. Nonetheless, the interested reader can refer to Webers et al. [2008] who define a support process for requirements engineering in the automotive industry. Requirements engineering was treated with particular attention in our approach as the basis for design. Bishop [2008] and Gao et al. [2007] respectively present and discuss model-based development with specific automotive industry need for simulation in perspective. Finally, in [Chalé Góngora et al. 2009, 2010] we presented a model-based approach used at Renault that considers safety aspects relative to ISO 26262. In particular, we identified the *need for a common data model between systems engineering and functional safety* as a basis for automotive safety critical systems design.

The general methods presented in this section, except for EDONA / MeMVaTex, treat the requirements as a non conceptual element. It is implied that additional tool support are used for requirements management and traceability. In fact, it reveals that the product development process in the automotive industry is heterogeneous in the sense that different tools (with different languages) are used. While UML / SysML with OCL have the potential and ambition to place themselves as *the* general purpose language that represents a unique formalism for systems engineering, we have seen that in the automotive industry other formalisms such as Matlab/Simulink ones (*e.g.*, block diagrams) are actually used and need to be connected. This poses a problem of interoperability between tools. In particular, requirements management and traceability activity illustrates this problem of interoperability but the issue is even more general: given we have interoperability, do we inter-operate in an *integrated* way ?

1.2.4 Conclusion

In this section the product development process has been presented with the focus on the design process that includes functional safety. Some development methods have been presented and we concluded on the needs for a common data model for systems engineering and functional safety and the problem of (tool's semantic) integration.

The next section presents the ontology paradigm as the underlying discipline to formalize a conceptualization.

1.3 Formalization of a Conceptualization

Formalization of knowledge addresses the problematic of harnessing and reusing knowledge, a key point for all domains. In the engineering domain, decision making is fundamental as engineers are confronted with multiple choices and only one solution will be subject to complete development. In order to choose the best solution, the engineers need the relevant necessary knowledge required to perform informed decision making. Currently, content created in the development process is ultimately archived and forgotten, knowledge is lost whereas archived. There is evident need to make use of this archived knowledge considering probable occurrence of confrontation with an already solved problem. To confront this issue, the formalization activity consists in better structuring knowledge, enabling computer treatment for more performing information retrieval that has to bring relevant information to the surface. This can be implemented in an elegant fashion using the ontology paradigm.

1.3.1 The Ontology Paradigm

In computer science, the term ontology is used to denote a paradigm (*i.e.*, a coherent model to represent knowledge about a world). The following definition of ontology is given in [Gruber 1993]. It is a well accepted definition that will be used throughout this thesis.

> Ontology: *An ontology is a formal, explicit specification of a shared conceptualization.*

The definition is explained as follows: formal means that the ontology is machine readable; explicit means that concepts and how they are constrained is explicitly defined; shared indicates that the ontology captures consensual knowledge; conceptualization refers to an abstract, simplified model of concepts in the world. Ontologies are used in many domains (*e.g.*, systems engineering, artificial intelligence, *etc.*) as a representation of knowledge about a world. More precisely, an ontology captures, structures and defines a set of concepts of a domain along with the relationships between those concepts. Finally, an ontology is a *formal description*, *i.e.*, a description that has the correct form, shape or structure which entails clearness and preciseness.

Formalization (structuring) of domain knowledge enables to reason on a certain level about properties of a domain. What we retain is that an ontology can be used to define a domain and reason on some level about properties of that domain.

1.3.1.1 Basic Elements of Ontologies

Even though ontologies are used in many different domains and implement different languages, most implement at least the following basic elements:

Individual: *An object (physical or logical) of a world, it is an instance of a class.*

Class: *A concept of the world.*

Attribute: *A property that individuals or classes can have.*

Relation: *A link that can exist between two classes or two objects.*

An individual is the most basic object of a world. An object can be either physical or logical. A class is a concept of the world that encompasses objects. Somehow classes describe a division of the world where objects can be categorized or typed. Attributes describe properties, characteristics, parameters that objects can have. These properties are intrinsic, and relations that describe the links between two classes or two objects enable an object to share properties with others.

1.3.1.2 What is an Ontology

An ontology situates at the level of representable knowledge. This is the universe of discourse for the human knowledge. In a specific domain, knowledge that should be represented appears in the domain of discourse. That is why the ontology's concepts should be close to objects in the discourse of the domain of interest. This is also true for individuals and relationships. Good practice identifies classes and individuals as nouns and relationships as verbs in sentences that describe the domain.

A knowledge base can be defined as an ontology that is populated with individuals. A distinction between an ontology and a knowledge base is subtle as the borderline separating the two is not clearly drawn. In this thesis we propose to not distinguish between the two and we define them as an attempt to formalize the universe of discourse. We will use the term ontology as knowledge base refers by analogy to database. Knowledge bases and databases do share some similarities (*e.g.*, information retrieval) and they are important but it is the other aspects shared by ontologies and knowledge bases that need to be highlighted (*e.g.*, deductive reasoning).

The semantic aspect of ontologies is quite specific, often forgotten and is of the greatest importance. An ontology is a semantic network that contains a set of concepts that describe a domain. Those concepts are linked together by usual taxonomy relations and additional semantic relations. An ontology compared to a database uses a more explicit representation for relationships. Ontologies therefore support knowledge sharing and reuse.

Formal Ontologies. In this work we are interested in a particular type of ontologies: *formal* ontologies. The term formal that is used refers to the mathematical meaning. A formal ontology is defined by *axioms* in a *formal language*. We use the following definition:

> Axiom: *A postulate for which the truth value is unquestioned and taken for granted.*
>
> Formal language: *A language which symbols and formulas stand in precisely syntactic and semantic relations to one another.*

The definition of an axiom differs from the one relative to formal logic where only axiomatic statements are considered. The axioms we use define assertions about a domain *which can also include the theory derived from traditional axiomatic (generative) statements.* This mathematical structure enables to perform mathematical reasoning. Two important properties of the deductive system in use are *soundness* and *completeness.* Soundness ensures that any sentence derived from the set of axioms is correct, *i.e.*, all provable sentences are true. Completeness vis-a-vis soundness is the converse of the latter and states that all true sentences are provable. Assertions are used to formulate knowledge. For instance, one can assert that an Electro-Mechanical Brake (EMB) is a braking actuator. Assertions' truth values are also taken for granted. Reasoning on axioms and assertions, a deductive system can infer or prove truth values for the domain sentences. For instance, if we assert, that an EMB delivers a maximum of max braking force, that a braking system is composed of four EMB, and that the braking force delivered by the braking system is equal to the sum of the braking forces that can be delivered by its brakes, then the braking system delivering a maximum braking force of $4 * max$ can be either inferred or proven to be true.

Open-World Versus Closed-World Assumption. Two assumptions can be taken when reasoning in a formal logic: *Closed-World* Assumption (CWA) and *Open-World* Assumption (OWA):

> Closed-world assumption: *What is not known to be true is false.*
>
> Open-world assumption: *What is not known to be true is unknown.*

These two assumptions are different by nature and enable a different understanding of a domain. We give credit to both assumptions as their usefulness are demonstrated for different intentions that do not necessarily overlap and therefore can be combined while mutually reinforcing each other uses. However, whatever the assumption taken, it should be made explicit as it dramatically changes the understanding of an ontology. We can only stress this remark as we actually observed that it was not necessarily the case. Finally, we advocate using the open-world assumption when describing a domain as it revealed really pleasant in use.

Let us give an example of the usefulness of the mathematical background of formal ontologies. For instance, checking the consistency of an ontology is a desirable capability when defining and using an ontology. It ensures that the ontology structure is correct thanks to the mathematical foundation. A person responsible to develop an ontology can be confronted to an element in the universe of discourse which is not yet represented in the ontology. When this person will try to describe the element in the ontology, consistency checking will ensure that the new element description is not in conflict with the rest of the ontology. Consistent ontologies are particularly fitting for engineers who need to completely master their system. One inconsistency and failure is not far.

When confronted to mechatronics systems that are heterogeneous in nature, a formal ontology has a central role to play. As actors in the development of mechatronics systems are interdisciplinary, agreement has to be made in order for all those actors to effectively contribute in the same direction. Even though interdisciplinary, those actors do share some common grounds. Indeed, they develop *the same* system. They can be working on different aspects, however it is evident that these aspects overlap. An ontology is the perfect artifact to conceptualize a shared understanding of all those disciplines in order to define the common grounds on which different actors can agree about.

1.3.2 Evolution of the World Wide Web towards the Semantic Web

In this thesis, we chose to use the semantic web for domain description. The Semantic Web project is a shared research plan that aims "to provide explicit semantic meaning to data and knowledge on the World Wide Web" [Berners-Lee et al. 2001]. The idea is that by providing semantic to Web information will enable useful and automated information processing. In our case, we simply wanted to use an ontology language with Open World Assumption (OWA) that reflects the progressive completeness nature of system's development. Ontology Web Language (OWL) provided by the semantic web is the actual standard *de facto* for both knowledge representation and (OWA). Figure 1.8 presents different layers that compose the semantic web.

In general, the semantic web adds information on *resources* identified with an Unique Resource Identifier (URI). At metadata level, Resource Description Framework (RDF) is

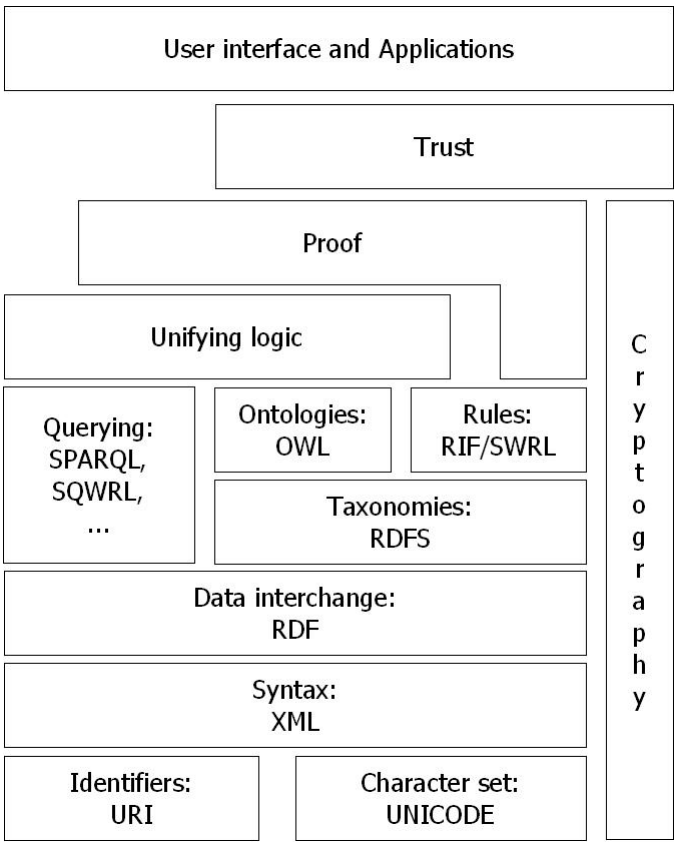

Figure 1.8: Semantic Web Layers

used to add metadata information about Web resources (including things that cannot be directly retrieved). At ontological level, RDF Schema, or RDFS, and OWL add meaning to Web resources and are used to define ontologies (note that RDFS is mainly intended to define taxonomies or vocabularies of a domain). The Semantic Web Rule Language (SWRL) adds rule capability to ontologies in OWL. Querying ontologies is done in parallel using a query language. Finally, at reasoning level, proof (or reasoning) is performed for meaning interpretation meaning at ontological level. Other layers are not relevant to our work but envisioned future web applications will be able to integrate data and knowledge automatically at the semantic level.

1.3.2.1 OWL

Web Ontology Language (OWL) [Patel-Schneider et al. 2004; Motik et al. 2009] was developed as an ontology language for constructing ontologies that provide high-level de-

scriptions of Web content. As any ontology language, OWL enables to define individuals, classes, attributes and relations. Classes are organized into hierarchies and are related to one another with properties (*i.e.*, relations). OWL provides mechanisms for reasoning at both class and individual levels and a powerful constraint language that enables to give a precise interpretation for the concepts in an ontology. Compared to XML, RDF and RDFS, OWL is more expressive and has greater machine interpretability [Stuckenschmidt and Harmelen 2005]. OWL adds more vocabulary for describing classes, instances and properties. Among others, relations between classes (*e.g.*, disjointness), properties cardinality (*e.g.*, exactly one), instances equality, richer typing of properties, properties characteristics (*e.g.*, transitivity), and enumerated classes. In the end OWL is used to explicitly represent the meaning of terms in vocabularies and the relationships between those terms. For example, with RDFS we can define classes like *Actuator* and *Command* with a property *sendsCommand* that has Command as its domain and Actuator as its range. With OWL, we can additionally define that Actuator and Command are disjoint classes, that *receivesCommand* is the inverse property of sendsCommand, and that Actuator is defined precisely as the individuals of Actuator that have at least one value with the property receivesCommand.

In terms of reasoning capabilities, our interest mainly lies in the verification of consistency under the OWA assumption. One characteristic of the World Wide Web is that information is incomplete. Assuming we defined an individual of Actuator, *"EMB"* for instance (EMB stands for Electro-Mechanical Brake), checking consistency will not result in CWA contradiction: the EMB is an actuator and it should receive a command which is not the case. But rather, OWA answer will be EMB is an actuator, it should receive a command which is unknown (not defined) at the moment. Understanding information with OWA seemed more precise and was retained as a characteristic of an ontology language to use. Reasoning with OWL is therefore oriented towards deducing knew knowledge from the definitions of the ontology. However, OWL provides limited deductive reasoning capabilities and relatively recent work has concentrated on adding rules to it.

1.3.2.2 SWRL

The Semantic Web Rule Language (SWRL) [Horrocks et al. 2004] complements OWL for the definition of an ontology. Similarly with many rule languages, SWRL rules are written as couples (antecedent,consequent). The antecedent is referred to as the rule body and the consequent is referred to as the head. The head and body consist of a conjunction of one or more atoms. SWRL rules reason about OWL individuals, primarily in terms of OWL classes and properties. It provides deductive reasoning capabilities that can infer new knowledge from an OWL ontology. For example, with OWL we define the property *interpretsCommand* and three individuals *"Brake request"*, *"Acceleration Request"* and *"EMB"* (all distinct). The first two individuals belong to Command and the last one belongs to Actuator. Also "EMB" can only interpret "Brake request" and we set receivesCommand between the EMB and the two commands. With SWRL we can capture that an actuator tries to interpret a command when it receives one. The rule in SWRL would be: $Actuator(?x) \wedge receivesCommand(?x, ?y) \rightarrow interprets\_Command(?x, ?y)$. The rule body is the conjunction on the left of the arrow and the head is the conjunction on the right. Executing the rule will try to match any individuals in the body (thus the ?) and set the property interpretsCommand for matching individuals. This interesting property already demonstrates two important aspects of rules. In terms of reasoning capabilities, SWRL does enable to deduce additional knowledge from an OWL ontology. However, unlike axioms, the truth value of the head should be questioned. As we defined that the EMB receives all the commands, the rule wants to conclude that it also interprets these commands. But we defined that the only command the EMB can interpret is the brake request. Therefore, inconsistent knowledge can be deduced from rules which should be treated with precaution. SWRL also provides numerous built-ins (that are user-defined methods) that can be extended. Implemented built-ins enable more expressiveness and address XML data-types. For instance, integer comparison operators are defined and enable to retrieve all the actuators with a weight greater than 5 kg as a general example. In the end, OWL and SWRL can be used in combination in the semantic web to define an ontology.

1.3.2.3 SQWRL

The Semantic Query Web Rule Language (SQWRL) [O'Connor and Das 2009] is a relatively powerful query language for OWL ontologies. It enables to extract information while understanding OWL's semantic. It provides operators in a query language fashion (*e.g.*, select, order by, count, *etc.*) for information retrieval. A query is defined similarly to SWRL rules. The body and the head corresponds respectively to retrieval specification query and to query execution. SWRL built-ins can also be used in the body for complex retrieval specifications. The body operates similarly to SWRL pattern matching and the head operates on OWL individuals that matched. For instance, the previous rule body can correspond to a retrieval specification of the commands received by an actuator. One can be interested in the number of commands received by each actuator which is expressed as follow: $Actuator(?x) \wedge receivesCommand(?x, ?y) \rightarrow sqwrl:select(?x) \wedge sqwrl:count(?y)$. Naturally, $sqwrl:select$ returns the list of actuators and $sqwrl:count$ returns the cardinal of commands received by each actuator. The interesting part is that SQWRL queries understand OWL and SWRL semantic. It results that the body will match not only asserted individuals in an ontology *but also* entailed ones. SQWRL also supports some form of closure for more expressiveness. In particular, queries with negation cannot be expressed with the previous structure. For instance, the usefulness of an actuator that does not receive any command can be questioned. SQWRL adds set operators to address closure. As a note, the use of these operators contradicts OWL's OWA therefore they cannot be used in SWRL rules. These operators are used in a second and third part of the body separated with $\circ$. Set construction operators are used in the body's second part and set operation operators are used in the third part. As an example, the following query returns the list of actuators that do not receive any command: $Actuator(?x) \wedge receivesCommand(?x, ?y) \wedge Actuator(?z) \circ sqwrl:makeSet(?s1, ?x) \wedge sqwrl:makeSet(?s2, ?z) \circ sqwrl:difference(?s3, ?s2, ?s1) \wedge sqwrl:element(?e, ?s3) \rightarrow sqwrl{:}select(?e)$. In the query, we match all the actuators that receive a command to $?x$ and all the actuators to $?z$. These individuals are grouped in respective sets $?s1$ and $?s2$ with construction operator $sqwrl{:}makeSet$. Then set $?s3$ is the resulting set difference of $?s2 - ?s1$ with $sqwrl{:}difference$. We return all the elements of $?s3$ that

correspond to the actuators that do not receive any command. In the end, SQWRL is a powerful query language that understands the semantic of OWL and SWRL and we only criticize the inability to define sub-queries (*i.e.*, reusing the result of a query in another query).

1.3.3 Conclusion

In this section, we presented the ontology paradigm as a way to formalize a conceptualization. We specifically presented formal ontologies that use a language with syntax and semantic formally defined. Finally, we presented the semantic web technologies with OWL, SWRL and SQWRL that we chose to use in this work. The interested reader is redirected to Gruber [2009] and Huth and Ryan [2004] who present some elements of logic and the domain of ontologies with many details. ISO [2007] presents Common Logic which is a framework for languages based on First Order Logic. Guarino [1998] and Motik et al. [2006] discuss formal ontologies, logic programming and description logics. Yu et al. [2006] discusses ontology checking. Finally, most of the resources on the semantic web can be accessed via the W3C semantic web activity website (`http://www.w3.org/2001/sw/`).

1.4 Chapter Conclusion

In this state of the art, we presented the product development process and ontologies as completely different domains. In section 1.2.3, we presented some general approaches for the development process. We concluded on the need for a common data model for systems engineering and functional safety and the unsolved problem of (tool's semantic) integration. As a conclusion, let us give the main characteristics of our approach with related works.

Our approach corresponds to a design process based on a common semantic data model for systems engineering and functional safety. The idea is quite simple and comes from the fact that actors involved in a system's development are working on the *same* underlying system. Tudorache [2006] and Sebastian et al. [2008] cover collaborative approaches which corroborate the automotive industry context and the idea of a same underlying system.

In our approach, the reference data model is realized with a formal ontology. Compared to other approaches, the data model's semantic is formally defined. Burr et al. [2005], Suwanmanee et al. [2005] and Driouche et al. [2007] discuss data integration in the design process. Data integration enables the exchange of information between applications and thus interoperability. It is shown that the real interoperability problem is to address semantic interoperability between heterogeneous tool's conceptualizations. For example, when two different tools communicate, their communication is based on the assumption that they *understand* one another, *i.e.*, similar concepts have the same meaning in each tool. The use of a formal ontology (and its formal semantics) simply transforms this assumption into a property. Previous integration solutions mostly address only syntactic integration which is actually still the case. For instance, in the case of Model Driven Engineering, Harmelen and Fensel [1995]; Evans et al. [1998]; Brucker et al. [2006]; Micskei and Waeselynck [2010] demonstrate that UML /SysML, even with OCL, have no formal semantics. It results that using MDE technologies such as the popular Eclipse platform with EMOF (Eclipse Meta Object Facility), can make the tools inter-operate. But these tools are not integrated at the semantic level. This may lead to error prone misunderstandings.

Ontologies and the semantic web technologies have been actively used to deal with the semantic interoperability problem [Sure et al. 2002; Bussler 2003; Stuckenschmidt and Harmelen 2005; Suwanmanee et al. 2005; Driouche et al. 2007]. In our approach, we build upon these ideas and go further by exploiting the semantic web during the design process, in order to ensure its consistency.

In this thesis, we do not propose a fundamentally new approach. As explained, related works have already identified the problem of semantic integration. We pursue the idea that, by analogy to Model Driven Engineering, it is possible to solve semantic integration in the design process by doing "Ontology Driven Engineering". It is relatively new and to our knowledge has only been partially presented by Gasevic et al. [2009]. We use the semantic web technologies to propose a design approach based on a common underlying ontology that enables to verify information consistency all along the design process. The next chapter presents our contributions: the production of a systems engineering and functional safety ontology that formalizes and integrates the domains presented in sections 1.2.1 and

1.2.2; the ontology centric design approach which is compliant to ISO 26262 standard and enables to guarantee information consistency at the semantic level.

Chapter 2

Contribution

Sommaire

2.1 Introduction

Problems of the Current Design Process

The model-based design approaches outlined in section 1.2.3 call for different design objects that have to be described as clearly as possible (*e.g.*, requirements, system architectures, safety goals, system use-cases, Feared Customer and System Events, fault trees, *etc.*). The first implementations of the process at Renault were mainly document-centric and depended largely on testing and simulation [Chalé Góngora et al. 2009]. Although these first attempts gave quite satisfactory results in building safe system architectures, the creation of the different objects of the process was somewhat troublesome and relatively time-consuming. The reason for this is that the objects were modeled by means of transformations of ad-hoc data and information contained in the different documents that were transmitted from one process step to the other.

The main difficulty in implementing the process consisted thus in the lack of semantic consistency among the different modeled objects. This *need* for a better formalization is further stressed by the fact that car manufacturers rely heavily on third parties to develop vehicle systems. A better formalization of processes and process objects would certainly contribute to avoid confusion and misinterpretations in the development of systems. All this led us to the conclusion that the use of formal and informal (but consistent) models can commit to a *common semantic model*, *i.e.*, a system and safety ontology, which purpose is to better understand all the aspects of safety-critical system design.

Content of the Chapter

This chapter presents the contributions. Based on identified issues of the current design process at Renault, we propose an approach that addresses those problems that can be generalized and applied by any company. In particular for Renault, the domains of systems engineering and of functional safety are the main point of focus. As such, the chapter is structured in order to cover two significant contributions of the thesis. The first is the realization of a domain ontology for Renault systems and safety engineers. The second is the improvement of the design process with the domain ontology as a basis.

Section 2.2 presents the formalization of the conceptualization of systems engineering and functional safety domains. Section 2.3 is the integration of the systems engineering and functional safety ontology into a domain ontology that supports the design process for systems and safety engineers. The domain ontology defines the sharing of a conceptualization that promotes synergies. Section 2.4 presents our approach: Ontology Based Design Process. Finally, section 2.5 gives some conclusions.

2.2 Domains Formalization

Formalization of knowledge addresses the problematic of harnessing and reusing knowledge, a key point for all domains. In the engineering domain, decision making is fundamental as engineers are confronted with multiple choices and only one solution will be subject to complete development. In order to choose the best solution, the engineers need the relevant necessary knowledge required to perform informed decision making. Currently, content created in the development process is ultimately archived and forgotten, knowledge is lost whereas archived. There is evident need to make use of this produced and archived knowledge considering the probable occurrence of confrontation with an already solved problem. There is two important points to verify in order to address this issue. First, some quality of knowledge needs to be assessed. For that aspect we want to verify the consistency of knowledge. Second, for easy reuse of knowledge, conceptualization of knowledge needs to be shared as reusing something that is not understood can be detrimental. The formalization activity consists in better structuring knowledge, enabling computer treatment for more performing information retrieval that has to bring relevant information to the surface. This can be implemented in an elegant fashion using the ontology paradigm.

This work focuses on two specific engineering fields of Renault Research and Advanced Engineering. The first domain is systems engineering. With systems being more and more complex at Renault, the need for system engineers and system thinking is becoming pressing matter. Systems engineering is a relatively new domain at Renault and deployment is in progress. Similarly, the second domain, functional safety, is subject to change required for conformity with ISO 26262 international standard. Sections 2.2.1 and 2.2.2 respec-

tively present the formalization of systems engineering and functional safety domains with ontologies that tackles previous issues in knowledge quality.

Even though the formalization (via ontologies) has been done using OWL and SWRL (see section 1.3), in this chapter, the formalization is presented using First Order Logic (FOL). The reader is assumed to be familiar with FOL. Using it enables to reason under usual closed world assumption, to express negative facts and to use logical disjunction ($\oplus$) which makes reading more natural. All these languages are formal in the sense that their syntaxes and semantics are defined formally which enables verifications such as consistency.

The formalization of systems engineering and functional safety is done following the partial axiomatization given in appendix A. Informally, the axiomatization defines *instance*, *class* and *property*. Instances are expressed with constant symbols. Variables are used only to refer to sets of instances. For example, the instance *constant* can be substituted to the variable x. A class formalizes a concept as a *set* of instances. A class is expressed with a unary predicate whose term is a concatenation of words with their first letter capitalized (*e.g.*, the class *Requirement*). $Requirement(x)$ denotes the set of instances that can be substituted to x that are in the set *Requirement*. A property formalizes a role from one instance to another. A property is expressed by a binary predicate whose term is a concatenation of terms. The first word is a verb in lowercase conjugated at the third singular person and the following words have their first letter capitalized. That corresponds to the relation name . It is followed by an underscore and another term and can be followed by another underscore and term, *e.g.*, $hasPart\_DomCoDom$ and $hasConstituant\_Dom\_CoDom$. In the case of only one underscore, the term after the underscore refers to the domain and the co-domain of an internal relation expressing a class attribute. Otherwise, the term after the first underscore refers to the relation domain relation and the term after the second underscore refers to the relation co-domain expressing a role from a domain instance to a co-domain one. For example, $hasPart\_DomCoDom(x, y)$ denotes the set of couples of instances that are a substitution of x and y with $x, y \in DomCodom$. It expresses a whole to part relation, so the user should understand the substitution of x being the whole and the substitution of y being the part. The terms class and concept, property and relation, instance and individual, and, range and co-domain, are used indifferently pairwise. All the

variables present in the axioms are assumed to be universally quantified if no quantifier is used.

2.2.1 Systems Engineering Ontology

The systems engineering ontology supports the general design process at the system level from the needs to the design of the system architectures. The choice upon methods and tools is still arbitrary and relative to each project but based on previous history, available manpower and expertise. Figure 2.1 presents a high level view of the system design process with the main activities of specification and design.

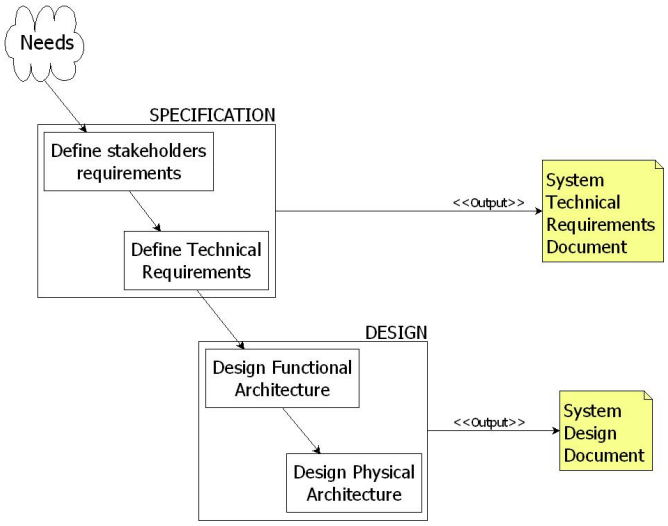

Figure 2.1: System design process at Renault

Indifferently of model-based approach or document-centric approach, two documents are required in the system design process (see figure 2.1): the *System Technical Requirements* (STR) document and the *System Design Document* (SDD). For now, these documents are the interfaces between the different fields involved in a project. The activities represented in the figure manipulate the general concepts of $Need, Requirement$, $Function$ and $Component$. In the following sections, we detail the content of these documents in order to make those concepts precise. Sections 2.2.1.1, 2.2.1.2, 2.2.1.3 and 2.2.1.4 present the formalization of those concepts independently of each other. Section 2.2.1.5 makes those concepts precise by presenting how traceability is established. Finally, section 2.2.1.6 presents some thoughts on the non functional aspects of a system.

2.2.1.1 On Needs

The design process first activity is the definition of the stakeholders requirements. The STR document contains sections on system's finality, missions, goals and strong concepts expressed in natural language. This information is used to guide the brainstorming for the finding and elaboration of the stakeholders requirements. They are a suitable form (usually better structured natural language) of stakeholders needs, for both the development and communication between developers and stakeholders. Another important section that enables to construct a better set of stakeholders requirements is the description of the system context. The system is taken as a black box and its environment is described so as to define its functional and physical boundaries. This enables to take into account external considerations in order to develop requirements that are relevant for the system (and for the elements outside the system perimeter). These external considerations are presented in sections 2.2.1.3 and 2.2.1.4.

The conceptualization is straight-forward. It is illustrated in figure 2.2.

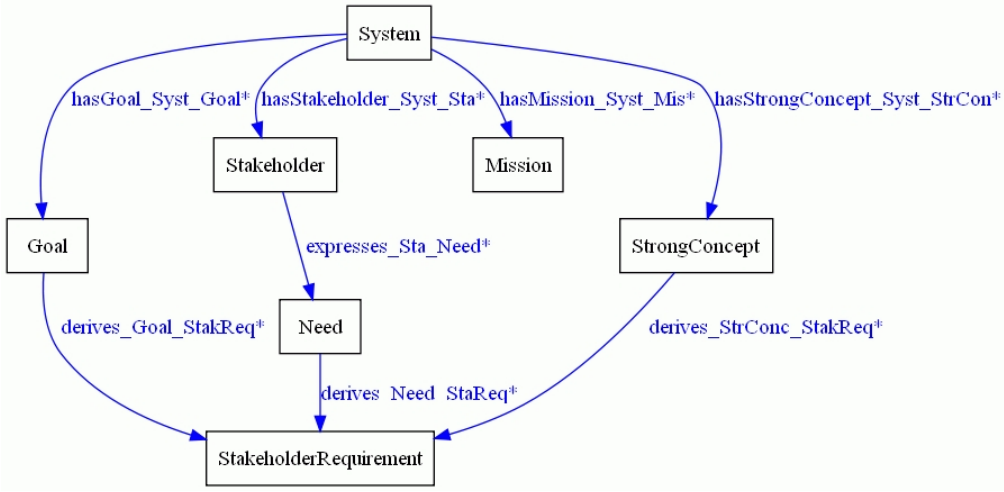

Figure 2.2: Formalization of the needs

We have the following disjoint classes: *System*, *Stakeholder*, *Mission*, *Goal*, *StrongConcept*, and *StakeholderRequirement* (represented by rectangles in figure 2.2). *Need* is not disjoint with the other classes. Each class in the ontology is given a natural language definition

coming from either INCOSE[1], AFIS[2] or Renault terminologies. Only controversial and unusual definitions are reported in this work.

A system is related to its stakeholder(s), mission(s), goal(s) and strong concept(s). This is formalized with respective *has* relations. Those relations are represented with labeled directed arrows in figure 2.2. Arrows origin and destination express domain and co-domain predicates. Also the relations are constrained for each class with respective range and a minimum cardinality of 1. In figure 2.2, the quantifications are represented with a star next to the label of the relation that corresponds to $1..*$ in look across notation (*i.e.*, each class instance that implements the relation, is related (across the relation) to 1 or more instances in the relation range). The range and the minimum cardinality can be expressed with a single axiom using the existential quantifier, denoted $\exists$. The following axioms for the class *System* are given as general examples for other classes as axioms that respect appendix A axiomatization are implicitly defined and essentially not reported.

$$\exists\, y \quad System(x) \wedge Stakeholder(y) \wedge hasStakeholder\_Syst\_Sta(x, y) \tag{2.1}$$

$$\exists\, y \quad System(x) \wedge Mission(y) \wedge hasMission\_Syst\_Mis(x, y) \tag{2.2}$$

$$\exists\, y \quad System(x) \wedge Goal(y) \wedge hasGoal\_Syst\_Goal(x, y) \tag{2.3}$$

$$\exists\, y \quad System(x) \wedge StrongConcept(y) \wedge hasStrongConcept\_Syst\_StrCon(x, y) \tag{2.4}$$

A stakeholder expresses some needs which is made explicit with relation *expresses_Sta_Need*. Similarly to the restrictions on the previous relations, a stakeholder expresses at least one need.

Finally, four respective *derives* relations are defined from *Mission*, *Goal*, *Need* and *StrongConcept* to *StakeholderRequirement* with the quantification restriction of at least one stakeholder requirement (*i.e.*, minimum cardinality of 1). Missions, goals and strong concepts define respectively special assignments given to the system (the main functionalities of the system), properties that the system should exhibit in the end (for example,

[1] International Council on Systems Engineering
[2] Association Française d'Ingénierie Système

a usual goal of a new braking system is for its price to be at most the same as an earlier braking system) and unavoidable concepts that constrain the solutions for the system in an unavoidable manner (such as using or not using specific technologies. For instance using electro-mechanical brake actuators on the front wheels of the vehicle is a strong concept for the braking system used in the case study). Each of the classes, *Need*, *Mission*, *Goal* and *StrongConcept*, account for the domain of *need*. They have to be taken into account by the system development which is done by making them more accurate as stakeholders requirements using respective *derives* properties.

Even though those classes can be better structured with other relations, we did not go further on the formalization of the needs (yet some precisions are given in section 3.3.2) as our interest was on the integration of functional safety to systems engineering which we start at the abstraction level of requirements.

2.2.1.2 On Requirements

Stakeholders needs are the entry point into the somewhat more formal world of requirements. The requirement concept is first defined as a general concept with coverage greater than what is contained in the STR document. Then, based on this general definition, the concept is made precise.

The Concept of Requirement As a general definition, a requirement is a statement that expresses a need and/or a constraint. As stated in section 1.2.1.1, requirements are still an active subject of research. They delimit the frontier between the informal and the more formal worlds in the development process. The difficulty is relative to the high degree of ambiguity of the informal world. No attempt is given to formalize the statement that constitutes the requirement. The presented conceptualization makes precise a typology of requirements and their relations with other concepts. From the universe of discourse, we only retain that a requirement can:

- be *decomposed* into other *requirements*;
- *derive* other types of *requirements*;

- *relate* to other *requirements* to detail a peculiar aspect of the former requirement;
- *relate* to *functions* or *flows* that implement the requirement;
- be *allocated* to a *system*, a *function*, a *component* or a *flow*.

Figure 2.3 illustrates the requirement concept formalization at the most general level of abstraction. As it can be seen in the figure, the concepts *Requirement*, *Function*, *Flow*

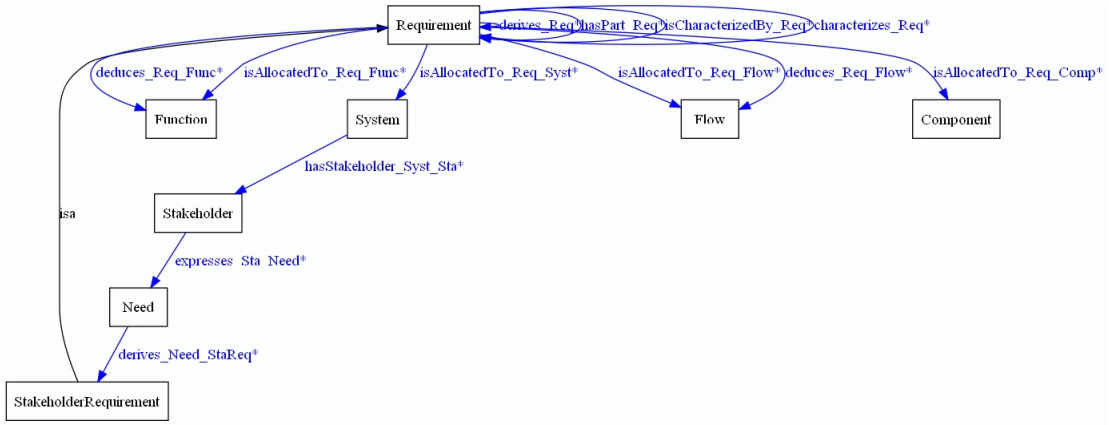

Figure 2.3: Formalization of the requirement concept

and *Component* are defined along with the previous concepts of *System*, *Stakeholder*, *Need* and *StalkeholderRequirement*. The rest of the formalization defines the relations of requirements with other concepts through properties *hasPart_Req* for the decomposition of a requirement in a whole to part manner, *derives_Req* to relate different kinds of requirements, *characterizes_Req* and *isCharacterizedBy_Req* for the precision of a requirement by another and the converse, *deduces_Req_Func* and *deduces_Req_Flow* to relate a requirement to a function or a flow identifiable in the requirement statement, and four respective *isAllocatedTo* properties for the allocation of a requirement to a system, a function, a component or a flow. There is no cardinality constraints on the properties (the stars in figure 2.3 next to the relations names correspond to $0..*$, *i.e.*, a class instance at the origin of an arrow is related to 0 or more class instances at the arrow destination). This is done intentionally for staying at the highest degree of abstraction by expressing the possible relations that a requirement is involved with. Making the requirement concept precise is actually removing this "possible" aspect of requirements descriptions.

Requirements Typology. Making the requirement concept precise is first done by identifying the different kinds of requirements that may be encountered in development processes.

Informal description of the requirement concept. As seen in fig 1.3, there is a distinction between the domains of the problem and of the solution for a system under development. The requirements that correspond to the domain of the problem are the stakeholders requirements. Once a set of stakeholders requirements has been agreed upon it is time to think of the solution. The next activity in the design process (see figure 2.1) is the definition of the technical requirements. The technical requirements are requirements that define the system characteristics. The usual partition into functional and non functional requirements is used and we make no attempt in the formalization of the non functional ones. In addition to technical requirements that describe the system, another type of technical requirements is relative to system elements characteristics. They are part of the technical requirements specified in the STR document but can have another substantial role (*i.e.*, subsystems requirements) later in the development process when going to the abstraction level of the subsystems. Finally, another type of requirements concerns system external elements. They are not part of the STR document. They are however necessary for the system to work correctly. At Renault, systems usually correspond to subsystems of the whole vehicle. As external requirements specify external elements, they are actually stakeholders requirements for the systems that are constituted of these elements.

Requirements abstraction types. The following distinctions apply:

- problem and solution domains of the system.
- system requirements and external requirements.
- system requirements and requirements of the elements of the system.
- functional and non functional requirements.

Figure 2.4 presents a first decomposition of the Requirement class that considers the precedent important distinctions.

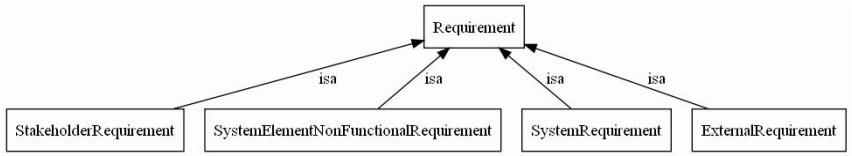

Figure 2.4: Requirements typology

In the figure, the black arrows with the relation name *isa* (that stands for "is a") correspond to the subsumption relation for classes. The four classes in the figure are subclasses of *Requirement* and are disjoints. They formalize different abstraction levels of requirements that correspond to the design process timeline. The *StakeholderRequirement* class formalizes the expression of the stakeholders needs into requirements that belong in the domain of the problem. At the next level of abstraction the classes *SystemRequirement* and *ExternalRequirement* formalize respectively system requirements and requirements of the external elements of the system that also belong to the domain of the problem. Then, the final level of abstraction concerns the requirements of the system elements. Those requirements describe only non functional characteristics of a system element (either functions or components) and are grouped in the class *SystemElementNonFunctionalRequirement* that accounts for a solution for the system.

Functional and non functional requirements. The different types of requirements that correspond to the design process timeline are further subsumed as illustrated in figures 2.5, 2.6, 2.7 and 2.8. The classes of each layer in the presented hierarchy are disjoints with each other.

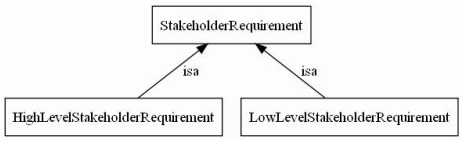

Figure 2.5: StakeholderRequirement subclasses

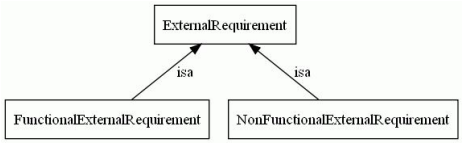

Figure 2.6: ExternalRequirement subclasses

Another important distinction for the requirement concept concerns the functional and non functional aspects of a requirement. The notions of "functional" and "non functional" requirement will be made formal in section 2.2.1.5 but informally, a functional require-

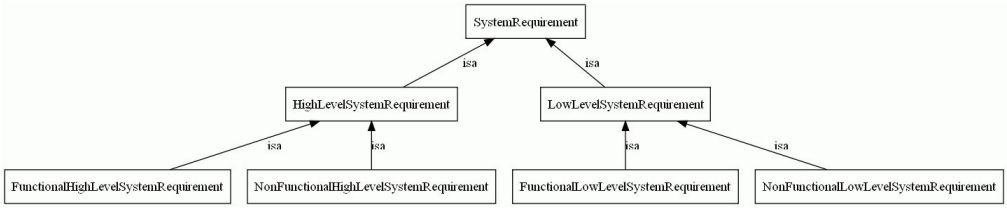

Figure 2.7: SystemRequirement subclasses

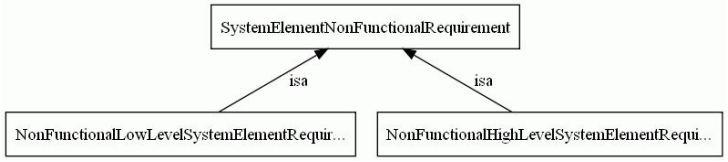

Figure 2.8: SystemElementNonFunctionalRequirement subclasses

ment describes some functionality to be implemented by a system and a non functional requirement describes a property to be satisfied by a system.

Stakeholders requirements can account for functional and non functional characteristics so they do not partition into functional and non functional subclasses. The class is however further defined with two subclasses: *HighLevelStakeholderRequirement* and *LowLevelStakeholderRequirement* (*cf.* figure 2.5). The terms "high level" and "low level" denote respectively requirements that are decomposed into other requirements and requirements that are not decomposed. This is explained in the next paragraph.

Similarly to the stakeholders requirements, using the *high level/low level* distinction, *SystemRequirement* is subsumed into *HighLevelSystemRequirement* and *LowLevelSystemRequirement* (*cf.* figure 2.7). The rest of the figure illustrates the further subsumptions of *HighLevelSystemRequirement* and *LowLevelSystemRequirement* classes, *i.e.*, *FunctionalHighLevelSystemRequirement*, *NonFunctionalHighLevelSystemRequirement*, *FunctionalLowLevelSystemRequirement* and *NonFunctionalLowLevelSystemRequirement*.

Illustrated in figure 2.6, the class *ExternalRequirement* is decomposed into the classes *FunctionalExternalRequirement* and *NonFunctionalExternalRequirement*.

Finally, in figure 2.8, the non functional requirements of the system elements are organized into high level and low level requirements: *NonFunctionalHighLevelSystem-*

ElementRequirement and *NonFunctionalLowLevelSystemElementRequirement*.

We do not go further on the different kinds of requirements. There is indeed a consequent number of different non functional requirements. Formalizing a relevant subset of the different types of non functional requirements that relates to critical mechatronic systems for the automotive domain has been quickly set aside. It has nonetheless been examined. Although it has not been formalized, it is set up partially in a formal manner in the ontology with explicit intentions that are presented in section 2.2.1.6.

Requirements Decomposition. The requirements are usually structured in a hierarchical manner, *i.e.*, a requirement can be decomposed into sub-requirements. As a sub-requirement has no conceptual addition to the *Requirement* class, it is not defined as a class. The *Requirement* class is simply structured in relation to itself by defining the *hasPart_Req* property with the *Requirement* as domain and range. This property is intended to express the invers relational quality of parthood as to follow the top-down development. Using $hasPart\_Req(x, y)$ states that "*x is a super-requirement of y*" and "*y is a sub-requirement of x*". There is no cardinality constraints on the *hasPart_Req* property (stars in figure 2.3 next to relations names correspond to $0..*$) therefore the relation of decomposition of a requirement describes both the possible sub-requirements and super-requirements of a requirement. In other words, a requirement can be in relation with 0, 1 or more sub-requirements and 0, 1 or more super-requirements. This is illustrated in figure 2.9 with squares representing requirements instances and the arrows representing the hasPart property.

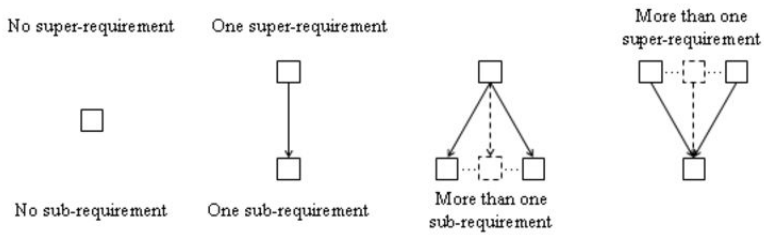

Figure 2.9: Possible constructions for the *hasPart* property

Complying with the axiomatization given in appendix A, the *hasPart* property is inherited

to the subclasses of *Requirement*. The subclasses of *Requirement* are presented in the previous paragraph. If not said otherwise, the range of hasPart is constrained for each subclass to that subclass. In other words, stakeholders requirements are decomposed into stakeholders requirements, system requirements are decomposed into system requirements, and so on (if not said otherwise). In the following we have a look at the decomposition relation for all the subclasses of Requirement.

Decomposition of the stakeholders requirements. As mentioned earlier, the stakeholders requirements are decomposed into stakeholders requirements and no cardinality restriction is defined. I distinguished between two types of stakeholders requirements: the high level and the low level stakeholders requirements (see figure 2.5).

The justification of this distinction is the same for all "high level" and "low level" classes and related to decomposition. This justification is given in the paragraph that follows the explanations of the decompositions. Here only the meaning of those terms is defined. As a general definition, "high level" and "low level" refer to objects that can be decomposed into objects of the same type. A high level object is always decomposed and a low level object is never decomposed. The correct analogy is a directed acyclic graph (those include trees) in graph theory. The leafs in the graph correspond to low level objects and the other nodes in the graph correspond to high level objects (see figure 2.10. The rectangles in the figure are objects and the arrows are a decomposition relation).

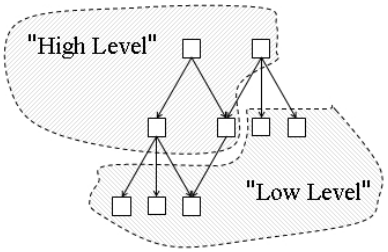

Figure 2.10: Analogy of directed acyclic graph to high level and low level objects

Coming back to the stakeholders requirements decomposition, the range of $hasPart\_Req$ property for $HighLevelStakeholderRequirement$ is refined to $StakeholderRequirement$ (*i.e.*, $HighLevelStakeholderRequirement \cup LowLevelStakeholderRequirement$)

and given a minimum cardinality of 1 (2.5). The class *LowLevelStakeholderRequirement* is not decomposed (2.6) which is equivalent to assign a maximum cardinality of 0 to $hasPart\_Req$ for the class.

$$\exists\, y \quad HighLevelStakeholderRequirement(x) \Rightarrow \\ StakeholderRequirement(y) \wedge hasPart\_Req(x, y) \tag{2.5}$$

$$\nexists\, y \quad LowLevelStakeholderRequirement(x) \wedge hasPart\_Req(x, y) \tag{2.6}$$

Decomposition of system requirements. They define some system characteristics of the system under development. Informally, systems requirements are allocated to the system under development. Systems requirements are organized into high level and low level system requirements, we have axioms (2.7) and (2.8). Then, for these two classes, we have the distinction between functional and non functional requirements. The system requirements that are functional are decomposed only into system requirements that are functional. Similarly, the system requirements that are non functional are decomposed only into system requirements that are non functional. The formalization is done as follows. The co-domain of $hasPart\_Req$ for $FunctionalHighLevelSystemRequirement$ is constrained to $FunctionalHighLevelSystemRequirement \cup FunctionalLowLevelSystemRequirement$. Likewise, the co-domain of $hasPart\_Req$ for $NonFunctionalHighLevelSystemRequirement$ is constrained to $NonFunctionalHighLevelSystemRequirement \cup NonFunctionalLowLevelSystemRequirement$. As a reminder, the cardinality restrictions are inherited as defined in appendix A (this corresponds to axioms (2.9), (2.10), (2.11) and (2.12)). Therefore, $FunctionalHighLevelSystemRequirement$ and $NonFunctionalHighLevelSystemRequirement$ are in relation with at least one element in the image of $hasPart\_Req$. The property is disabled for $FunctionalLowLevelSystemRequirement$ and $NonFunctionalLowLevelSystemRequirement$.

$$\exists\, y \quad HighLevelSystemRequirement(x) \Rightarrow \\ SystemRequirement(y) \wedge hasPart\_Req(x, y) \tag{2.7}$$

$$\nexists\, y \quad LowLevelSystemRequirement(x) \wedge hasPart\_Req(x, y) \tag{2.8}$$

$$\exists\, y \quad FunctionalHighLevelSystemRequirement(x) \Rightarrow \\ hasPart\_Req(x, y) \wedge [FunctionalHighLevelSystemRequirement(y) \vee \\ FunctionalLowLevelSystemRequirement(y)] \tag{2.9}$$

$$\nexists\, y \quad FunctionalLowLevelSystemRequirement(x) \wedge hasPart\_Req(x, y) \tag{2.10}$$

$$\begin{aligned}\exists\, y \quad & NonFunctionalHighLevelSystemRequirement(x) \Rightarrow \\ & hasPart\_Req(x, y) \wedge [NonFunctionalHighLevelSystemRequirement(y) \vee \\ & NonFunctionalLowLevelSystemRequirement(y)]\end{aligned} \tag{2.11}$$

$$\nexists\, y \quad NonFunctionalLowLevelSystemRequirement(x) \wedge hasPart\_Req(x, y) \tag{2.12}$$

Decomposition of the external requirements. Those requirements define some characteristics of the elements outside the system perimeter. They are not part of the STR document that accounts only to the system under development. They are however necessary for the system to work correctly and also to understand the requirements globally. As the external requirements specify external elements they are actually stakeholders requirements for these elements. Their consideration is the object of other developments. For this reason, the external requirements are not the object of decomposition as the project is not responsible for their development. It is formalized by giving a minimum and maximum cardinality of 0 to *hasPart_Req* for *ExternalRequirement* (2.13). The two subclasses of *ExternalRequirement* (functional and non functional) presented in figure 2.6 inherit the previous axiom.

$$\nexists\, y \quad ExternalRequirement(x) \wedge hasPart\_Req(x, y) \tag{2.13}$$

Decomposition of the non functional requirements of the system elements. Those requirements refer to the system elements, *i.e.*, functions, components and flows (see figure 2.3). Non functional requirements describe an aspect of the system that is not a function. The present conceptualization only considers the design process at system level. Non functional requirements of the system elements are either high level or low level (*cf.* figure 2.8). The range of *hasPart_Req* property for *NonFunctionalHighLevelSystemElementRequirement* is customized to *NonFunctionalSystemElementRequirement* and the minimum cardinality of the relation is constrained to 1 for *NonFunctionalHighLevelSystemElementRequirement* (2.14). The relation is disabled for *NonFunctionalLow-*

$LevelSystemElementRequirement$ (2.15).

$$\exists\, y \quad NonFunctionalHighLevelSystemElementRequirement(x) \Rightarrow hasPart\_Req(x,y) \wedge [NonFunctionalHighLevelSystemElementRequirement(y) \vee NonFunctionalLowLevelSystemElementRequirement(y)] \tag{2.14}$$

$$\nexists\, y \quad NonFunctionalLowLevelSystemElementRequirement(x) \wedge hasPart\_Req(x,y) \tag{2.15}$$

Requirements Internal Traceability. The traceability established in this paragraph structures the different kinds of requirements with each other. This structure enables to follow the trail left by a requirement. This trail corresponds to a sequence of relations that is called a *trace*. By requirements internal traceability we refer to the capability to establish such traces from any requirement of a level of abstraction to another so we focus on the relations that only involve requirements only. Section 2.2.1.5 discusses traceability between requirements and other elements .

Stakeholders requirements traceability. The abstraction change from the stakeholders level to the system and external level is done through the $derives\_Req$ relation. This relation denotes some abstraction change in the requirement concept. More formally, the range of the relation $derives\_Req$ for $StakeholderRequirement$ is restricted to $SystemRequirement \cup ExternalRequirement$. In section 2.2.1.2 we presented the typology for requirements and the distinction that is made between functional and non functional requirements. In the universe of discourse, the stakeholders requirements account for the most abstract, as yet imprecise, type of requirement. The distinction between functional and non functional is made at the next level of abstraction by using the derivation relation. Even though we restricted the image of $derives\_Req$ to $SystemRequirement \cup ExternalRequirement$, recall that the system and external requirements are further decomposed into functional and non functional classes enabling to define precise requirements with functional or non functional aspect in addition with the information on the relevance to the system or to external components.

As we have seen in the former paragraph, the stakeholders requirements are structured in a hierarchy and can be the object of decomposition or not. To ensure that the

stakeholders requirements are effectively taken into account, all the low level stakeholders requirements need to be derived in at least one instance of $SystemRequirement \cup External\text{-}Requirement$ (this is equivalent to assigning a minimum cardinality of 1 to $derives\_Req$ for $LowLevelStakeholderRequirement$). It is possible by following the structure of the stakeholders requirements to establish a trace from any high level stakeholder requirement to a low level stakeholder one (the structure is a directed acyclic graph and it is not possible to construct a high level object that is not decomposed, see figure 2.10 for an example). By extension, as we just axiomatized that all the low level stakeholders requirements are related to system or external requirements, all stakeholders requirements are either directly (the low level ones) or indirectly (the high level ones) related to those kinds of requirements.

It is interesting to observe that this formalization is only possible by constructing high level and low level classes. Indeed, if the classes high level and low level were not defined, the expression "all the low level stakeholders requirements are the object of at least one relation derives" would be formalized with axiom (2.16).

$$\begin{aligned} \exists\, y \quad & StakeholderRequirement(x) \wedge StakeholderRequirement(z) \wedge \\ & [SystemRequirement(y) \vee ExternalRequirement(y)] \wedge \\ & \neg hasPart\_Req(x,z) \wedge derives\_Req(x,y) \end{aligned} \tag{2.16}$$

This part of the axiom, $\exists y\ StakeholderRequirement(x) \wedge [SystemRequirement(y) \vee ExternalRequirement(y)] \wedge derives\_Req(x,y)$, is equivalent to assigning a minimum cardinality of 1 to $derives\_Req$ for *all* the stakeholders requirements, contradicting with the intended restriction for only the stakeholders requirements that are not decomposed. In other words, the theory only allows to restrain the cardinality of a relation for a whole class and not for only some elements of a class.

Finally, a high level stakeholder requirement can participate to the $derives\_Req$ relation with some system and/or external requirements. The semantic is that all the sub-requirements (of the stakeholder requirement) implement the relation with the same images (of the stakeholder requirement). More formally, the stakeholder requirement transitive closure through $hasPart\_Req$ derives the images of the stakeholder requirement (2.17). This is a desired capability to relate a set of stakeholders requirement.

$$\begin{aligned} & HighLevelStakeholderRequirement(x) \wedge hasPart\_Req(x,y) \wedge \\ & \quad derives\_Req(x,z) \Rightarrow derives\_Req(y,z) \end{aligned} \tag{2.17}$$

Functional system requirements traceability. Functional system requirements (*i.e.*, $FunctionalHighLevelSystemRequirement$ and $FunctionalLowLevelSystemRequirement$) are detailed through the decomposition relation (with requirements whose statements are more precise and more atomic) and by other requirements that account for non functional characteristics that supplement the functional statement of the functional requirement. For instance, the requirement *"The braking system shall decelerate the vehicle"* is functional. The requirement *"The braking system shall decelerate the vehicle and shall allow a deceleration* $\geq 5.8m.s-2$*"* amounts for an additional non functional characteristic to the decelerate function. It is non functional.
A functional system requirement can also be related to some external requirements. A relation with functional or non functional external requirements exhibit some stakes (functional or non functional) for the proper functioning of the system that are emergent properties at a greater level of abstraction than the system under development. For example, if the system commands another system, the system is responsible for sending commands to the other system that is responsible for receiving and handling the commands. If the other system does not receive or handle the command, the functionality (at the level of abstraction that comprises both the system and the external system) is lost. Similarly, if the other system handles the command too slowly, the functioning property is also lost.

The abstraction change from functional system requirements to non functional system requirements is formalized with the property $isCharacterizedBy\_Req$. The codomain of $isCharacterizedBy\_Req$ is restricted to $NonFunctionalHighLevelSystemRequirement \cup NonFunctionalLowLevelSystemRequirement$ for $FunctionalHighLevelSystemRequirement$ and $FunctionalLowLevelSystemRequirement$. A minimum cardinality of 1 is given to $isCharacterizedBy\_Req$ for $FunctionalLowLevelSystemRequirement$. By construction, all the functional system requirements are related directly or indirectly (through the decomposition relation) to at least one non functional system requirement, ensuring that non functional aspects of a functional requirement are taken into account in the design. Axiom (2.18) is the capability to characterize a set of functional

system requirements.

$$FunctionalHighLevelSystemRequirement(x) \wedge hasPart\_Req(x,y) \wedge isCharacterizedBy\_Req(x,z) \Rightarrow isCharacterizedBy\_Req(y,z) \quad (2.18)$$

Relation *derives_Req* is used once again between functional system requirements and functional external requirements. The range of the relation is constrained to $FunctionalHighLevelSystemRequirement \cup FunctionalLowLevelSystemRequirement$.

Finally, the relation $derivesNonFuncExtReq\_FuncSystReq\_NonFuncExtReq$ is added with $FunctionalHighLevelSystemRequirement \cup FunctionalLowLevelSystemRequirement$ as domain and $NonFunctionalExternalRequirement$ as co-domain. It enables to express some non functional stakes on the external elements of the system from the functional system requirements if needed. Therefore the relation cardinality is not constrained.

Functional external requirements traceability. First, remember that the external requirements can be related directly with the stakeholders requirements (and now they can be related to functional system requirements). Therefore, it exists some functional external requirements that are not the object of the *derives_Req* relation for the functional system requirements. It is important however to ensure that all the functional external requirements are related to some functional system requirement (this will be explained in section 2.3.2). This is realized by defining the relation $isDerivedFrom\_FuncExtReq\_FuncSystReq$ with $FunctionalExternalRequirement$ as domain and $FunctionalHighLevelSystemRequirement \cup FunctionalLowLevelSystemRequirement$ as co-domain. $FunctionalExternalRequirement$ implements the relation and the image of $isDerivedFrom\_FuncExtReq\_FuncSystReq$ is defined for the class to $FunctionalHighLevelSystemRequirement \cup FunctionalLowLevelSystemRequirement$ and given a minimum cardinality of 1. As the external requirements are not decomposed those axioms are sufficient to ensure that all the functional external requirements are related to at least one functional system requirement.

Non functional system requirements traceability. The *derives_Req* property is used one last time to relate non functional system requirements to non functional requirements of the system elements. The relation range is customized to $NonFunctionalHighLevelSystemElementRequirement \cup NonFunctionalLowLevelSystemElementRequirement$

for $NonFunctionalHighLevelSystemRequirement$ and $NonFunctionalLowLevelSystem-Requirement$. The relation is given a minimum cardinality of 1 for $NonFunctionalLow-LevelSystemRequirement$ to ensure that all the non functional system requirements are related directly or indirectly (through $hasPart\_Req$) to at least one non functional requirement of a system element. It is once again given the possibility to relate a set of non functional system requirements to a non functional requirement of a system element through $derives\_Req$ as defined by axiom (2.19).

$$NonFunctionalHighLevelSystemRequirement(x) \wedge hasPart\_Req(x,y) \wedge isCharacterizedBy\_Req(x,z) \Rightarrow isCharacterizedBy\_Req(y,z) \quad (2.19)$$

2.2.1.3 On Functional Architecture

In the previous STR document, the system has been considered as a black box to identify the necessary interactions with the environment. The level of detail vary from lists of external functions and elements to system context diagrams. As mentioned, this definition of the system context accounts for the system interactions with its environment. The environment has been described but not detailed. In this section, we detail the definition of the system functional architecture, see figure 2.1. We are interested only in the functional aspect of the environment and in the whitening of the functional aspect of the previous black box system that has to interact with the environment.

As an introduction, the SDD contains a duplicate of the description of the system missions and context. Then the system architecture is presented. The functional architecture describes the functions or transformations that the system must perform. The flows (*i.e.*, information, energy or material flows) that are relevant to the functions are specified as well. The system internal behavior is also described and corresponds to the logical execution of the system functions.

We used the most common representation for functional architectures as a basis for the formalization. The functional architecture description is usually done with a kind of block diagram. The notation makes use of two basic elements, blocks and arrows. Blocks represent respective functions and arrows represent respective flows. An Arrow origin designates the function that produces the flow. Similarly, an arrow destination

designates the function that consumes the flow. Different levels of detail are supported with a decomposition of the blocks into sub-blocks. The formalization presented below enables to support this conceptualization.

The Concepts of Function and Flow. The universe of discourse of the functional architecture is more complex at Renault and specific to mechatonics systems. Nonetheless, we decided to stay sufficiently general with a small vocabulary as the opposite was going beyond our purpose (in section 2.2.1.5). The important concepts that we detail afterwards are *Function* and *Flow*. They are presented in figure 2.11.

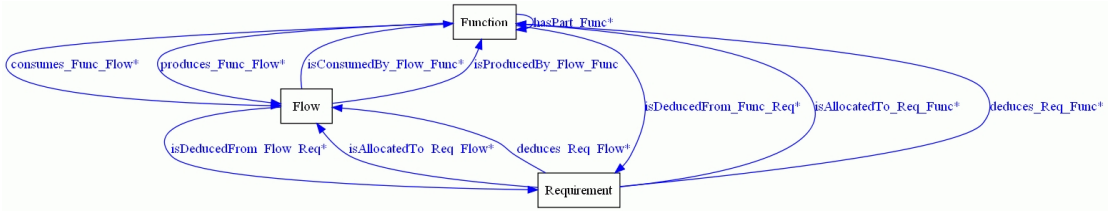

Figure 2.11: Formalization of the functional architecture

The class *Requirement* is also present in figure 2.11 to relate with the previous section and to emphasize that those concepts are tightly related to others (see section 2.2.1.5). Informally, as can be seen in figure 2.1, a prerequisite to the definition of the functional architecture is the specification activity. The specification (*i.e.*, a set of requirements) is actually the basis to the definition of the functions and the flows. This relation is materialized with properties $deduces\_Req\_Func$, $deduces\_Req\_Flow$, $isDeducedFrom\_Func\_Req$, $isDeducedFrom\_Flow\_Req$, $isAllocatedTo\_Req\_Func$ and $isAllocatedTo\_Req\_Flow$ that memorize the function origin or the flows in the specification. Those properties are not further detailed in this section, refer to section 2.2.1.5 on traceability for the complete description of the relations between the requirements and the functional architecture. A function can be decomposed into sub-functions through the relation $hasPart\_Func$. A function can produce and/or consume some flows. This is defined with respective relations $produces\_Func\_Flow$ and $consumes\_Func\_Flow$. We add the definition that a function has to produce or consume at least one flow (2.20). Otherwise, this function does nothing which is incorrect. The relations $isProducedBy\_Flow\_$-

$Func$ and $isConsumedBy\_FLow\_Func$ are also defined to record which functions are producers or consumers of a particular flow. Only $isProducedBy\_Flow\_Func$ is constrained quantitatively (the stars in the figure have no particular meaning) with an exact cardinality of 1. This is equivalent to defining a minimum and maximum cardinality of 1 to $isProducedBy\_Flow\_Func$ for $Flow$ and means that a flow is produced by only one function.

$$\exists\, y \quad produces\_Func\_Flow(x, y) \vee consumes\_Func\_Flow(x, y) \tag{2.20}$$

Details of Function and Flow Concepts. As with the requirements, the functional architecture accounts for both the system under consideration and its external environment. Also, based on the observations of the domain, we considered that it was not important to distinguish between different concepts of functions or flows resulting in the arguably low number of concepts illustrated in figure 2.12.

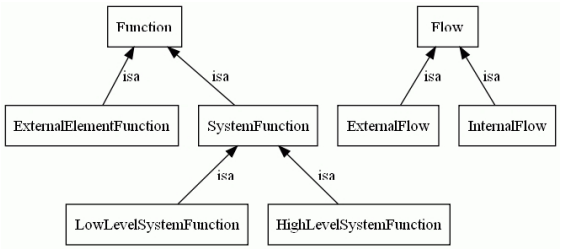

Figure 2.12: Formalization of the functional architecture

At the top of the hierarchies, we find the main classes $Function$ and $Flow$. They are subsumed along their external/internal connotation with the system (see section 2.2.1.4 for the distinction between external and internal flows). $ExternalElementFunction$ and $SystemFunction$ are defined as subclasses of $Function$ respectively for the functions of the external system elements and for the system functions. Similarly, $ExternalFlow$ and $InternalFlow$ are defined as disjoint subclasses of $Flow$. Finally, the system functions can be the object of decomposition and are organized in high level and low level functions, *i.e.*, $HighLevelSystemFunction$ and $LowLevelSystemFunction$.

First, the system functions. They correspond to functions realized by system constituents. The relations $consumes\_Func\_Flow$ and $produces\_Func\_Flow$ are not customized for the class nor its subclasses. Indeed, a system function can produce and consume either internal or external flows. In the case of external flows, they correspond to the interaction between the system and its environment. As seen with the requirements, the functions can be decomposed into sub-functions. We restrict the range of $hasPart\_Func$ to $System\_Function$ for $System\_Function$. This class is subsumed into $HighLevelSystemFunction$ and $LowLevelSystemFunction$ that respectively denotes system functions that are decomposed and the opposite, see axioms (2.21) and (2.22). By using those relations, the system functional architecture can be described at different levels of granularity. The finest grained description of the functional architecture corresponds to low level system functions. Coarser grained descriptions of the functional architecture use the high level system function concept abstracting the insignificant details. As such, a flow produced by a sub-function is apparent at the level of the super-function if it is consumed by a function that is not part of the super-function. Respectively, a flow consumed by a sub-function is apparent at the level of the super-function if it is produced by a function that is not part of the super-function. With the system functions hierarchy viewed as a directed acyclic graph, let $reach(v, R)$ be the reachable set of vertices (or nodes) starting from v and using R, we have axioms (2.23) and (2.24). With this structure, the finest grained description of the functional architecture corresponds to low level system functions and to functions of external elements. In order to memorize precisely which function is the producer and / or the consumer of a particular flow, we restrict $isProducedBy\_Flow\_Func$ and $isConsumedBy\_FLow\_Func$ to $LowLevelSystemFunction \cup ExternalElementFunction$ enabling to stay consistent with the cardinality restriction of exactly 1 for $isProducedBy\_Flow\_Func$ (otherwise, as we describe the functional architecture at different levels of abstraction, it would be possible for a flow to be produced by two different functions at different levels of abstraction).

$$\begin{aligned} \exists\, y \quad & HighLevelSystemFunction(x) \Rightarrow \\ & hasPart\_Func(x, y) \wedge [HighLevelSystemFunction(y) \vee \\ & LowLevelSystemFunction(y)] \end{aligned} \tag{2.21}$$

$$\nexists\, x \quad LowLevelSystemFunction(x) \wedge hasPart\_Req(x, y) \tag{2.22}$$

$$SystemFunction(w) \wedge hasPart\_Func(x,w) \wedge produces\_Func\_Flow(w,z) \wedge consumes\_Func\_Flow(y,z) \wedge (y \notin reach(x, hasPart\_Func)) \Rightarrow produces(x,z) \quad (2.23)$$

$$SystemFunction(w) \wedge hasPart\_Func(x,w) \wedge consumes\_Func\_Flow(w,z) \wedge produces\_Func\_Flow(y,z) \wedge (y \notin reach(x, hasPart\_Func)) \Rightarrow consumes(x,z) \quad (2.24)$$

Second, the external functions. For the same reasons that the external requirements are not decomposed (the project is responsible only for the system it develops), the external functions are not decomposed. *hasPart_Func* is disabled for *ExternalElementFunction* by defining a cardinality of 0 to the relation for the class. As external functions are outside the system perimeter, they produce and consume only external flows, which is made formal with the corresponding restrictions of the range of *produces_Func_Flow* and *consumes_Func_Flow* to *ExternalFlow* for the class *ExternalElementFunction*.

Finally, the only customization for the flows is the definition of the range of *isProducedBy_Flow_Func* to *SystemFunction* for the internal flows (*i.e.*, the internal flows are produced by only one system function).

2.2.1.4 On Physical Architecture

The physical architecture is the final abstraction level of the design. The system physical architecture is usually portrayed in the form of block diagram models similarly to the functional architecture and is part of the SDD. The physical architecture is the materialization of a solution that explains by what means the system functional architecture is realized. The solution also satisfies completely the technical requirements (in particular the non functional system requirements which were not yet taken into account by the design like cost, weight, dimensions, forbidden or authorized use of material, *etc.*).

The Concept of Component. Similarly to the functional architecture, we opted for a small vocabulary that applies in any scenario for the components description. This is represented in figure 2.13.

The classes *Requirement* and *Function* are also present in figure 2.13 as once again the concept of component is precise only through its relations with other concepts. The

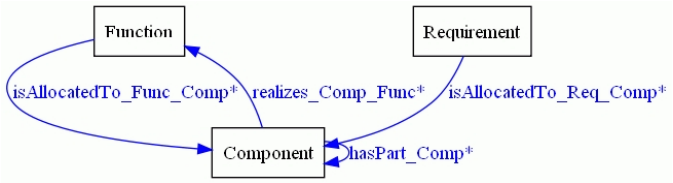

Figure 2.13: Formalization of the component concept

Component class is defined as general as possible and accounts for any type of component. The two *isAllocatedTo* properties describe part of the relation between the requirements, the functional architecture and the physical architecture. Ultimately, the only properties presented for the *Component* concept express the possible decomposition of a component into others and the set of functions that a component realizes, *i.e.*, $hasPart\_Comp$ and $realizes\_Comp\_Func$ respectively. The axioms on the range of these two relations for *Component* are represented in the figure by the class at the destination of the arrows that correspond to the relations. In this section we are only interested in the component concept (and its decomposition), the relations with the other concepts are presented in the next section.

Components Typology and Decomposition The physical architecture is presented in terms of components and how they are interconnected. In the universe of discourse, we first have the same distinction between internal and external components, respectively for system constituents and for external elements that contribute to the system. Then, the interconnections of those components make use of specific types of components that need to be distinguished. Finally, the rest of the typology concerns the mechatronics systems at Renault with the commonly encountered types of components. The different types of components present in the ontology are illustrated by a hierarchy in figure 2.14.
At the top of the hierarchy we have the general concept of *Component*. In the previous paragraph we mentioned that a component can be decomposed into subcomponents through the relation $hasPart\_Comp$ with *Component* as domain, co-domain and range (see figure 2.13).

The distinctions between internal, external components and their interconnections are respectively represented in the next level of the hierarchy in figure 2.14 with *System-*

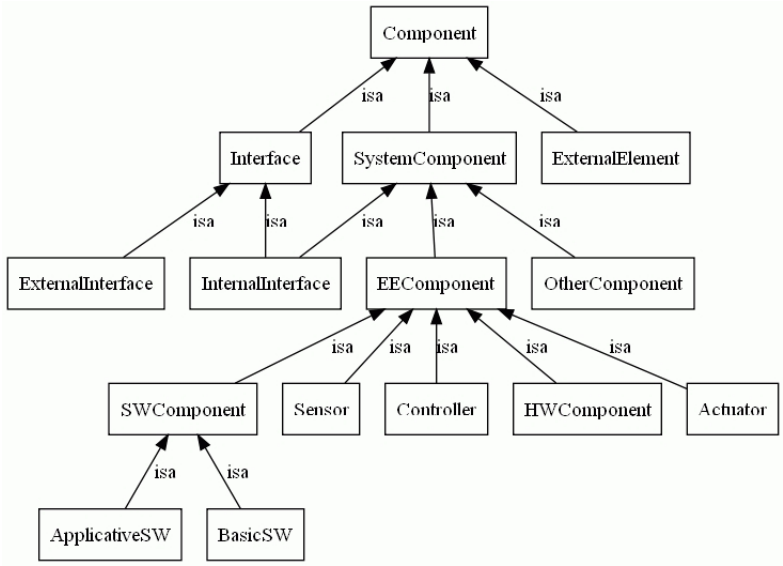

Figure 2.14: Components typology

Component, *ExternalElement* and *Interface*. *SystemComponent* and *ExternalElement* are the main types of components and the classes are disjoints with one another. The axiom on range for the different subclasses of *Component* is inherited to the subclasses following the axiomatization in appendix A. We assume that the range of the *hasPart_Comp* property is customized for each subclass to this specific subclass if not stated otherwise.

System and external components. The different types of system components concerns mechatronics systems at Renault where we distinguish between Electrical/Electronic components and components from other technologies. They are represented in the ontology with the disjoint classes *EEComponent*, *InternalInterface* and *OtherComponent*, subclasses of *SystemComponent*. The E/E components disjoint subclasses are *Sensor*, *Actuator* and *Controller*. They account only for the terminology used for mechatronics systems and we did not distinguish their differences in a formal way. The two other disjoint subclasses of *SystemComponent* are *HWComponent* and *SWComponent* that distinguish the hardware and software components. From the point of view of the development process, these two classes represent the system components that will be accounted for after the design phase as subsystems of the system. The components chosen to be further devel-

oped in a respective sub-development process are asserted as elements of *HWComponent* and/or *SWComponent* which is why the subclasses of *SystemComponent* are not all disjoint with each other. The range of *hasPart_Comp* is not customized for *Sensor*, *Actuator*, *Controller* and *HWComponent* (*i.e.*, the relation range is *EEComponent* for those classes) as a sensor and an actuator can have a controller as a part, a sensor can be a part of a controller and a hardware component can be composed of a software component. Finally, *SWComponent* is further subsumed into *BasicSW* and *ApplicativeSW*. The latter corresponds to traditional application software, in opposition with basic software that refers to a non functional abstract layer relative to the operating system, device driver, memory management, *etc.*

Interface components. An interface is a special type of component that defines the physical juncture between two or more components (internal or external). $Interface$ is a subclass of *Component* and is partitioned into disjoint subclasses $ExternalInterface$ and $InternalInterface$. These concepts are presented in figure 2.15.

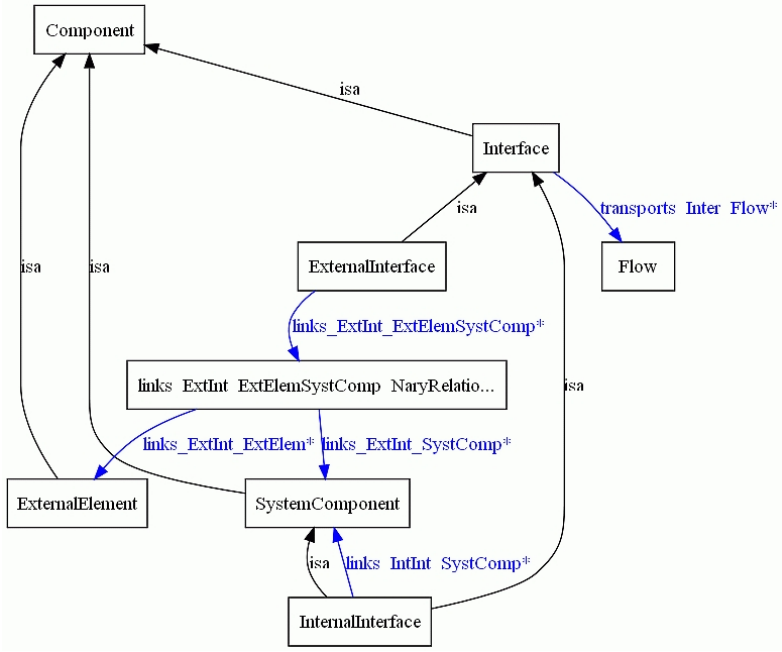

Figure 2.15: Formalization of the interface concept

First, we chose to disable the decomposition of interfaces into sub-interfaces by defining

the cardinality of $hasPart\_Comp$ to 0 for the class $Interface$.

Second, the external/internal distinction does not refer to the system as usual. The connotation refers to the physical juncture between components and corresponds to interfaces that link system components only (internal interfaces) or components which at least one is an external element (external interfaces). Therefore, an internal interface is a system component and an external interface can be a system component or an external element of the system depending on whether its development is the responsibility of the project or not. Formally, $InternalInterface$ is asserted as a subclass of $SystemComponent$ disjoint with $ExternalElement$ (see figure 2.14).

The interface considered as the physical juncture between components is formalized with respective links properties in figure 2.15. An internal interface is related to system components through $link\_IntInt\_SystComp$ with $InternalInterface$ and $SystemComponent$ as domain and co-domain. The relation is constrained with a minimum cardinality of 2. The relation range is customized to system components that are not interfaces.

For the external interfaces, we introduce an intermediate class $links\_ExtInt\_ExtElemSystComp\_NaryRelation$ in order to model the 3-ary relation between $ExternalInterface$, $ExternalElement$ and $SystemComponent$. As the class models a relation its name starts with a lowercase letter. The property $links\_ExtInt\_ExtElemSystComp$ relates $External\_Interface$ with the 3-ary relation as can be seen in figure 2.15. The property is given an exact cardinality of 1 (this is equivalent to define a minimum and maximum cardinality of 1) as an external interface that does not links components is incorrect. Then properties $links\_ExtInt\_ExtElem$ and $links\_ExtInt\_SystComp$ enable to assert which components are linked through external interfaces. The domain of those relations is $links\_ExtInt\_ExtElemSystComp\_NaryRelation$. Their range is respectively constrained to external elements and systems components that are not interfaces (*i.e.*, $ExternalElement \setminus Interface$ and $SystemComponent \setminus Interface$). The two correct cases accounted are either, the external interface links only one external element, which implies that it also links a system component, or the external interface links two external elements. Formally, the relation $links\_ExtInt\_ExtElem$ is only given a minimum cardinality of 1 as the axiomatization in appendix A does not allow to express that the external

interfaces link at least two components with one being an external element.

Finally, an interface is defined by the flows it transports which is formalized with the property $transports\_Inter\_Flow$ with $Interface$ and $Flow$ as domain and co-domain. The relation is given a minimum cardinality of 1 for the class $Interface$. It is not further customized for external interfaces as they can transport any type of flows. For $Internal\text{-}Interface$, the property range is constrained to $InternalFlow$ (that forms with $External\text{-}Flow$ a partition of the class $Flow$).

2.2.1.5 On Traceability

The main concepts of systems engineering have been presented in the previous sections. The remaining of the SDD document presents the allocation of the functions on the constituents and the allocation of the flows on the interfaces that transport them based on the functional and physical architectures. Lastly, a traceability table explains how requirements, functions and constituents are related.

Allowing the verification that all the requirements are taken into account by the design is one objective of *traceability*. Recall the definition of traceability from section 1.2.1.1.

> Traceability: *The ability to trace (identify and measure) all the stages that led to a particular point in a process that consists of a chain of interrelated events.*

In this work, traceability is implemented by using the different semantic relations that have been presented and the ones that relate the main concepts of systems engineering (presented in this section). As mentioned, a *trace* corresponds to a sequence of relations. By traceability we refer to the capability to define such traces which is inherently made possible by defining an ontology. In addition to this implicit capability, the ontology defines a hierarchical structure for the elements of design (*i.e.*, requirements, functions and components) and has been designed such that traceability is established from the requirements, through the functions, to the components along with the informal property that any non leaf element of a hierarchy is taken into account by the next level of abstraction if and only if all the leaf elements of the hierarchy are related to the next abstraction level. The cardinality constraints are used to define the existence of an element in relation to

another which ensures the consideration all the leaf elements in the next level of abstraction following the design process timeline.

The difficulty to define the ontology was to prevent misuse and misinterpretation due to the different possible, and sometimes incompatible, domain conceptualizations. The previous choices for the conceptualization were made to establish the traceability of the main elements of systems engineering and remove ambiguity while staying general enough to support different compatible conceptualizations. In this section, we present the justifications of the choices made in the previous sections for the conceptualization and how traceability is defined and established in the design process.

Traceability Between Needs and Requirements. In section 2.2.1.1, the concept of need has been formalized with the classes *Need*, *Mission*, *Goal* and *StrongConcept*. Figure 2.16 presents the properties that enables to move from the world of need to the world of requirements.

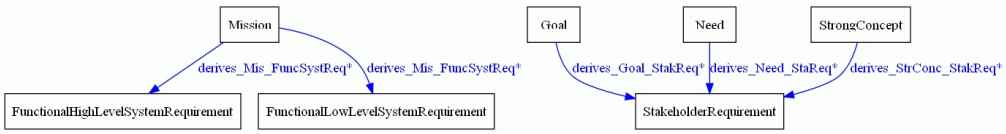

Figure 2.16: From the needs to the requirements

The derivation relations to the stakeholders requirements has already been presented in section 2.2.1.1. As a reminder, we defined minimum cardinality restrictions of 1 for the relations of each class. For the class *Mission*, we decided to relate directly to the functional system requirements with property $derives\_Mis\_FuncSystReq$ that ranges over $FunctionalHighLevelSystemRequirement \cup FunctionalLowLevelSystemRequirement$ and a minimum cardinality of 1. This makes explicit that a system mission should account only for functional aspect of the system. For *Need*, *Goal* and *StrongConcept*, we do not reason on the functional or non functional quality of the information recorded by those concept as they are related to stakeholders requirements. This distinction is done at the next level of abstraction when a stakeholder requirement is made more precise by the system or external requirements it is derived into.

Traceability at the Requirement Level. The structure and the traceability of the different requirement types have been defined in section 2.2.1.2. In that section we made precise for each type of requirement if it could be decomposed or not and how they were related to one another. In this paragraph, we do not return on those definitions but discuss on the objective of traceability that we want to establish and on the ambiguity due to requirements that remain informal text in this conceptualization.

Objective of traceability. As previously mentioned, the objective of the traceability defined with the ontology is to ensure that all requirements are taken into account by the design. With the ontology, we make sure that the elements of an abstraction level have left an explicit trail to the most abstract elements of the ontology, *i.e.*, there exist a trace from the most abstract ontology elements to the other ones. Informally, we distinguish between the abstraction level of need that enables to enter the world of requirements. Then, all the requirements need to be related to the functional level of abstraction and to the physical one.

In section 2.2.1.2, we presented the requirements structure that allows by its general nature to work with specifications (sets of requirements) that have a tree structure, those that allow multiple parents or those that have no sub-requirements. Then the relations that establish traceability between different types of requirements have been defined semantically to remove ambiguity from requirements structured along the project timeline. The decomposition relation ($hasPart\_Req$) is intended to express the hierarchical structure of a specific type of requirements and is assumed to establish part of the whole traceability in the design process that explains how an element is taken into account via traces. The relation expresses the inverse relational quality of parthood. It captures a whole to part paradigm of the class that corresponds to the natural understanding that the whole is equivalent to the sum of its parts. Informally, in the ontology, we captured the informal property that a node requirement is taken into account by its sub-requirements, *i.e.*, it is necessary and sufficient for any requirement to be taken into account that all its sub-requirements are taken into account. By extension, in this conceptualization, only the leafs requirements of the decompositions need to be taken into account by the design for all the

requirements to effectively be considered.

For example, in section 2.2.1.2, we defined the traceability between stakeholders requirements and requirements on the system and on the external elements, as the trace that ends with a *derives_Req* relation. We partitioned the stakeholders requirements into high level and low level ones to account for the different possible structures of requirements used in different projects. If the stakeholders requirements are not decomposed then they all correspond to low level stakeholders requirements and the traces correspond to the derivation relations. The other case presents the stakeholders requirements as a hierarchy (an element can have multiple parents or not) and each one is related, directly through derivation, or indirectly through decompositions and derivation, to the other level of abstraction (system or external requirements). In the end, all the low level stakeholders requirements are the object of at least one *derives_Req* relation in the formalization. This results in the property that all the stakeholders requirements are related by construction (directly or indirectly) to the next level of abstraction, *i.e.*, it exists a trace from any stakeholder requirement to at least one system or external requirement. Informally, all the stakeholders needs are effectively taken into account in the next level of abstraction as system or external requirements.

Ambiguity in the establishment of traceability. In our formalization, *a super-requirement is equivalent to the sum of its parts.* This is not always the case in the universe of discourse. Consider as an example the following functional high level system requirement: *"The electric brake system shall compute the driver's global deceleration will from the driver's command interfaces."* It is decomposed into four functional low level system requirements with the *hasPart_Req* relation: *"The electric brake system shall compute the driver's global deceleration will from the positions of the brake pedal, of the acceleration pedal and of the parking brake"*; *"The electric brake system shall interpret the position of the brake pedal"*; *"The electric brake system shall interpret the position of the acceleration pedal"* and *"The electric brake system shall interpret the position of the parking brake"*. It is clear that by considering those low level requirements, the high level one is also taken into account. With this general whole to part structure, it results by construction that

the nodes have effectively been taken into account if the leafs are related to the next level of abstraction. The incorrect use of the decomposition relation for the requirements corresponds to the interpretation that the whole is greater than the sum of its parts. In the example, this would correspond to removing the low level requirement that accounts on the computation of the driver's deceleration will. Only considering the low level requirements is erroneous as the aspect on computation is contained into the high level requirement. In this case, for the high level requirement to be taken into account, it is not enough that the low level requirements are related to the next abstraction level.

Considering the different possible structures of specifications, we have chosen that particular conceptualization as it is as general as possible and, precise and minimal, in the sense that only the necessary and sufficient requirements (the leafs) have to be related to the next abstraction level. To establish a meaningful traceability, the whole to part definition needs to be kept in mind in order to correctly use the ontology semantic relations: *a super-requirement is equivalent to the sum of its parts.*

Traceability Between Requirements and Design. Figure 2.17 presents the general concepts of systems engineering that are used for traceability, *i.e.*, *Requirement*, *Function*, *Flow* and *Component*. The respective *hasPart* relations enable to define the hierarchical structures of the information they record. The rest of the relations enable to establish the traceability between the different types of requirements and the elements of design.

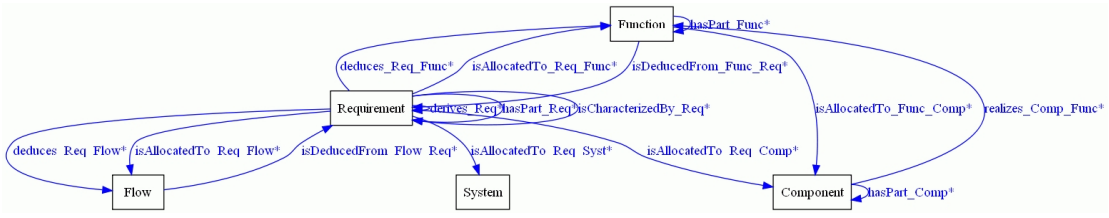

Figure 2.17: Traceability relations of the design process

First, the requirement concept records information at the system, functional or physical level of abstraction that correspond to the different levels of granularity in the systems engineering design process. This is formalized with the four *isAllocatedTo* properties that have *Requirement* as domain. The co-domains respectively correspond to the system,

functional and physical level, *i.e.*, $System$, $Function$ and $Flow$, $Component$. Those properties capture a partition of the requirements at these different levels of detail therefore all the requirements (except the stakeholders requirements) are allocated either to the system or some functions or some flows or some components (2.25). Moreover, those properties (except $isAllocatedTo\_Req\_Syst$) contribute to the establishment of traceability between requirements and design (they directly capture how an element of design is an answer to the consideration of a requirement).

$$\begin{array}{l}\neg StakeholderRequirement(x) \wedge \\ (isAllocatedTo\_Req\_Syst(x, syst) \oplus isAllocatedTo\_Req\_Func(x, func) \oplus \\ isAllocatedTo\_Req\_Flow(x, flow) \oplus isAllocatedTo\_Req\_Comp(x, comp))\end{array} \quad (2.25)$$

Second, the properties $derives\_Req$ and $isCharacterizedBy\_Req$ have been presented in section 2.2.1.2 and are used to establish the traceability between the different types of requirements illustrated in figures 2.4, 2.5, 2.6, 2.7 and 2.8. Informally, the different types of requirements are used respectively to record information corresponding to the system or to external system elements at system, functional or physical level of detail.

Third, some requirements can be involved by properties $deduces\_Req\_Func$ and / or $deduces\_Req\_Flow$. They semantically record the abstraction change from the requirement to the system functional architecture. In terms of traceability those properties capture how the requirements are realized by a solution for the system at functional level, in other words, how requirements are taken into account by functions and flows. Properties $isDeducedFrom\_Func\_Req$ and $isDeducedFrom\_Flow\_Req$ express that a function or a flow come from a requirement and they are used to establish backwards traceability (*i.e.*, all the relevant functions and flows can be related to a requirement).

Finally, the abstraction change from the functional to the physical level is formalized with $isAllocatedTo\_Func\_Comp$. In terms of traceability, the previous relation $deduces\_Req\_Func$ captures how the requirements are realized by a solution for the system at the functional level. The property $isAllocatedTo\_Func\_Comp$ captures how the the solution at the functional level is realized by a solution at the physical level. By transitivity, those two relations capture how the requirements are taken into account by the components (as they realize the functions that themselves take into account those requirements). The property $realizes\_Comp\_Func$ expresses for a component the functions it

realizes. This property has a minimum cardinality of 1 for *Component*, as a component that does not realize any function is useless.

The design process develops a solution for the system under development. This solution is described with the ontology main concepts that correspond to design process activities, see figure 2.1. The specification activity produces textual requirements. In the design process, these requirements are of utmost importance and need to be of high quality, *i.e.*, precise as defined by the ontology. As our objective is to ascertain that all requirements have effectively been considered by the development process, the traces we want to establish shall record changes from system level of abstraction, to functional and physical ones. We have the usual trail from requirement, to function, to component and also traces that record the consideration of non functional requirements. The first kind of trace actually starts with the functional requirements and explains how they are taken into account by the functional architecture, and then how the functions of the functional architecture are taken into account by system components. The non functional aspect of the system is more problematic. The consideration of non functional system requirements by the design is not as straightforward as the functional counterpart because there is not yet a precise conceptualization of the different types of non functional requirements and how they are taken into account by the design. Nevertheless, the main idea for the non functional requirements consideration is that they are expressed at the system level of abstraction and they should trace to the adequate level of abstraction (functional *or* physical).

Another kind of traces concerns backwards traceability where we ensure that elements of design effectively trace back to requirements. This is important to describe as downwards and backwards traceability do not define a bijection in the ontology. In this paragraph, we presented traceability at the level of detail presented in figure 2.17. The remaining of this section presents traceability at the system, functional and physical level by explaining all the relevant relations in figure 2.17. Note that section 2.2.1.6 gives more detail on non functional requirements.

Traceability at the System Level. System requirements (see figure 2.7) correspond to requirements at system level. They express the stakeholders needs. These needs are either

stakeholders requirements or system missions. They are related to system requirements through *derives_Req* (section 2.2.1.2) or *derives_Mis_FuncSystReq* (previously in this section). In section 2.2.1.2, we explained that the stakeholders requirements were the most abstract type of requirements. As such, they can record information at any level of detail (system, functional or physical level) and the allocation relations have been disabled to consider this imprecision of the concept (*isAllocatedTo_Req_Syst*, *isAllocatedTo_Req_Func*, *isAllocatedTo_Req_Flow* and *isAllocatedTo_Req_Comp* have a cardinality of 0 for *StakeholderRequirement*). The traceability at the system level corresponds simply to system requirements that define some characteristics of the system under development. Figure 2.18 only presents relations that are used to establish a trace from the system requirements to the system.

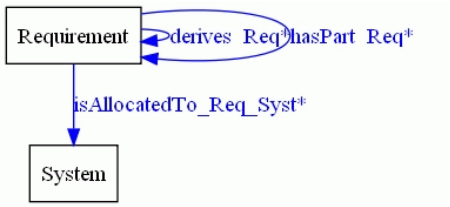

Figure 2.18: From the system requirements to the system

In this conceptualization, all system requirements are related to the system under development. This is formalized by giving an exact cardinality of 1 (minimum and maximum cardinality of 1) to the *isAllocatedTo_Req_Syst* property for the *SystemRequirement* class (2.26). The other relations of allocation (*isAllocatedTo_Req_Func*, *isAllocatedTo_Req_Flow* and *isAllocatedTo_Req_Comp*) are disabled for the class (the relations are constrained with a cardinality of 0 for *SystemRequirement*). Note that this is consistent with axiom (2.25). An informal rule to show that a requirement is at system level is that system requirements statements start with: *"The ___ system shall ..."*. The capability to allocate a set of system requirements to the system was also requested and corresponds to axiom (2.27).

$$\exists!\ y \quad SystemRequirement(x) \Rightarrow isAllocatedTo\_Req\_Syst(x, y) \tag{2.26}$$

$$\begin{aligned} &SystemRequirement(x) \wedge hasPart\_Req(x, y) \wedge \\ &\quad isAllocatedTo\_Req\_Syst(x, z) \Rightarrow isAllocatedTo\_Req\_Syst(y, z) \end{aligned} \tag{2.27}$$

Traceability at the Functional Level. The system description at the functional level corresponds ultimately to the system functional architecture presented in section 2.2.1.3. The functional architecture presents the functions and the flows of the system and the relevant functions and flows that are external to the system. Figure 2.19 depicts the relations used to establish traceability at the functional level.

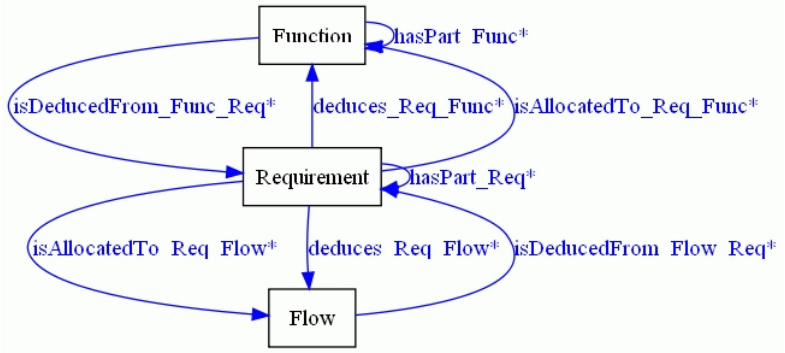

Figure 2.19: Traceability relations between requirements and functional architecture

Informally, traceability at the functional level relates the requirements to the functional architecture. The different kinds of requirements corresponding to functional level are functional system requirements, part of system's elements non functional requirements, and external requirements. Backwards traceability relates the functional architecture to functional requirements. In the following paragraphs we define how traceability is established precisely for each concept by using 2.19 relevant relations.

From the functional system requirements to the functional architecture. Functional system requirements correspond to the classes $FunctionalHighLevelSystemRequirement$ and $FunctionalLowLevelSystemRequirement$. As previously mentioned, these requirements express system level information. For these requirements to be taken into account at the functional level, we use the properties $deduces\_Req\_Func$ and $deduces\_Req\_Flow$. The relations used to establish a trace from functional system requirements to the functional architecture are presented in figure 2.20.

As system requirements account only for the system under consideration, the range of $deduces\_Req\_Func$ is restricted to $SystemFunction$ for $SystemRequirement$. The

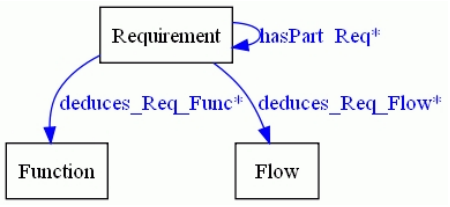

Figure 2.20: Traceability relations for the functional system requirements

range of $deduces\_Req\_Flow$ is not further customized as the described flows can be internal to the system or at the interface of the system and the environment. In this ontology, for all the functional system requirements to be considered at the functional level, it is necessary and sufficient only for the low level functional system requirements to be related to the next level of abstraction through our relations of deduction. More formally, $deduces\_Req\_Func$ and $deduces\_Req\_Flow$ are given a minimum cardinality of 1 for $FunctionalLowLevelSystemRequirement$. By construction, there is a trace from any functional system requirements to at least one function and one flow. These traces are composed of possible $hasPart\_Req$ and end with $deduces\_Req\_Func$ or $deduces\_Req\_Flow$. It is possible to relate a set of functional requirements with the usual semantics that the functions and flows are effectively related to the transitive closure of the functional requirements through $hasPart\_Req$ (low level requirements of the transitive closure are linked to functions and flows), see axioms (2.28) and (2.29).

$$FunctionalHighLevelSystemRequirement(x) \wedge hasPart\_Req(x,y) \wedge deduces\_Req\_Func(x,z) \Rightarrow deduces\_Req\_Func(y,z) \tag{2.28}$$

$$FunctionalHighLevelSystemRequirement(x) \wedge hasPart\_Req(x,y) \wedge deduces\_Req\_Flow(x,z) \Rightarrow deduces\_Req\_Flow(y,z) \tag{2.29}$$

From non functional requirements on elements of the system to the functional architecture. In section 2.2.1.2, we presented the non functional system requirements that correspond to non functional system characteristics. Those requirements are expressed at system level. The relation $derives\_Req$ is used to record the abstraction change from the non functional requirements at the system level to the non functional requirements at the level of the elements of design ($Function$, $Flow$ and $Component$).

This is formalized by disabling $isAllocatedTo\_Req\_Syst$ (a cardinality of 0 is defined) for the class $SystemElementNonFunctionalRequirement$. As can be seen in figure 2.8, the system element non functional requirements are partitioned into high level and low level ones. $hasPart\_Req$ expresses a whole to part relation with the implicit interpretation that the whole is equivalent to the sum of its parts. This results that, for all the non functional requirements on the elements of the design to be taken into account by the design, it is necessary and sufficient for only the low level ones to be related to design elements. Figure 2.21 presents the relations used to define a trace from non functional requirements on the system elements to the functional architecture.

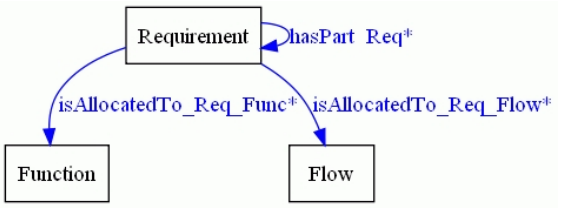

Figure 2.21: Traceability relations for the non functional requirements on the elements of the system

As it is not possible to distinguish between requirements that are taken into account by functions or flows or components, we have axiom (2.30) that uses the relations $isAllocatedTo\_Req\_Func$, $isAllocatedTo\_Req\_Flow$ and $isAllocatedTo\_Req\_Comp$ (with respective range $SystemFunction$, $Flow$ and $SystemComponent \cup InternalInterface$) in order to record how non functional requirements of the system elements are taken into account by the design (functional and physical). For the functional architecture, only $isAllocatedTo\_Req\_Func$ and $isAllocatedTo\_Req\_Flow$ are used as illustrated in figure 2.21. This imprecision for the allocation of the non functional requirements demonstrates that the concept is still ambiguous and that there are actually more types to be defined for the non functional requirements. We do not go further on the subject as section 2.2.1.6 is devoted to non functional requirements. The reader is invited to read this section for explanations concerning this concept. The capability to allocate a set of non functional requirements on system elements was also requested and corresponds to axioms (2.31) and

(2.32) respectively for a function and a flow.

$$\begin{aligned}&\exists func\ \exists flow\ \exists comp\\&NonFunctionalLowLevelSystemElementRequirement(x) \wedge\\&(isAllocatedTo\_Req\_Func(x, func) \oplus isAllocatedTo\_Req\_Flow(x, flow) \oplus\\&isAllocatedTo\_Req\_Comp(x, comp))\end{aligned} \tag{2.30}$$

$$\begin{aligned}&NonFunctionalHighLevelSystemElementRequirement(x) \wedge\\&hasPart\_Req(x, y) \wedge isAllocatedTo\_Req\_Func(x, z) \Rightarrow\\&\quad isAllocatedTo\_Req\_Func(y, z)\end{aligned} \tag{2.31}$$

$$\begin{aligned}&NonFunctionalHighLevelSystemElementRequirement(x) \wedge\\&hasPart\_Req(x, y) \wedge isAllocatedTo\_Req\_Flow(x, z) \Rightarrow\\&\quad isAllocatedTo\_Req\_Flow(y, z)\end{aligned} \tag{2.32}$$

From external requirements to the functional architecture. In section 2.2.1.2 and 2.2.1.3, we explained that it was not the project responsibility to develop its own environment. As such, external requirements and external functions are not the object of decomposition (they correspond to lists or enumerations with no specific structure, in particular they are not hierarchical). Similarly to the system, we distinguish between functional and non functional requirements, see figure 2.6. Relations used to define a trace from some external requirements to the functional architecture are presented in figure 2.22.

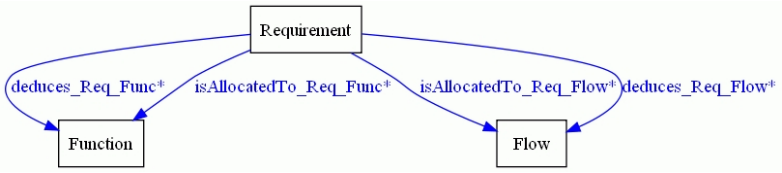

Figure 2.22: Traceability relations for the external requirements

Functional external requirements are related to the functional level through the properties *deduces_Req_Func* and *deduces_Req_Flow*. The range of *deduces_Req_Func* is constrained to *ExternalElementFunction* for *FunctionalExternalRequirement* and a cardinality of exactly 1 is defined as this function is outside the scope of the system (it is not under the responsibility of the project development team). In this way, the development team does not overlap in an intrusive way with other system projects. The range

of $deduces\_Req\_Flow$ is constrained to $ExternalFlow$ as flows produced or consumed by external functions are outside the scope of the system. A minimum cardinality of 1 is defined to constrain the relation for $FunctionalExternalRequirement$. For the non functional external requirements, it is not possible to distinguish between requirements that are taken into account by functions or flows or components. $isAllocatedTo\_Req\_Func$ and $isAllocatedTo\_Req\_Flow$ are the relations used to relate non functional external requirements and the functional architecture as shown in figure 2.22. The ambiguity for the concept is not solved but we define axiom (2.33) that explains the allocation of a non functional external requirements to either functions or flows or components. The ranges of $isAllocatedTo\_Req\_Func$, $isAllocatedTo\_Req\_Flow$ and $isAllocatedTo\_Req\_Comp$ are respectively constrained to $ExternalElementFunction$, $ExternalFlow$ and $ExternalElement \cup ExternalInterface$.

$$\begin{aligned}&\exists func\ \exists flow\ \exists comp \quad NonFunctionalExternalRequirement(x) \wedge \\ &isAllocatedTo\_Req\_Func(x, func) \oplus isAllocatedTo\_Req\_Flow(x, flow) \oplus \\ &isAllocatedTo\_Req\_Comp(x, comp))\end{aligned} \tag{2.33}$$

From the functional architecture to requirements. As previously mentioned, downwards and backwards traceability do not define a bijection. As such, we cannot use the same relations to ensure that the functional architecture is effectively related to the requirements.

For now, with downwards traceability, we ensured that the stakeholders needs were expressed as requirements on the system and on the system external elements. Then, the functional part of these requirements can trace to the functional architecture directly. Using cardinality constraints and requirements structure, we ensured that all the relevant functional requirements (low level functional system requirements and functional external requirements) were taken into account by the functional architecture (functions and flows). The converse is not true as it is still possible to have some functions and / or some flows defined in the functional architecture that are not related to any functional requirements. To ensure that this is not possible, we use the relations $isDeducedFrom\_Func\_Req$ and $isDeducedFrom\_Flow\_Req$ presented in figure 2.23.

As the flows are not the object of decomposition, we define $isDeducedFrom\_Flow\_$-

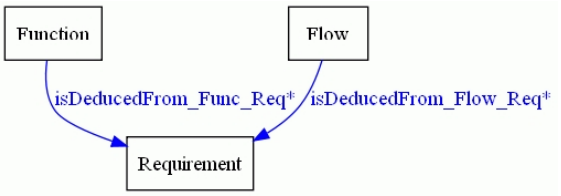

Figure 2.23: Traceability relations for the functions and flows

Req (with *Flow* and *Requirement* as domain and co-domain) as the inverse relation of *deduces_Req_Flow* (2.34). The range of $isDeducedFrom\_Flow\_Req$ is constrained to $FunctionalHighLevelSystemRequirement \cup FunctionalLowLevelSystemRequirement \cup FunctionalExternalRequirement$. Also, a minimum cardinality of 1 is given to $isDeducedFrom\_Flow\_Req$ for *Flow* ensuring that all the flows are related to at least one requirement.

For the functions, we define $isDeducedFrom\_Func\_Req$ with *Function* and *Requirement* as domain and co-domain. As a reminder, figure 2.11 presents the different types of functions that mainly correspond to system functions and the ones realized by elements external to the system. For the functions of the external elements, the range of $isDeducedFrom\_Func\_Req$ is customized to $FunctionalExternalRequirement$. The minimum cardinality of the relation is defined to 1 for $ExternalElementFunction$ ensuring that all functions of the external elements are related to requirements. For the system functions, the range of $isDeducedFrom\_Func\_Req$ is constrained to $FunctionalHighLevelSystemRequirement \cup FunctionalLowLevelSystemRequirement$. Similarly to the requirements decomposition, the decomposition of the system functions is done with the relation $hasPart\_Func$ that captures a whole to part paradigm of the class that corresponds to the natural understanding that the whole is equivalent to the sum of its parts. With this understanding, for all the system functions to be related to the requirements, it is sufficient for only the low level system functions to be related to the requirements. Formally, $isDeducedFrom\_Func\_Req$ is given a minimum cardinality of 1 for the class $LowLevelSystemFunction$. A high level system function deduced from a requirement means that its transitive closure through $hasPart\_Func$ is deduced from this requirement (2.36). And finally, because the system functions are structured with a hierarchy, we define only an

implication between $deduces\_Req\_Func$ and $isDeducedFrom\_Func\_Req$ rather than an equivalence (2.35). Informally, if a requirement is used to deduce a function, then the function is deduced from this requirement. With $deduces\_Req\_Func$ we record the functions *directly* deduced from a requirement. With $isDeducedFrom\_Func\_Req$ we record which requirements any low level system function is related to, *directly or not* hence the implication.

$$deduces\_Req\_Flow(x, y) \Leftrightarrow isDeducedFrom\_Flow\_Req(y, x) \tag{2.34}$$

$$deduces\_Req\_Func(x, y) \Rightarrow isDeducedFrom\_Func\_Req(y, x) \tag{2.35}$$

$$\begin{aligned} &HighLevelSystemFunction(x) \wedge hasPart\_func(x, y) \wedge \\ &isDeducedFrom\_Func\_Req(x, z) \Rightarrow isDeducedFrom\_Func\_Req(y, z) \end{aligned} \tag{2.36}$$

Traceability at the Physical Level. The system description at the physical level corresponds to the physical architecture. In section 2.2.1.4, we presented only the component concept as a hierarchy of components. However, the physical architecture is actually the description of a solution in (terms of components) that explains by what means the system functional architecture is realized. This description corresponds to the functional architecture with additional information on which components realize which functions and on some non functional characteristics of the components. Figure 2.24 presents the different relations used to establish traceability from the requirements to the physical architecture.

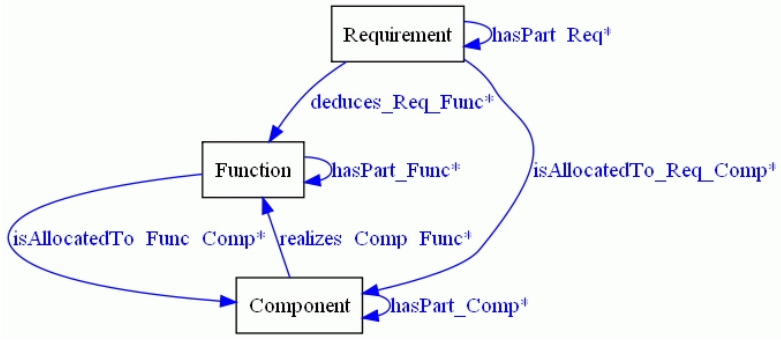

Figure 2.24: Traceability between requirements and physical architecture.

Similarly to the traceability at the functional level, the traceability at the physical level relates the requirements to the physical architecture. The physical architecture is the most

precise and final system abstraction in the design process. As such, the property that all the requirements have been addressed by the design (in particular the non functional system requirements that are not yet taken into account) must hold completely. Finally, backwards traceability relates the physical architecture to the functional requirements. In the following paragraphs we define how traceability is established precisely for each concept by using the relevant relations of figure 2.24.

From functional system requirements to the physical architecture. As previously presented in figure 2.20, the functional system requirements are taken into account by the functional architecture in terms of functions and flows. Figure 2.25 presents the conceptualization that explains how the functional system requirements are taken into account by the physical architecture. Only the relation *deduces_Req_Func* that pertains

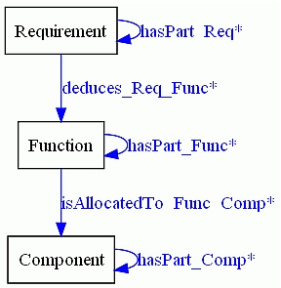

Figure 2.25: Traceability relations for the functional system requirements

to the functions of the functional architecture is presented in figure 2.25 as we define the abstraction change from the functional to the physical level with *isAllocatedTo_Func_Comp* to record how a system function is taken into account by the physical architecture. The information on the flows involved by the requirement is already captured by the function concept. Functional system requirements and system functions are also structured in a hierarchy with *hasPart_Req* and *hasPart_Func*. The *high level / low level* distinction enables to identify the leaf elements of those two hierarchies. We already explained that, by construction, for all the functional system requirements to be taken into account by the system functions, it is necessary and sufficient, only for low level functional system requirements to be the object of *deduces_Req_Func*. Now we need to make sure that all the systems functions are considered in the physical architecture. Because system

functions are inside the system perimeter, they need to be taken into account by system components (including interfaces). The range of $isAllocatedTo\_Func\_Comp$ is therefore restricted to $SystemComponent \cup Interface$ for the class $SystemFunction$. Similarly to $hasPart\_Req$, $hasPart\_Func$ captures a whole to part paradigm that corresponds to the whole being equivalent to the sum of its parts. Given this interpretation, for all the system functions to be taken into account by the physical architecture, it is necessary and sufficient that all low level system functions are related to components. Allocating the same function on more than one component is incorrect. This actually corresponds to an error in the universe of discourse as such functions actually correspond to two *different* functions that happens to be similar. Formally, an exact cardinality of 1 is given to $isAllocatedTo\_Func\_Comp$ for $LowLevelSystemFunction$. As usual the capability to allocate a set of functions was desired with the usual semantic that it is effectively the transitive closure of the high level system function through $hasPart\_Func$ (in particular the low level system functions that belong to this transitive closure) that is allocated to the concerned component (axiom 2.37). Finally, the traces that are defined using $hasPart\_Req$, $deduces\_Req\_Func$, $hasPart\_Func$ and $isAllocatedTo\_Func\_Comp$ ensures that all the functional system requirements have actually been taken into account by the physical architecture.

$$\begin{aligned} &HighLevelSystemFunction(x) \wedge hasPart\_Func(x,y) \wedge \\ &isAllocatedTo\_Func\_Comp(x,z) \Rightarrow isAllocatedTo\_Func\_Comp(y,z) \end{aligned} \tag{2.37}$$

From non functional requirements on system elements to the physical architecture. As a reminder, part of the non functional requirements on the elements of design corresponds to the system description at functional level (see figure 2.21). As already mentioned, it is not possible to distinguish between the requirements that are taken into account by functions or flows or components however the non functional requirement should be precise enough to be related exclusively to functions or flows or components (2.30). To record how a non functional requirement is taken into account by the physical architecture, we use $isAllocatedTo\_Req\_Comp$ (see figure 2.24. Section 2.2.1.6 addresses the non functional requirements so we do not elaborate on the subject. As usual, the capability to allocate a set of non functional requirements (at the physical level) to a component

is provided (2.38).

$$\begin{aligned} &NonFunctionalHighLevelSystemElementRequirement(x) \wedge \\ &hasPart\_Req(x,y) \wedge isAllocatedTo\_Req\_Comp(x,z) \Rightarrow \\ &\quad isAllocatedTo\_Req\_Comp(y,z) \end{aligned} \tag{2.38}$$

From external requirements to the physical architecture. As the development of the environment is not the responsibility of the project, decomposition of external elements is not done in the ontology. Similarly to system requirements, external requirements are partitioned between functional and non functional ones (see figure 2.6) and their traceability is established using the same relations presented before. The functional external requirements are traced to the physical architecture indirectly through the functional architecture with relations $deduces\_Req\_Func$ and $isAllocatedTo\_Func\_Comp$ (see figure 2.25). Their range is respectively customized to $ExternalElementFunction$ and $ExternalElement \cup ExternalInterface$ recording how a functional external requirement is related to an external function which is taken into account by the physical architecture. An external function that is not allocated to an external element is incorrect so $isAllocatedTo\_Func\_Comp$ is given an exact cardinality of 1. Concerning the non functional external requirements, they are related to the physical architecture with $isAllocatedTo\_Req\_Comp$ which ranges over $ExternalElement \cup ExternalInterface$. As previously defined, the non functional external requirements are allocated to either functions or flows or components (2.33).

From the physical architecture to requirements. With downwards traceability, we ensured that it was possible to follow the trail left by any requirement for its consideration by the functional and / or physical architectures. Conversely, we have to make sure that all the components of the physical architecture are effectively related to some requirements. We use property $realizes\_Comp\_Func$ to record for a component the functions it realizes. This property has a minimum cardinality of 1 for $Component$ as a component that does not realize any function is incorrect in our conceptualization. A component realizes the functions that are allocated to it (2.39) and a component that is decomposed into subcomponents realizes the functions of its subcomponents (2.40). Finally, we customized the range of $realizes\_Comp\_Func$ for $SystemComponent$ and $ExternalElement$ respec-

tively to $SystemFunction$ and $ExternalElementFunction$.

$$Function(x) \wedge isAllocatedTo\_Func\_Comp(x, y) \Rightarrow realizes\_Comp\_Func(y, x) \tag{2.39}$$

$$Component(x) \wedge hasPart\_Comp(x, y) \wedge realizes\_Comp\_Func(y, z) \Rightarrow realizes\_Comp\_Func(x, z) \tag{2.40}$$

2.2.1.6 On Non Functional Requirements

In section 2.2.1.5 we presented how the requirements were interrelated with the elements of design. The consideration of functional requirements is straightforward. They result into identified elements of design (*i.e.*, functions, flows and components, see figure 2.25) with adequate semantic. Informally, they describe some functional characteristics that correspond to some functions and flows. These functions and flows are designed into a functional architecture by describing for each function the flow it consumes and produces. Then this functional architecture is realized by defining the components that implement those functions. As we were mainly interested in the functional aspect of the system, we did not make any attempt to be as explicit with non functional requirements. The main encountered difficulties concern, on the one hand, the number of different types of non functional requirements and, on the other hand, the intentional limited expressive power given by axiomatization in appendix A. We therefore opted to formalize the non functional requirements in a general manner by recording how they were taken into account by the elements of design with respective allocation properties on the system, functions, flows and components (see figure 2.17 and axiom (2.25)). This section explains how the non functional requirements concept are intended to be manipulated in the ontology and we give some leads for their further formalization.

Formalization of Non Functional Requirements. In the following, we describe the formalization of system non functional requirements. Those descriptions are similar for the non functional requirements that are relevant to the environment. As a first attempt to capture the different types of non functional requirements, we defined $hasType\_Req$ as an attribute of a requirement. The relation ranges over an enumerated set that defines

the type of the non functional requirement that we call constraint. We listed *Temporal performance*, *Cost* and *Weight*. For each class of requirement (see figures 2.6, 2.7 and 2.8) we give a minimum cardinality of 0 or 1 to $hasType\_Req$ depending on the requirement being functional or not. As we have seen in section 2.2.1.5, a non functional requirement is first allocated to the system and then needs to be addressed by the architecture (functional or physical).

Temporal performance constraint. The notion of temporal performance constraint expresses how much time is needed to perform a task. It is associated to the functional aspect of the system therefore a requirement with information on temporal performance is related to exactly one functional system requirement through $characterizes\_Req$ (see figure 2.3). Given the axiomatization in appendix A, it is not possible to customize the relation just for the non functional requirements with temporal performance type so we have axiom (2.41). In order for all the systems requirements to be related to temporal aspects, it is necessary and sufficient for all the low level ones to be related to a temporal constraint. We define $characterizes\_Req$ as the inverse property of $isCharacterizedBy\_Req$ (2.42) and constrain the former relation with a cardinality of 1 for the functional low level system requirements. This enables to describe at system level temporal constraints corresponding to functional requirements. For example, we have the following non functional requirement that characterizes a similar functional requirement that does not describe the temporal performance aspect: *"The electric brake system shall compute the driver's global deceleration will from the driver's command interfaces in less than 0.02 seconds."* This temporal performance needs to be taken into account at design level. The non functional requirement at system level is expressed at the architectural level by using $derives\_Req$: *"Computing the driver's global deceleration request shall be performed in less than or equal to 0.02 seconds."* In this particular case, the non functional requirement on the system elements corresponds to the functions that are deduced from the functional requirement. In this example we have only one function with the following label: *"Compute driver's global deceleration request"*. The correspondence is made explicit by allocating the requirement to the function with $isAllocatedTo\_Req\_Func$. In the end,

a non functional requirement on system elements, that is derived from a non functional system requirement which itself characterizes a functional requirement, is allocated to the functions deduced from the functional requirement (2.43).

$$\exists!\ y \quad Requirement(x) \wedge hasType\_Req(x, "Temporalperformance") \Rightarrow characterizes\_Req(x, y) \tag{2.41}$$

$$Requirement(x) \wedge characterizes\_Req(x, y) \Leftrightarrow isCharacterizedBy\_Req(y, x) \tag{2.42}$$

$$hasType\_Req(x, "Temporalperformance") \wedge characterizes\_Req(x, y) \wedge derives\_Req(x, z) \wedge deduces\_Func(y, f) \Rightarrow isAllocatedTo\_Req\_Func(z, f) \tag{2.43}$$

In order to formalize the concept of temporal performance in terms of semantic relations, we chose to define an attribute for *System* and *Function* with properties *hasExecutionTime_Syst* and *hasExecutionTime_Func* that ranges over positive integers (all the execution times of the system and functions shall be expressed with the same unit, seconds in the example). The intention is similar to the consideration of functional requirements by the system and functions deduced from requirements. The development of the defined functions implicitly answers the requirements. In this conceptualization, a temporal performance constraint (a non functional requirement with type attribute asserted to temporal performance) is taken into account by the development of the system functions that respects the execution time attribute. By following the definitions given in 2.2.1.5, for all the temporal performance constraints at system level to be taken into account by the design, it is necessary and sufficient for all the low level ones to derive at least one temporal performance constraint at architectural level. The range customization of the derivation relations for only temporal performance constraints at system level into temporal performance constraints at architectural level is not possible with the axiomatization given in appendix A, so we assert axiom (2.44). In the end, for all the temporal performance requirements at architectural level to be taken into account by the design, it is necessary and sufficient for all the low level ones to be allocated to at least one function. Finally, the system and all the functions need to be associated to an execution time so *hasExecutionTime_Syst* and *hasExecutionTime_Func* are customized with an exact cardinality of 1.

$$Requirement(x) \wedge hasType\_Req(x, "Temporalperformance") \wedge derives\_Req(x, y) \Rightarrow hasType\_Req(y, "Temporalperformance") \tag{2.44}$$

For now, we did not attach any semantic with the structure of the functions and their execution times so the concept of temporal performance is still informal in this sense (it results that the definition of the functions execution time remains entirely manual). The positive aspect is that the concept is defined in a general manner allowing to use the desired interpretation, with some caution. For instance, one can implicitly associate the interpretation that the execution time of a super function is equal to the sum of the execution times of its direct sub-functions, and that the execution time of the system is equal to the sum of the execution times of the top level functions. However, in the universe of discourse, there is many more licit interpretations that are not always compatible, but are possible given the general nature of the ontology. We have for example functions that are executed in parallel that can be described as the decomposition of a super-function. The execution times of the sub-functions do not add up in the super function.

Weight and cost constraints. In the ontology, the notions of weight and cost correspond to the physical architecture. Weight is intricately related to the notion of mass that is attached to materials and, intuitively, it is related to the physical components. In the universe of discourse, it is possible to talk about the cost of a function however we chose to let the functional architecture as an abstract object that needs to be materialized by components that have a price. As previously mentioned in section 2.2.1.5, a non functional requirement is first expressed at the system level. Then it needs to be addressed at the functional or physical level. This abstraction change is done by using $derives\_Req$ from non functional system requirements to non functional requirements on system elements. Informally, non functional requirements with weight and cost types at system level are intended to respectively derive only weight and cost constraints on the elements of design (2.45) and (2.46). These constraints are finally taken into account by the components of the physical architecture by using $isAllocatedTo\_Req\_Comp$. Four attributes are defined to formalize the concepts of weight and cost with $hasWeight\_Syst$ and $hasPrice\_Syst$ for the system, and $hasWeight\_Comp$ and $hasPrice\_Comp$ for the components. The relations co-domain are positive integers (all the weights and costs shall be expressed with the same unit). Those four attributes enable to formalize the informal constraints expressed

by non functional requirements with information on weight and cost. Constraints must be taken into account when developing the components that have to respect weight and cost attributes. Given the typology of components presented in section 2.14, $hasWeight\_Comp$ and $hasPrice\_Comp$ are customized with a cardinality of 0 for the external elements and interfaces. For all the other classes, $hasPrice\_Comp$ is given an exact cardinality of 1. The interfaces in the ontology are physical interfaces so $hasWeight\_Comp$ is given an exact cardinality of 1 for $InternalInterface$ and also for the other technology components. Then we have the hardware and software components. The former has some weight while the latter has not so the cardinality of $hasWeight\_Comp$ is respectively 1 or 0. Property $hasWeight\_Syst$ is not further constrained as a system can be constituted of only software components with no weight and $hasPrice\_Syst$ is customized with a cardinality of 1.

$$\begin{array}{l} Requirement(x) \wedge hasType\_Req(x, "Weight") \wedge \\ derives\_Req(x, y) \Rightarrow hasType\_Req(y, "Weight") \end{array} \tag{2.45}$$

$$\begin{array}{l} Requirement(x) \wedge hasType\_Req(x, "Cost") \wedge \\ derives\_Req(x, y) \Rightarrow hasType\_Req(y, "Cost") \end{array} \tag{2.46}$$

Similarly to the temporal performance constraints, we did not attach any semantic to the cost concept as costs of two similar software components do not add up into system cost. For the weight, the intuitive interpretation is given that the weight of a component is equal to the sum of the weight of its sub-components (2.47) and the system weight is equal to the sum of the weight of top level components (2.47). The property $hasConstituent\_Syst\_Comp$ is used to record the system constituents (it excludes external elements) and sum is a function that returns the sum of a set of integers. As a note, in the universe of discourse, the sub-components of a component can correspond to the whole transitive closure through $hasPart\_Comp$. This is equivalent to define the relation as transitive (*i.e.*, $hasPart\_Comp(x, y) \wedge hasPart\_Comp(y, z) \Rightarrow hasPart\_Comp(x, z)$) which would result in counting several times the same quantity. This is one of the reasons why the decomposition relations are not transitive in the ontology.

$$\begin{array}{ll} let & \\ & Z = \{z \mid hasPart\_Comp(x, y) \wedge hasWeight\_Comp(y, z)\} \\ & a = sum(Z) \\ in & Component(x) \wedge hasPart\_Comp(x, y) \wedge \\ & hasWeight\_Comp(y, z) \Rightarrow hasWeight\_Comp(x, a) \end{array} \tag{2.47}$$

$$
\begin{array}{ll}
let & \\
 & Z = \{z \mid hasConstituent\_Syst\_Comp(x,y) \wedge \\
 & \quad \neg hasPart\_Comp(nonexist,y) \wedge hasWeight\_Comp(y,z)\} \\
 & a = sum(Z) \\
in & System(x) \wedge Component(y) \wedge hasConstituent\_Syst\_Comp(x,y) \wedge \\
 & \quad \neg hasPart\_Comp(nonexist,y) \wedge hasWeight\_Comp(y,z) \Rightarrow \\
 & \quad\quad hasWeight\_Syst(x,a)
\end{array}
\tag{2.48}
$$

Generalization for the Consideration of Non Functional Requirements. In a general manner, for a requirement to be taken into account by the design, it needs to be allocated to the correct elements of the design process (*i.e.*, system, functions, flows or components). If this is sufficient for the functional requirements, the non functional requirements address additional characteristics of the elements of design that need to be formalized. The previous paragraphs served as examples for the consideration of non functional requirements with three different types of non functional requirements. Those three types of non functional requirements provide additional information on the elements of the design process such as system and components prices. This additional information needs to be formalized in order to be recorded. For the types of non functional requirements presented, the concept formalization has been done by defining attributes on the elements of the design process. Ultimately, once the design phase of the development process is completed, it is possible to realize a product based on the design. The resulting product needs to exhibit all the characteristics (functional and non functional) defined in the design phase, hence the importance for the design to be as precise and as explicit as possible. Moreover, the formalization (from the universe of discourse to a conceptualization) of the non functional aspects of the system and the architecture provides the necessary basis onto which system verification and validation can be defined. This provides some elements of answer concerning characteristics for requirements to be verifiable while reducing the gap between the informal world of requirements and models.

Finally, the ontology enables to record all the non functional requirements and makes precise how those tightly related to the elements of design are addressed. Some other types of non functional requirements can be recorded into the ontology but have not been formalized as a concept. Consider for example process requirements that are systems requirements. They do not need to be taken into account by the design as they are related

to other concepts such as processes, activities, manpower and roles, *etc.* There is two alternatives to formalize all the different types of requirements. We chose to use a type attribute to represent some different types of non functional requirements which is more natural at Renault. However, the axiomatization given in appendix A does not allow complex axioms to be formulated. It therefore requires the assertion of those formulas outside the framework given by the axiomatization ((2.44) for instance). Another alternative would be to restrict ourselves to the axiomatization framework which results in the introduction of a class for each type of non functional requirement. Both approaches are equally valid because we used FOL to describe the ontology. In practice, it depends on the expressiveness and semantic of the implementation language(s) (for instance, axiom (2.41) cannot be written in OWL and SWRL and needs to be verified manually with SQWRL).

2.2.1.7 Conclusion

In this section, part of systems engineering domain has been formalized with the focus on the system during the design process and its functional aspect. The general concepts of $Need$, $Requirement$, $Function$ and $Component$ have been presented in detail with their semantic relations that enable traceability between design process elements. This formalization enables to record quality information produced by the design process as structured knowledge which facilitates understanding information and therefore its re-usability. At Renault, systems engineering definition and deployment is currently undergoing. The development of a system depends on different engineering fields with conceptualizations relevant to their domain. In this picture, systems engineering should be viewed as the orchestrator of the different other engineering fields that contributes to the system development. The formalization of systems engineering is therefore fundamental as it contains the different system descriptions which are studied from the point of view of other engineering fields. In order to illustrate the centrality of systems engineering, we formalized another engineering field into a different ontology. This is the object of section 2.2.2 that presents functional safety domain for safety critical systems.

2.2.2 Functional Safety Ontology

The consideration of functional safety in the automotive domain has recently been given central attention with the recent arrival of ISO 26262 international standard. As the main objective of this work is to improve the design process of safety critical systems at Renault, the conceptualization of functional safety has been formalized to benefit from the opportunities provided with the intended compliance with ISO 26262. Figure 2.26 presents the general process for functional safety currently followed at Renault.

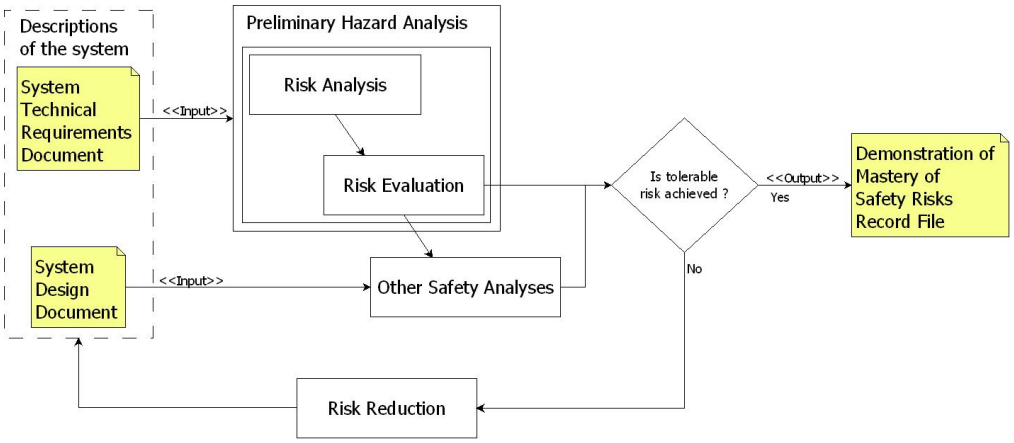

Figure 2.26: Functional safety process at Renault

The objective of the functional safety process is to demonstrate that the system is safe (from unacceptable risks). Generally speaking, the different system abstractions during the design process are analyzed with adequate techniques and results of the activities that demonstrate freedom from unacceptable risk are recorded in the *Demonstration of Mastery of Safety Risks Record file* (DMSRRF) as can be seen in figure 2.26. Safety analyses study the misuse or dysfunctional aspects of the system. If the results of these safety studies do not demonstrate that the corresponding system description is safe, risk reduction is performed which impacts the description(s) of the system. The first activity, Preliminary Hazard Analysis (PHA), implements risk analysis and risk estimation from the risk management process (see figure 1.6). It serves to identify system hazards and to estimate their associated risk. Then risk evaluation gives an integrity attribute to the system. The other safety activities are not completely supported by the ontology so they are represented in a

general manner in the figure. These safety activities implement the overall process of risk management at finer grained levels of abstraction. Those activities must bring evidence that the root causes of potential failures do not impact on the system integrity attribute demonstrating that the system is safe. Positive results are recorded into the DMSRRF and negative ones lead to redesign with the risk reduction step. The functional safety ontology presented in this section has been designed to promote a more systematic execution of this process with the focus on the fundamental concepts of functional safety domain and the PHA. Section 2.2.2.1 presents risk analysis with the different system descriptions, their respective failure models, the identification of hazardous events, the estimation of corresponding risks, and the related fundamental notions of functional safety. Section 2.2.2.2 presents risk evaluation and is devoted to the notion of integrity. Finally, section 2.2.2.3 gives the conclusion to the formalization of functional safety.

2.2.2.1 Risk Analysis

In the overall process of risk management (see figure 1.6), risk analysis starts with the definition of intended use and reasonably foreseeable misuse of the system. The system hazards are identified based on this definition. Then the system's risk is estimated. This general process can be applied at different levels of detail. The formalization presented below captures a conceptualization of this general process and enables to support the functional safety studies performed during system design process.

System's Descriptions. Functional safety, as its name implies, is interested in the study of the functional aspects of a system with respect to safety in order to handle functional and technological issues. These issues are relative to different system's descriptions which are the responsibility of systems engineering. As such, the STR and the SDD are prerequisites to functional safety examination as can be seen in figure 1.6. The ontology therefore contains similar classes presented in section 2.2.1. Figure 2.27 presents the elements that correspond to different system descriptions considered by functional safety .

The class *Requirement* records the system requirements. A safety attribute is defined for the class with property $safety\_Req$ that ranges over booleans *true* and *false*. It has two

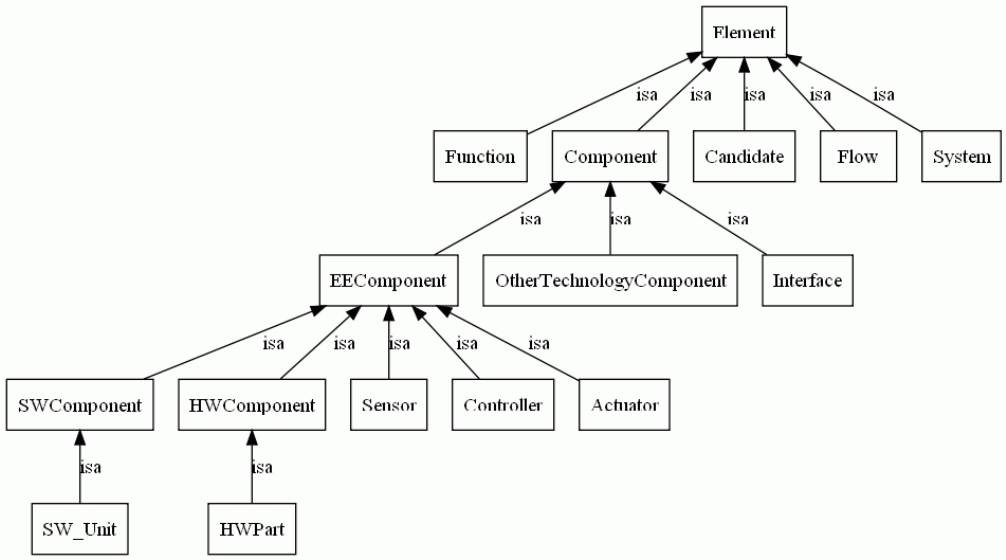

Figure 2.27: Elements of functional safety

subclasses, *FunctionalRequirement* and *SafetyRequirement*. A functional requirement corresponds to the concept presented in systems engineering. The class is used to record a functional description of the system in terms of requirements. A safety requirement is a requirement that is recognized to have an impact on safety and has the safety attribute defined to true, *i.e.*, the range of $safety\_Req$ is constrained to be *true* for each instance of the class. As a functional requirement can have an impact on safety, it is possible for a requirement to belong to the two classes so they are not disjoint with one another. The concept of requirement in functional safety is further defined in section 2.2.2.2, for now, we are interested only in functional requirements that correspond to a functional system description. As mentioned, under the notion of element, we find concepts similar with systems engineering ones such as *System*, *Function*, *Flow* and *Component*. The definitions of those concepts has been presented in section 2.2.1. The following focus on the new notions introduced by ISO 26262. The class *Candidate* is used to record existing systems that can potentially be the solution of the development process. ISO 26262 is applicable for Electric and Electronic (E/E) systems and corresponds to Renault conceptualization of the components presented in section 2.2.1.4. Classes HW_Part and SW_Unit are introduced respectively as the lowest level components for hardware and software architectures. Finally, this formalization enables to partially capture three different system descriptions:

its functional specification (a set of functional requirements), its functional architecture and its physical architecture. Based on these descriptions, functional safety seeks to identify and handle functional and technological issues that impact safety, which is the study of dysfunctional aspects of the system.

Failures and their Consequences. The system descriptions presented above can be denoted as *functional* in the sense that they present the system assuming that it performs correctly (without errors) and safely. Functional safety is interested in the demonstration that the system exhibits the safety property, *i.e.*, it is free from unacceptable risk, when it performs normally and also when it does not. The notion of risk therefore needs to be made precise along with the descriptions of erroneous functioning. Theoretically speaking, we want to formalize the fundamental chain of fault, error and failure (see figure 1.5) that explains dependability threats (naturally they include safety threats). This chain is related to system descriptions in the following way. A fault is dormant in a requirement, function or component. When activated, the fault becomes an error. The error is propagated and when it gets to the system boundary, it becomes a failure. We use a failure model to describe the system dysfunctional aspects in terms of failures. The relation with safety is then given by the identification of hazardous events and the estimations of their risks to the system.

Failure model. In order to describe the system dysfunctional aspects, we implemented a failure model that adds up to the system functional descriptions. Figure 2.28 presents the different failure modes considered by our failure model.

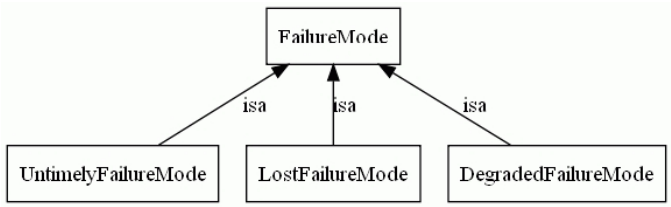

Figure 2.28: Failure model

A failure mode is the manner in which an element (*e.g.*, requirement, function or

component) fails. As it can be seen in figure 2.28 we consider three different types of failure modes: *degraded*, *lost* and *untimely*. The term *mode* corresponds to modal logic and expresses modality on the correct functioning of the system (services). Degraded, lost and untimely express respectively a reduction in the quality of the provided service (*e.g.*, underperforming or overperforming braking service), a service loss and the providing of a service at inappropriate time (in general untimely corresponds to too early or too late but the term is also used to refer to service provided timely but in a jerky manner. For instance when calling for the braking service, it can sometimes manifest itself in quick successions of braking and no braking actions felt like jolts by the driver). As such, this failure model is applied to systems descriptions. For reasons of conciseness, only relations with requirements are presented. Similar classes and relations that describe the dysfunctional aspect (*i.e.*, the failures) of the system, function, flow and component concepts exist but add nothing to the discussion. Figure 2.29 illustrates the application of the failure model when dealing with the system description in terms of functional requirements.

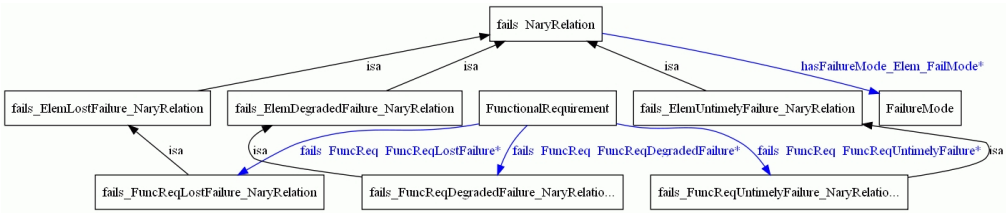

Figure 2.29: Application of the failure model on the functional requirements

The class $fails\_NaryRelation$ and its subclasses represent the failure model application on considered elements. In particular, the three leaf classes represent respective applications of the three different failure modes on functional requirements and correspond to failures as defined in section 1.2.2.2. The three relations from $FunctionalRequirement$ express failure modalities of functional requirements respectively in terms of degradation, loss or untimely execution of functionality. As the whole set of functional requirements can actually correspond to different system descriptions, the relations are not yet constrained in terms of cardinality but the capability to describe the dysfunctional aspect of a functional requirement is set up. The property $hasFailureMode\_Elem\_FailMode$ records which modality is applied to a functional requirement and is customized with a cardinality

restriction of exactly 1. The range of the relation is constrained for the leaf classes to their adequate failure mode (see figure 2.28). This automatically records the correct failure mode of the failure model that is applied as a modality of the functional requirement. Finally, this formalization enables to systematically consider the dysfunctional aspect of the system by applying the failure model on its functional requirements. The next paragraphs formalize the remaining of risk assessment (see figure 1.6) with the change from failures (given by the application of the failure model) to feared system events and their correspondence with the general notion of risk (*i.e.*, risk of harm).

Hazardous events identification. The system dysfunctional aspect obtained by application of our failure model is the prerequisite to the identification of hazardous events. Feared events and hazardous events can be used indifferently in this ontology as we do not consider feared events that have no relation with harm. A hazardous event occurs when the hazard's potential to cause harm is realized. Hazardous events are identified considering previously produced failures that lead to harm. Figure 2.30 presents the conceptualization that corresponds to the hazardous events.

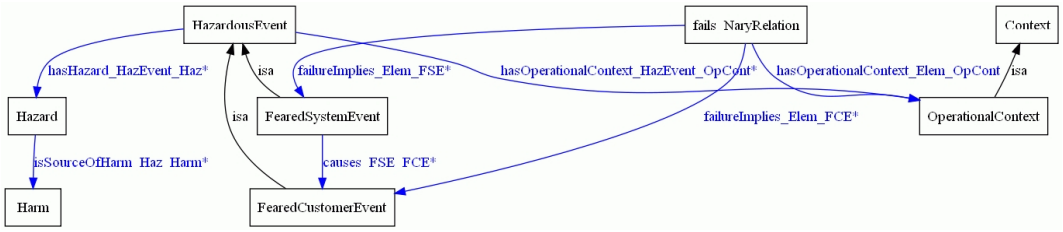

Figure 2.30: Formalization of the hazardous events

A hazardous event is defined as multiple combinations of one hazard in an operational context. The relations *hasHazard_HazEvent_Haz* and *hasOperationalContext_HazEvent_OpCont* are used to record all the hazards and operational contexts of one hazardous event so they are constrained with a minimum cardinality of 1. As presented in section 1.2.2.2, a hazard is defined as the potential source of harm. At Renault, a hazard traditionally corresponds to a collision so elements of *Hazard* are enumerated and we consider *rollover* in addition with *collision with, infrastructure, the same size passenger car, a smaller passenger car, a bigger passenger car, a truck or bus, a pedestrian, a cyclist,*

a motorcyclist. As a remark, other hazards need to be formalized such as electrocutions which are naturally present when we consider E/E systems. A hazard is related to the notion of harm which is represented in figure 2.30 with $isSourceOfHarm\_Haz\_Harm$. This property is constrained with a minimum cardinality of 1 to record the relation with harm. As presented in section 1.2.2.2, harm can be formalized with the AIS (see table 1.1). This table, developed by the automotive industry, is not usually used at Renault so we did not define $Harm$ as an enumeration to keep the notion general. We did not go further on the fundamental definitions of functional safety but hazard and harm are of utmost importance for functional safety activities to be relevant.

Now, let us go back to the failures presented in the previous paragraph. As can be seen in figure 2.30, the failures captured in the class $fails\_NaryRelation$ are relevant to one operational context. We define $hasOperationalContext\_Elem\_OpCont$ and a cardinality restriction of exactly 1 for the class $Element$. The failure of an element (*i.e.*, system, functional requirement, function, flow or component) is then considered with a given operational context (typically, a phase of system life) to identify hazardous events with properties $failureImplies\_Elem\_FSE$ and $failureImplies\_Elem\_FCE$. The respective range of the relations are $FearedSystemEvent$ and $FearedCustomerEvent$. Automotive jargon commonly uses the term system to designate a subsystem of the whole vehicle. Safety studies are performed at subsystem level of abstraction, but (working at this abstraction level) this can erroneously result in the identification of hazardous events that are unrelated to the safety property. This is naturally understood by the engineering field of functional safety at Renault hence hazardous events are made precise with Feared System Event (FSE) or Feared Customer Event (FCE) respectively for the subsystem (the system under consideration) or for the customer or vehicle system point of views. The FCEs are the most important hazardous events as, in the automotive industry, safety is a property that is emergent (can be observed) at customer or vehicle level of abstraction. Also, a FCE is the ultimate consequence of a failure that enables to intuitively understand that this consequence is harmful, which has been made explicit with the previous definition of the hazards of a hazardous event ($hasHazard\_HazEvent\_Haz$) and the harms of a hazard ($isSourceOfHarm\_Haz\_Harm$). In other words, the FCEs are the hazardous events,

which, when they occur, are perceivable by the customers and may result in harm. Figure 2.31 presents FSEs and FCEs identification based on the application of the failure model and the propagation of failures.

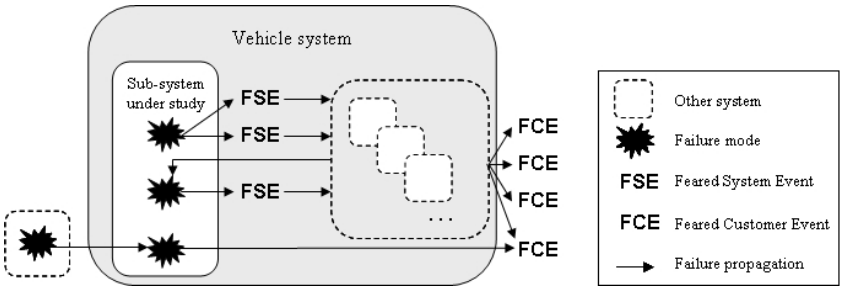

Figure 2.31: Hazardous events at the vehicle system and subsystem points of view

As it can be seen in figure 2.31, the application of a failure mode defines a failure that is propagated and leads to FCEs, with possible intermediate FSEs. This is formalized with the previous relations $failureImplies\_Elem\_FSE$ and $failureImplies\_Elem\_FCE$. The identification of the hazardous events should result in at least one FCE for a failure so $failureImplies\_Elem\_FCE$ is given a minimum cardinality of 1 for the class *Element*. Identifying a FSE has some sense only if it can be related to a FCE which is formalized with relation $causes\_FSE\_FCE$ and a minimum cardinality restriction of 1. Finally, we directly relate a failure that identifies a FSE to the FCE(s) of the FSE with axiom (2.49) enabling to correctly relate all the failures to adequate FCE(s).

$$failureImplies\_Elem\_FSE(x,y) \wedge causes\_FSE\_FCE(y,z) \Rightarrow failureImplies\_Elem\_FSE(x,z) \tag{2.49}$$

Risk estimation. Once system hazardous events have been identified, it is time to estimate the magnitude of their consequences. ISO 26262 introduces three criteria for estimation: *controllability*, *probability of exposure* and *severity*. As presented in section 1.2.2.3, these three criteria are qualitative measurements at the discretion of people assigned to the activity. Risk estimation is performed by analyzing different scenarios which is done by filling table 2.1.

A scenario corresponds to a row in table 2.1. It estimates the three criteria from ISO 26262: probability of exposure, severity and controllability, respectively denoted by E, S

Scenario	FCE	Operational Situation	Consequence	Possibility of avoidance	E	S	C

Table 2.1: Risk estimation table

and *C*. These criteria are contextual so we define the context under the term *operational situation* that is composed of one of the operational context of the FCE under consideration and an *aggravating circumstance*. The operational context corresponds to a phase of system life, *e.g.*, freewheel mode that corresponds to the context for the braking function where there is no generated energy for propulsion. An aggravating circumstance is a realistic situation that in combination with an operational context enables to consider a hazardous event under a different perspective for the estimation to be as precise as possible, *e.g.*, a slippery road situation has more potential to result in harm when braking. Given an operational situation (*i.e.*, one operational context and one aggravating circumstance) the *consequence* (*i.e.*, the accident) and the possibilities of avoiding this consequence in this operational situation are documented. Consequences are expressed in terms of harm (*e.g.*, death, light injury, *etc.*[3]). Possibility of avoidance concerns the driver's opportunities to perform an action to avoid the impending accident. Reasonable driving skills should be considered (typically a driver with no particular training is considered). These informations (operational situation, consequence and possibility of avoidance) are respectively related to the three criteria E, S and C documented in ISO 26262. We did not formalize this correspondence as we were lacking domain knowledge however, intuitively, an operational situation should be equivalent to a probability of exposure, a consequence should be equivalent to one severity level and a possibility of avoidance should be equivalent to a controllability level. Such a formalization would solve both discrepancies and errors during risk estimation by making this activity automatic and removing personal judgment during estimation. In the end, a row in table 2.1 is defined along the formalization in figure 2.32.

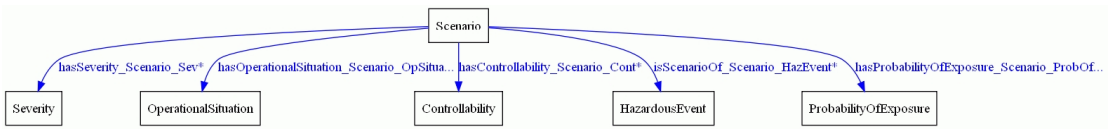

Figure 2.32: Concepts and relations for risk estimation

[3]For a definition of harm, it is possible to reuse the AIS (table 1.1)

Properties in figure 2.32 are constrained with an exact cardinality of 1 (for their respective domain) correctly representing a row of table 2.1. Unnecessary information about consequence and possibility of avoidance are not represented but can easily be added to the ontology in the same manner (for our purpose, only severity and exposure criteria are of importance). As we defined the operational contexts for the identification of the hazardous events, we ended up defining the context in terms of operational situation, operational context and aggravating circumstance as can be seen in figure 2.33.

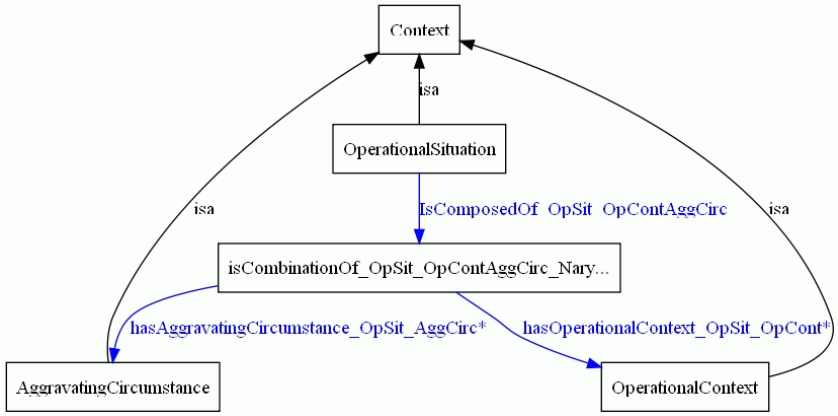

Figure 2.33: Formalization of the context concept

The class *Context* is used to represent any type of context. In order to restrain the interpretations to the Preliminary Hazard Analysis, it is partitioned into *OperationalSituation*, *OperationalContext* and *AggravatingCircumstance*. The two last classes are given the previous definitions of a phase during the system life and a circumstance that gives light to the estimation of the risk. An operational situation is a couple of an operational context and of an aggravating circumstance so we introduce the intermediate class $isCombinationOf\_OpContAggCirc\_NaryRelation$. An operational situation is related to the latter class with $isComposedOf\_OpSit\_OpContAggCirc$ which is itself related to an operational context and an aggravating circumstance with $hasOperationalContext\_OpSit\_OpCont$ and $hasAggravatingCircumstance\_OpSit\_AggCirc$. These three relations are constrained with an exact cardinality of 1 for their respective domain correctly representing the relation between an operational situation and the couple it is composed of. Finally, figures 2.34, 2.35 and 2.36 present the three parameters for risk

estimation that are defined in ISO 26262 for relevance with the automotive domain. Each subclass has only one element which is defined with respective enumerations.

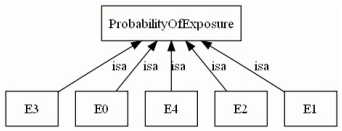

Figure 2.34: Probability of exposure subclasses

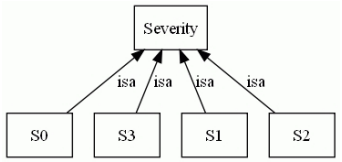

Figure 2.35: Severity subclasses

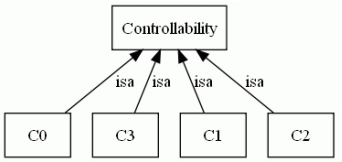

Figure 2.36: Controllability subclasses

In this section, we presented how the risk (of harm) of a system can be identified in a more systematic way with the help of a failure model applied on system descriptions and how this risk is classified by estimating different parameters of each identified hazards. The next section completes our description of the covered functional safety concepts with the evaluation of the risk through an integrity attribute specific to the automotive domain and by explaining how this integrity attribute is taken into account during the design process.

2.2.2.2 Risk Evaluation and Safety Concept

Section 2.2.2.1 has presented the risk analysis part of the PHA (see figure 2.26). It ends up with probability of exposure, severity and controllability parameters estimated for each identified hazardous event. The following concludes on risk assessment with risk evaluation that ultimately returns the system integrity level, *i.e.*, the level of confidence a user can have in the system.

ASIL Determination. Risk evaluation in the automotive domain is adapted from the general notion of Safety Integrity Level (SIL) (see section 1.2.2.3). Risk evaluation is called ASIL determination as the specific adaptation of the general SIL based approach. ASIL stands for Automotive Safety Integrity Level and, as presented in section 1.2.2.3, the whole standard ISO 26262 is constructed based on the ASIL of the system. Depending on this ASIL, a corresponding specification is systematically produced, which, if satisfied, allows asserting the absence of unacceptable risks. ASIL determination concludes risk assessment

by completing table 2.1 with an ASIL column filled by following table 1.4 defined in ISO 26262 in chapter 1 reinstated here for reading convenience.

Severity	**Exposure**	**Controllability**		
		C1	C2	C3
S1	E1	QM	QM	QM
	E2	QM	QM	QM
	E3	QM	QM	A
	E4	QM	A	B
S2	E1	QM	QM	QM
	E2	QM	QM	A
	E3	QM	A	B
	E4	A	B	C
S3	E1	QM	QM	A
	E2	QM	A	B
	E3	A	B	C
	E4	B	C	D

Table 1.4: Automotive Safety Integrity Levels

The class *ASIL* and its subclasses are defined as illustrated in figure 2.37.

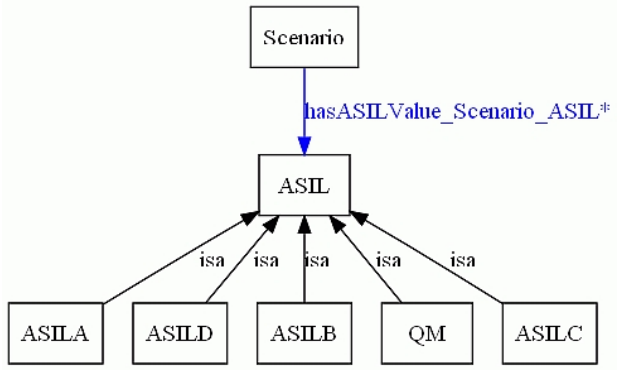

Figure 2.37: ASIL determination

Figure 2.37 presents the concept of ASIL as a scenario attribute. The subclasses of *ASIL* define the four different levels defined by the standard. They range from A to D with D being the most restrictive. An ASIL is evaluated for a scenario with property *has-ASILValue_Scenario_ASIL*. This evaluation is systematic and based upon estimated probability of exposure, severity and controllability criteria during risk estimation as defined by table 1.4. Each line in the table establishes a direct correspondence between a triple of the previous parameters for risk estimation and an ASIL. Each line is defined with

an axiom similar to (2.50). This axiom is given as an example and it corresponds to the last line in table 1.4 expressing that an ASIL D is determined for a given scenario with E4 exposure, S3 severity and C3 controllability.

$$\begin{aligned} &Scenario(x) \wedge hasSeverity\_Scenario\_Sev(x, "Severity3") \wedge \\ &hasProbabilityOfExposure\_Scenario\_ProbOfExp(x, "Exposure4") \wedge \\ &hasControllability\_Scenario\_Cont(x, "Controllability3") \Rightarrow \\ &\quad hasASILValue\_Scenario\_ASIL(x, "ASIL\_D") \end{aligned} \tag{2.50}$$

This concludes risk assessment with the risk evaluated for each scenario in terms of ASIL. Let us note that this analysis can be done at other levels of abstraction by applying the failure model on more precise artifacts such as functions and components. Such analyses are typically done at Renault by using the Failure Mode Effects and Criticality Analysis technique (FMECA). The PHA is fundamentally different with all the other safety studies. It is performed with the assumption that the system does not implement any safety measures (that would mitigate the risk) in order to obtain fundamental results, independent from the system's implementation choices, *which can be re-used as is*. Currently, problems for reuse stem from a lack of formalization which results in the PHA being subject to arbitrary assessment and human errors. All the other safety studies are performed during the design process in order to verify preceding fundamental results that are always expressed with ASIL. The realization of these other safety activities is not supported by the functional safety ontology. Let us note that efficient tools are available on the market. However, the ontology supports their fundamental result which is the ASIL. The following precises the notion of ASIL and how it is intricately present in the design process as an answer to reduce the risk associated to identified hazards.

Realization of a Safety Concept. ISO 26262 specifies in a general manner how ASIL is taken into account by the design. In the previous paragraph, we ended up with ASIL as an attribute of the different scenarios for hazardous events. The notion of ASIL associated to a scenario specifies the target level of reduction of the risk associated to the hazards to an acceptable level. These hazards are clearly relative to the system. In order to integrate the safety attribute into the system genes (ensuring that the system does take into account its hazards and is developed towards acceptable risk), the DMSRRF (see

figure 2.26) documents the top level safety goals and two safety concepts which relates to the notion of requirement. Next paragraphs present how the ASIL of the hazardous events are reflected into the system descriptions and focus on requirements.

ASIL Assignment. In the previous section, we concluded on risk assessment with the ASIL attribute evaluation for the scenarios of the FCEs. Now we need to assign the correct ASIL to FCEs. Figure 2.38 presents the concepts and relations used to systematically assign an ASIL to a FCE.

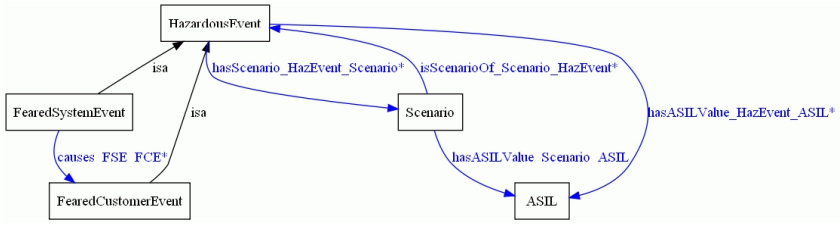

Figure 2.38: ASIL assignment on the hazardous events

In figure 2.38, we find the previous formalization of a scenario that records to which hazardous event it corresponds and the evaluated ASIL of the scenario. Property $hasScenario\_HazEvent\_Scenario$ is defined as the inverse of $isScenarioOf\_Scenario\_HazEvent$ (2.51) and is constrained with a minimum cardinality of 1 for the subclass $FearedCustomerEvent$. With axiom (2.51), the hazardous event is automatically related to all the scenarios it is involved with. We actually defined the ASIL as an attribute of all the hazardous events with $hasASILValue\_HazEvent\_ASIL$ which is constrained with an exact cardinality of 1 for $HazardousEvent$. The scenarios are only defined for the FCEs therefore, for the FSEs to be related to at least one scenario, we relate the scenarios of a FCE to the FSE that causes the FCE (2.52). Then the highest ASIL of the scenarios of a hazardous event has to be assigned to the hazardous event (in particular, to the FCE). Let $maxASIL$ be the function that returns the highest ASIL of a set. We have axiom (2.53) that assigns automatically the highest ASIL of the scenarios of a hazardous event to this hazardous event.

$$\begin{aligned} &hasScenario\_HazEvent\_Scenario(x,y) \Leftrightarrow \\ &isScenarioOf\_Scenario\_HazEvent(y,x) \end{aligned} \tag{2.51}$$

$$\begin{array}{l} FearedSystemEvent(x) \wedge causes\_FSE\_FCE(x,y) \wedge \\ hasScenario\_HazEvent\_Scenario(y,z) \Rightarrow \\ \quad hasScenario\_HazEvent\_Scenario(x,z) \end{array} \tag{2.52}$$

$$\begin{array}{ll} let & \\ & Z = \{z \mid hasASILValue\_Scenario\_ASIL(y,z)\} \\ & a = maxASIL(Z) \\ in & hasScenario\_HazEvent\_Scenario(x,y) \wedge \\ & \quad hasASILValue\_Scenario\_ASIL(y,z) \Rightarrow \\ & \quad\quad hasASILValue\_HazEvent\_ASIL(x,a) \end{array} \tag{2.53}$$

These axioms make it possible to relate all system hazardous events to an integrity level. In other words, we defined the target levels of risk reduction associated to hazards of hazardous events (of the system functional requirements). These target levels need to be adapted for the different abstraction levels for the system development to be directed towards acceptable risks. Therefore, related target levels need to be defined for the system and its different descriptions, *i.e.*, the system and its requirements, functions, flows and components. The notion of ASIL is dependent on its context represented with different classes in the ontology. The relations $hasASILValue\_Elem\_ASIL$ and $hasASILValue\_Req\_ASIL$ are used to define the ASIL of elements (system, functions, flows and components, see figure 2.27) and requirements. They are constrained with an exact cardinality of 1 respectively for *Element* and *Requirement*. ISO 26262 remains overall very general so that the different actors that manipulate this standard can all relate to it however with specific customizations. Concerning ASIL assignment, ISO 26262 defines with little details the structure of the requirements, how they are related to architectural elements, how the ASIL is assigned to safety goals (a type of requirements) and how the ASIL is assigned to architectural elements. The product development process defined in ISO 26262 is actually composed of three complementary V cycles for the system, hardware and software levels. At system level, the system is ultimately described with a physical architecture that contains the system components. The components that are subject to development are partitioned between hardware and software components that are developed with the appropriate product development process at hardware and / or software level. The requirements defined in ISO 26262 are actually dependent of the ASIL. The notion is used out of context in the standard but intuitively refers to the ASIL of the system and to

the ASIL of the components at hardware and software level. For the ASIL of the system (the target level of risk reduction associated to system hazards), it is implied that it should correspond to the highest ASIL of the system hazardous events. For the components, the standard requires the definition of functional safety requirements allocated to the components. A component is then assigned the highest ASIL of the functional safety requirements in relation. In order for the system development to be directed towards acceptable risk, the standard requires the definition of top level safety requirements called safety goals. They are formulated for each hazardous event (if their ASIL is different than QM). With this definition, the same safety goal can actually be defined for different hazardous events. The ASIL attribute of a hazardous event is assigned to its safety goal and in the case of a safety goal that covers different hazardous events, it is the highest corresponding ASIL that is assigned to the safety goal. Figure 2.39 presents the formalization of the safety goals.

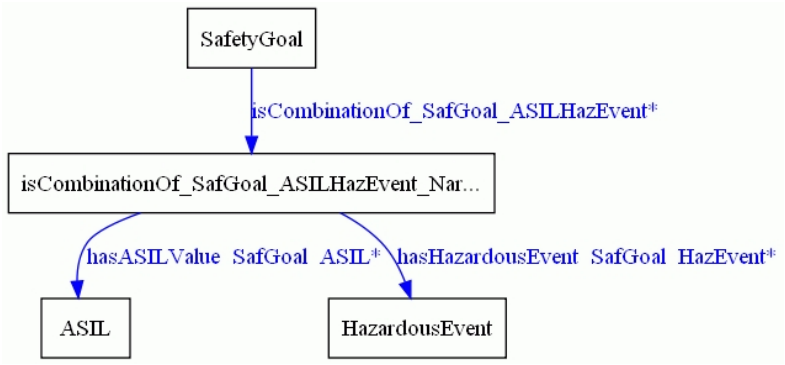

Figure 2.39: ASIL assignment on the hazardous events

The class $SafetyGoal$ is a subclass of $SafetyRequirement$. It is related to the hazardous events it covers and one ASIL with the intermediate class $isCombinationOf\_SafGoal\_ASILHazEvent\_NaryRelation$ and the properties $isCombinationOf\_SafGoal\_ASILHazEvent$, $hasASILValue\_SafGoal\_ASIL$ and $hasHazardousEvent\_SafGoal\_HazEvent$, that represent this nary relation. The first two relations are constrained with an exact cardinality of 1 (a safety goal has only one ASIL) and the last one is constrained with a minimum cardinality of 1 (a safety goal covers at least one hazardous event). Hazardous events that are evaluated with QM (for Quality Management, this class denotes no requirement according to ISO 26262) are not the object of a safety goal therefore

the range of $hasASILValue\_SafGoal\_ASIL$ is $ASILA \cup ASILB \cup ASILC \cup ASILD$. Using these relations, axiom (2.54) assigns the highest ASIL of the hazardous events related to a safety goal to this safety goal. They correspond to the hazardous events of the system so we also assign the highest ASIL of the safety goals to the system (2.55).

$$\begin{array}{l} let \\ \quad Z = \{z \mid hasASILValue\_HazEvent\_ASIL(y, z)\} \\ \quad a = maxASIL(Z) \\ in \quad isCombinationOf\_SafGoal\_ASILHazEvent(w, x) \wedge \\ \qquad hasHazardousEvent\_SafGoal\_HazEvent(x, y) \wedge \\ \qquad hasASILValue\_HazEvent\_ASIL(y, z) \Rightarrow \\ \qquad\quad hasASILValue\_Req\_ASIL(x, a) \wedge hasASILValue\_SafGoal\_ASIL(y, a) \end{array} \tag{2.54}$$

$$\begin{array}{l} let \\ \quad Z = \{z \mid hasASILValue\_Req\_ASIL(x, z)\} \\ \quad a = maxASIL(Z) \\ in \quad SafetyGoal(x) \wedge System(y) \wedge \\ \qquad hasASILValue\_Req\_ASIL(x, z) \Rightarrow \\ \qquad\quad hasASILValue\_Elem\_ASIL(y, a) \end{array} \tag{2.55}$$

The product development process in ISO 26262 continues with the definition of the *functional safety concept*, the *technical safety concept*, and then the system design specification. The functional safety concept is constructed based on the safety goals. It contains *functional safety requirements* that are derived from the safety goals. Similarly, the technical safety concept is constructed based on the functional safety requirements. It contains *technical safety requirements* that are derived from the functional safety ones. Finally, the system design specification is constructed in accordance with the technical safety requirements. It contains *system safety requirements* that are derived from the technical safety ones. The functional safety requirements are expressed at the functional level of abstraction, *i.e.*, they are independent from the implementation. They are allocated to an architectural element which is developed with the highest ASIL of the functional safety requirements it is associated to. The technical safety requirements provide the technical implementation of the associated functional safety requirements. These two types of requirements are fulfilled by the system design. The system safety requirements provide the system level implementation of the technical safety requirements. They are allocated to an architectural element which is developed with the highest ASIL of the system safety

requirements it is associated to. The conceptualization described in ISO 26262 does not go much further and numerous questions need to be answered. For instance, we have an ASIL for the safety goals, but what is the ASIL of the three other types of requirements? If a functional safety requirement is allocated to a function, which is an architectural element, it shall be developed with the highest ASIL of the functional safety requirements it is associated to. How is a function developed ? And so on. At Renault, the DMSRRF contains the safety goals presented before and one safety concept that is a set of requirements. Figure 2.40 presents the different types of requirements supported by this ontology and their previous relation with the ASIL.

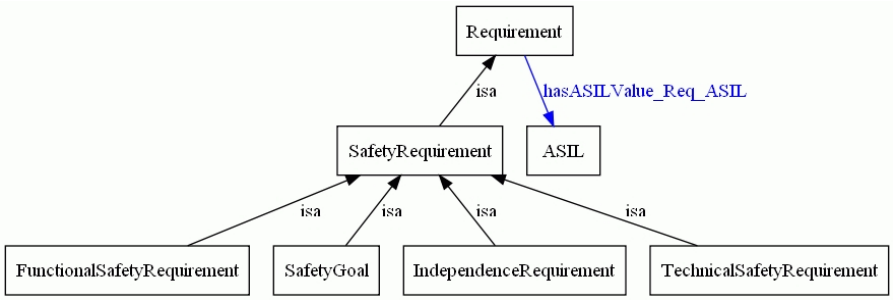

Figure 2.40: Safety requirements

In figure 2.40, we find the safety goals presented before. We promote the use of functional safety requirements and technical safety requirements as respectively relative with implementation independent and implementation abstraction levels. The system safety requirements defined in ISO 26262 were not a fundamental concept and are not modeled in the ontology. Finally, *IndependenceRequirement* is relative to ASIL decomposition which is the object of the next paragraph. The ontology only defines the ASIL as an attribute of the requirements and the elements (in particular, the system, functions and components). This attribute needs to be defined for each requirement and element. It gives the capability to assign an ASIL. Finally, we did not go further on the formalization of ASIL assignment in this ontology. On the one hand, the concepts manipulated are not precise enough to enable a correct (in all cases) automatic assignment of the ASIL. On the other hand, we immediately anticipated that there was enormous potential for reuse of the concepts defined in the systems engineering ontology to make the concepts manipulated by functional safety engineering field more precise. Therefore, the semantic relations that

enable to structure the concepts of functional safety and to define the remaining of ASIL assignment is presented in section 2.3.

ASIL Decomposition. ISO 26262 allows the tailoring (*i.e.*, the reduction) of the ASIL which is incredibly advantageous and unfortunately necessary because of the cost (manpower, labor, time, tools, *etc.*) associated to the development of safety critical systems. Tailoring of the ASIL is performed with two different techniques named *ASIL decomposition* and *criticality analysis*. The ontology only supports ASIL decomposition as it corresponds to design creative phase where tailoring is performed a priori while criticality analysis tailors the ASIL a posteriori. Figure 2.41 comes from ISO 26262. It illustrates the decomposition of a safety requirement with a defined ASIL into two other safety requirements with respective defined ASILs which are not inherited from the initial requirement (before decomposition).

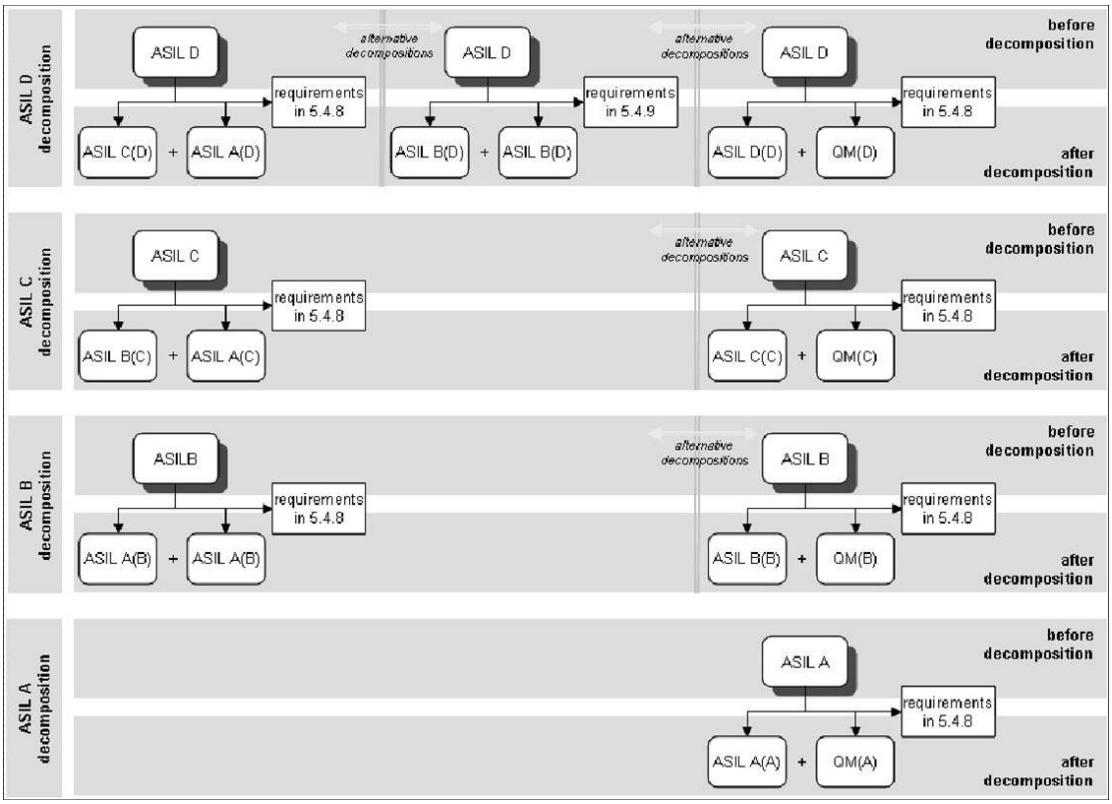

Figure 2.41: ASIL decomposition schemes

A decomposition following figure 2.41 involves a requirement, two sub-requirements and additional requirements. In particular, a requirement of independence (*IndependenceRequirement* in figure 2.40) is defined for the safety studies to provide the evidence that the resulting requirements are implemented by sufficiently independent elements. As can be seen in figure 2.41, the ASIL before decomposition is recorded (in parenthesis) by the resulting requirements. The relation $hasASILObjectiveValueBeforeDecomposition\_Elem\_ASIL$ is defined with $Element \cup Requirement$ and $ASILA \cup ASILB \cup ASILC \cup ASILD$ as domain and co-domain. Along with relations $hasASILValue\_Elem\_ASIL$ and $hasASILValue\_Req\_ASIL$, it records that the target level of risk reduction, if attained, is sufficient for the more constraining previous one. The relation is not useful for the classes *System*, $SafetyGoal$ and *IndependenceRequirement*, therefore it is constrained with an exact cardinality of 0 for the three classes. Figure 2.42 presents the relations used to define ASIL decomposition.

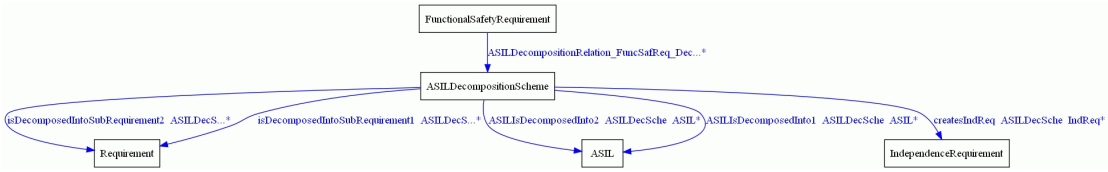

Figure 2.42: Formalization of ASIL decomposition

As shown in figure 2.42, functional safety requirements are decomposed using the intermediate class $ASILDecompositionScheme$ and six relations. Property $ASILDecompositionRelation\_FuncSafReq\_DecScheme$ records the functional safety requirement which is decomposed and the property is constrained with a maximal cardinality of 1 as not all the functional requirements are the object of ASIL decomposition. Properties $isDecomposedIntoSubRequirement1\_ASILDecSch\_Req$ and $isDecomposedIntoSubRequirement2\_ASILDecSch\_Req$ record the two sub-requirements in relation. Properties $ASILIsDecomposedInto1\_ASILDecSche\_ASIL$ and $ASILIsDecomposedInto2\_ASILDecSche\_ASIL$ record the ASIL of the respective sub-requirements. $createsIndReq\_ASILDecSche\_IndReq$ records the requirement of independence involved by the decomposition. These five relations are respectively defined with an exact cardinality of 1. Then the different decompositions of figure 2.41 are formalized with the respective sub-

classes of *ASILDecompositionScheme* presented in figure 2.43 that are disjoints with one another.

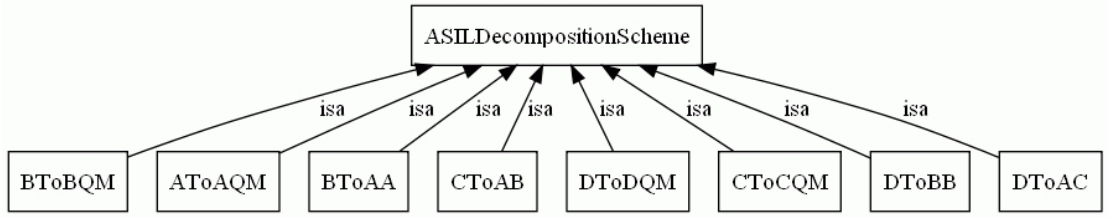

Figure 2.43: ASIL decomposition scheme subclasses

The range of $ASILIsDecomposedInto1\_ASILDecSche\_ASIL$ and $ASILIsDecomposedInto2\_ASILDecSche\_ASIL$ are respectively constrained to the adequate ASIL for each subclass of *ASILDecompositionScheme* as specified in figure 2.41. Axiom (2.56) is given as an example. It ensures that the decomposition scheme which decomposes an ASIL A into an ASIL A and a QM one is used by the functional safety requirements that have an ASIL A. Similar axioms are defined for the other decomposition schemes of figure 2.41. Naturally, properties $hasASILValue\_Req\_ASIL$ and $hasASILObjectiveValueBefore-Decomposition\_Elem\_ASIL$ should be correctly configured for each sub requirement according to figure 2.41, however the configuration actually depends on the structure of the requirement which is defined in section 2.3.

$$\begin{aligned} &ASILDecompositionRelation\_FuncSafReq\_DecSche(x,y) \wedge \\ &AToAQM(y) \Rightarrow hasASILValue\_Req\_ASIL(x,"ASIL\_A") \end{aligned} \tag{2.56}$$

2.2.2.3 Conclusion

In this section, part of the functional safety domain has been formalized with the focus on the Preliminary Hazard Analysis (PHA) which is the basis of all the safety activities. The functional safety ontology has been defined in order to be compliant with ISO 26262, to promote a more systematic realization of the risk estimation activity with a mandatory failure model, and to automatically perform risk evaluation as defined in ISO 26262. It is important to note that the information provided by the PHA is actually completely reusable *as is* for any similar system. We gave some leads on the further automation of the PHA throughout the section. Ultimately, the PHA results in the evaluation of the

ASIL relative to each system hazard. We ended setting the ontology up to support the assignment of the ASIL on the system, its requirements, its functions and its components. For reasons of unresolved ambiguities that correspond to licit incompatible interpretations, the assignment of the ASIL remains manual. One objective of the realization of a domain ontology is obviously to remove as much as possible these problems of ambiguity. This removal has been actually purposely deferred to section 2.3 that takes advantage of the two domain ontologies of systems engineering and functional safety to improve the preciseness of their concepts which mutually reinforce one another.

2.2.3 Conclusion

Sections 2.2.1 and 2.2.2 present the respective formalizations of the systems engineering and functional safety domains into ontologies. These formal ontologies define in a formal way the concepts manipulated by the two engineering fields and how those concepts are related to one another. The formal syntax and semantics of the language enables to automatically define consistency as the property that the asserted axioms are non contradictory. Computer treatment enables to verify the ontologies consistency answering either if the ontology is consistent or not. In case of inconsistency, the contradiction(s) are presented. The information that aims to be recorded in our ontologies benefits implicitly from this consistent structure. Abiding with the structure is equivalent for the information to have correct structure (*i.e.*, the ontology at conceptual and information level is consistent), ensuring quality. These ontologies are the carrier of structured knowledge. As the semantic aspect of formal ontologies ensures that the structure of the concepts is understood in a unique manner, the two ontologies bring together the understanding of the domains at the conceptual level with the information that needs to be understood at this conceptual level. In the case that the information is susceptible for reuse, it will be done with correct understanding. For instance, reusing a component comes with all the functions it realizes.

In the end, these two ontologies correspond to two sides of the development process of one single system. We concluded on systems engineering as the orchestrator of the other engineering fields that contribute to the development. Systems engineering contains the different system descriptions. The other engineering fields study the system based on these

descriptions. In particular, functional safety is responsible for assigning an ASIL to all the elements of systems engineering and demonstrating the absence of unacceptable residual risks. Also, the results provided by functional safety engineers guide the design of the system. As such, communication between these two engineering fields is of the utmost importance. Section 2.3 presents the integration of these two domains into a domain ontology for systems and functional safety engineers that enables precise communication based on this shared conceptualization.

2.3 Global Domain Formalization

SE deployment and the increased concern to comply with ISO 26262 standard have acted as a catalyst for developing synergies. Renault engineers from different fields, academics and other companies staff have been drawn together to work on these subjects. At the most basic level, the need to share meaning on terms arises as barriers such as dissimilar vocabularies, representations or languages can imped those synergies. This can be answered by defining an *ontology*. It is a formal, explicit specification of a shared conceptualization [Studer et al. 1998] pertaining to a domain. Our ontology is constituted by a specific vocabulary of terms used in the domain with an explicit specification of their meaning, *i.e.*, definition of the concepts of the domain and their relationships. It defines a structure of the domain and constrains the possible interpretations of terms. Informally, an ontology enables a precise and non ambiguous communication as everybody shares the same language. This section addresses the integration of the two domain ontologies presented in section 2.2.1 and 2.2.2 into a domain ontology that supports the design process for both systems and functional safety engineers. The intent is to involve the functional safety engineers in the design process as soon and as much as possible. Actually, other engineering fields also impact system design but we support the value that nothing is more important than safety. Section 2.3.1 presents the concepts integration concepts and focuses mainly on the requirements which remained ambiguous. Section 2.3.2 concludes on ASIL assignment. Finally, section 2.3.3 gives the conclusion on the formalization of systems engineering and functional safety. The interested reader may also consult the ontology at the following address: `http://cedric.cnam.fr/~taofif_o/`.

2.3.1 Systems Engineering and Functional Safety Domains Integration

The conceptualization of systems engineering and functional safety has been formalized in sections 2.2.1 and 2.2.2. These two domain ontologies enable system engineers and functional safety engineers to communicate using a defined structured vocabulary of terms. These engineers are involved in the design process of safety critical mechatronics systems with processes that correspond to each respective field presented in figures 2.1 and 2.26. In particular, these two engineering fields manipulate some concepts that are similar, however with a different focus and conceptualization. These different conceptualizations are translated in loss of knowledge at the processes interfaces which impeds communication between the two domains. The systems engineering and functional safety ontology presented in this section provides the two domains with a common semantic model where concepts ambiguities are resolved enabling correct communication. Section 2.3.1.1 presents the integration of the systems engineering and functional safety ontologies into a single domain ontology for systems and functional safety engineers. Section 2.3.1.2 addresses integration at the information level of detail.

2.3.1.1 Conceptual Integration

OWL is equipped with an importation mechanism that enables to manipulate different ontologies into the same universe of discourse. The systems engineering and functional safety ontologies are imported into a single ontology named systems engineering and functional safety ontology. Their importation results in the attribution of different unique namespaces that precede each identifiers. Here, *p1:* and *p2:* correspond respectively to the namespaces of systems engineering and functional safety. For instance, $p1{:}Requirement$ corresponds to the class in the systems engineering ontology while $p2{:}Requirement$ is used for the class in the functional safety ontology. In the resulting ontology, it is now possible to use any term from the previous ontologies inside the same universe of discourse. The remaining work is the formalization of a conceptualization which is simply to define more axioms (*i.e.*, classes, properties and constraints). One difficulty when defining an ontology lies in the fact that the formalized conceptualization comes from a consensus between different actors that agree upon, and therefore share, the ontology. When integrating two

ontologies, the difficulty is at the next level. A consensus involves more people and therefore is more difficult to attain. Also, the defined conceptualizations can be incompatible which forces to rework those ontologies to enable their integration. In the end the ontologies of systems engineering and functional safety that have been presented correspond to the final domains conceptualization where incompatibilities have been removed. The integration of these two ontologies addresses the resolution of concepts ambiguities that remain.

The concepts that are at the interface of systems engineering and functional safety correspond to system descriptions (architectures). The difficult part corresponds to the requirements which is left to the end. The system, functions, flows and components are similar / equivalent concepts in the two ontologies. To express this equivalence between two classes we use set equivalence (symbolized by $\equiv$). The sets A and B are equivalent ($A \equiv B$) means that $A \subseteq B$ and $A \supseteq B$. In terms of classes, classes A and B are equivalent ($A \equiv B$) means that A is a subclass of B and B is a subclass of A. Then, according to the axiomatization given in appendix A, the properties and constraints defined for a class are semantically defined for the equivalent class, and the subclasses of a class that is equivalent to another class semantically inherit all the properties of the two equivalent classes. For the concepts of system, function, flow and component we have the following equivalences:

$$
\begin{array}{lcl}
p1 : System & \equiv & p2 : System \\
p1 : Function & \equiv & p2 : Function \\
p1 : Flow & \equiv & p2 : Flow \\
p1 : Component & \equiv & p2 : Component \\
p1 : Interface & \equiv & p2 : Interface \\
p1 : EEComponent & \equiv & p2 : EEComponent \\
p1 : Actuator & \equiv & p2 : Actuator \\
p1 : Controller & \equiv & p2 : Controller \\
p1 : Sensor & \equiv & p2 : Sensor \\
p1 : HWComponent & \equiv & p2 : HWComponent \\
p1 : SWComponent & \equiv & p2 : SWComponent \\
p1 : OtherComponent & \equiv & p2 : OtherTechnologyComponent
\end{array}
$$

In order to benefit from the definitions provided by the two ontologies for systems engineering and functional safety, we anticipated the reuse of relations that should have their similar counterpart in the other ontology, which is why those counterparts are not defined. The integration of the preceding concepts pose no particular difficulty as we did not detect any incompatibility. The properties and constraints are defined correctly for

all the classes. From the point of view of systems engineering, the concepts are enriched with failures, ASILs and ASIL objectives information. From the point of view of functional safety, the concepts are enriched with traceability and decomposition information.

The concept of requirement was relatively more difficult to integrate as its understanding from the functional safety point of view is a bit ambiguous. The systems engineering ontology defines the requirement concept from a fundamental point of view that is used as a basis to understand requirements from functional safety. As such, we defined an equivalence only between $p1$:*Requirement* and $p2$:*Requirement*. Figure 2.40 from section 2.2.2.2, presents the different types of requirements that we retained for functional safety. Following the design process timeline, safety goals are the top level safety requirements. They are used to derive functional safety requirements which are used themselves to derive technical safety requirements. From the fundamental point of view of systems engineering, safety goals actually correspond to low level stakeholders requirements (from functional safety). This is formalized by defining $p2$:$SafetyGoal$ as a subclass of $p1$:$Low\_Level\_StakeholderRequirement$ (*cf.* figure 2.44).

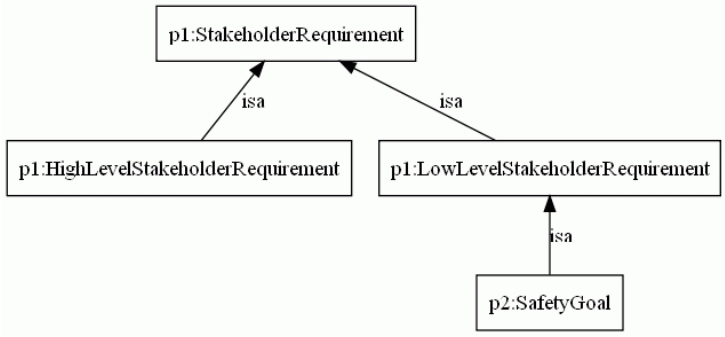

Figure 2.44: Integration of the safety goals

Functional safety requirements correspond to functional system requirements that are used to deduce functions which tackle the dysfunctional behavior of the system. The fundamental functional system requirements are formalized with the classes $p1:FunctionalHighLevelSystemRequirement$ and $p1$:$FunctionalLowLevelSystemRequirement$. As can be seen in figure 2.45, in the functional safety ontology, we actually defined the class $p2$:$FunctionalSafetyRequirement$ as a subclass of $p2$:$FunctionalRequirement$ which is itself a subclass of $p2:Requirement$. The classes $p1:FunctionalHighLevelSystemRe-$

quirement and $p1:FunctionalLowLevelSystemRequirement$ are defined as subclasses of $p2$:*FunctionalRequirement*. They are not disjoint with $p2$:$FunctionalSafetyRequirement$ as a functional safety requirement is either a high level or a low level functional system requirement (as it can be decomposed or not). The derivation of a safety goal into a functional safety requirement is possible as stakeholders requirements (that comprise safety goals) are used to derive system and external requirements (which comprise functional safety requirements).

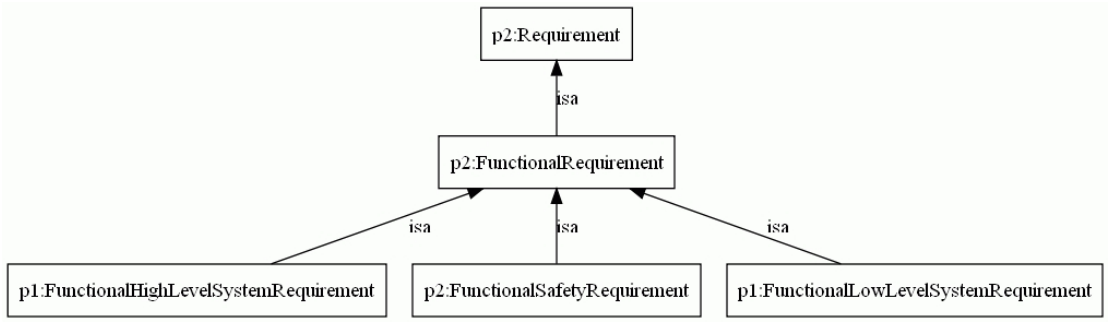

Figure 2.45: Integration of the functional safety requirements

Functional safety requirements are used to derive technical safety requirements which are formalized by constraining the range of $p1$:$derives\_Req$ to $p2$:$TechnicalSafetyRequirement$ for $p2:FunctionalSafetyRequirement$. In order to be consistent with the derivation relation defined for the functional system requirements into non functional requirements, we defined $p2:TechnicalSafetyRequirement$ as a subclass of $p1:SystemElementNonFunctionalRequirement$ as can be seen in figure 2.46. The non functional requirements on the system elements are either high level or low level ones depending on their decomposition. This is formalized by the fact that $p2$:$TechnicalSafetyRequirement$ is not disjoint with the non functional requirements on the system elements.

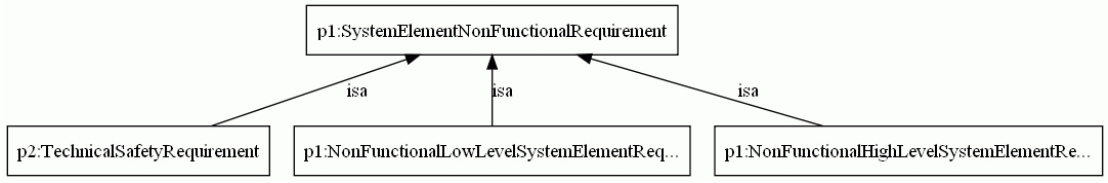

Figure 2.46: Integration of the technical safety requirements

As can be seen in figure 2.44 and 2.46, safety goals and technical safety requirements are defined as subclasses of respective requirements from the systems engineering ontology. As such, they automatically inherit all the relations and constraints that have been previously defined for their superclasses and are sufficiently precise from the point of view of systems engineering. Moreover, technical safety requirements are meant to be defined as either high level or low level non functional requirements which will result for each technical safety requirement to implement the axioms defined for the precedent high level or low level requirements. Functional safety requirements need to be asserted as either functional high level system requirement or low level ones. This assignment needs to be done manually. It enables functional safety requirements to be automatically defined with all the axioms of a functional high level or low level system requirement making its structure precise. The class $p2:IndependenceRequirement$ needs to be made more precise by considering the general concept of requirement presented in figure 2.3 from section 2.2.1.2. We disabled $p1{:}hasPart\_Req$, $p1{:}derives\_Req$, $p1{:}isAllocatedTo\_Req\_Syst$, $p1{:}isCharacterizedBy\_Req$, $p2{:}hasASILValue\_Req\_ASIL$ and $p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_ASIL$ by defining an exact cardinality of 0 for all these properties. We left the other properties of systems engineering open for use as independence is actually an important concept of functional safety that is not yet completely formalized. For now, the concept of independence can be associated to ASIL decomposition presented in section 2.2.2.2. It involves a functional requirement decomposed into two functional sub-requirements which are specified as independent with an independence requirement. An independence requirement therefore can be used to characterize the involved functional sub-requirements so we constrained the range of $p1{:}characterizes\_Req$ to $p2:FunctionalRequirement$. It can also be related to $p1:deduces\_Req\_Func$ and $p1:deduces\_Req\_Flow$ to the functions and flows in relation with these requirements. And finally, the notion of independence has some meaning related to the elements of the functional and physical architectures of the independent functional requirements. The related elements can be recorded with $p1{:}isAllocatedTo\_Req\_Func$, $p1{:}isAllocatedTo\_Req\_Flow$ and $p1{:}isAllocatedTo\_Req\_Comp$.

The same customizations are made for the different types of requirements of systems

engineering from the point of view of functional safety. The relations that correspond to the failure model application (see figure 2.29 in section 2.2.2.1) are precise enough and do not need further customization. As we defined $p1$:*Requirement* and $p2$:*Requirement* as equivalent classes, the different types of requirements defined in the systems engineering ontology (see figure 2.4 in section 2.2.1.2) inherit the relations $p2$:$hasASILValue\_Req\_ASIL$ and $p2$:$hasASILObjectiveValueBeforeDecomposition\_Elem\_ASIL$. Following the design process timeline, the latter property is first used on functional requirements when ASIL decomposition is used. Therefore, it is disabled for the preceding stakeholders requirements by constraining the property with an exact cardinality of 0 for the class. For the other types of requirements, the property is not further customized as it will be used situationally depending on the use of ASIL decomposition or not. Still following the design process timeline, $p2$:$hasASILValue\_Req\_ASIL$ is first defined for the safety goals (and constrained with an exact cardinality of 1) which are low level stakeholders requirements. Only the low level stakeholders requirements are the object of $p2$:$hasASILValue\_Req\_ASIL$ so the property is constrained with an exact cardinality of 0 for the high level stakeholders requirement. We chose to assign an ASIL to all the other requirements that correspond to the system design and is formalized by customizing $p2$:$hasASILValue\_Req\_ASIL$ with an exact cardinality of 1 for the classes $p1$:*SystemRequirement*, $p1$:*ExternalRequirement* and $p1$:*SystemElementNonFunctionalRequirement*.

2.3.1.2 On Individuals Integration

In section 2.3.1.1, we presented how the similar concepts of systems engineering and those of functional safety are made more precise by means of customization and by reusing the definitions of one domain into the other. The other concern is the knowledge intended to be recorded inside the ontology. The conceptual level of the systems engineering and functional safety ontology enables the engineers of both domains to share a conceptualization, hence enabling formal communication. At the level of the information contained inside the ontology, individuals (or instances) of the classes and their structure (*i.e.*, how they are related to one another using the properties defined at the conceptual level) are defined. For the ontology to be consistent, it is mandatory that they respect the structure

defined at the conceptual level. This section discusses the expressiveness at the individuals level of granularity and the integration of individuals with the focus on the satisfaction of cardinality constraints.

Integrity Constraints. For readability reasons and for readers non familiar with Open World Assumption (OWA), we presented the ontology using First Order Logic (FOL) and Closed World Assumption (CWA). As a reminder, CWA means that what is not known to be true is consequently false. In the ontology the integrity constraints are expressed with minimum and maximum cardinality axioms (an exact cardinality is equivalent to asserting a minimum and a maximum cardinality axiom with the same value). Because the integrity constraints are defined as axioms in the ontology, verifying the ontology consistency (including the individuals) automatically verifies that the integrity constraints are respected. It is clear that checking the ontology consistency will not result in a positive answer until all the individuals have been asserted and structured using the semantic relations of the domain ontology. This is not realistic as all possible individuals define a large set available only at completion. This is one of the reasons why we chose to implement the ontology using OWL that uses OWA so that ontology consistency can be checked early. In the following, we explain how the integrity constraints are checked in the ontology implemented in OWL and SWRL. Under OWA, what is not known to be true is not asserted to be true and therefore left unknown. When defining an individual that needs to be in relation with at least one other individual (axiom of the domain ontology that defines a minimum cardinality of 1), checking the ontology consistency under OWA will not result in a contradiction of this axiom. OWA acknowledges incompleteness of the information. It is understood that the individual is related to another, this individual is simply not yet defined and / or put in relation with the former one. The integrity constraints are nonetheless used in order to bring missing information to the surface such as the list of low level functional requirements that are not related to at least one function for instance. It is possible to implement this using a query language that has a CWA flavor: we use SQWRL on top of OWL and SWRL. Finally, SQWRL queries are used to check integrity constraints and if they return no result the integrity constraints are satisfied.

Unique Name Assumption. Our ontologies have actually been implemented using OWL and SWRL which do not assume the Unique Name Assumption (UNA). It means that individuals with different names can refer to the same underlying individual or not: they can be distinct. Even though it is possible to semantically infer that two individuals are actually the same, the concepts defined in the ontology do not allow to exploit such reasoning. UNA is commonly used at Renault by all the engineers so the intuitive way to use the ontology in order to define individuals is to assert that they are all distinct. However, even though the systems and functional safety engineers perform their activities following the top-down design process, it is actually possible for different names to be used to refer to the same underlying individual. On the one hand, as the activities are performed as much as possible in parallel to reduce development time, they suffer from iteration as prerequisite information necessary for their realization can change. For instance, the definition of the system architectures are first based on the requirements from systems engineering that do not address the dysfunctional behavior of the system. Based on these requirements, the functional safety engineers study the dysfunctional behavior of the system and use mental representations or models of the system architectures to produce safety requirements. These architectures are actually provided by system engineers but if they are not yet defined, safety engineers can make some assumptions on the architectures to continue their activities. At one point these architectures will have to match. As the two engineering fields are working on similar information, they can use different but more meaningful names for their respective domain that need to be integrated for validation. On the other hand, the implementation of the design process at Renault still mainly remains document-centric and suffers from human errors. Communication at the interfaces of systems engineering and functional safety processes is done by exchanging documents (*e.g.*, the System Stakeholder Requirement document is a prerequisite to the Preliminary Hazard Analysis, see figure 2.26). The information contained inside the documents is transformed manually into specialized domains models to produce or complete documents which suffers from human errors as the same underlying information can be named differently. Not using UNA represents this problem. It allows for the different domains to use the name of their choice to refer to an object. The information can be validated if the same individuals and

the distinct ones are asserted. As previously mentioned, using UNA is done by asserting that all the individuals are distinct. For the information to be validated, one has to check that no different names are used for the same underlying individual. We chose to allow engineers to use the name of their choice to refer to the same underlying individual. Moreover, tools that manipulate the information contained inside the ontology can take the opposite assumption (by asserting all the individuals, that are not asserted nor inferred to be the same, to be distinct). In particular, this is done when checking the integrity constraints.

2.3.1.3 Conclusion

In this section, we presented the systems engineering and functional safety ontology. This domain ontology captures a non ambiguous conceptualization shared by the systems and functional safety engineers that enables quality communication. From systems engineering, the ontology contains the different system descriptions (in terms of requirements, functions, flows and components) and how they are related to one another. From functional safety, the same descriptions are manipulated with additional ASIL information that comes from the supported PHA activity. As of now, the concepts of the ontology are integrated ,*i.e.*, they are precise because they were disambiguated. Section 2.3.2 is an attempt to involve the safety engineers as soon as possible during the design process. It presents how safety aspects of the system are defined as soon as possible in a more systematic manner and how we eased their progressive association with finer and finer grained system descriptions based upon the ontology.

2.3.2 Ontology Based ASIL Propagation

In section 2.2.2.2 we presented how functional safety is based upon the notion of ASIL that we represented as an attribute of the system descriptions. ISO 26262 can be viewed as the state of the art for the practice of functional safety in the automotive domain. It remains very general in order to reach all the actors in the entire field. The implementation of the standard is specific to an individual actor according to its characteristics. This section presents how the general design process of ISO 26262 at system level is understood

at Renault. In particular, ASIL assignment activity is made precise and we define a semi-automated analysis called *ASIL propagation* that assigns the correct ASIL to the concepts of the system which is possible only due to the semantic commitment with the systems engineering and functional safety ontology. Section 2.3.2.1 focuses on the introduction of safety aspects into the functional "genes" of the design and returns on traceability to remind about the different system descriptions that will inherit these genes. Section 2.3.2.2 defines the semi-automatic propagation of the ASIL throughout the different system descriptions. Section 2.3.2.3 gives the conclusion.

2.3.2.1 Systems Elements Traceability Establishment.

We presented the traceability during the design process from the requirements to the system architectures in section 2.2.1.5. Traceability records how the requirements are materialized into architectural elements of the system. As we address the development of critical systems, we advocate a top-down approach for the design process as it enables to consider safety which is an emergent property of the system that is impossible to observe at a finer grained level of detail. The intent is to introduce the safety aspects into the functional "genes" of the design in order for the system to be developed towards acceptable risk. In ISO 26262, this is done by defining safety goals as top level safety requirements. Then, these safety goals are made more precise into other safety requirements materialized into architectural system elements. At Renault, the objective is to involve safety engineers as early as possible so we use the functional requirements that are directly related to a system mission to start the safety activities. Axiom (2.57) expresses that each of these functional requirements are respectively the object of at least one degraded, lost and untimely failures which represent the systematic application of the failure model presented in section 2.2.2.1. The continuation of the PHA ends up with the ASIL being assigned to the safety goals.

$$\begin{aligned} \exists\, x\, \exists\, y\, \exists\, z \quad & p1{:}derives\_Mis\_FuncSystReq(m,r) \Rightarrow \\ & p2{:}fails\_FuncReq\_FuncReqDegradedFailure(r,x) \wedge \\ & p2{:}fails\_FuncReq\_FuncReqLostFailure(r,y) \wedge \\ & p2{:}fails\_FuncReq\_FuncReqUntimelyFailure(r,z) \end{aligned} \tag{2.57}$$

In ISO 26262, the safety goals are used to derive functional safety requirements that

are themselves used to derive technical safety requirements. Naturally, our intention is to respect this structure and propagate the ASIL from the safety goals to the functional safety requirements and then to the technical safety requirements. The structure recorded by the system and safety ontology enables to record traceability information that explains more precisely how the safety goals (which are low level stakeholders requirements) are considered. One feature of our ontology is that the safety engineers have been solicited as early as possible in the design process. The classifications of the safety goals as stakeholders requirements, the functional safety requirements as functional system requirements, and the technical safety requirements as non functional requirements on the system elements, highlight and promote their most important contribution in the system design. For instance, the safety requirements do not relate with the usual notion of a non functional requirement. We partitioned the requirements between functional and non functional requirements and defined that a functional requirement is used to deduce at least one function. It is under this interpretation that the safety goals are considered. As any other stakeholder requirement, they are taken into account by the functional system and external requirements (see section 2.2.1.2). In other words, the derivation of a safety goal into a functional safety requirement makes more sense from the point of view of systems engineering. Fundamentally, these requirements are functional system requirements which are considered by functions and components that respectively belong or are external to the system (see section 2.2.1.5). The safety engineers perform their safety studies which result in additional elements of systems engineering to ensure the safety property for the system. Ensuring their participation in the activities of systems engineering will improve the system design (without safety aspects) by benefiting from their expert knowledge (for instance, they can point out when a design is of poor quality because they know it will be hard to secure).

The safety goals actually come from the analysis of the functional system requirements that are related to a system mission. The design of the safety goals corresponds to the design of those functional requirements into functions and the components that realize these functions. The consideration of non functional requirements is naturally still mandatory for the design but can be treated in parallel. They are inconsequential with respect to functional safety. Functional safety analyses address both the functional system descriptions

and their physical implementation. Finally, the establishment of traceability should record how the safety goals are taken into account by the design using the relations presented in figure 2.47.

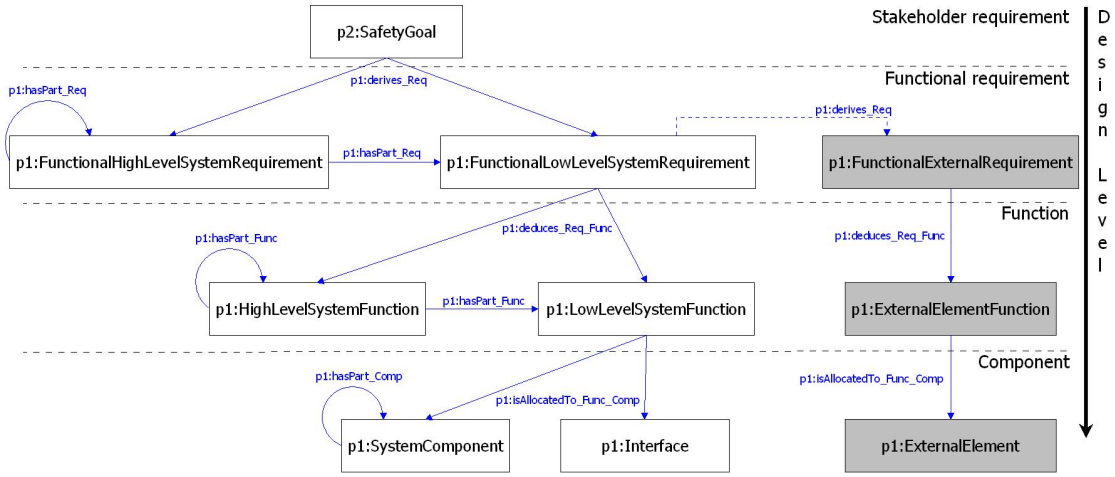

Figure 2.47: Traceability for functional safety

The figure presents the different concepts and relations that are used to establish the traceability relative with functional safety. Following the design process timeline, traceability is established between the different levels of detail of the process. The safety goals are taken into account by the development of the functional system requirements that were analyzed during the PHA. Informally, each safety goal should be related through a derivation relation with at least one functional system requirement. We enabled the decomposition of the functional system requirements into high and low level ones and explained that it was sufficient to establish traceability with the low level ones for all these requirements to be considered. The low level functional system requirements are used to deduce the system functions that have similar hierarchical structure. Downwards traceability is then established between functions and system components. Similarly, it is sufficient to establish traceability between the low level system functions and the components by means of allocation relations. These concepts and relations account for the system under development. On the right hand side of figure 2.47, the gray rectangles represent the external counterpart of the previous concepts gathered on the left hand side. The relation with the system environment enables to design functional safety as an emergent property that

can be observed at the adequate level of detail. The functional requirements, functions and components that are external to the system have the same structure with the only difference that we disabled their decomposition. The only point of detail concerns the functional external requirements. They can be related directly with the safety goals (as stakeholders requirements) but we really want to record the relation with the functional system requirements to explain that these external elements are indirectly involved in the functional safety of the system under development.

Finally, in addition to the informal semantics that the structure of the information needs to capture, we have to verify that traceability has been established in a consistent manner. The axioms on cardinality used in the ontology reflect the notion of integrity constraints that need to be fulfilled to conclude on the completeness of the information recorded inside the ontology. If the ontology (including the individuals) is consistent then all the integrity constraints are satisfied (and reciprocally). Clearly, this verification is possible only at the end of the design process when all the information has been given and all individuals introduced. Each cardinality constraint can and should however be verified independently following the design process timeline when necessary information is available by using a query language.

2.3.2.2 ASIL Propagation.

As mentioned in section 2.2.2.2, ISO 26262 is vague concerning ASIL assignment. It only specifies how the ASIL is assigned to the safety goals and how the ASIL is assigned to architectural system elements from the ASIL of the functional safety requirements in relation. We already presented the assignment of the ASIL to the safety goals. For the architectural elements, the rule is that the maximum ASIL of the functional safety requirements in relation with an architectural element is assigned to it. We retain this general idea that *the assignment of the ASIL should be conservative of the maximum ASIL of the elements in relation.*

Traceability records how functional safety is taken into account by the design process from the safety goals to the elements of systems engineering. If traceability is correctly established as presented in the previous section, it becomes possible to semi-automatically

assign an ASIL to the elements of systems engineering by propagating the ASIL from the safety goals. This is done using traceability relations. We called this a priori analysis *ASIL propagation*. The analysis is semi-automatic because ASIL can be reduced *a posteriori* using criticality analysis. Making this analysis automatic would result in the ASIL propagated (assigned) *a priori* to contradict the ASIL obtained with criticality analysis. For example, suppose that we only have one safety goal with an ASIL D in the ontology. Using ASIL propagation based on the traceability closure of the information, all the elements of systems engineering would be assigned the same ASIL D. However, if criticality analysis concludes that the ASIL of a specific architectural element can be tailored, for instance to ASIL C, then this element would have two different assigned ASIL, which contradicts the cardinality constraints of exactly one ASIL for an element. Let us note that the automatic propagation will propose highest ASIL required. The fact that some elements will be assigned two different ASIL will point out the necessity to check precisely its effective value. In the following, ASIL propagation is presented with different axioms and, as a result of the state of affair with ASIL tailoring, some of them must be considered with particular attention which is stressed when needed. ASIL propagation is performed using the traceability relations presented in figure 2.47, from the most abstract level of detail to the most concrete.

The ASIL of the safety goals can be propagated to the functional system requirements related by $p1$:$derives\_Req$. As different safety goals can be related to the same requirement, the maximum ASIL of these safety goals has to be assigned to the functional system requirement. Let $maxASIL$ be the function that returns the highest ASIL of a set. Axiom (2.58) can automatically assign the highest ASIL of the safety goals of a functional system requirement to this functional system requirement. However, the safety goals can also come from other interacting systems which functional safety can be impeded depending on the correct functioning of our system. This is problematic and it results that this axiom is actually not used for the automatic propagation. It is nonetheless saved as it represents

the intention for the development to follow a top-down approach.

$$\begin{array}{ll} let & \\ & Z = \{z \mid p2{:}hasASILValue\_Req\_ASIL(x,z)\} \\ & a = maxASIL(Z) \\ in & p2{:}SafetyGoal(x) \wedge p2{:}FunctionalRequirement(y) \wedge \\ & \quad p1{:}derives\_Req(x,y) \wedge p2{:}hasASILValue\_Req\_ASIL(x,z) \Rightarrow \\ & \quad\quad p2{:}hasASILValue\_Req\_ASIL(y,a) \end{array} \tag{2.58}$$

The functional system requirements are structured hierarchically using $p1{:}hasPart\_Req$. This hierarchical structure corresponds to the definition of the specification for the system where requirements are made more precise in order to be developed. Ideally, the safety goals are related to the most abstract functional system requirements (they are at the top of the hierarchy) as they are the result of the PHA done on the most abstract functional requirements that correspond to system missions. We want to propagate the ASIL from these functional requirements to their whole decomposition closure. There are three points of detail to be discussed.

First, the safety goals are expressed for risks evaluated with an ASIL greater than QM (that stands for Quality Management). To also record the information relative to ASIL QM (meaning that this information is not safety related), we use axiom (2.59). Informally, the functional system requirements that are the object of the PHA are assigned with the highest ASIL evaluated for their identified FCEs (see sections 2.2.2.1 and 2.2.2.2). Note that this ASIL propagation from FCEs to the functional system requirements in relation captures the intention of axiom (2.58) as the safety goals (of the system) are naturally related to these functional system requirements.

$$\begin{array}{ll} let & \\ & Z = \{z \mid p2{:}hasASILValue\_Req\_ASIL(y,z)\} \\ & a = maxASIL(Z) \\ in & p2{:}FunctionalRequirement(x) \wedge p2{:}failureImplies\_Elem\_FCE(x,y) \wedge \\ & \quad p2{:}hasASILValue\_HazEvent\_ASIL(y,z) \Rightarrow \\ & \quad\quad p2{:}hasASILValue\_Req\_ASIL(x,a) \end{array} \tag{2.59}$$

Second, while axiom (2.59) captures indirectly the propagation of the ASIL from the safety goals to the functional system requirements with the additional information that some of them can be assigned an ASIL QM, the axiom does not capture the reality completely as some safety goals can come from interacting systems. These safety goals can

actually be related to more precise requirements than the functional system requirements at the top of the hierarchy which has to be considered for the propagation. For instance, assume that we are developing a brake system. The most abstract requirement that corresponds to the system mission of deceleration is *"The braking system shall decelerate the vehicle"*. Using the relation of decomposition, this requirement is made more precise and the deceleration is actually possible by capturing information about the vehicle speed: *"The braking system shall capture the longitudinal speed of the vehicle"*. Naturally, deceleration is safety critical and the requirement should be assigned an ASIL D that is propagated throughout its decomposition closure. For the purpose of the example, we assign an ASIL C to the two requirements. The information on the longitudinal speed is also needed by the steering system. At Renault, a safety goal actually corresponds to the direct negation of a feared customer event with an ASIL. This enables to address functional safety at the customer level where the safety property is observable. The dysfunctional studies for the steering system resulted in the definition of the safety goal *"No loss of steering, ASIL D"* and the assignment of this same integrity level to the external functional requirement *"The braking system shall send the longitudinal speed of the vehicle"*. For the braking system, this requirement is a safety goal (and a stakeholder requirement) that relates with a functional system requirement part of the decomposition of the capture of the longitudinal speed. In this case, propagating the ASIL of the functional requirements will result in the assignment of two different ASILs to some requirements. Naturally, the most stringent has to be assigned.

Third, ASIL decomposition can be applied toany high level functional system requirement, see section 2.2.2.2. If a requirement is the object of ASIL decomposition, information on objective ASIL (*i.e.*, the ASIL preceding ASIL decomposition) must also be recorded and propagated.

For space reasons, further axioms that are used for ASIL propagation are given in appendix B. ASIL propagation in the hierarchy of functional system requirements addresses the previous issues and is performed with the axioms gathered in appendix B. Note that the safety goals of the system are intended to be related to all the top level requirements in the hierarchy of the functional system requirements. This was not enforced but can be

easily checked.

Axiom (B.1) expresses that a functional system requirement is assigned the highest ASIL among the ASILs of its direct super-requirements that are not the object of ASIL decomposition, its related safety goals, and of the relevant part of ASIL decomposition the requirement (as a sub-requirement) is the object of. Then, if a requirement is the object of ASIL decomposition, its sub-requirements that have their ASIL tailored and their hierarchical closure are assigned an objective ASIL (*i.e.*, the ASIL of the super-requirement).

Axiom (B.2) states that a functional system sub-requirement is assigned the highest ASIL as an objective ASIL among the ASIL of the super-requirement that is the object of ASIL decomposition. This also concerns the sub-requirement and the objective ASIL of its direct super-requirements.

Following figure 2.47, we now study ASIL propagation from the functional system requirements to the system functions. The propagation of the ASIL from the safety goal throughout the hierarchy of functional system requirements is correctly done, *i.e.*, the highest ASIL has been assigned and all the low level functional system requirements have an ASIL attribute instantiated. In addition, information about objective ASIL has also been propagated if it applies. As we explained, in order to take into account all the high level functional system requirements, it is sufficient that only the low level ones are related to the functions. This still applies to safety aspects so the propagation is defined from the low level functional system requirements throughout the hierarchy of system functions.

Axiom (B.3) expresses that a system function that has no ASIL assigned, is assigned the highest ASIL among the ASILs of both its direct super-functions and its related low level functional system requirements.

Axiom (B.4) assigns objective ASIL to system function in the same way.

As previously discussed, the system functions (as architectural elements) can be subject to ASIL tailoring as a result of criticality analysis. If axiom (B.3) is used to effectively assign the ASIL to the system functions, ASIL tailoring will possibly assign a different ASIL to some of them and result in a contradiction. Moreover, part of the propagation

could also be too restrictive as an ASIL that is too high is assigned to the sub-functions of the function that have seen its ASIL tailored by criticality analysis. In order to ensure that we do not assign two different ASIL to the same function, the consequences of axiom (B.3) (*i.e.*, the assignment of the ASIL to the functions) are treated only as entailed conclusions that must not be added to the ontology until criticality analysis has ended for the functional architecture.

Let us note that it is possible that not all the system functions are assigned with an ASIL (even as entailed conclusions) as the structure enables to relate the functional system requirements to sub-functions. Traceability (more precisely, the cardinality constraints and the use of axiom (B.3)) ensures that all the low level system functions have an ASIL which is enough to present ASIL propagation. We nevertheless defined that all the functions should have an ASIL so the capability to backward-propagate the ASIL from the low level functions to the top of the hierarchy exists but is not presented.

Similarly, now that all the low level system functions have an ASIL assigned (this is sufficient for the consideration of all the system functions), it is propagated to the system components.

Axiom (B.5) expresses that a system component that is not already assigned with an ASIL is assigned the highest ASIL among the ASILs of its direct super-components and its related low level system functions.

Axiom (B.6) assigns objective ASIL to system components in the same way.

Criticality analysis can also be used on the components for the tailoring of their ASIL, the consequences of axiom (B.5) are treated only as entailed conclusions until criticality analysis has ended for the physical architecture. Similarly, functional system requirements can be related to sub-functions, low level system functions can be allocated to system sub-components resulting in the possible existence of a component with no ASIL assigned to it. It is important that all the components have an assigned ASIL. The components partitioning into hardware and software reflects the selection of which components will be subject to further development (at hardware and software level) with specific requirements of ISO 26262 defined by the ASIL of the component. As with axiom (B.5), all low level

components are assigned to an ASIL, we can fully (*i.e.*, completely) backward-propagate the ASIL of the components for all the components to have an ASIL.

Axiom (B.7) (respectively axiom (B.8)) states that a system component that is not already assigned with an ASIL (respectively objective ASIL) is assigned the highest ASIL (respectively objective ASIL) among the ASILs (respectively objective ASILs) of its direct sub-components. The consequences of these axioms can be produced but are also treated as information that is not added to the ontology as they can be subject to criticality analysis.

Finally, we address the propagation of the ASIL to the elements that are part of the system environment. As can be seen in figure 2.47, the functional external requirements are derived from the functional system requirements. This traceability between these two types of requirements should be inspected with care. It is possible and it makes sense to derive a functional external requirement from an abstract functional system requirement. However it is possible, because of ASIL decomposition, that we assign an ASIL that is too high to an external requirement. Therefore the traceability has to be established with the functional system requirement that is the less abstract.

Axiom (B.9) propagates the correct ASIL to all the functional external requirement (the highest ASIL of the functional system requirements in relation with one functional external requirement is assigned to this requirement).

Axiom (B.10) does the same for the objective ASIL of the functional external requirements.

Axiom (B.11) propagates the ASIL from the external requirements to the external functions

Axiom (B.12) is the similar counterpart of axiom (B.11) for the objective ASIL.

Applying axioms (B.13) and (B.14) will respectively result in propagating the ASIL and objective ASIL at component level from the external functions to the external components (external system elements).

2.3.2.3 Conclusion

This section covers the interpretation of ISO 26262 specifically for Renault. In this approach, the safety studies start as early as possible. Safety aspects represented by the ASIL are integrated in the most abstract system elements that will be made precise along the design process timeline. Based on the system and safety ontology, ASIL assignment is presented as the semi-automatic propagation of the ASIL from these most abstract elements to the most concrete ones at system level. If the general idea for ASIL assignment is relatively shared by all the actors of functional safety, it is still done manually in practice as a result of document-centric approaches. Here, ASIL propagation is defined precisely for ASIL assignment to be done correctly.

2.3.3 Conclusion

In this section, we presented the domain ontology for both systems and functional safety engineers. It is defined by integrating the two ontologies from section 2.2. This made apparent the ambiguities of the universe of discourse that had to be solved when these two domains are confronted with one another. The result is a systems engineering and functional safety ontology. It defines formally the concepts and relations of the two domains enabling precise communication between the systems and safety engineers. Specifically, the now shared conceptualization solves loss of knowledge at the processes interface. Moreover, it is an asset for the synergy of all the engineers and encourage for their tighter collaboration. As an example of these synergies, ASIL propagation has been presented. It exploits traceability to support the correct top-down propagation of the safety aspects (*i.e.*, the ASIL) to the elements of systems engineering. The same conclusions can be formulated for any domain ontology. The two domains integration, however, increases the scope of these conclusions. This exceeds the reach of this section, but, finally, consistency is now a property shared by the two domains.

The systems engineering and functional safety ontology is one of the concrete contribution of this work. As an answer to a need for better formalization, it is the most precise solution. The ontology is realized with the help of Protégé 3.4.4[4] which is an editor of

[4] `protege.stanford.edu`

OWL ontologies. This version of Protégé includes in particular a plug-in for SWRL and an interface with the rule engine Jess[5] to execute SWRL rules. As of now, the ontology consists in 140 classes, 103 properties and 391 constraints. The generated documentation is available at the following address: `http://cedric.cnam.fr/~taofif_o/`. In section 2.4, we present the extent of having our domain ontology to improve the design process at Renault.

2.4 Ontology Centric Design Approach for Safety Critical Automotive Mechatronics Systems

The development of automotive mechatronic systems requires the participation of different professional fields (*e.g.*, vehicle architecture, mechanics, electronics, software, *etc.*), each having its own language, its own jargon. Knowledge and information are often implicit to one specific professional field. They are known to experts or specialists of the profession, but are not always well capitalized and, therefore, they are unknown to the other fields or, even worse, lost if those experts or specialists change of position. It is the role of system engineers to effectively take into account all those system stakeholders (*i.e.*, the professional fields concerned with the system) and orchestrate their contributions in the big picture as to develop a correct system solution. We underline in particular the heterogeneous nature of the automotive industry where many tools (and languages) can be used by Renault and its suppliers. For example, used or considered tools include (but are not limited to) Reqtify for requirements management and traceability, UML / SysML Papyrus or Enterprise architect for model development, ArKItect for the whole development process including safety activities, Matlab/Simulink for the analysis of functional behavior, Statemate for the analysis of physical behavior, Aralia for the development of fault trees, and so on... System engineers must overcome a consistency problem to integrate this heterogeneous environment.

From a syntactic point of view, the consequences are not too severe. Syntax consistency problems arise when two different terms are used to denominate one same thing. As a usual example, we often work with documents and models that have terms in English and French

[5] `protege.cim3.net/cgi-bin/wiki.pl?SWRLTab/`

languages. Consequently, we might say that working with two models with different names *just* takes more time. As the meaning is not altered, we can somehow understand how it all comes down together. It is just a matter of realizing *"he calls this thing that way"* and living with that. From the semantic point of view, however, the problem takes a completely different dimension. The problem can be resumed to the utilization of one same term by two different professional fields to designate respectively two different concepts. This can lead to situations that are so contradictory that we might end up trying to solve a problem with no solution. Ultimately, when the actors of different professional fields exchange some information or knowledge; some are lost either by communication omission or by misinterpretation of this information [Burr et al. 2005]. The other possible consistency problem is less fundamental but equally important and consists in inter-domain consistency. As the development is executed in parallel by different fields, each of them relies only on the information relevant for their studies. The information manipulated by the different fields (used or produced information) can intersect and the difficulty is to guarantee that all the fields are working with consistent information ensuring consistency of the design process [Papadopoulos et al. 2001].

Section 2.4.1 presents the systems engineering and functional safety ontology as the reference model placed at the heart of the system design process. The ontology enables to *guarantee the design process consistency*. These ideas were the subject of a precedent communication that can be found in [Chalé Góngora et al. 2011]. Section 2.4.2 presents our design approach for safety critical automotive mechatronics systems at Renault which is more precise than the current general one. Finally, section 2.4.3 gives the conclusion.

2.4.1 Place of the Ontology in the Design Process

Renault is currently transitioning to a Model-Based System Engineering (MBSE) process for the development of its vehicle systems. The use of formal and informal (but consistent) models to create a common semantic model is expected to facilitate systems engineering activities and to avoid the encountered drawbacks of previous document-centric implementations of the process, which were lacking semantic consistency among the different modeled objects [Chalé Góngora et al. 2009, 2010]. The objective of MBSE is to

produce and control a consistent, correct and complete global model of the system, which contains all information that specify, design or will allow verifying and validating the system. The main benefits, as they are emphasized by Estefan [2008] and Friedenthal et al. [2008], of implementing MBSE include the following:

- improved quality through a more rigorous and costless traceability between requirements, design, analysis and testing
- increased productivity through the reuse of models and automated document generation
- enhanced communication by integrating views of the system from multiple perspectives

The risk of developing inconsistent models that have different conceptualizations of the same system according to their own viewpoint still remains very present. Incompatibilities or inconsistencies between models discovered too late in the development process may produce huge costs. Consistency is then a crucial issue and needs to be maintained at all levels in the development process. In a MBSE approach, the consistency problem can be formulated as the demonstration of the consistency of any models couples. As shown in figure 2.48, we propose to introduce the systems engineering and safety ontology as the central element of the system design process. In this figure, we separate into two branches the activities pertaining to system design and safety presented in sections 2.2.1 and 2.2.2. The ontology is instantiated for the system under development. This instantiation serves as the *consistency reference model* for the project.

2.4.1.1 Use of the Reference Model

The actors of a development project, independently of their respective fields or area of expertise, will refer to the ontology (a shared conceptualization of the system and safety engineering domain and of the system under development) to verify and validate the compliance, the completeness and the consistency of the information (*i.e.*, documents and models) produced by the system design and safety activities.

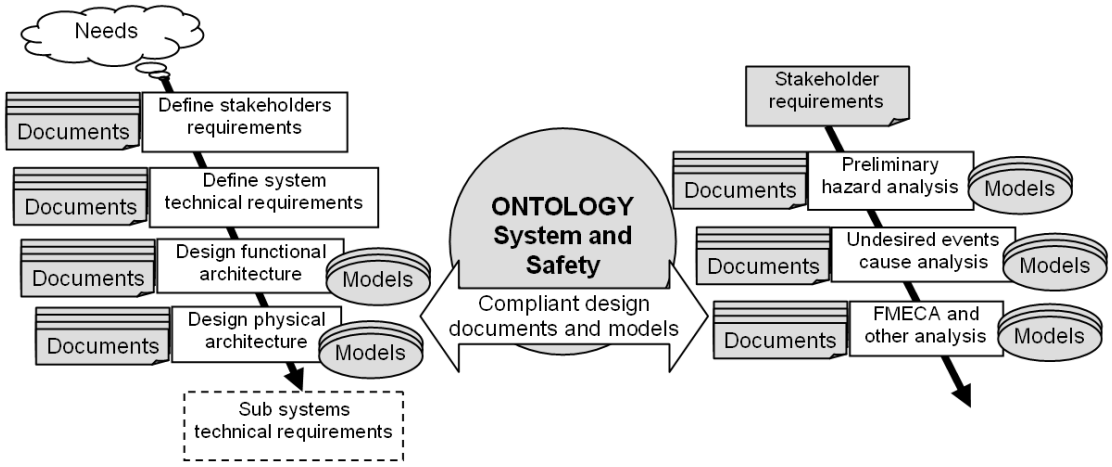

Figure 2.48: Central role of the system and safety ontology in the design approach

Figure 2.49 illustrates the possible uses of an ontology in a model-based approach. The figure presents the example of two Simulink models, but the approach is applicable to other types of models. In this example we are interested in the signals (*i.e.*, the solid straight arrows) of Simulink models. The ontology models this concept with the flow class and its attribute (not represented in the figure) maxValue. We can define semantic consistency relations with the help of transformations or mappings between the domains of the ontology and of the language of the Simulink tool, on the one hand, and between the instances of the ontology and the instances of Simulink elements, on the other hand.

Assuming we have defined that Simulink signals are equivalent to the ontology flows, it is then possible to:

1. Enrich the ontology: All the signals of a Simulink model will enrich the ontology instances. In the figure, the signal *Torque_ Frein_ Electrique* of the Simulink model defines the flow *Flow_ 001* in the ontology. For this flow, we define a unique maximal value of the braking force *maxTorque*.

2. Use the knowledge in the ontology: A second Simulink model will be able to use the flows of the ontology and gather the information previously defined. In the figure, the flow *Flow_ 001* of the ontology and the signal *Electrical_ Brake_ Torque* of the second model are equivalent. In Simulink, this signal should connect to a port that enables to type the flow. In our example, this value has an upper bound equivalent

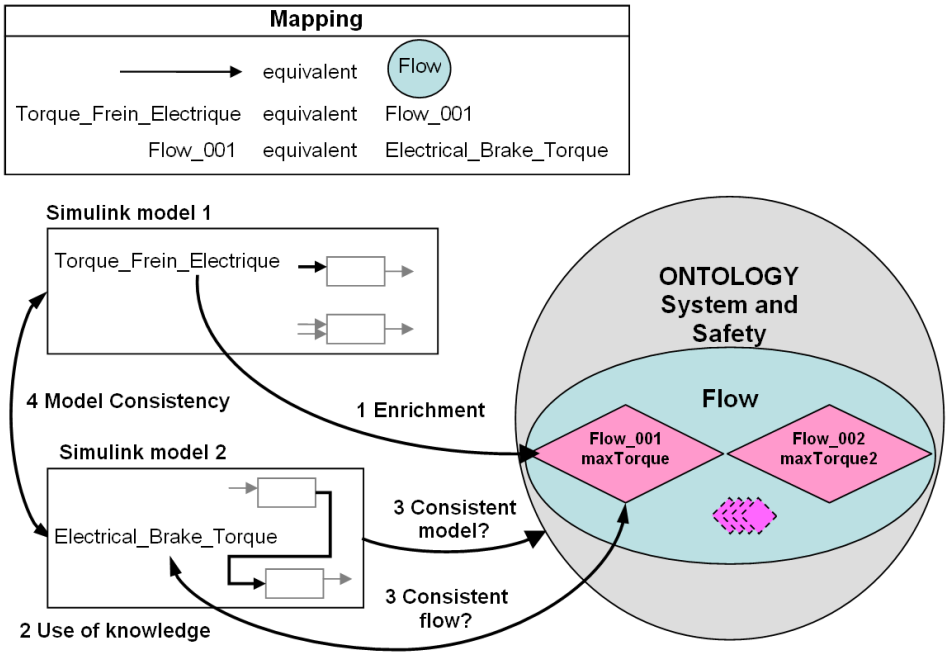

Figure 2.49: Uses of an ontology as a reference model of a MBSE approach

to *maxTorque*.

3. Verify the consistency of a model with respect to the ontology: If the signal *Electric_ brake_ Torque* (that models a flow representing the braking torque of the electrical engine) does not exceed in simulation the maximum value *maxTorque*, then the signal is coherent with the ontology (for the *maxValue* relation). Generalizing to all the relations defined into the ontology, we can assess the consistency of a model compared to the ontology.

4. Verify that two models are consistent: If we have defined a mapping between *Flow_ 001* and *Torque_ Frein_ Electrique* and between *Flow_ 002* and *Electrical_ Brake_ Torque*, then a user can notice that those two instances are equivalent since they represent the same element in the system, even though two designations are used in the models (one in French and the other in English). Defining an equivalence between those two instances will however result in an inconsistency. In figure 2.49, two different values have been defined for *maxValue* (*i.e.*, *maxTorque* is different from *maxTorque2*) whereas in the ontology we specified that a flow can only have one maxValue. In

the opposite case (*i.e.*, *maxTorque* is equal to *maxTorque2*) and generalizing, the two models are consistent and, once again, they describe the same system and we have some evidences that a solution for the system exists. Generalizing even more, it becomes possible to verify some form of consistency of the whole system design through an ontology.

In a general manner, the project actors create information in documents and models. The ontology will enable them to verify the consistency, the completeness and the conformity of the produced information. Once verified, the new information can be imported into the ontology [Kergosien et al. 2010]. The ontology can also be viewed as a knowledge base. Queries can then be developed to bring relevant information to the surface such as the list of requirements related to one specific function for example. Reference information can be exploited to produce new views (system views) and generate new information [Sure et al. 2002]. In our ontology, we generate information about a priori ASIL for instance. The ontology is therefore the reference (model) that, on the one hand, contains the reference information that describes the system under development and, on the other hand, connects the information it contains with the information present in the documents and models produced during the course of the system development project.

Another key element in MBSE is the transformation of models which allows the definition and implementation of operations on models. Using model transformation enables the automated or computer assisted development of a system from its corresponding models. Similarly to its role to ensure semantic consistency of models, the ontology can ensure semantic integrity when using model transformation. This is the object of the next section.

2.4.1.2 On Model Transformation

Model transformation is an essential part of the MDA framework (see section 1.2.3.2). In this framework, models are based on meta-models that comply with the Meta-Object Facility (MOF) standard of the OMG that uses the layered concepts of instance, model, meta-model and meta-meta-model. Model transformation is the automatic generation of a target model (the result of the transformation) from a source model (the input of the

transformation) by a transformation engine according to a transformation model (a set of transformation rules), see figure 2.50 below.

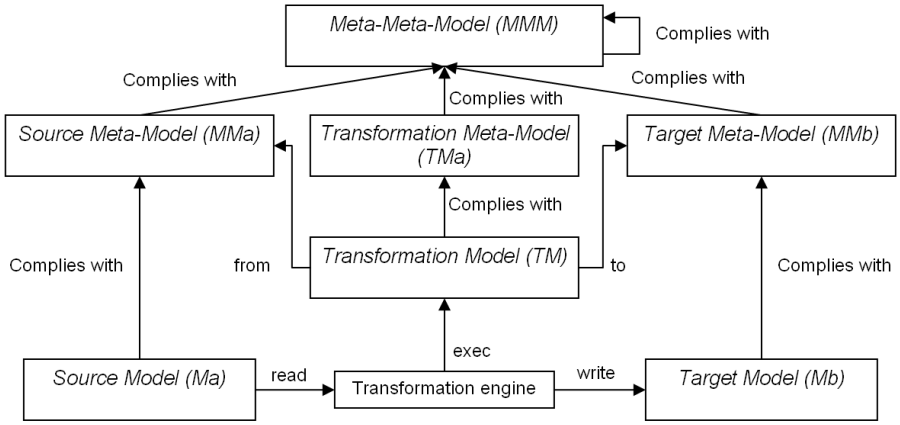

Figure 2.50: Elements of model transformation

Transformation rules are defined whenever possible for the meta-model level and written as expressions of transformation languages. In the MDA framework, transformation rules are entered into a transformation tool, which can then automatically interpret them and execute the transformation. For that purpose, a formal syntax for writing transformation rules must be defined [Anneke et al. 2003]. In the case of automatic model transformations, the mapping between the different concepts has to be developed only once for a pair of meta-models, not for each model instance [Levendovszky et al. 2002]. Therefore, the specification of meta-models is a prerequisite for the execution of automatic model transformation.

In MBSE, a model allows capturing the relevant aspects of a system from a given perspective, and at a precise level of abstraction. During the system development, different model types are realized to represent specific possible system views for any of the design process activities (specifications analysis, system architectural design, validation and safety analysis). The models should contain only the aspects needed to support the design process phase they are used in, hiding unnecessary complexity. Models are supported by languages that have at least a well defined structure (*i.e.*, syntax) and in some cases a well defined meaning (*i.e.*, semantics). When the syntax and semantics are well defined (*i.e.*, mathematically defined) the language is connoted formal. In MDA, meta-models are used

to define the syntax and semantics of languages. Most meta-models are semi formal in the sense that their syntax is formal but their semantics is not. Following those semi formal meta-models, model transformations operate essentially at the syntax level but always embed implicitly some semantic knowledge [Roser and Bauer 2005] that ensures the inter-model consistency. We argue that the system and safety ontology can make explicit the part of the semantic knowledge that is common to the source and target domains involved in a transformation.

We propose a framework that ensures model transformations consistency (at the semantic level) with the ontology. Figure 2.51 illustrates this framework. We build upon the framework of model transformation as defined by the OMG so we can see two meta-models, source and target, with the transformation model in the center. For the purpose of the example we represented two transformation models in the figure. At the top of the figure, we place the system and safety ontology that defines those domains with their concepts and relations. Actually, both meta-models and ontologies can be used to define concept and relations [Söderström et al. 2001] so meta-models and ontologies can be used independently as the meta-model in MOF for model transformations. The terms mapping and transformation are interchangeable and can be used indifferently but the term mapping is encountered more frequently in ontology literature so we will use this term when the transformation involves an ontology. In the framework, a model transformation is still defined independently from the ontology. However, the meta-models involved in the transformation need to be mapped with the ontology, so we define one mapping from the ontology to each meta-model (*Source Mapping* and *Target Mapping*). In the figure, we only represented some concepts of the meta-models and the ontology but the idea can and has to be generalized to relations. Mapping the ontology to the meta-models involved in a transformation enables to define the concepts of the meta-models that are equivalent with respect to the ontology. For instance, the concept C of the source meta-model and the concept i of the target meta-model are equivalent as they are respectively mapped with the same concept *3* in the figure. Those equivalent concepts enable to define the consistency of a transformation with respect to the ontology.

In the figure, *Transformation Model 1* is consistent with the ontology as all the transfor-

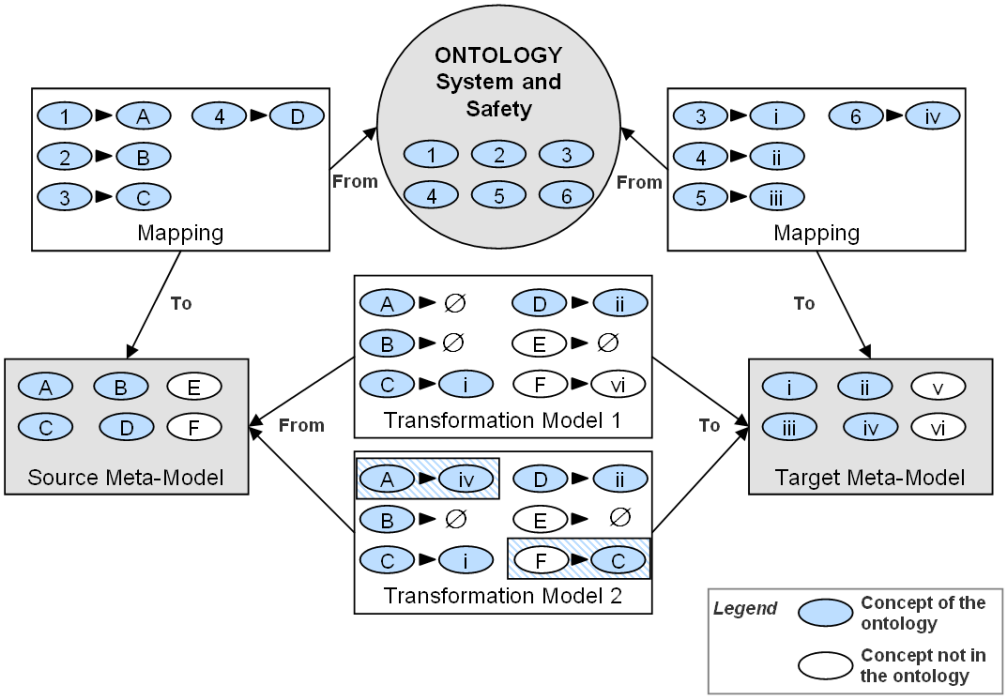

Figure 2.51: Model transformation framework

mation rules are consistent with the ontology, *i.e.*, all the source concepts that are mapped with the ontology are transformed into their equivalent target concept and all the target concepts that are mapped with the ontology have been transformed from their equivalent source concept. If a source concept is not subject to transformation (*e.g.*, concept *A* is not the object of transformation) the consistency property still holds. The framework does not allow checking the consistency of concepts outside the ontology so we disregard the transformations of those concepts as they are neither fundamental for systems engineering nor for safety. *Transformation Model 2* illustrates the contrary as source concept *A* is transformed into target concept *iv* and those concepts are not mapped to the same ontology concept (*1* is mapped to *A* and *6* is mapped to *iv*. *1* and *6* are not equivalent). The transformation is inconsistent with respect to the ontology. If a source concept not mapped with an ontology concept is transformed into a target concept mapped with an ontology concept then the transformation is inconsistent (*e.g.*, source concept *6* is transformed into target concept *iii*). Reciprocally, if a source concept mapped with an ontology concept is transformed into a target concept not mapped with an ontology concept, the

transformation is inconsistent.

This framework enables to guarantee tools interoperability at the semantic level. The ontology presented in the previous section explains how system and safety engineering have to be understood. Within the framework, we can verify that the SE design tools implement correctly the ontology at the semantic level (*i.e.*, the meta-model implemented by the tool is consistent with the ontology). Moreover, the framework shows how those tools can inter-operate seamlessly via model transformations that are consistent with the ontology. Finally, this framework enables to evaluate if the language used and the transformations proposed by a tool correctly implement the ontology and, therefore, if this tool can be used to support the system design process at Renault.

2.4.1.3 Conclusion

In this section, we presented the numerous advantages of an ontology based development. In a heterogeneous environment where different domains have to communicate, *ontologies can solve the problem of semantic integration.* This is naturally very constraining as mappings have to be defined between the ontology and each meta-model used in specific tools. This issue is however inherently related to tool interoperability that is criticized with "the ambient paradigm of distribution ("tools that hardly work together except hopefully one day")" [Albinet et al. 2010]. Tools are actually developed, independently (different tools), and for different manufacturers. Making them work together remains manufacturer specific. The ontology can help answering if a tool actually corresponds to Renault needs (it is consistent with the ontology), through mapping definitions, and therefore if it can be integrated in a development process for Renault. Finally, not only ontologies enable to ensure seamlessness during the design process, it does so at the semantic level. The next section presents our recommendations to improve the quality of the complex design activity. It is a general proposition that could be used at Renault which is used as a specific example.

2.4.2 Ontology Centric Design Process

In order to improve Renault design process and facilitate its transition from document-centric to model-based systems engineering, we chose to introduce an ontology that can guarantee the whole design process consistency at the semantic level. We expect this ontology to improve the design process in a number of ways. First, the activity of writing requirements and establishing their traceability is done with the semantic interpretation defined in the ontology and can somewhat bridge the gap between the informal world of discourse and the more formal worlds of models. Secondly, making usually implicit knowledge explicit further enables the front loading of activities, while engineers remain capable to make informed decisions, and help to identify areas susceptible to automatic or semi-automatic procedures used in Model-Based Design. In addition and more generally, going towards more formalization (at the semantic level) and sharing the formalized conceptualization is an enabler for reuse. Thirdly, committing to the ontology makes it possible to work at inter-disciplinary level, the foundation coming from the fact that the system is indeed the common rally point to different professions. For instance, analysis such as multi-model consistency and coherence gives confidence that it is the same system that is being built. Finally, incorporated concepts from ISO 26262 help to demonstrate the compliance with this standard.

With respect to the amount of work that will be required for these improvements to come to life, it seems particularly fitting and it is our recommendation to introduce a new actor, an ontology engineer, whose role will consist in ensuring a seamless consistent development. Considering the chaotic nature of current systems engineering practice (for instance tools such as Powerpoint, Visio, Matlab/Simulink, Enterpise Architect, Parpyrus and so on can be used to define the system architecture), the actual challenge for the transition to model-based systems engineering is to evaluate formalisms adequacy to Renault needs in anticipation of a future domain specific language for Renault. As such, the general design process remains unchanged and we introduce the ontology engineer role in parallel to systems and functional safety engineer roles as can be seen in figure 2.52 that uses Business Process Modeling Notation (BPMN).

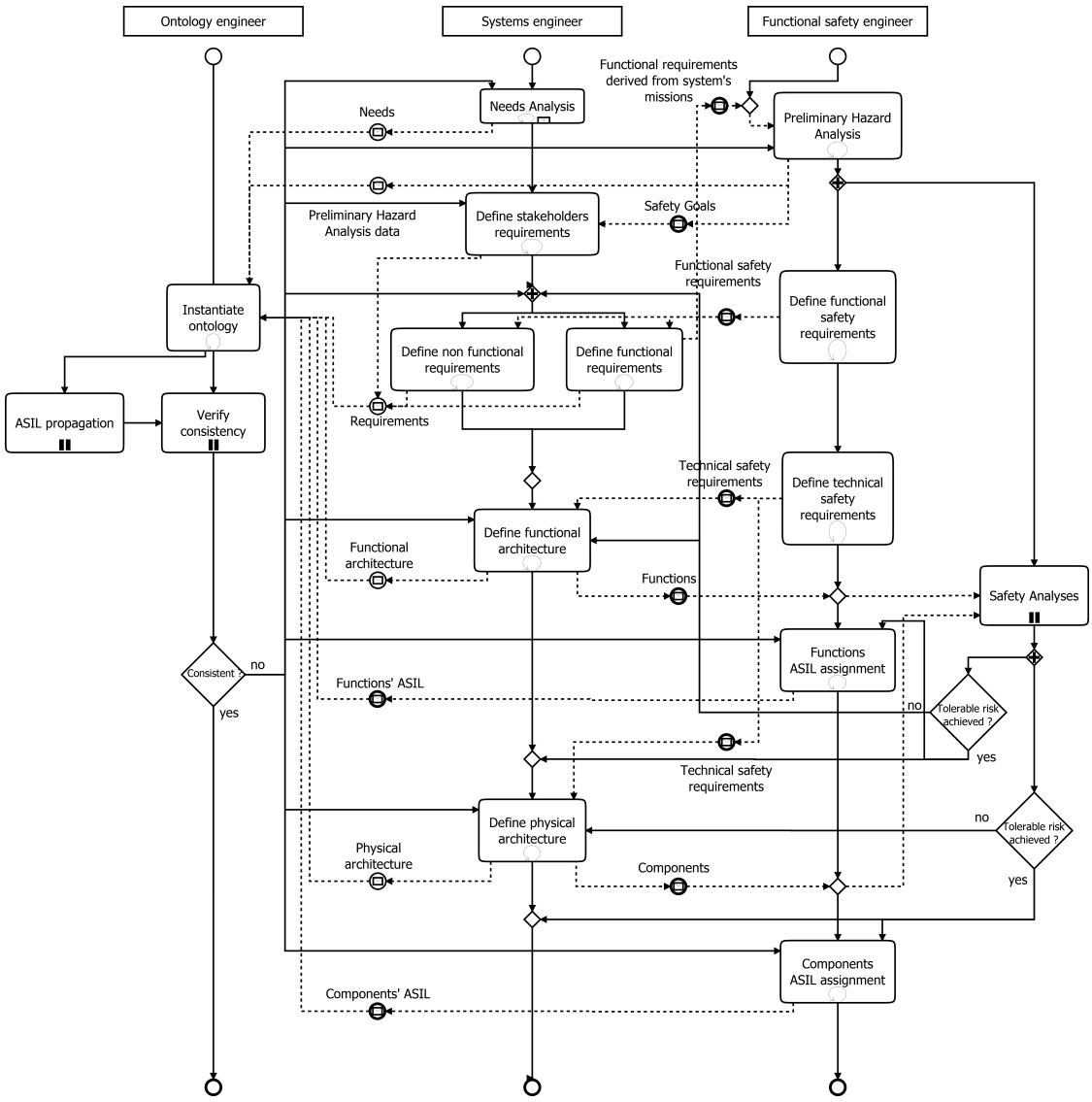

Figure 2.52: Design process' BPM

Activities are represented with rectangles. System engineers and functional safety engineers activities are represented in the middle and on the right hand side of the figure. Ontology engineers activities are on the left hand side of the figure. Information exchanges are represented with dashed arrows. To keep it simple, ontology engineers activities are to instantiate the ontology with received information, verify its consistency and perform other possible analyses given the semantic structure of the ontology (such as ASIL propagation). In case of a detected inconsistency, they should communicate the issue to the other engineers to the impacted activities. In the remaining of this section, we discuss about specific

aspects of our approach and give our recommendations on some activities that need to be brought to maturity.

2.4.2.1 Understanding Design Process Knowledge

It is important to advocate a top-down approach for the design process. On the one hand, it enables to consider emergent properties at system level (where they are observable) so that they are effectively taken into account at finer grained levels of detail. On the other hand, it enables to understand the information produced during the design process following the logic of a top-down approach which facilitates reuse. Reuse is done by selecting some elements that will be reused and producing missing information using a dual bottom-up approach. The general idea is that activities are sequenced backwards to produce information. For instance, reusing a component will require the production of related functions and requirements. As good design usually requires a complete understanding of the system, theoretically, information should display a top-down logic advocated in systems engineering. Already having information that follows a top-down logic is therefore an enabler for reuse. That being said, the design process and producing knowledge is not linear in reality. Figure 2.53 presents the design process as a spiral to represent its iterative nature. Activities are sequenced clockwise and reciprocal arrows represent possible fallbacks when an issue is detected.

Knowledge is actually constructed for parts of the system with more and more details being added. For example, system stakeholders are identified with respect to available knowledge. For instance, adding a stakeholder means that a relation to a need has to be defined (see figure 2.2). If the need does not exist it has to be defined. Then this need has to be related to a stakeholder requirement. If the stakeholder requirement does not exist, it has to be defined. And so on, until knowledge is complete. As knowledge about the system is produced, new stakeholders can be identified which can be the origin of a design iteration to consider such new information. The capability to state that everything is consistent, and the opposite, is therefore essential.

The integrity constraints that we defined can seem to be very strong. For instance, the different types of requirements defined in the ontology account for system, functional and

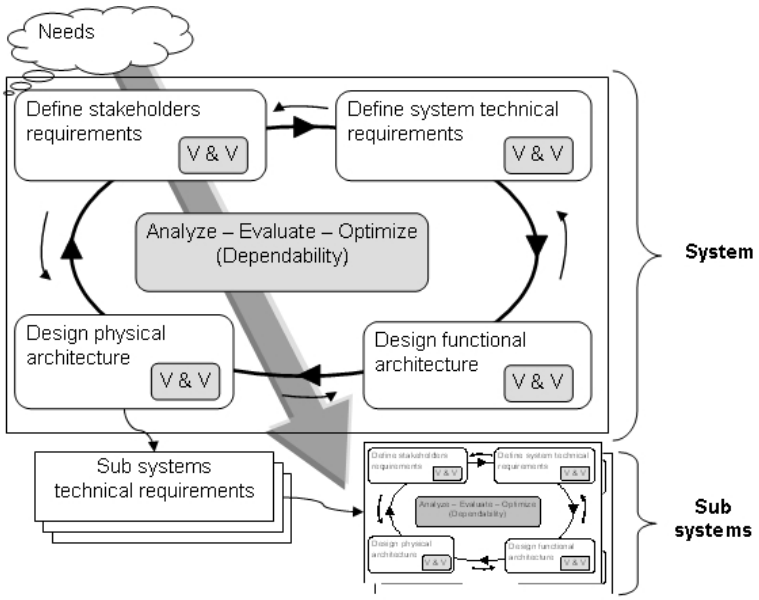

Figure 2.53: Spiral model of the system design process

physical level of detail. Some requirements can however appear directly at intermediate levels, like exported requirements, and it can be confusing to trace back these requirements to former levels [Albinet et al. 2007]. It results in our approach that, even though such information was not produced following a top-down logic, the top-down logic should be constructed nonetheless both for reuse and to comply with the rigor necessary for safety critical systems development.

2.4.2.2 Need Analysis and Stakeholders Requirements Definition

The first activity is *need analysis*. Stakeholders that are identified express their needs. The objective is to remove ambiguities, lacks and inconsistencies likely to be present in stakeholders needs. The concepts presented in section 2.2.1.1 are documented. As can be seen in figure 2.52, we use system missions to already define functional requirements so as to start safety activities as soon as possible.

Define stakeholders requirements transforms the needs into requirements which are more structured than natural language. We recommend the use of boilerplates as defined by Hull et al. [2004] for writing requirements. Boilerplates add structure to natural language and leads to the use of conceptual elements. The following boilerplate is given as an example

for stakeholders requirements:

The <stakeholder type> (shall or may) be able to <capability>

where *stakeholder type* is a stakeholder, verbs *shall or may* represent different priorities of requirements (high when using shall and low when using may) and *capability* represents the functionality to be developed.

One particular type of stakeholders requirements are safety goals. Functional safety engineers should be solicited by system engineers whenever risk of harm can be foreseen in advance or even when they are not certain that it is the case. Usual safety critical systems in the automotive are braking and steering. However the scope begins to enlarge as new systems or even systems of systems are being developed at Renault such as electric vehicles and automatic vehicle fleets. As explained in section 2.2.2.2, safety goals result from the PHA. They are composed of a Feared Customer Event and an ASIL. In terms of requirements, a safety goal actually corresponds to the direct negation of a feared customer event with an ASIL so we propose the following boilerplate:

The <stakeholder type> shall not be subject to a <FCE>, ASIL <ASIL level>

where *stakeholder type* is a stakeholder, *FCE* is the feared customer event in question and *ASIL level* corresponds to A, B, C or D as defined in ISO 26262.

In general, ontology engineers receive all informations produced (here needs and stakeholders requirements) and perform *instantiate ontology* in parallel (see figure 2.52). The difficulty comes from the heterogeneous and numerous formalisms used that are specific to each project. For now, Renault general system design process is defined but it is instantiated depending on the project. One project manager can choose to use specific tools or use suppliers that can come with their respective tools. Other project managers can make different choices. In order to instantiate the ontology, mappings have to be defined. For need analysis general purpose modeling languages such as SysML can be used but still informal formalisms are more commonly used. For instance, techniques such as APTE[6] method's "bête à cornes"[7] can be used where some stakeholders are identified and system

[6] `http://www.methode-apte.com/`
[7] `http://www.methode-apte.com/bete_a_cornes.htm`

goals defined. In both cases, these formalisms can present a delta with the ontology and some level of interpretation is applied to define a mapping. Implementing a mapping remains arbitrary as complete coverage of the diversity of languages is unlikely. It is actually the same for any activity. The rule in model-based design is to use a modeling language but the actual transition will not be made until modeling languages and tools to be used are clearly defined revealing necessary mappings implementation.

Assuming the ontology engineers have instantiated the ontology, the next activity is *verify consistency*. This can be easily performed using a reasoner that understands the ontology language. But more specific verifications can be made. As an example we defined *ASIL propagation* in section 2.3.2 which is a useful analysis concerning functional safety as we will see further in the discussion. Other relevant verifications are left to the engineer's insight.

2.4.2.3 System Level Requirements Definition

In figure 2.52, we represented requirements definition in general focusing on functional and non functional requirements. Naturally, requirements structure defined in section 2.2.1.2 is applicable. In the remainder, we focus on the consideration of functional safety in the design process as illustrated in figure 2.47 so non functional requirements definition is not brought into the discussion.

Preliminary Hazard Analysis. In order for safety aspects to be included into the functional "genes" of design, *preliminary hazard analysis* is performed as soon as missions have been transformed into functional requirements. We have three recommendations for this activity. First, the application of a failure model as presented in section 2.2.2.1 to make the analysis more systematic. Second, PHA being completely reusable as is if done at vehicle level, we cannot stress enough the reuse of PHA's lines from previous similar projects. Finally, PHA output being safety goals, they should be written by applying the specific boilerplate as presented in previous section 2.4.2.2.

Safety goals come with an assigned ASIL determined by following table 1.4 specified in ISO 26262. ASIL determination is formalized in the ontology (see section 2.2.2.2) there-

fore, once PHA data are instantiated in the ontology, ontology engineers can verify both that ASIL determination and safety goal's ASIL assignment have been done correctly. In addition, they propagate the ASIL from hazardous events to their original functional requirements.

Functional Requirements Definition. Integrating systems engineering and functional safety helped identify an important synergy between the two kinds of engineers. Now that safety aspects are incorporated into stakeholders requirements, the system can actually be developed towards acceptable risk following a top-down approach. Even though it should be system engineers role to define the system functional requirements, they can benefit greatly from functional safety engineers expertise to develop functional requirements that intuitively correspond to additional functions that exist only to ensure safety. The goal is to reduce design iterations by making systems and functional safety engineers work together in a more collaborative manner rather than strictly in parallel. Systems engineer position does not yet exist at Renault and systems engineering activities are actually partially performed by different actors depending on the project. For instance, the project manager can be in charge of writing requirements while the electric and electronic architect (a position at Renault) is responsible of functional and physical architecture production. Actually, many design iterations exist for system architecture definition and related requirements. System architecture are first produced without considering safety, and then *secured* architectures are developed to take safety into account. Corresponding requirements are naturally also defined iteratively. Safety engineers activities are performed in parallel and consist in, Preliminary Hazard Analysis, definition of safety requirements, ASIL assignment and other safety analyses. Usual communication between system engineers and safety engineers are represented in figure 2.52 with dashed arrows between the two roles. Using a common ontology can indirectly play the role of collaboration as consistency between design documents and models can be ensured. Two scenarios are conceivable. First, project actors assigned the role of system engineers and functional safety engineers (they are distinct for safety critical systems according to ISO 26262) can also play the role of ontology engineers. They add the information they are responsible for to the ontology, verify its consistency and use

it as a knowledge base. Second, ontology engineering can be viewed as a new professional field in the company and dedicated actors are assigned to the role of ontology engineers. They receive information from systems and safety engineers, instantiate the ontology with this information, verify its consistency, use it as a database and communicate points of concerns when inconsistencies are detected. Using an ontology ensures that systems and safety engineers are working on the same system in a more synchronized way. This can lead to the front loading of safe architectures definition with their requirements and to design iterations reduction as systems and safety engineers are working on information verified to be consistent.

Coming back to functional requirements definition, we recommend the use of the following boilerplate at system level:

The <system> shall <function>

where *system* is the system in development and *function* is a system function displaying the functional nature of the requirement.

As presented in section 2.2.2.2, the main point of involving safety engineers in functional requirements definition is to perform ASIL decomposition on functional requirements. Safety engineers have a better understanding when and how a function (a functionality, not necessarily a system function) captured by the functional requirement can be decomposed into functional requirements (which correspond to a safety mechanism, redundancy for instance). This decomposition (or tailoring) is done a priori, independently of any architecture, and further safety studies will be performed later in the process to verify that tolerable risk is achieved (in function of the ASIL). Performing ASIL decomposition is an asset of ISO 26262 as it follows systems engineering principle to think about the problem before thinking about the solution. To impose oneself with independence constraints early in the design process can result in elegant and more performing solutions in terms of architectures.

For ontology engineers, on reception of functional requirements, they instantiate the ontology and verify ontology consistency. In particular, they have to make sure that semantic relations usage (decomposition, derive *etc.*) upon natural discourse makes sense.

They can verify the structure of the functional requirements and perform ASIL propagation from top level functional requirements to low level ones (see section 2.3.2.2).

2.4.2.4 Functional and Physical Architecture Definition

In *define functional architecture* and *define physical architecture* activities, functional requirements are materialized into functions that are allocated to the components responsible for their execution. As explained in the previous section, we propose more synchronization for systems and safety engineers through an ontology. Systems engineer are responsible for system architectures definition. Functional safety engineers are responsible for these architectures to be reasonably safe, *i.e.*, they have to prove that both the architectures development and the architectures respect evaluated risk for the system. Let us note that risk evaluation was historically performed on system architectures devoid of safety-related functions and safety mechanisms in order to obtain a correct appraisal of system's risk. Even though the habit is still followed by safety practitioners, the approach proposed in this work front loads risk evaluation on functional requirements that correspond to system missions. It seems even more fitting for the analysis pertinence as it is more probable that safety-related functions and safety mechanisms are not considered. As can be seen in figure 2.52, system architectures are defined by systems engineer that take into account safety aspects provided by safety engineers in the form of functional and technical safety requirements.

The hardships concerning requirements management and traceability, identified by Albinet et al. [2010] for instance, is still relevant. Making sure that project actors justify their work in relation to system requirements gives more confidence that the project does not deviate. Stemming from project tools heterogeneity, the capability to manage and trace requirements from a textual document to different modeling objects in different tools is a necessity. Tools such as Telelogic DOORS[8] or Geensoft Reqtify[9] are used for requirements management. Naturally, for manufacturer specific needs, they have to be customized. The ontology is a highly formalized attempt to express how traceability should be established and understood.

[8] `http://www-01.ibm.com/software/awdtools/doors/productline/`
[9] `http://www.3ds.com/products/catia/portfolio/geensoft/geensoft-product-lines/reqtify/`

Safety engineers role is to assign an ASIL on functions and components from the safety goals. Precautions must be taken if a function or component failure can lead to the violation of a safety goal. They are defined in ISO 26262 with respect to assigned ASIL. The ontology formalizes how ASILs are assigned (see sections 2.2.2.2 and 2.3.2) which remained ambiguous previously. In general, the ontology is an incredible asset for understanding a domain. If creating an ontology requires for the formalized conceptualization to be shared, new people (both internal or external to any company) that have to familiarize with the domain will be able to do so with a more precise artifact than natural language (*i.e.*, oral communication or documents) or informal models.

Concerning ontology engineers, they perform ASIL propagation from the functional requirements to the architectures exploiting the structure of instances as defined in section 2.3.2. As can be seen in figure 2.52, they also receive functions ASIL assigned by safety engineers. While we consider that functional safety engineers at Renault perform ASIL assignment correctly (*i.e.*, they do share our conceptualization), using suppliers is also a possibility. In addition, be it engineers from Renault or suppliers, both are likely to make human errors as ASIL assignment remains a manual activity. So, after performing ASIL propagation and verifying the ontology consistency, ontology engineers point out to safety engineers that one or more assignments may be erroneous if an inconsistency is found. For instance, instantiating a function with an ASIL D as done by the safety engineer, an inconsistency will be found if a different ASIL is defined for the function by ASIL propagation.

2.4.2.5 Conclusion

In order to improve the design process of safety critical systems at Renault to answer ISO 26262 requirements, we first went through appropriating and capturing the domains of systems and functional safety engineering. This enabled to identify favorable areas for improvement. In this section we gave recommendations and discussed our take on a design process adapted to the context at Renault. The process presented in this section is more systematic, improves requirements writing, reduces design iterations by means of tight collaboration between systems and safety engineers, promotes and facilitates infor-

mation reuse and complies with ISO 26262. A new role has also been introduced to answer the heterogeneous nature of a project. *Ontology engineers* guarantee the design process consistency, mostly when some activities are ordered from suppliers. In tight interaction with systems and safety engineers, they ensure that the enterprise needs are effectively considered.

2.4.3 Conclusion

The general design process presented in this section addresses different issues that stem from the transition to model-based systems engineering at Renault and international standard ISO 26262. Defining a process is equivalent to defining activities, supporting techniques and supporting tools (see figure 1.1). The challenge identified at Renault, that is most probably also present in other companies, is to evaluate the adequacy of the actual state of systems engineering practice to the company needs in anticipation of a future specific seamless process (with adequate tools). Ontology engineers, responsible to define ontologies that formalize needs and mappings between the ontology and different formalisms is our answer to this challenge. Based on ontology consistency, any delta with the conceptualization is detected by ontology engineers and brought to the surface. Ultimately, independently from any specific languages and tools, the approach presented in this section ensures semantic data integration.

2.5 Chapter Conclusion

In this chapter, we presented our contributions. The observation that semantic consistency was lacking among modeled objects in implementations of Renault design process led us to formalize in a very precise way systems engineering domain. While system engineers and ontology engineers professions do not yet exist at Renault, the company has started its transition towards model-based systems engineering and formalization helped to clarify ambiguous understandings of the domain. Safety critical systems were central considerations at the origin of this work with the arrival of standard ISO 26262 on functional safety. It has now been published in November 2011. To consider ISO 26262, we also decided to formalize functional safety as to define Renault interpretation of the standard.

Finally, conceptual information manipulated in the design process of safety critical automotive mechatronics systems have been formalized by integrating systems engineering and functional safety. The formalization was done by means of formal ontologies. The result is the object of section 2.2 were we present partially two ontologies realized for systems engineering and functional safety, while section 2.3 presents the integration of these two domain ontology that is used as an ontology for Renault systems and safety engineers when they develop a safety critical system. Based on the integrated ontology, section 2.4 returns on the design process semantic consistency problem and shows that ontologies formal semantics actually addresses and solves the problem of semantic integration. We present an approach where the systems engineering and functional safety ontology is central to the design process. The ontology (with the project instances) is considered as the consistency reference model of a project to ensure semantic consistency. This approach is consistent and in accordance with systems engineering and ISO 26262. Then, we presented a process that implements this approach by introducing a new role of ontology engineer. We believe it has become essential as the development process is a really complex activity, not yet fully understood, that impacts directly on the competitive advantage of an enterprise, and that increases in complexity with systems becoming larger and new rules, standards and regulations emerging continuously. An ontology engineer may guarantee the consistency of heterogeneous languages / formalisms used during the design process by bringing these different languages within the ontology formalism.

The approach presented in this chapter aims to enhance the design process while leaving it unaltered. Although very promising, it is relatively difficult to completely validate the approach. First, the formalization quality needs to be estimated. The real challenge for the domains formalization to be relevant is not trivial. The defined conceptualizations have to be both *general* and *precise* enough. On the general aspect, the conceptualizations would be adequate if they can be used to describe the diversity of developed systems. On the precise aspect, the conceptualizations should enable to record necessary and sufficient level of detail. These actually correspond to expert knowledge and the quality of the conceptualizations can only be validated against real projects. Second, while the approach strives to leave the design process unaltered, we have been confronted to ambiguities and incom-

patibilities between the different conceptualizations (of systems engineering and functional safety). These are resolved when the two conceptualizations are integrated. The next chapter presents a case study where the approach is used on an already existing system. It demonstrates that the information produced during the design process can effectively commit to the proposed formalization and the applicability of the approach. Nonetheless, it remains that the design process used for the case study was executed without compliance to the ontology resulting mostly in missing information. Therefore, the next chapter does not validate the approach but contributes to validation by giving some elements of answer concerning applicability. The validation would require more executions of the approach on new systems while complying to the (integrated) ontology.

Chapter 3

Case Study: the Regenerative Combi-Brake System

Sommaire

3.1 Introduction

In this chapter we present the study case that inspired the work presented in this thesis. This work is an answer to concerns at Renault due to changes required by recent international standard ISO 26262 on functional safety. In the automotive industry, braking and steering were the last vehicle functionalities to remain purely mechanical despite the advent of mechatronics. This comes from strong regulations as these functionalities can lead to the most serious incident: an accident. Our study case is the Regenerative Combi-Brake (RCB) project which is a braking system relevant for this context. Section 3.2 first goes over previous electric braking projects. Then the RCB that takes advantage from these preceding projects is presented. In section 3.3, the RCB is used to illustrate the application of our design approach: the ontology centric design process. Finally, section 3.4 concludes the chapter.

3.2 Presentation of the Case Study

3.2.1 Antecedent Projects History

In this section we cover two antecedent braking projects. It presents the main functionalities and components of electric braking, some safety concepts and what has been learned during these projects.

3.2.1.1 FREL

The first Renault electric braking project started in 2002. The FREL project (*FReinage ELectrique* that stands for electric braking) consisted in a braking system intended for the medium car range. It was the first project to apply standard IEC 61508 on functional safety (see section 1.2.2.3). The project had the following characteristics:

- Four Electro-Mechanical Brake (EMB) actuators, one for each wheel;
- No backup braking actuators that implement traditional hydraulic or mechanic systems;

– No mechanical link between the driver's commands (*i.e.*, brake pedal or parking brake interfaces) and the EMB actuators;

– A Pedal Feel Emulator (PFE) that implements force feedback on the brake pedal.

The FREL system is a *by-wire* braking system. By-wire means that some traditional mechanical and hydraulic control parts have been replaced with electronic control systems. Driver's commands are not linked mechanically or hydraulically with braking actuators. It has the unwanted result of making the braking pedal *soft* (*i.e.*, absence of force feedback) disabling driver's capability for different level of braking. Thus the existence of the PFE. As a braking system, FREL has to ensure an optimum vehicle deceleration. FREL is also responsible for immobilization and release of the vehicle depending on the context of the vehicle. Those high level functions had to be implemented while preserving and interacting with other vehicle control functions (*e.g.*, ABS, ASR, ESP, *etc.* See section 3.2.2.2). The project focused on the high-level control software developed for the EMB control system following a model-based design approach. Figure 3.1 presents a simplified system architecture of the project.

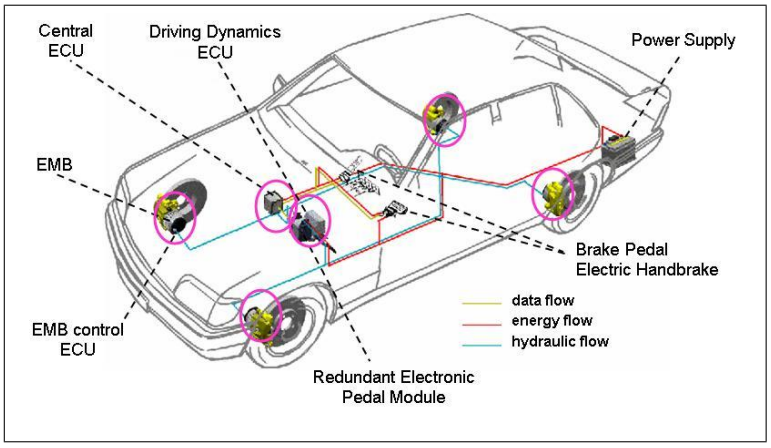

Figure 3.1: FREL simplified architecture

We identify the key components required to perform the high level functions deceleration, immobilization and release:

– Four EMBs that are used both for deceleration and immobilization of the vehicle;

– The driver's command interfaces for the driver to interact with the system;

– The calculators, *i.e.*, Electronic Control Units (ECU), that are used to interpret driver's will / intention and to compute braking commands;

– And the power supply that is the energy source of the ECU and the four EMB.

Particular attention has been given to dependability and safety as new behavior and risks were introduced. For instance, in figure 3.1, there are two hydraulic networks that correspond respectively to the two couples of wheels in diagonal. Those two diagonals are independent and a failure on one diagonal is safe as braking on a diagonal has been evaluated to be safe.

The project demonstrated technical feasibility of *safe* electric braking as shown with the secured architecture in figure 3.2.

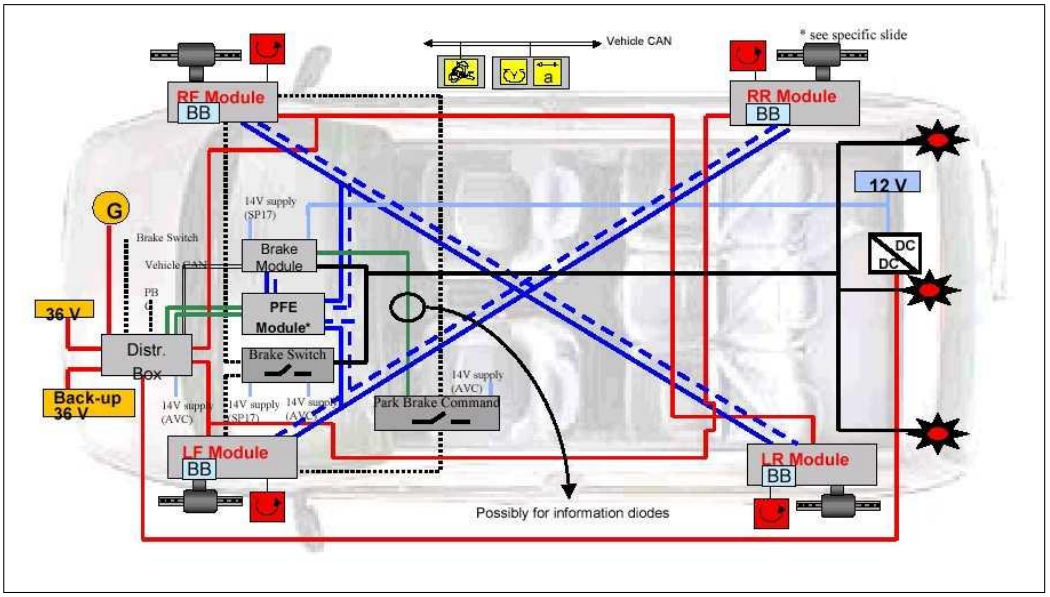

Figure 3.2: FREL architecture

We identify:

– Four EMB in their respective wheels modules Right Front, Left Front, Rear Right and Left Right (RF, LF, RR and LR);

– the brake pedal in the PFE module and the hand brake;

– the ECU in the brake module;

- the power supply (36V) that has been moved to the front;
- And the diagonal architecture that makes possible to brake independently on the four wheels.

The hydraulic network has been completely gotten rid off and the link between braking interfaces and the braking system is by-wire. As a result, the brake pedal is hidden inside a PFE module and a data flow network (the blue straight and dashed lines) replaces the hydraulic network with the same diagonal architecture. The whole system power source is redundant with a backup power supply (Backup 36V) for the four EMB and another battery (12V) that backs up the brake module.

Safety design choices are implicit as shown in figure 3.2. Important aftermath in subsequents projects lies in the choice of a braking command. Different braking functions may run simultaneously and ask for different braking services. Commands priority has been defined to always select the most adequate one.

In addition to those architectural choices, FREL has been developed to exhibit the following properties:

- The system shall ensure a minimum deceleration of the vehicle in case of a failure that affects the deceleration on four wheels functionality
- The system shall not display undesired braking action (*i.e.*, braking action without a demand from the driver)
- The system shall ensure that a driver's (braking) request implies a braking action
- The system shall be fail silent (*i.e.*, no error propagation)
- The system shall ensure that immobilizing failures are limited to less than 5%

We won't go into any further details on the FREL project. The project ran until the end of 2003 and concluded on the extra costs of the technology to be too expensive for relatively small braking service and safety gains.

3.2.1.2 X by Wire.

The preceding by-wire project FREL concluded on the extra costs of the technology (*i.e.*, by-wire technology) not to be affordable. Mainly, costs came from the necessity to secure the architecture thus the idea to level off the costs by gathering all the by-wire systems and searching for services at the inter-system level, idea that materialized in the X by Wire project.

The project "*Commandes Découplées*", that is French for X by Wire (XbW), the X stands for the multiple functions, has been initiated in mid-2004. The objective was to prepare the company to the introduction of electric commands (*i.e.*, by-wire) for braking and steering functions inside Renault lines/fleet. Those two functions are the last functions at vehicle level that have kept full mechanic or hydraulic parts as they are intimately related to safety. Introduction of by-wire technology seemed inevitable at the time therefore the two objectives of the project were:

- To find customer services that one customer will find acceptable compared to the cost of the vehicle;
- To construct the development process adapted to such safety critical systems.

Over the course of the development, the project has gone through different phases. The project has focused on steering and braking functions, first as a whole, and then separately. The respective systems denominations have been used: Steer and Brake by Wire (SBbW) which was the first project that intended to apply systems engineering, Steer by Wire (SbW) and Brake by Wire (BbW). SbW and BbW systems make for all necessary and sufficient functions of steering and braking. However, their interconnection enables the introduction of new services (such as regenerative braking, lane keeping, automatic cruise control, *etc.*) by means of software blocs and, if necessary, by adding sensors, thus SBbW system that takes SbW and BbW considerations at a more abstract level. Implementation of new functionalities result in an increase in system complexity (*i.e.*, introduction of hardware components). From the beginning, XbW system disposes of all hardware components that enable most of the services that will be depicted in section 3.2.2.

Those services therefore present the opportunity to be dematerialized into software. We can already assume that current trend in Commercial Off The Shelf (COTS) development will make a cultural resistance to the development of specific (software) solutions even though COTS development implies more integration work and dependency on a supplier.

Figure 3.3 depicts the hardware components of the BbW system.

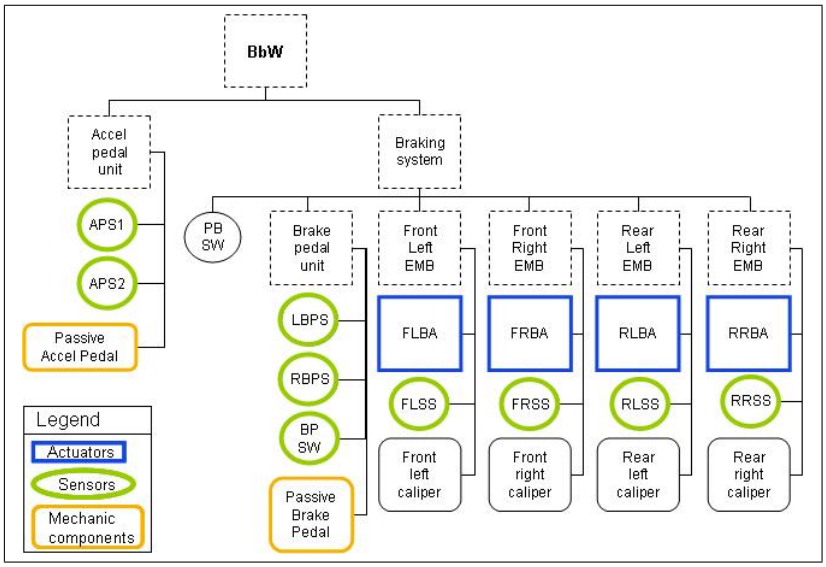

Figure 3.3: BbW components

As just stated, hardware should be more or less the same for any BbW system. In the braking system branch of the figure, we recognize the four EMB, Front Left (FL), Front Right (FR), Rear Left (RL) and Rear Right (RR). The EMB is a mechatronics system. It is fundamental for a by-wire solution. It is composed of an ECU (that do not appear in the figure), a Speed Sensor (SS) and a Brake Actuator (BA) mounted on a brake caliper. Interfaces with the driver are the usual brake pedal and a button for the parking brake that is represented by the Parking Brake SWitch (PBSW) in the figure. The brake pedal position is measured with two, Left and Right, Brake Pedal Sensors (LBPS and RBPS). The Brake Pedal SWitch (BPSW) monitors if the brake pedal is pressed or not and can be used as a safety control measure to detect failure of the LBPS and/or the RBPS. The XbW project first objective was to find (new) customer services. Innovation was possible with the tight interaction of different systems which was established with either CAN or FlexRay

communication systems. As such, the BbW project perimeter integrated the Accelerator Pedal Unit (Accel Pedal Unit). This unit is composed of the driver's accelerator pedal interface and two sensors, Accelerator Pedal Sensor 1 and 2 (APS1 and APS2).

The inclusion of the accelerator enables to offer new services for advanced braking strategies. For instance, BbW can detect an emergency reaction of the driver and activate the emergency braking service. The scenario is the following: the driver goes quickly from the accelerator to the brake pedal in response to an emergency. First, the system detects a quick release of the accelerator and engage the brakes by itself. Then second, the driver wants to brake and starts to push the brake pedal. The system detects a rapid change from the accelerator to the brake pedal which corresponds to an emergency, it activates the emergency braking service which applies the maximum braking force until Anti Block System regulation. Other strategies have been developed that benefit from the system external environment. In the previous example, we had to introduce new sensors on the brake pedal. The system can also make use of existing components making the notion of *new* virtual service.

The project stopped in January 2008. The reasons that were mentioned pointed out interests divergence between different departments at the company level. For instance, profitability studies did not show that the company would break even and product marketing considered that the economical balance with regards to the services would be understood only by specialists and therefore was not viable. Moreover, the project met internal negativism from the engine department which considered the project fruitless as it did not bring radical change in the architecture, the product remaining more or less the same, whereas the improvement was for the development method to be adapted to safety critical systems. More generally, downstream directions did not involve themselves sufficiently for the project to be transfered into Renault vehicles fleet.

Nevertheless, the project was highly proficient. The objectives were met. On new services or innovations, by March 2008, the project had generated a total of 140 patents in France that were the source for more innovations (24 other patents referenced patents from the project). Two prototypes implementing BbW were developed and helped in the

realization of braking command laws adapted to independent braking on the four wheels. On the construction of a development process, the project enabled all the Renault actors to tightly work together in a cohesive manner, thus identifying new roles that were inexistent (for instance system engineer) and what activities each actor was responsible for. The project concluded that the development of safety critical systems requires the automobile constructor to position itself as a system developer rather than an integrator of commercial solutions (*e.g.*, COTS) that are not completely mastered and that the company is liable for. Important results on the development of safety critical systems therefore came from systems engineering and international standard ISO 26262 that are respectively treated in sections 1.2.1 and 1.2.2.3.

3.2.2 A Regenerative Combi-Brake System

3.2.2.1 Towards an Electric Vehicle

By the end of 2008, the automotive industry has been strongly affected by the crisis that occurred starting from the financial world. The situation led the direction board of Renault to take radical decisions in order to better position the company for a world in mutation. We can mention two decisions that are important for this section. First, the concentration of the engineering effort over a full electric vehicle (*i.e.*, equipped with an electric engine). That is the Electric Vehicle (EV) project. Second, the affirmation of a new identity with the company signature "*Drive the change*".

The EV project started in 2008. EV is a vehicle project. Under systems engineering vocabulary, the system (under development) is the whole vehicle. It is a first on many points. It is the first vehicle with an electric engine developed internally to be included in Renault line/fleet. It is the first project onto which systems engineering method has been applied at the vehicle level. It is the first vehicle project onto which standard ISO 26262 (see section 1.2.2.3) has been applied.

EV project comes from the desire of the company to drive the change in bringing an affordable electric vehicle on the market (*i.e.*, with a selling price equivalent to the one of a fuel motorized vehicle of the same category). As the project calls for originality (by developing an electric engine), new risks are also introduced. The automobile world

is currently doing extensive work on risks management for these risks to be acceptable. More generally, the development process has to be improved in order to be suitable to the specificities of automotive safety critical systems. The company signature is a tremendous asset for the improvement of the development process as it pulls all the actors in the same direction which was not necessarily the case before as demonstrated in the XbW project.

3.2.2.2 Regenerative Combi-Brake

The EV project has been decomposed into many sub-projects. The high level safety critical functions of a vehicle lie in steering and braking. We are interested in the braking subsystem of the vehicle. Multiple braking projects that implement different functionalities have started in the context of the EV project. We focus on the Regenerative Combi-Brake project (RCB).

Functionalities. At the vehicle level of abstraction, we have the braking systems usual functions presented in figure 3.4. Namely, the functions are: to decelerate the vehicle, to contribute to the stability of the vehicle, to stop the vehicle and to hold the vehicle stopped. The last function of the figure is particular to RCB. The RCB project intends to implement a regenerative braking function into the braking system. The general idea is for the braking system to use the resistance of the electric engine against the kinetic energy of the vehicle (making the electric engine operate backward) in order to recharge the battery of the vehicle. In other words, the new function, that is put at system level, is to retrieve as much energy as possible during vehicle deceleration phases. The intentions

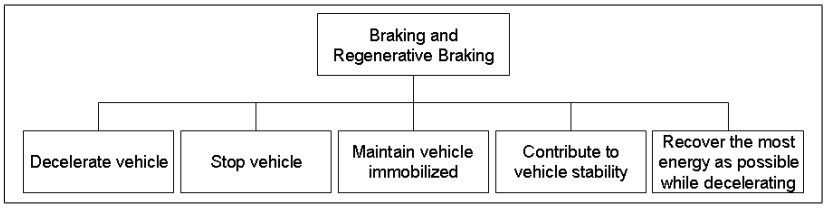

Figure 3.4: RCB functional schema

behind the addition of a regenerative characteristic into the braking system are, first, an amelioration of the vehicle autonomy which is one of the principal feature that a customer looks at in the purchase of an electric vehicle, second, an amelioration of energy retrieval

possibilities while using the technique, and third, a cost reduction compared to actual commercial solutions in regenerative braking.

System perimeter. There are two characteristics that are introduced in the RCB project. As the name of the project implies, a combi-brake system will be developed. The usual meaning is that the system can be used in two or more different ways. Here, the system will be used as a traditional braking system using the brake actuators or the system will pilot the engine in regenerative mode. Combi can also be interpreted as combination which is the other characteristic of the RCB system as it was decided to use a combination of regenerative actuators, EMB actuators, and traditional friction actuators with their hydraulic network because it is a well controlled technology. Figure 3.5 depicts the RCB system perimeter and its environment.

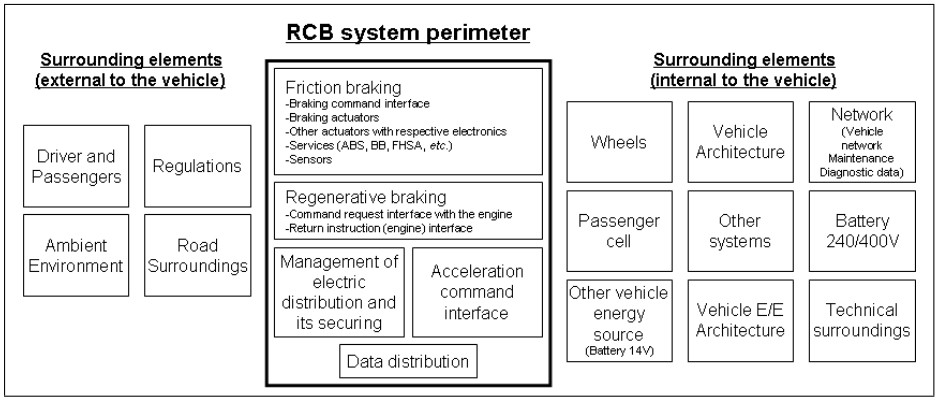

Figure 3.5: RCB system perimeter

The following elements are part of the RCB system (see figure 3.5): under *Friction Braking* we identify, the driver's command interfaces (*i.e.*, brake pedal and parking brake), the braking actuators (*e.g.*, disks, drums, pads, *etc.*), the other actuators with their respective electronics (*e.g.*, external signals), the sensors (*e.g.*, speed sensor) and the different braking services (*e.g.*, ABS, ASR, ESP, *etc.* See below). Under *Regenerative Braking* we have the interfaces between the RCB and the electric engine. The RCB can send command requests to the electric engine and can receive return instructions from the engine. Advanced regenerative strategies will depend upon the accelerator pedal therefore an interface with the system is inside the perimeter. *Data Distribution* (and acquisition) is connected

to the vehicle network (*e.g.*, CAN). Finally, *Management of Electric Distribution and its Regulation* have to be developed following regulations (*e.g.*, redundant electric network and backup power supply).

From a functional and safety point of view, the surrounding elements of importance are, regulations that constrain the system, parameters that influence system configuration (*e.g.*, Road Surroundings such as icy or wet), vehicle communication and energy distribution networks that need to be connected, and all the remaining elements interfaced with the vehicle (*e.g.*, tools used in after-sale service, dashboard and external signals, *etc.*).

Services. We have seen that services are of the utmost importance. As a braking system, the RCB functions are to decelerate the vehicle, to contribute to the stability of the vehicle, to stop the vehicle and to hold the vehicle stopped. Functionality although necessary is not sufficient from the customer point of view. Indeed, those high level functions would be meaningless without their features or properties (*e.g.*, driving pleasure). Table 3.1 lists most of the services that intervene in the braking function of a Renault vehicle.

Acronym	Meaning	Description
ABS	Anti-lock Braking System	The ABS function regulates the braking forces of each actuator in order to stop the wheel (or wheels) from locking and to restore vehicle maneuverability. The function monitors constantly the individual wheel speeds and determines a reference vehicle speed, from which wheel locking can be detected.
ACC	Adaptative Cruise Control	The ACC function controls the vehicle to be at defined speed. Its action is twofold. First, it elaborates commands for the engine; either to accelerate the vehicle in order to attain defined speed or to reduce the engine torque instruction when the driver is pushing the accelerator too far. Second, when following another vehicle, the system automatically uses the braking system to decelerate in order to maintain a safe distance with the other vehicle. The system is deactivated by pressing the brake pedal.
ASR	Anti-Slip Regulation	This function elaborates braking and deceleration commands for the braking system and the engine control in order to avoid wheel slipping during an acceleration. The objective is to optimize torque transfer to the wheels on roads with particularly degraded grip (slip).
		continued on next page

Table 3.1: Braking services

continued from last page

Acronym	Meaning	Description
AYC	Active Yaw Control	The AYC function splits and transfers the engine torque to the wheels that have the best adherence.
BB	Basic Brake	The BB function is in charge of interpreting the drivers will to brake the vehicle through his actions on the Human Machine Interfaces of the system (*i.e.*, the brake pedal and the parking brake command). It elaborates a global deceleration command and includes a braking distribution function that outputs braking force commands to the four braking actuators. It is also in charge of the parking brake mechanism.
BFD	Brake Force Distribution	The BFD function takes into account many parameters (*e.g.*, speed, global braking command, weight, *etc.*) to compute separate braking commands for each wheels. More or less braking pressure is applied to each wheel in order to maximize stopping power while maintaining steering.
CLD	Closed Loop Deceleration	This function modulates the global deceleration coming from the BB function in order to give a *constant* response in terms of deceleration to the driver's command. More precisely it elaborates a robust braking command that is insensitive to variations of the mass (weight) of the vehicle, the braking efficiency of the brake pads, or the slope of the road.
EBA	Emergency Braking Assistance	The EBA detects an emergency braking situation through the speed and the force of the driver's braking demand, and it boosts the braking force to its maximum level.
ESP	Electronic Stability Program	This function detects the discrepancies between the driver's will and the actual vehicle trajectory, then corrects such discrepancies by acting on the brakes and/or on the engine management system. The idea is to bring the vehicle back to a *normal* trajectory and to avoid instabilities due to inadequate actions from the driver.
FHSA	Full Hill-Start Assistance	This function fulfills two main objectives: keeping the vehicle still after it reaches a full stop, and assisting the driver during a takeoff or a maneuver by keeping the vehicle from sliding in the wrong way or by limiting the slope-induced acceleration while ramping. The FHSA function can also manage the parking brake mechanism in order to relieve the electrical motors of the EMB actuators to preserve the level of charge of the battery.
MSR	Motor Skid Regulation	The MSR function regulates wheels skidding by acting on the engine.

continued on next page

Table 3.1: Braking services

continued from last page

Acronym	Meaning	Description
SL	Soft Landing	The SL function upgrades the braking comfort when the vehicle is about to stop by diminishing the amplitude or the speed of the vehicle pitch. To do this the function modulates the braking command of each actuator optimizing the compromise between the driver's demand and the increase of the braking distance.

Table 3.1: Braking services

The services in table 3.1 will not all be integrated into the RCB project. They are nonetheless presented as all of them can actually be dematerialized into software elements. It will be possible to integrate each software element, while taking the necessary precautions, with no additional costs in terms of hardware. The project will integrate the ABS and the ESP (the latter supervises vehicle trajectory using a combination of ASR, MSR and AYC) as they are required by the regulation authorities. BB and BFD are solutions to regulatory requirements (*e.g.*, presence of a service brake, an emergency brake and a residual brake) and are therefore mandatory. The EBA will also be implemented ulteriorly. As it is a crash preventing system it will put additional confidence in the vehicle.

The PFE from the FREL project (section 3.2.1.1) will not be integrated into the RCB project. We mentioned earlier that the design choice to use traditional friction actuators with their hydraulic network had already been made. Usual pressure force (from the hydraulic network) can therefore be applied on the brake pedal and a PFE is irrelevant.

Last but not least, the regenerative braking which is the novelty of the system but has yet to have an acronym for referencing.

Architecture. The previous paragraph is a representative list of the services that are related to braking. Two deliverables in the Renault development process are the *functional architecture* that structures the functionalities of a system and the *physical architecture* that structures the physical components of a system. They are the respective objects of figures 3.6 and 3.7.

From a functional point of view, a coarse grained abstraction of mechatronics systems can be represented under a data flow diagram. In figure 3.6, the *boxes* are the functions that produce and consume the flows which are represented with the usual *arrows*. The *circles* represent the sensors and the brake actuators. Compared to the high level functions of figure 3.4, the functional architecture structures finer grained functions that processes the flows.

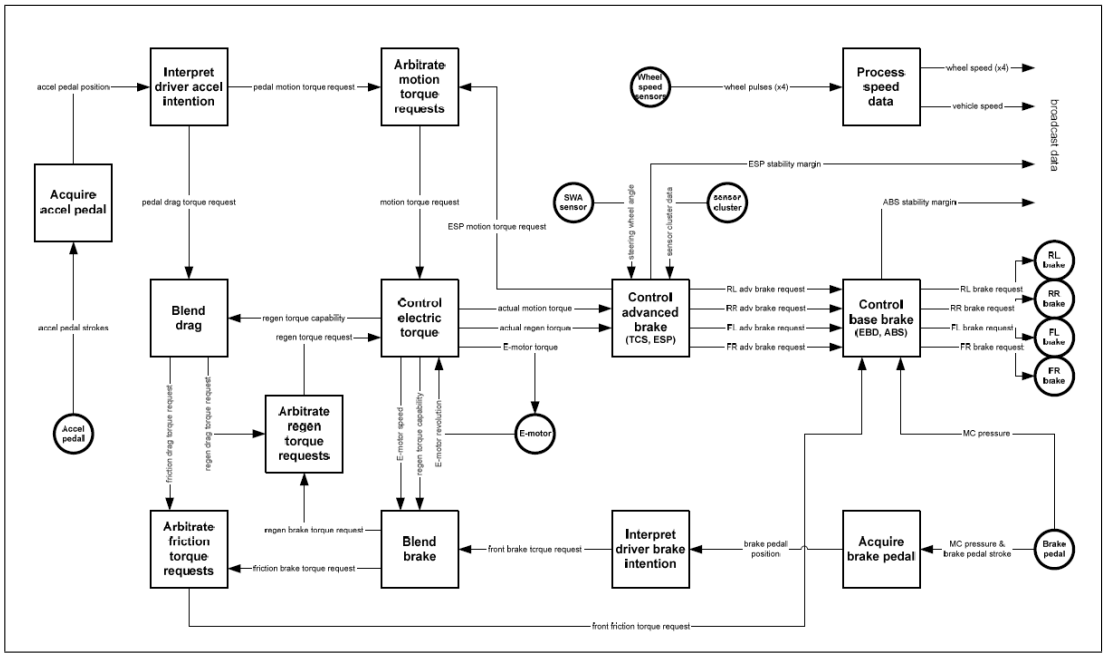

Figure 3.6: RCB functional architecture specification

We describe the functions following the fundamental chains from the sensors, through the flows processing, to the actuators.

First, we have the chain from the brake pedal to the brake actuators. The brake pedal sensor on the bottom right hand side of the figure observes the brake pedal stroke and the pressure exerted on the brake pedal. Those flows are transformed by the acquire brake pedal box into a brake pedal position. The main functionality to be designed when a human is in the loop, is to interpret his intention. The interpret driver brake intention box consumes the brake pedal stroke and the pressure on the brake pedal to produce a brake pedal position flow. The latter is interpreted as the driver brake intention and a request for front (wheels) brake torque is produced. The blend brake function consumes the re-

generative torque capability, the electric motor speed and the precedent front brake torque request. The function produces two requests for the friction brake and the regenerative brake respectively. Those two flows are then arbitrated to produce a front friction torque request and a regenerative brake torque request respectively. The request on regenerative torque is consumed by the off-system controller that is responsible for the electric motor torque, controller that commands the electric motor brake actuator. Meanwhile, the front friction torque request goes to the base brake controller which is responsible for services such as the BFD and the ABS. Finally, the controller assign brake requests on the four wheels brakes.

Second, we have the chain from the accelerator pedal to the brake actuators. The accelerator pedal sensor is on the left side of the figure. From there we have the same functions than before: acquire the accelerator pedal that produces the accelerator pedal position and interpret driver acceleration intention that produces two requests for the electric motor, one for (positive) motion the other for drag (*i.e.*, negative motion). On the one side, the request for (positive) motion is off-system and shown only because of tight interconnection with the ESP that can request another motion torque. The arbitrate motion torque requests function produces a motion torque request to the controller of electric torque which commands the electric motor. On the other side, the pedal drag torque request is consumed with the regenerative torque capability flow by the blend drag function that produces two torque requests for the friction brake and the regenerative brake respectively. The rest of the chain is similar to the first one: Those two flows are arbitrated to produce a front friction torque request and a regenerative brake torque request respectively. The request on regenerative torque is consumed by the electric motor controller that commands the electric motor brake actuator. Meanwhile, the front friction torque request goes to the base brake controller that assigns brake requests on the four wheels brakes.

It remains two functions on the figure: control advanced brake and process speed data. The latter digitalizes the vehicle (global) speed and the four wheel speeds. On the control advanced brake, it can be interesting to introduce the notion of a *virtual driver* as the function emulates the reactions of a driver. The function consumes many parameters that

are represented under the sensor cluster. It produces individual advanced brake requests for the four wheels. Those flows are consumed by the base brake controller that assigns brake requests on the four wheels brakes.

The parking brake fundamental chain is not represented in the functional architecture as it is relatively equivalent to the brake pedal one. The rear friction torque request is not represented either as one important architectural choice has been decided early in the project: to use traditional drum brakes with their hydraulic network on the rear wheels in combination to a BbW system with two EMB on the front wheels.

Figure 3.7 displays this choice of architecture. It is a high level physical architecture but sufficient for a presentation.

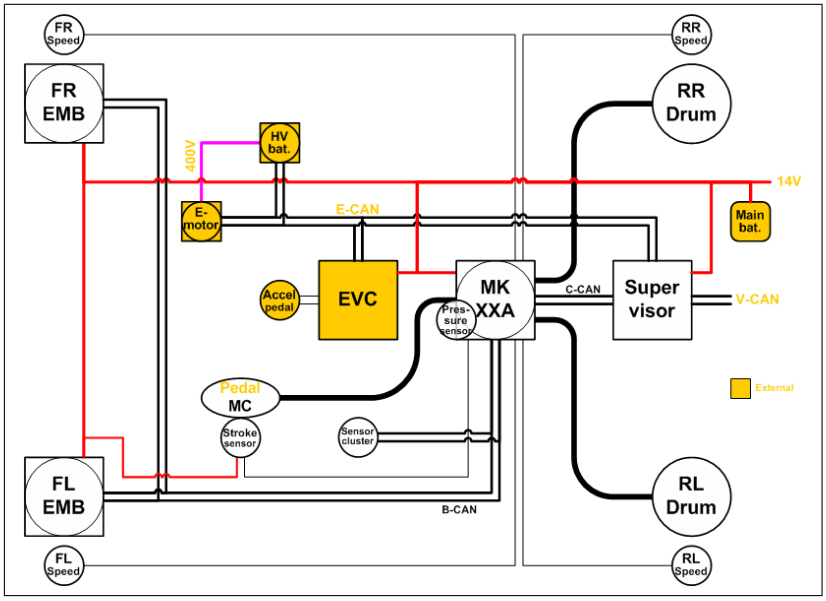

Figure 3.7: Example of the RCB physical architecture

The yellow elements are outside the system scope. The brake actuators are, the electric motor for regenerative braking, the two EMB on the front wheel-axle unit, and the two drum brakes on the rear wheel-axle unit. We find most of the sensors that are present in the functional architecture: four speed sensors (one for each wheel), the accelerator, a stroke sensor on the brake pedal, and a pressure sensor for the brake pedal. The controllers complete the usual triplet used to classify the E/E components of mechatronics systems. In the figure we have three controllers represented by the big squares. The Electric Ve-

hicle Controller (EVC) serves mainly to control the electric engine and is off-system but important for interconnection purpose. The MKXXA is responsible for the usual braking functions of a braking system: to decelerate the vehicle, to contribute to the vehicle stability, to stop the vehicle and to hold the vehicle stopped. Even more, it is responsible for providing all the services presented before. As for the supervisor, it is responsible to the last function of the RCB, to recover the most energy as possible while decelerating.

An E/E system is dependent on a power source (which is outside the scope of the RCB system). There are two batteries that for technological reason are necessary (*e.g.*, cell management requires a redundant cell configuration). The main battery powers the calculators and the two EMB. It is represented with red lines in the figure. As for the High Voltage (HV) battery, it is tightly interconnected to the system therefore the part of interest is the communication network. The communication network is represented with straight black lines. The only elements that appear not related in any way to the communication network are the drum brakes on the rear end of the vehicle and the main battery. Except for those, all the elements are interconnected with CAN links. Finally, the big black lines represent the hydraulic network that links the rear drum brakes and the brake pedal.

One last thing about typical components of E/E systems, the EMB is a COTS. Incidentally, it is not dimensioned specifically for the RCB project and houses in a controller and a speed sensor which is not used at the time but will probably be identified as a fundamental part of the system later in the development. More generally, automobile constructors heavily relies on parts suppliers. In the RCB project, Continental has been selected to develop the system. Renault actors were in charge to deliver *requirements* to Continental's actors. Continental is then in charge of the development and the realization of the product. It was also decided that Continental was in charge of the safety studies. Once the product will be delivered, Renault will have to verify and validate that the product conforms to the given requirements and make an integration effort in order to use the product in a vehicle. The fundamental importance of the requirements is transparent as they have to convey the constructor intentions for the supplier to realize a good product and they are used for the product verification, validation and integration. Furthermore, the supplier often delivers a product for which only black box knowledge is known about, resulting in difficulties to es-

timate the relevance of the solution. Therefore, requirements need to be verifiable, precise and complete. This is especially the case when safety is concerned. Consider the case of a faulty product from a supplier that leads to an accident; it is not the supplier that will be accountable for the accident but the constructor.

This physical architecture must implement all the functions from the functional architecture. Thereby it is one among the many possible physical architectures. Obviously, the physical architecture in figure 3.7 is not a haphazard choice. One can argue that the combi-brake solution is more complex than a solution that does not mix technologies but here are some justifications that make a good compromise on complexity, safety and costs.

First, the FREL project mapped the braking service that was provided by the use of any combination of the four wheels. There is less degrees of freedom in the RCB project as one can brake using a combination of the three braking subsystems: the front wheel-axle unit, the rear wheel-axle unit and the regenerative brake. The system is thus easier to analyze even when it comes to safety. In fact, three types of braking systems are mandatory: a service brake used in normal driving conditions, an emergency brake (or secondary brake) for use in case of failure of the service brake, and a residual brake that is operational even in the case of a power failure. The decomposition is obvious, the service brake is ensured by all the braking subsystems, the emergency brake is ensured by the traditional rear wheel-axle unit, and the residual brake is also part of the rear wheel-axle unit which is usual.

Second, the choice to use drum brakes on the rear wheel-axle unit greatly simplifies the system analysis as it is a well controlled technology. For instance, it enables to get rid of the PFE from antecedent projects by taking advantage of the hydraulic network of the drum brake technology. It suffices to increase the distance between the drums and the brake pads in such a manner that light braking will not make the drum and the pads contact resulting in braking on the regenerative brake or/and the front wheel-axle unit only. The force feedback on the brake pedal is ensured by the retracting springs of the drum pads. This solution contributes to recover the most energy as possible by promoting regenerative braking over the other brakings. Moreover, the combi-brake solution answers the regulation

requirement to have an *emergency brake* independent with the *service brake* by treating the front and rear wheel-axle units separately. This architectural separation facilitates the demonstration that the two brakes are independent. Even more, drum brakes are not subject to the permanent energy leakage of the disk brake technology that is due to a light brush of the pads on the disk for self cleansing. Finally, drum brakes present the advantage (even though they offer less precise braking than disk brakes) to be auto-amplifying. It is a desirable property as the drum brakes will be used mostly in hard braking situations where the maximum braking effort must be reached as fast as possible.

The result is an elegant solution that is highly compartmentalized enabling to analyze the system by components of smaller size.

3.2.3 Conclusion

As we said in section 3.2.2.2, braking and steering functions are safety critical. The RCB project that develops a regenerative braking system illustrates the current state of affair inside Renault. As such, this section should be taken as an example and be generalized to Renault's mechatronics systems. This section helped to characterize three points. One, Renault's role when developing critical systems as a developer of systems rather than an integrator of commercial solutions. Two, the particular relation between a constructor and a supplier. Three, the increasing system's complexity due to mechatronics considerations of multiple domains and the resulting introduction of new risks. Defining a development process for mechatronics systems that takes into account Renault specificities and that is conform to ISO 26262 was the objective of this work. The next section presents the application of our approach on the RCB system.

3.3 Application of the Approach

In this section we present the application of the approach from section 2.4.2 on the RCB project. Actually, the approach based on an ontology was realized after the project. Therefore documents and models do not always comply with the ontology. Nonetheless, we had to anticipate for the formalized conceptualization to be general enough to consider

the RCB, in particular, but also other systems. As a result, we present applicable concepts of the ontology and examples when information is absent. In addition, the documents presented in the previous section were produced at Renault and served as specifications for third parties to evaluate systems engineering tools and to produce a prototype. Information was therefore produced by different actors and media that were also not always available.

ArKItect tool is evaluated on the RCB project and other ones. It is our main source of information. ArKItect[1] is a modeling tool that features a meta-model editor for customization. More and more customizations are added to the tool to support systems engineering and safety activities.

Then we have the ontology editor Protégé 3.4.4. It enables to edit OWL ontologies that are XML structured documents. Software series 3.4 and 4 are two main development branches and we ended up using the 3.4 one as it has closer use to engineers modeling tools. The OWL ontology for systems and safety engineers is instantiated with project information.

This section presents the ontology engineer role and the different ontology uses. We first present Protégé in section 3.3.1. Then we follow the design process from need analysis and stakeholders requirements definition in section 3.3.2 through section 3.3.3 system requirements definition, section 3.3.4 functional architecture definition and physical architecture definition in section 3.3.5. Finally, section 3.3.6 concludes on the experimentation.

3.3.1 Protégé Presentation

The ontology has actually been developed to formalize the conceptualizations behind available documents, models and ArKItect meta-models. We went through different ontology versions that are not always compatible with the one presented in this thesis. The ontology in this work is the less ambiguous one as we removed or made precise non satisfactory concepts. In particular, need analysis and, requirements management and traceability are activities that have many shortcomings so we made great efforts towards future improvement in these activities. Using an adequate ontology editor can greatly improve ontology management. An OWL ontology is structured using XML and axioms that need

[1] `http://www.k-inside.com/web/`

not to be ordered in an *owl* file. Protégé interface enables to manage an ontology with different *tabs* for respective views of classes, properties, individuals (instances) and SWRL rules. Figure 3.8 and 3.9 are two different views about OWL classes tab. As can be seen in both figures, the subclass explorer on the left hand side displays classes hierarchy. In the bottom right hand side, we have the option to display a class either with *logic view* or properties view in the class editor window. In figure 3.8, the logic view displays the axioms and constraints of a class. The properties view is actually really useful as property constraints for a class are displayed for each property as illustrated in figure 3.9. Defining and browsing through classes seems more natural in this view which is one of may reasons for our choice between Protégé 3.4 and 4.

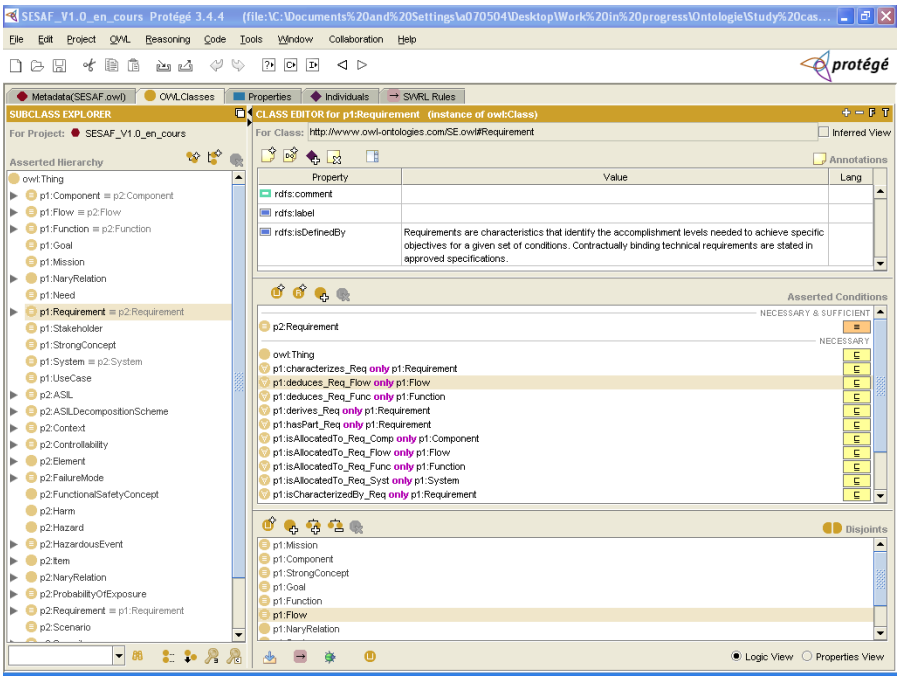

Figure 3.8: Protégé class tab – logic view

The properties tab is the object of figure 3.10. On the left hand side we find the property browser that displays all the relations between classes. On the right hand side, additional axioms can be defined in the property editor, namely, relations domain, range and property (*e.g.*, a relation is defined symmetric if and only if for any x in relation with y, we have also y in relation with x).

The two tabs with the SWRL tab in figure 3.11 enables to define an OWL ontology

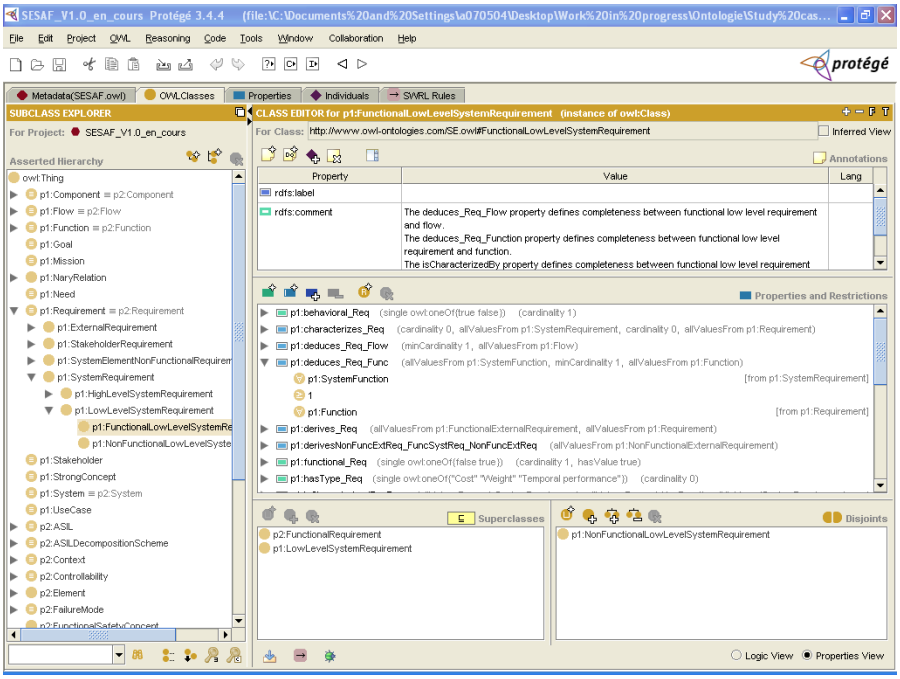

Figure 3.9: Protégé class tab – properties view

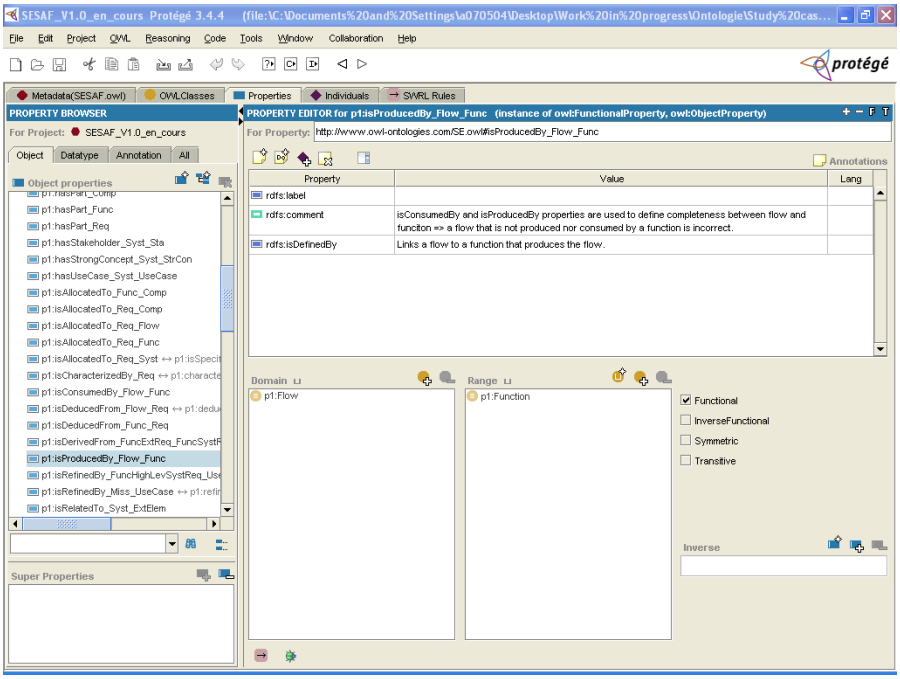

Figure 3.10: Protégé properties tab

with additional SWRL rules. This tab enables to write SWRL rules and SQWRL queries. The buttons on the top right hand side labeled *SQ* and *J* enable respectively to interpret

and execute SQWRL queries and SWRL rules. We will return on these functionalities in the following.

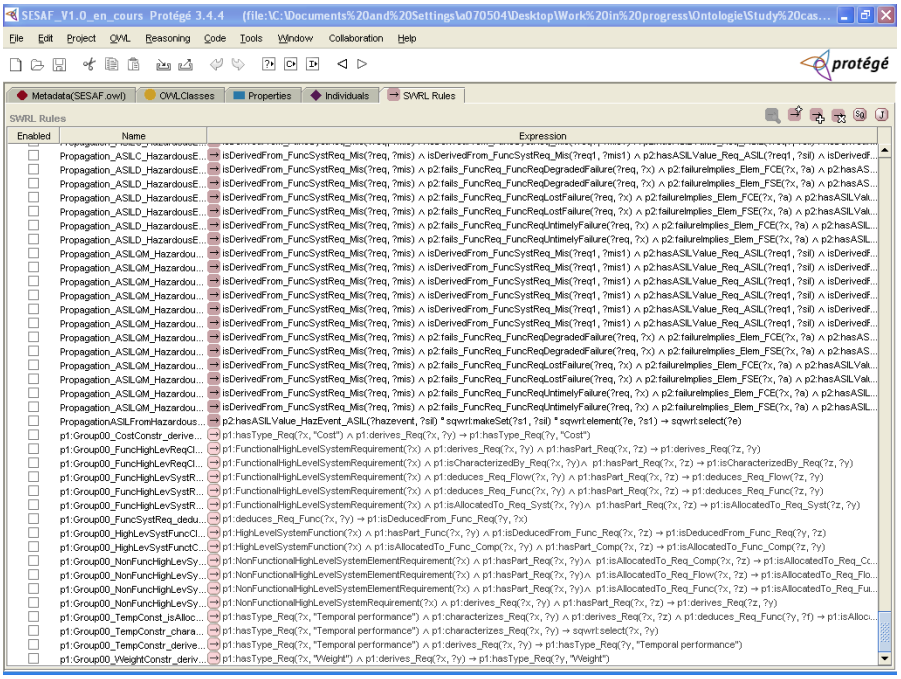

Figure 3.11: Protégé SWRL tab

Finally, the individuals tab enables to define class instances, as shown in figure 3.12. The class browser on the left hand side enables to select the active class to which an individual can be defined in the instance browser in the middle of the figure. Then the individual editor on the right hand side displays the relations that can be instantiated for an individual. Let us note that the relations framed in red denote non compliant relation with defined constraints, here cardinality constraints. This is provided by the tool and is a helpful indication that information is missing. Yet, as OWL is interpreted under Open World Assumption (OWA), consistency checking will still conclude on the ontology consistency as it is understood that these properties should be instantiated which is simply not the case at the time.

The ontology (in terms of classes and properties) is fixed so we do not need to revisit OWL classes and properties tabs. The application of the approach is done by defining class instances and their relations, and by executing SWRL rules and SQWRL queries in Protégé individuals and SWRL tabs.

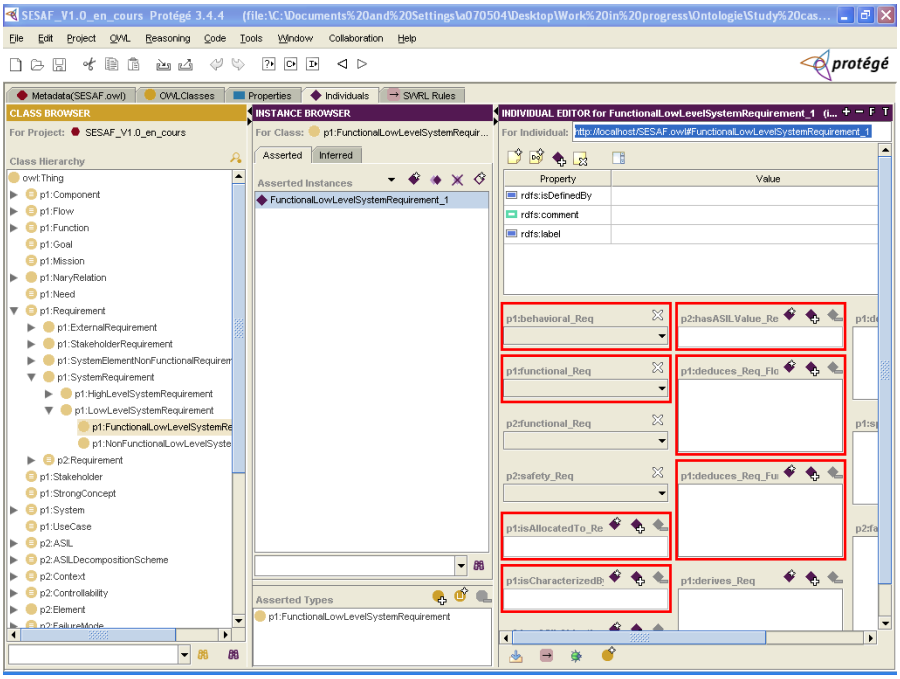

Figure 3.12: Protégé individuals tab

3.3.2 Need Analysis and Stakeholders Requirements Definition

Need analysis and stakeholders requirements definition were not realized entirely during the project. We had two STR documents available which are actually both incomplete and ambiguous. For instance, in one document, stakeholders requirements have not been defined and in the other they are named external requirements. The documents were realized in succession which displays iterations that reflect, first, a manual production, and then, a generated document based on an ArKItect model.

The concepts manipulated in these activities have been represented in figure 2.2. In general, ontology engineers will have to match information with the ontology concepts, *i.e.*, to define mappings between information and the ontology. This can potentially be straightforward, difficult or impossible. As the ontology was produced from these documents, obvious mappings can be defined which are then applied manually in this experimentation but can be automated once the process will be fixed (in terms of languages and tools).

As a starting point, we define a system class instance with *RCB* name in the individuals tab. Then we follow the structure of the STR document.

First, System's missions are presented. To remain simple, we present only two RCB missions. An OWL individual is identified by a Uniform Resource Identifier (URI) which we use as identifiers. We have *MIS_ RCB_ 1: Enable vehicle's deceleration* and *MIS_ RCB_ 2: Enable energy regeneration during deceleration phases.*

Then we have a list of the system goals understood as measures of effectiveness at Renault. For instance: *OBJ_ RCB_ 1: The RCB system shall improve the vehicle's autonomy through energy regeneration during deceleration phases (around 30%).*

We then have a description of the system boundaries where the system is represented as a black box in interaction with its environment. These architecture (functional and physical) enable to pose some *strong* constraints on the system in terms of solution. They are defined as system's strong concepts. For instance: *CON_ RCB_ 1: Electro-Mechanical Brake actuators (EMB) on the front wheels* and *Con_ RCB_ 4: Drum brake actuators on the rear wheels.*

Then we have a list of the stakeholders, for instance, *system architect*, *safety* and *customer*. This list is finally followed by the stakeholders requirements which have to be accepted as the expression of the system missions, goals and strong concepts.

These information correspond to the system so all these individuals are put in relation with the system individual using the different *has* properties with $System$ as domain (see figure 2.2). Depending on the number of instances, it can be advantageous to automate these properties instantiation with SWRL rules. For instance, executing the following SWRL rule associates the system to all the goals: $p1{:}System(?x) \wedge p1{:}Goal(?y) \rightarrow p1{:}hasGoal\_Syst\_Goal(?x, ?y)$. The result in Protégé is presented in figure 3.13 where the RCB individual is displayed.

As one can observe with the system's goal given in example, it is relatively easy to confuse the goals, strong concepts and missions as requirements. In fact, there are no traceability relations between the goals, strong concepts and missions with the stakeholders requirements. This is a problem for the system validation which is the activity that ensures that the system meets the stakeholders needs. In general, the stakeholders requirements express the stakeholders needs which are understood at Renault in terms of

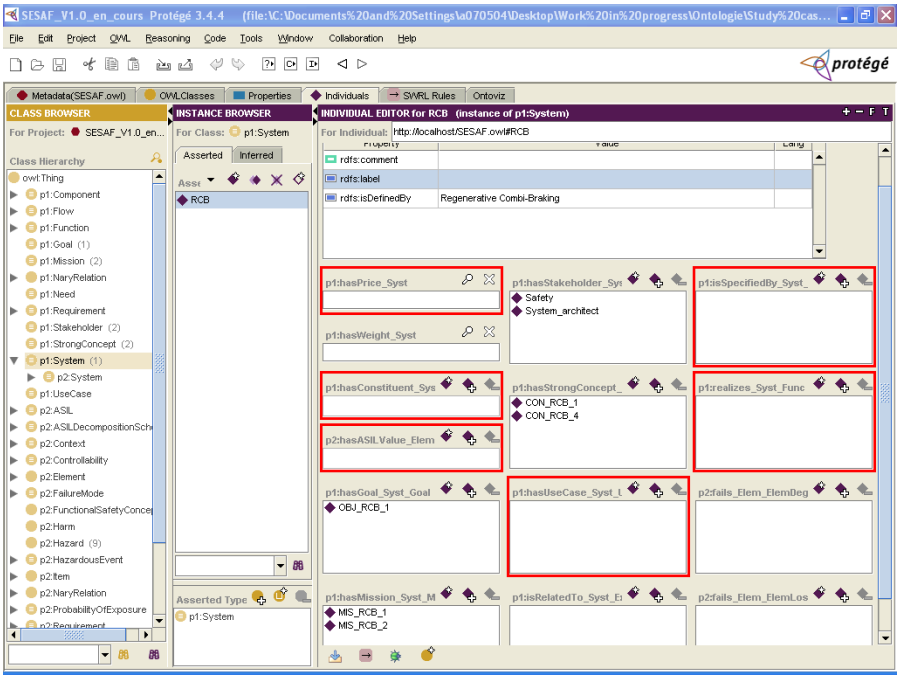

Figure 3.13: RCB individual in Protégé

missions, goals and strong concepts and shall therefore be traced to corresponding stakeholders requirements. As an example, we identified that no stakeholders requirements was defined for *CON_RCB_1*. The stakeholder need was nonetheless considered by a system requirement which was questionable as it was not traced to a stakeholder requirement. We have added these missing traceability relations in the ontology for additional coverage analysis and the general *Need* concept for consideration.

Figure 3.14 corresponds to the stakeholders requirements view in ArKItect.

Rectangles with rounded corners represent stakeholders and the other rectangles contained in the previous ones are stakeholders requirements of a specific stakeholder. This corresponds to the boilerplate structure for stakeholders requirements: *The <stakeholder type> (shall or may) be able to <capability>*. For instance: *REQ_EXR_0001: The customer shall be able to decelerate the vehicle.* Intuitively, this stakeholder requirement actually comes from *MIS_RCB_1: Enable vehicle's deceleration.* In protégé, we can define this provenance by asserting the type (*i.e.*, the class) of MIS_RCB_1 to *p1:Need* (*i.e.*, MIS_RCB_1 is an instance of *p1:Need*) which enables to use *p1:derives_Need_StaReq*

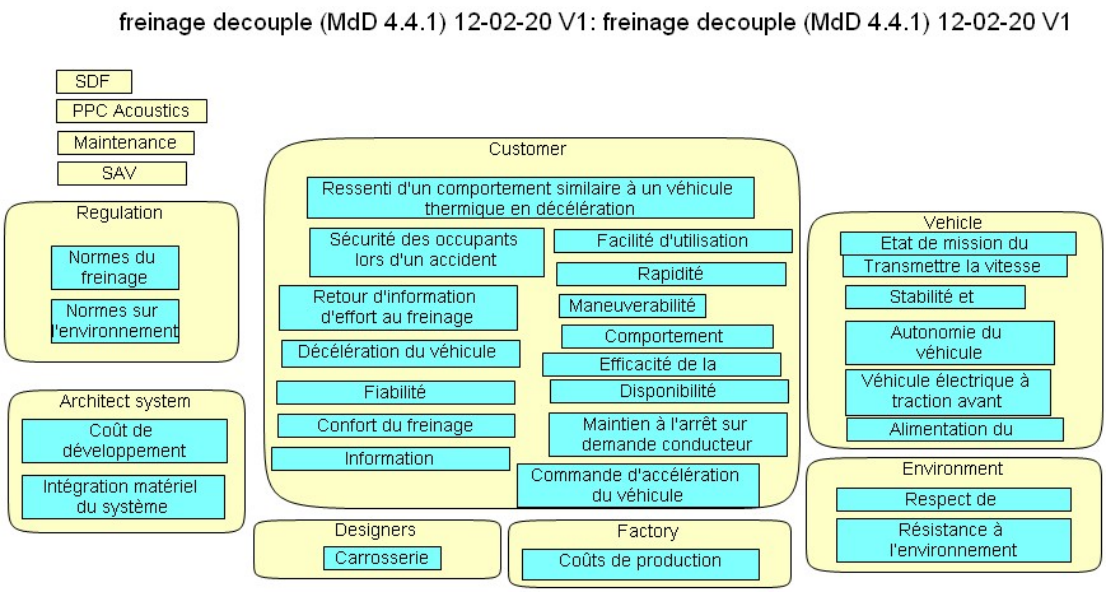

Figure 3.14: Stakeholders requirements view in ArKItect

for traceability. As it can be seen in figure 3.15, we added $p1{:}Need$ in MIS_RCB_1 asserted types which results into two new tabs in the individual editor. These tabs correspond to the different understandings of MIS_RCB_1 as a mission or a need.

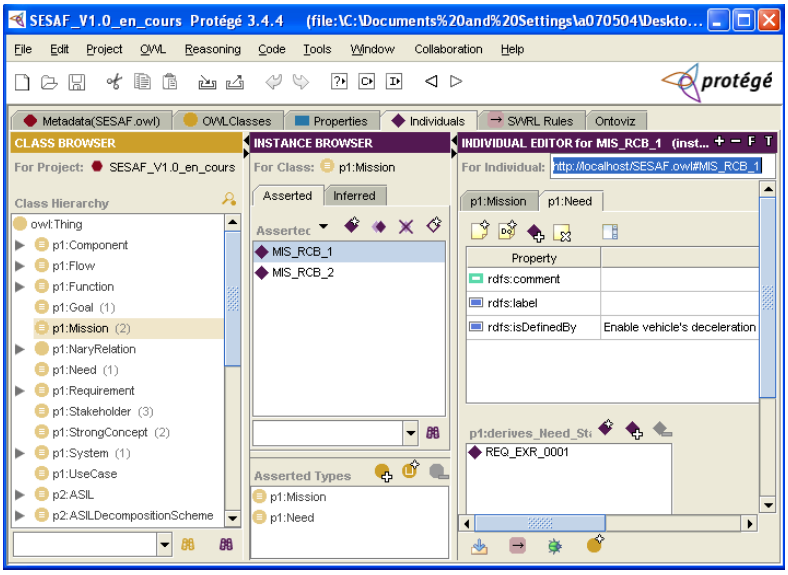

Figure 3.15: Individuals asserted types in Protégé

The stakeholders requirements are defined with a hierarchical structure in ArKItect that is recorded with *high/low level* classes and *hasPart* property. For instance, we have

REQ_EXR_0003: The customer shall be able to decelerate the vehicle using emergency brake and *REQ_EXR_0004: The customer shall be able to decelerate the vehicle using a footbrake* that are sub-requirements of REQ_EXR_0001.

Although we used cardinality constraints to define integrity constraints in the ontology, it is not possible to verify integrity constraints under OWA. Consistency checking concludes on the ontology consistency as the incomplete nature of information is assumed with OWL. To verify these integrity constraints, it is possible to use query languages supported by Protégé such as SQWRL (with set difference) or SPARQL. For instance, executing $p1:Need(?x) \wedge p1:derives\_Need\_StaReq(?x,?y) \wedge p1:Need(?z)^{\circ}sqwrl:makeSet(?s1,?x) \wedge sqwrl{:}makeSet(?s2,?z)^{\circ}sqwrl{:}difference(?s3,?s2,?s1) \wedge sqwrl{:}element(?e,?s3) \rightarrow sqwrl{:}select(?e)$ returns the list of needs that are not related to a stakeholder requirement. This query is composed of two parts, the body and the head, that respectively precede and succeed the middle arrow. Body parts are separated using °. The first part uses pattern matching which matches all the instances of the class Need to ?z and all the instances of the class Need that are related to a stakeholder requirement to ?x. The second part enables to use set construction operators ($sqwrl{:}makeSet$) so that all the individuals that matched in the first part of the body are recorded into a set, set ?s1 for ?x and set ?s2 for ?z. The third part enables to use set operation operators. $sqwrl{:}difference$ enables to perform set difference so ?s3 is the set of elements that are in ?s2 less those in ?s1. The query effectively returns the elements in ?s3 which are the needs not related to a stakeholder requirement. Defining a query for each type of cardinality constraints is straightforward and has been automated in open source reasoner Pellet[2] with Integrity Constraints Validator tool (ICV[3]) that produces the adequate SPARQL queries. In addition, we actually expect traceability management to be done in specific tools such as DOORS, Reqtify or others so we did not define all the queries for checking integrity constraints in the ontology.

3.3.3 System Requirements Definition

System requirements are produced to consider the stakeholders requirements. In this section, we only present the functional system requirements but we bring the non functional

[2] `http://clarkparsia.com/pellet`
[3] http://clarkparsia.com/pellet/icv/

ones in the discussion when it is interesting. For this activity, system and functional safety engineers roles overlap and the actors have to tightly collaborate in order to produce the whole system requirements set. In our approach, functional system requirements are written using a general boilerplate: *The <system> shall <function>*.

3.3.3.1 Preliminary Hazard Analysis

Following the design process timeline, system requirements are first produced by system engineers from the stakeholders requirements. PHA starts with a subset of system requirements as input. Our approach actually defines this subset as the functional system requirements that are traced to system missions. This enables to start safety activities as early as possible in the process as other system requirements need not be defined to start the PHA. Naturally, such traceability is one particular aspect of our approach. For instance, for *MIS_RCB_1: Enable vehicle's deceleration* we have the corresponding system requirement *REQ_RCB_1: The RCB system shall decelerate the vehicle.* These early system requirements are communicated to functional safety engineers as PHA input

PHA begins with hazardous events identification. Each functional system requirement input is analyzed by applying the different failure modes of the failure model in figure 2.28 and different operational context to identify hazardous events. For instance, REQ_RCB_1 is analyzed with failure mode *lost* and operational context *C: driving mode.* This led to the identification of multiple customer level hazardous events or Feared Customer Events (FCE) such as *EIC_FREL_02: absence of brake release* and *EIC_FREL_09: total absence of braking* (these were already identified in the preceding braking project FREL). All these informations are recorded in the ontology with appropriate concepts and relations as illustrated in figure 3.16.

The analysis continues with risk evaluation. Each FCE is evaluated with different scenarios that define an operational situation (*i.e.*, an operational context with an aggravating circumstance), the feared event consequence given the operational situation, and the possible ways to avoid or limit the consequence given to the system user. The operational situations, consequences and possibility of avoidance are respectively evaluated in terms of exposure (E), Severity (S) and controllability (C). Two scenarios are given as example in

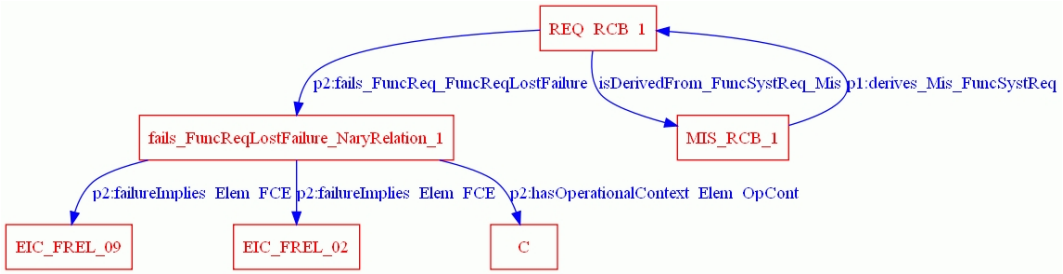

Figure 3.16: Hazardous events relative to deceleration – Protégé individuals view

table 3.2.

Scenario	FCE	Operational Situation	Consequence	Possibility of avoidance	E	S	C
02_1	EIC_-FREL_02	C, highway leftmost line	Death	Warning lights, pull back in hard shoulder	E1	S3	C2
09_1	EIC_-FREL_09	C, bend	Death	None	E4	S3	C3

Table 3.2: PHA example

E,S and C levels are now defined in ISO 26262 (see table 1.4) and were used in the example. However, prior to this standardization, other levels were defined by each project. Recording these informations (if different orders are used) in the ontology can therefore be subject to interpretation if a bijection is not possible. Another concern of ontology engineers is to consider inter-project knowledge. PHA information is reusable as is, yet, PHA (as other process activities) can be outsourced to suppliers and sharing confidential information becomes a problem which often results in starting again a PHA. Risk evaluation in the PHA is a manual activity done by an analyst. This evaluation can have different results when done by different analysts. Nonetheless, inter project consistency is desirable and can be supported with the ontology. For instance, a similar scenario of another PHA can be asserted to be the same as a scenario in our example. Any inconsistency such as different exposure levels will be detected automatically. Such usages of the ontology need to be considered by ontology engineers who have access to additional information (from different domains and projects).

PHA output are safety goals. Safety goals are top level safety requirements that are the negation of the FCEs with an assigned ASIL at Renault. Table 1.4 defines the corresponding ASIL to triples (E,S,C) for ASIL determination to assign an ASIL to the scenarios. Then safety engineers perform ASIL assignment on the safety goals which assigns each FCE with the highest ASIL of its scenarios. All these informations are recorded in the ontology. Additionally, we defined table 1.4 as SWRL rules in the ontology. This enables to verify that ASIL determination has been performed complying with ISO 26262. Executing the rules will assign an ASIL to the scenarios and detecting an inconsistency reveals an error. For example, let's assume that the ASIL of EIC_FREL_02 was defined to D. Figure 3.17 illustrates ASIL determination via rules execution in Protégé. Resulting inferred axioms are displayed at the bottom of the figure. The conclusions are that EIC_FREL_02 is not safety related (it is the responsibility of quality management) whereas EIC_FREL_09 is ASIL D. Adding these axioms to the ontology results in an inconsistency as illustrated in figure 3.18 (note that red is not good). Simply put, EIC_FREL_02 is associated to quality management and ASIL D whereas we defined that a hazardous event is related to exactly one ASIL.

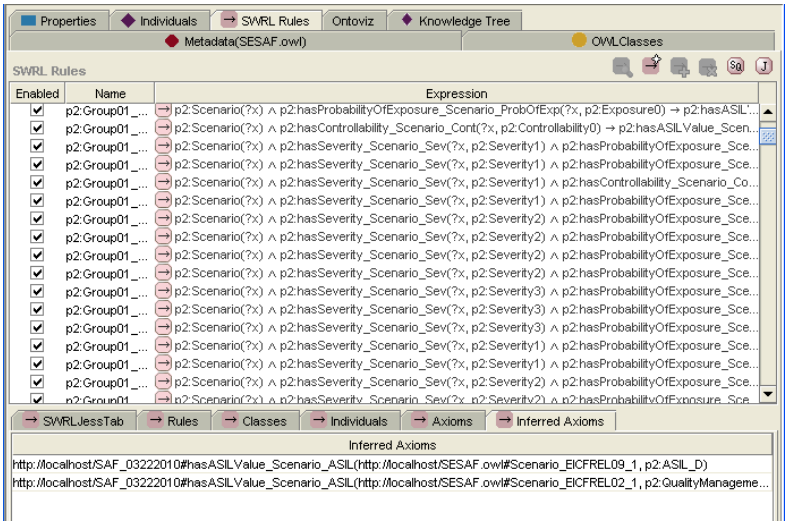

Figure 3.17: SWRL rules execution in Protégé

For ASIL assignment, we had to deal with OWL Open World Assumption. Assigning the highest ASIL to a FCE requires negation (or closure) as one would want to assign ASIL D to a FCE if it is related to any scenario with ASIL D, ASIL C if it is *not* related to a

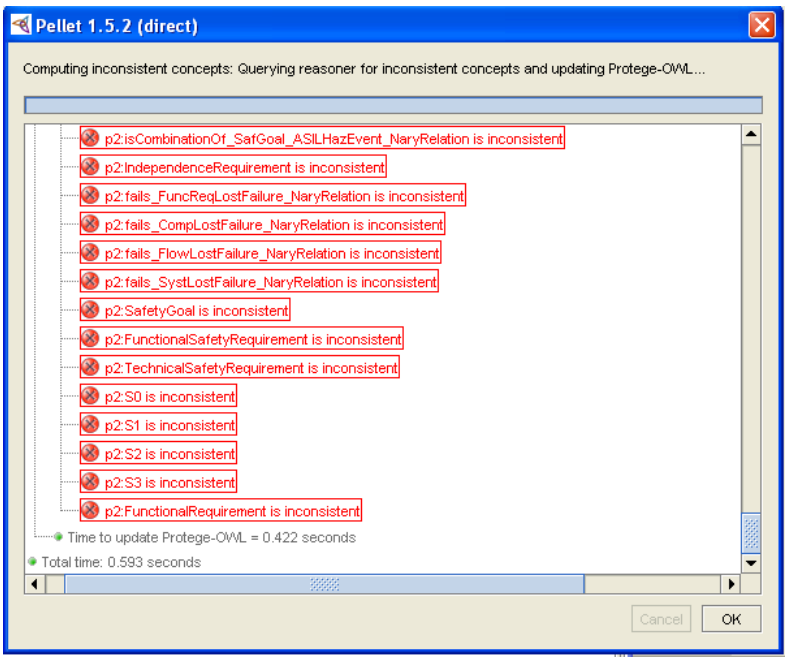

Figure 3.18: Inconsistent ontology in Protégé

scenario with ASIL D and related to a scenario with ASIL C, and so on. But it is impossible to express negation with OWL and SWRL. In this thesis we presented the ontology with first order logic and such properties as axioms (see axioms (2.25) and (2.53) for examples of negation and closure). However these properties cannot be written in OWL so it is natural to use (SQWRL) queries to return these results. However, in Protégé, it is not possible to add these results to the ontology. Fortunately, it is possible to deviate from SWRL and use SQWRL queries as rules, which inferred axioms can be added to the ontology. This must be done consciously and is only semi automatic as rule selection is important. For example, we actually have two rules to assign an ASIL C to a FCE. The first one has to be executed when no scenario has an ASIL D. It simply assigns ASIL C to FCEs that are related to ASIL C scenario(s). The second one accounts for FCEs that are assigned with ASIL D and are however related to ASIL C scenario(s). This rule only assigns ASIL C to FCE that are related to ASIL C scenario(s) and that are *not* related to ASIL D scenarios. Executing the first rule in this context would result in assigning two different ASIL to the FCE in relation with an ASIL D. Selecting the appropriate rule to execute remains manual. Ultimately, ASIL assignment is partly supported by the actual ontology and Protégé. The

ontology enables to detect errors done during this activity. Complete support of these axioms as they are defined in this thesis is possible with a dedicated application in parallel to the OWL ontology which exceeded the scope of this work.

Finally, safety goals are stakeholders requirements from the system engineers viewpoint and their consideration is similar with relative system requirements. From the safety point of view, a safety goal is the negation of a FCE with an ASIL. For example for EIC_FREL_09 we have the following safety goal: *No total absence of braking, ASIL D.* In the ontology, the two viewpoints are merged such that the more precise conceptualization from systems engineering predominates. Safety goals are therefore expressed using the stakeholder requirement boilerplate: *The safety engineer shall be able to verify that there is no total absence of braking, ASIL D.* This stakeholder requirement is naturally put in relation with the functional system requirement that accounts for vehicle deceleration (REQ_RCB_1: The RCB system shall decelerate the vehicle). Finally, the PHA defines safety goals from the functional system requirements in relation with a mission. As explained in section 2.3.2.2 ontology engineers start ASIL propagation to assign the highest ASIL of the FCEs of a functional system requirements to each requirement. This enables to also assign ASIL QM (for quality management) as safety goals are defined for ASIL more stringent than QM.

3.3.3.2 Functional System Requirements Definition

As it can be seen in figure 2.52, system engineers are responsible for this activity. Nonetheless, safety engineers are also responsible to produce functional system requirements that are relative to safety. Safety engineers further develop the safety goals (that are stakeholders requirements from systems engineering viewpoint) into the so called functional safety requirements as unacceptable risk has been detected. These two activities are performed in parallel but functional system requirements and functional safety requirements concepts overlap therefore we made them precise in the ontology by stating that a functional safety requirement is a type of functional system requirement (a subclass). Under this interpretation, functional safety requirements are expressed with the boilerplate for functional system requirements that involves a function. In fact, the term functional

safety requirements accounts for functional and non functional requirements as illustrated in figure 2.52 where the functional safety requirements are sent over to system engineers as input for functional and non functional system requirements definition. We lose this definition in the ontology so that ontology engineers cannot simply record the functional safety requirements defined by safety engineers as instances of functional system requirements. Only functional ones (*i.e.*, related directly or indirectly to a function with appropriate properties) are recorded as instances of the class functional safety requirement. Others are recorded in non functional classes. Similarly, functional system requirements developed by system engineers can actually be safety related (if they are assigned with an ASIL different than QM) so these have to be asserted as functional safety requirements by ontology engineers.

In our conceptualization, functional system requirements derived from system missions correspond to the top level elements of the functional requirements hierarchy. We concluded the previous section with the ASIL assignment to these elements. In this activity, system engineers produce more precise functional system requirements as usual. The interesting part concerns functional safety and our application of ASIL decomposition on functional requirements presented in section 2.2.2.2. Functional safety engineers produce more and more precise functional system requirements. Additionally, they can emit independence requirements on the system functionalities (in the general sense of the term as system functions do not yet exist) which will be materialized in the system architecture (functional and physical). This enables to consider safety in the domain of the problem (specification) to guide the production of conceptually safe architectures (design). This improves the current outdated view on system design process were system engineers first define the system architecture, which is then analyzed by safety engineers who emit recommendations for securing the architectures (if tolerable risk is not achieved), which are taken into account by system engineers with a definition of a *secured architecture* of the system. In the ontology, the recommendations correspond to system requirements (and independence requirements between functional requirements) and the system architecture corresponds to the final secured architecture.

These concepts correspond to our interpretation of ISO 26262 for Renault. ASIL decom-

position is relatively new and we had no real data to work with so the following examples concerning ASIL decomposition and ASIL propagation were defined to present different uses of the ontology. The functional system requirements are hierarchically structured with sub-requirements being more accurate than super-requirements. For instance, we have super-requirement REQ_RCB_107 that has REQ_RCB_179 and REQ_RCB_180 as sub-requirements:

REQ_RCB_107: The RCB system shall know the estimated position of the service brake command HMI from the pressure in the hydraulic network or the mechanical position of the service brake HMI.

REQ_RCB_179: The RCB system shall know the pressure in the hydraulic network.

REQ_RCB_180: The RCB system shall know the mechanical position of the service brake HMI.

These requirements are actually sub-requirements of precedent top level functional system requirement *REQ_RCB_1* that is ASIL D. Intuitively, the failure of *REQ_RCB_107* leads contextually to catastrophic consequences and should be assigned with ASIL D (assuming ASIL decomposition was not previously done for its super-requirements). The two sub-requirements are actually functionally redundant as it is still possible to estimate the position of the service brake with only one piece of information. Functional safety engineers can perform ASIL decomposition on these three requirements with the ASIL D being decomposed into two independent ASIL B (requirements). Given the ASIL of top level functional system requirements and information when ASIL decomposition applies, ontology engineers can execute SWRL rules (fundamentally SQWRL queries) to perform ASIL propagation in the functional system requirements hierarchical structure. The ASIL D of *REQ_RCB_1* is propagated to all its sub-requirements and when it reaches *REQ_RCB_107*, it is decomposed into two ASIL B for *REQ_RCB_179* and *REQ_RCB_180* (if they are not related with another super-requirement with a higher ASIL in which case it is the highest that is assigned). Figure 3.19 is a partial view of these informations.

ASIL propagation has been defined to gain more insights from future use of ASIL decomposition by safety engineers. This is completely new as assigning ASIL to functional

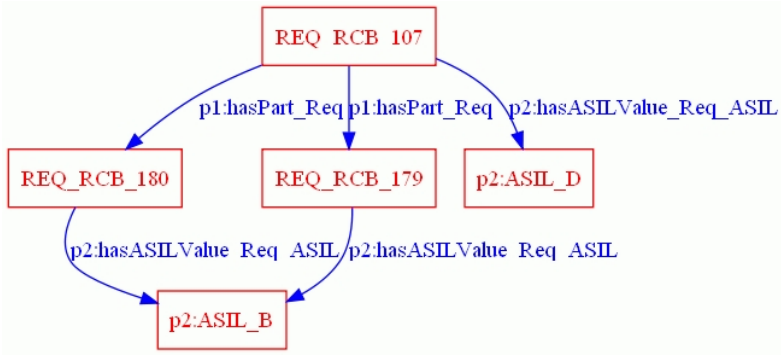

Figure 3.19: ASIL decomposition example – Protégé individuals visualization

requirements and expressing independence between requirements is not part of functional safety current practice at Renault (in fact independence requirements are expressed at the architectural level by defining two functions or two components). However it seems appropriate considering ISO 26262. For now, ASIL decomposition and ASIL propagation can be performed in the ontology up to the system architecture. This enables to define *a priori* ASIL that serve as specifications. Moreover, as we will see in the following, it enables to detect inconsistencies between system specification and solution.

3.3.4 Functional Architecture Definition

During this activity functional system requirements are the basis for the design of a functional architecture that organizes functions and flows. Figure 3.20 corresponds to the functional architecture view in ArKItect. Parallelograms and arrows respectively represent system functions and flows. System functions are organized hierarchically into sub-functions that are encapsulated into super-functions which is not represented in the figure.

Figure 3.21 is a view that lists system functions. Sub-functions appear with an offset to the right compared to their super-functions.

Finally, figure 3.22 is a view that focuses on one flow to present functions that produce or consume this flow.

In section 2.2.1.5 we presented the traceability relation between functional requirements and functions. Functional requirements are used to deduce one or more functions. For instance, functional system requirements *REQ_RCB_179* and *REQ_RCB_180* are

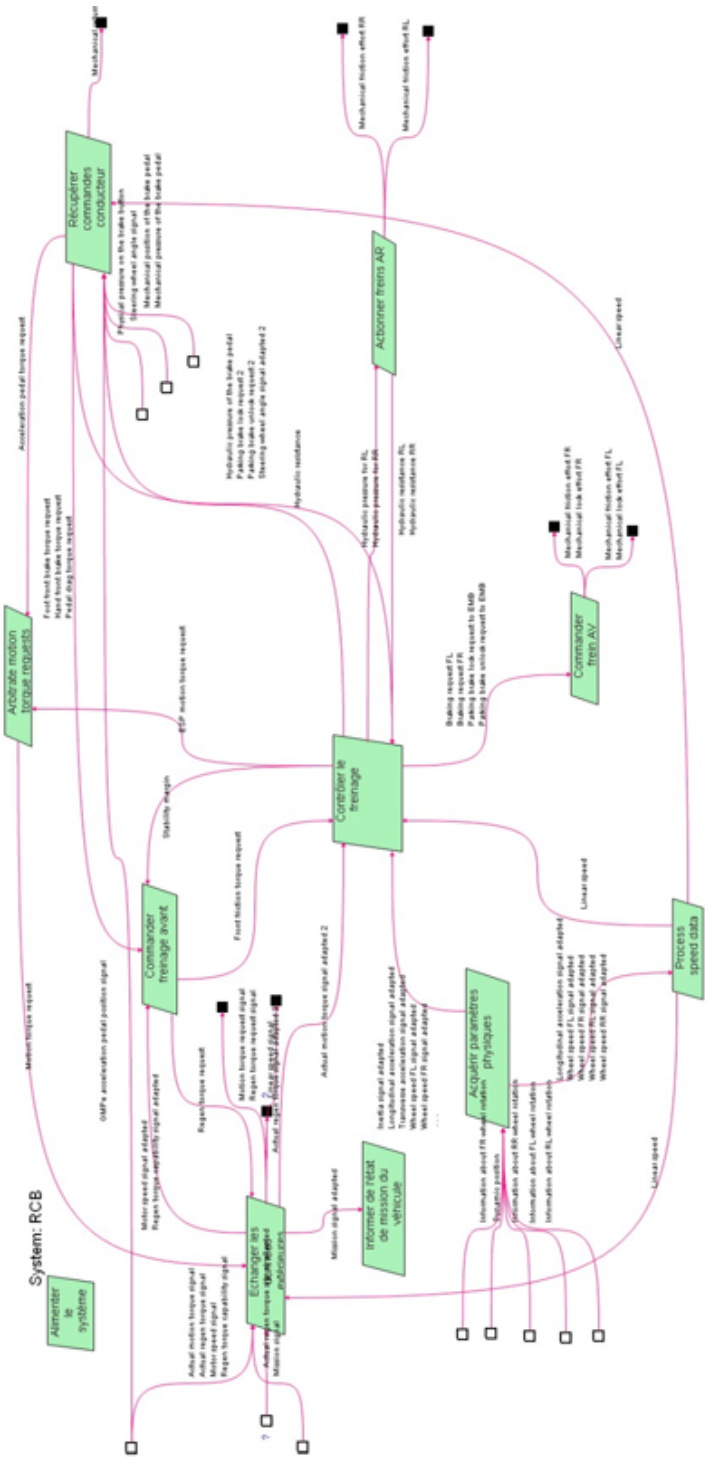

Figure 3.20: RCB functional architecture view in ArKItect

System: RCB [System]

	ArkId
-- Acquérir cluster data	Fun_INT_0036
Adapter le signal d'accélération longitudinale	Fun_INT_0040
Adapter le signal d'accélération transversale	Fun_INT_0041
Adapter le signal d'inertie	Fun_INT_0042
Capter l'accélération longitudinale du véhicule	Fun_INT_0037
Capter l'accélération transversale du véhicule	Fun_INT_0038
Capter l'inertie du véhicule	Fun_INT_0039
+ Acquérir les vitesses de rotation des roues	Fun_INT_0043
-- Actionner freins AR	Fun_INT_0052
Actionner frein hydraulique ARD	Fun_INT_0053
Actionner frein hydraulique ARG	Fun_INT_0054
Retourner une résistance hydraulique ARD	Fun_INT_0055
Retourner une résistance hydraulique ARG	Fun_INT_0056
Alimenter le système	Fun_INT_0089
Arbitrate motion torque requests	Fun_INT_0070
-- Commander frein AV	Fun_INT_0057
+ Actionner frein EMB	Fun_INT_0061
+ Actionner verrou EMB pour le blocage des roues AV	Fun_INT_0058
Commander EMB AVD	Fun_INT_0068
Commander EMB AVG	Fun_INT_0067
+ Superviser EMB	Fun_INT_0064
-- Commander freinage avant	Fun_INT_0001
Arbitrate brushing	Fun_INT_0006
Arbitrate friction torque requests	Fun_INT_0003
Arbitrate regen torque requests	Fun_INT_0004
Blend brake	Fun_INT_0005
Blend drag	Fun_INT_0002
+ Contrôler le freinage	Fun_INT_0084

Figure 3.21: RCB functions listing in ArKItect

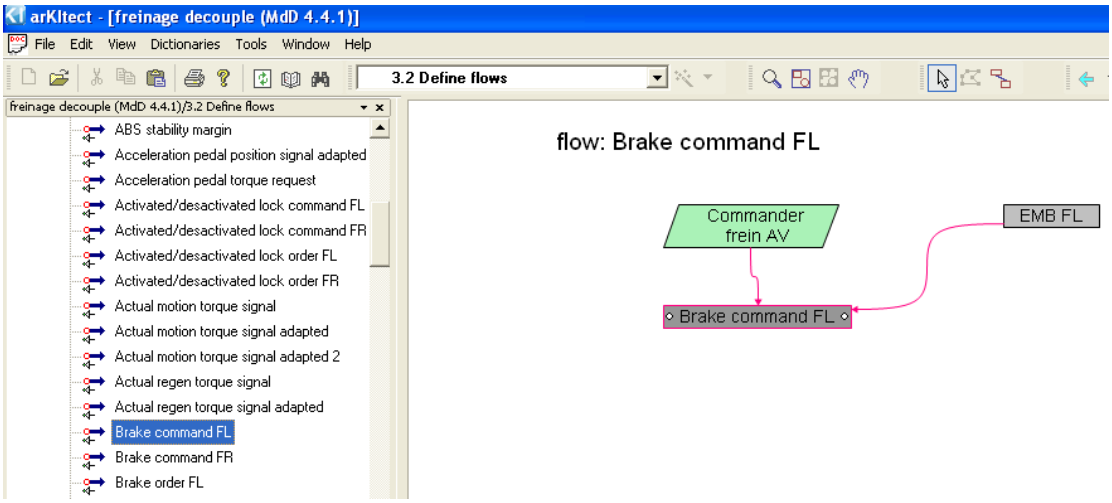

Figure 3.22: Flow view in ArKItect

respectively related to functions *FUN_INT_13* and *FUN_INT_18*.

REQ_RCB_179: The RCB system shall know the pressure in the hydraulic network.

REQ_RCB_180: The RCB system shall know the mechanical position of the service

brake HMI.

FUN_INT_13: sense hydraulic pressure of the master cylinder.

FUN_INT_18: Convert mechanical pressure (of the brake pedal) into hydraulic pressure.

Even though, system functions are mostly deduced from requirements, the functional architecture definition is still a creative activity in its own right. For instance, the preceding functions have been encapsulated into the following super function: *FUN_INT_11: acquire driver's commands.*

Naturally, the tight collaboration between safety engineers and system engineers is required in this activity. Safety engineers have to demonstrate that the risk is tolerable for a given functional architecture and when it is not the case, they should give recommendations for changes. To determine if the risk is tolerable or not, they base their analysis on the ASIL. As defined in ISO 26262, they have to assign to a function the highest ASIL of the functional safety requirements in relation. We had little access to functional safety information so this assignment remains obscure. However, the ontology gives a precise definition that enables to perform this assignment through ASIL propagation. Figure 3.23 presents the result of ASIL propagation to the previous functions. ASIL propagation is performed from the leaves in the hierarchy of functional system requirements so *FUN_INT_13* and *FUN_INT_18* are assigned the highest ASIL of their respective functional safety requirements (ASIL B in this example).

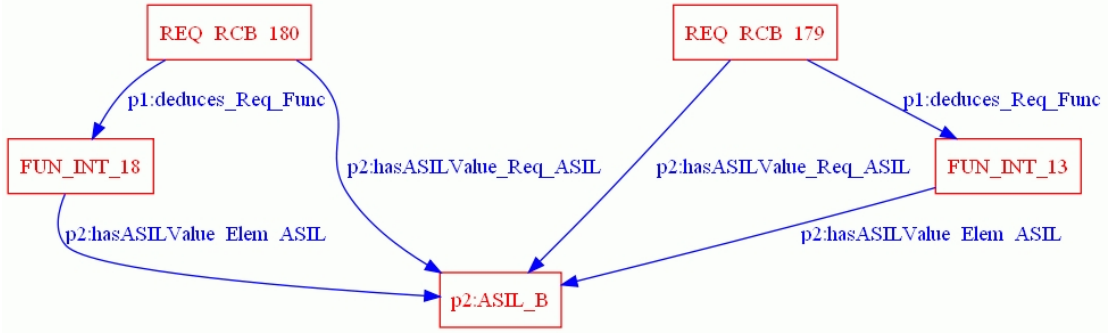

Figure 3.23: ASIL propagation on the functional architecture example – Protégé individuals visualization

With ASIL propagation it is possible to verify that functions ASIL assignment has been

done consistently compared to the ontology. Ontology engineers can record ASIL assigned to functions, execute ASIL propagation based on the functional system requirements and functional architecture structure, and detect any inconsistency revealing important matters to consider. Safety engineers can however perform a posteriori ASIL tailoring with criticality analysis. This safety study enables to reduce the ASIL of an architectural element (*i.e.*, a function or a component) that can invalidate ASIL propagation. For example, as a result of criticality analysis, *FUN_INT_13* can be assigned with ASIL A. This analysis is done later in the process compared to ASIL decomposition and is therefore more relevant (as the system is better understood). It results that the correct ASIL to be assigned to *FUN_INT_13* is really A and not B. Nevertheless, performing ASIL propagation will result in a contradiction that will lead to question why it is not ASIL B that is assigned ensuring that necessary information (criticality analysis data) that resulted in ASIL tailoring exist.

Finally, the ontology can also be used to verify inter model consistency and gain more insights from this level of detail. As it can be seen in figures 3.6 and 3.20 different descriptions and models can be developed concerning similar information. The ontology at the instance level is actually the global system model. Let us assume that figure 3.6 is another functional architecture and that we defined adequate mappings to record the two descriptions into the ontology (for example circles in figure 3.6 represent sensors that can be interpreted as sensing functions, *e.g.*, *sense MC pressure and pedal stroke* for the brake pedal circle). One difficulty for ontology engineers is that they have to identify equivalent individuals and assert that they are the same in the ontology. This is done only once and it formalizes the interconnection between different documents and models. It should also be relatively easy as the common point of all the process actors is that they develop the same system. If it is not the case, ontology engineers might ask whether or not it is really the same system that is being developed. In this example, *sense MC pressure and pedal stroke* is asserted to be the same as *FUN_INT_11*.

FUN_INT_11: acquire driver's commands.

Checking the ontology consistency ontology will guaranty that all models are consistent. Given our hypothesis, consistency checking will be negative in this example. At the bottom right hand side of figure 3.6, we find two flows that are produced by *sense MC pressure and*

pedal stroke: *MC pressure* and *pedal stroke*. In the model in figure 3.20, flow *Flo_INT_19: Hydraulic pressure of the brake pedal* is produced by *FUN_INT_13*. *Flo_INT_19* and *MC pressure* are similar and asserted to be the same which result in a contradiction as a flow can only be produced by one function and *FUN_INT_11* and *FUN_INT_13* are different (this is normal because figure 3.6 is used as a specification for the system and is not a description of the architecture).

3.3.5 Physical Architecture Definition

This is the last activity presented in the design process where system engineers seek to find a solution (in terms of components) that supports the functional architecture. Figure 3.24 is the physical architecture view in ArKItect. Rectangles, cylinders and arrows respectively correspond to components, interfaces and connections that are actually flows. It is important to consider interfaces as distinct components as they are adequate for transportation of specific types of flows. Similarly to requirements and functions, components are structured hierarchically and sub-components are encapsulated into super-components.

Figure 3.25 lists the RCB components without the interfaces. In this activity, system functions are allocated to components. Allocation relation enables to trace the functions to the components that will realize them.

Figure 3.26 presents the internal view of the rear right wheel drum actuator, view that is composed of the actuator allocated functions. In arKItect, only leaf functions of the hierarchy are allocated to exactly one component which suits the ontology definition that leaf functions have to be allocated to one component.

Returning to the example, *FUN_INT_13: sense hydraulic pressure of the master cylinder*, and *FUN_INT_18: Convert mechanical pressure (of the brake pedal) into hydraulic pressure* have been allocated to *Sub_INT_9* that is a control block that comprises a calculator (ECU for Electronic Control Unit) and a sensor.

Sub_INT_9: Conti ECU and hydraulic component that contains the master cylinder pressure sensor.

Let us note that due to implementation choices their super-function (*FUN_INT_11:*

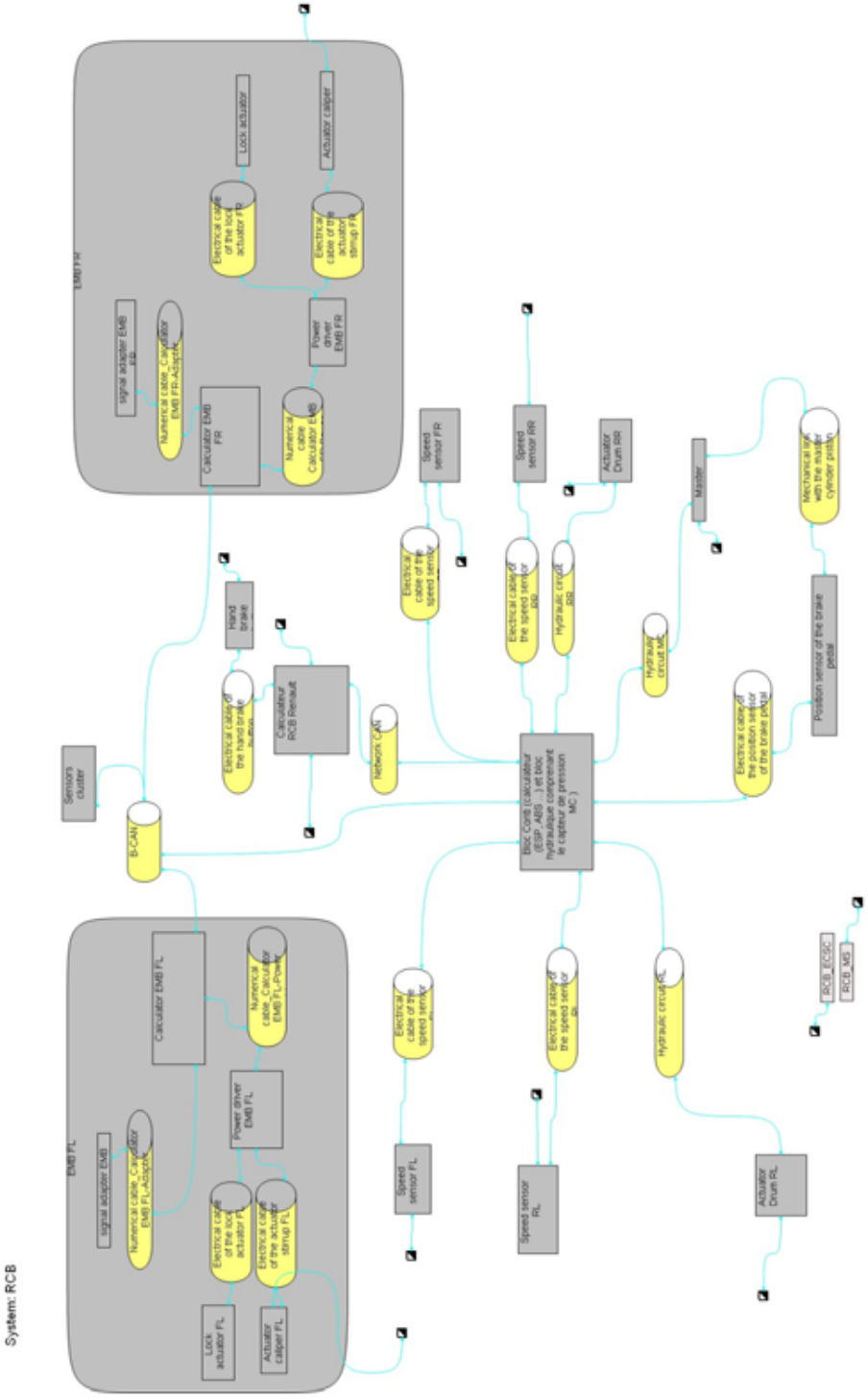

Figure 3.24: RCB physical architecture view in ArKItect

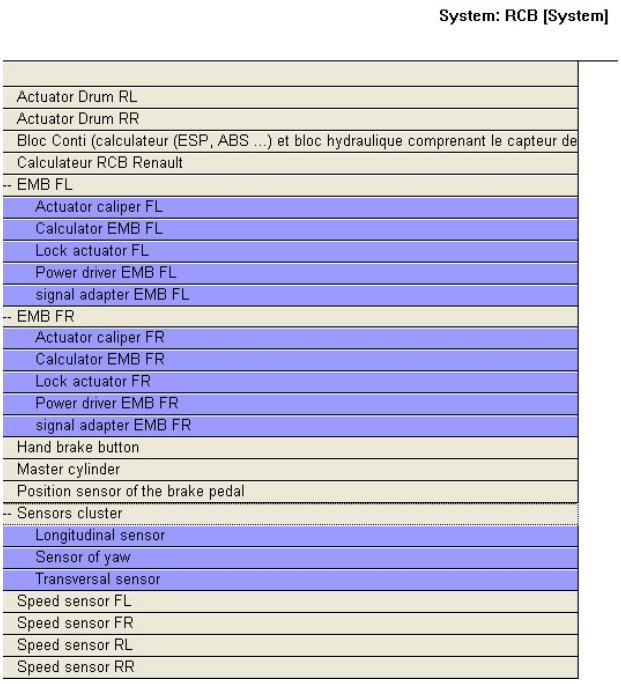

Figure 3.25: RCB components listing in ArKItect

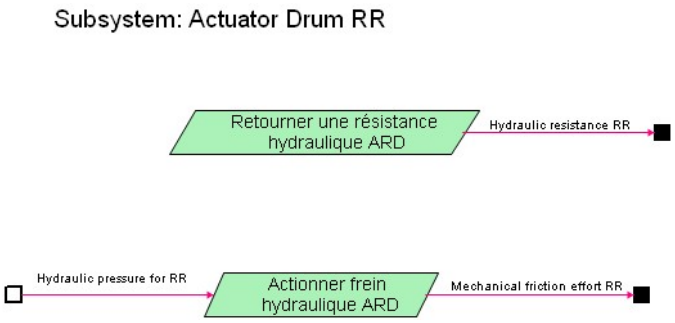

Figure 3.26: Component internal view in ArKItect

acquire driver's commands) has not been allocated to a single component as it is composed of other acquisition sub-functions that are allocated to other components, the capability to allocate a super-function exists nevertheless.

Safety engineers collaborate with system engineers in this activity keeping in mind that the goal is a safe architecture. In parallel, they have to assign an ASIL to each component similarly to the functions. In ISO 26262, an architectural element is assigned the highest ASIL of its related functional safety requirements. As explained in the previous section, these functional safety requirements differ from the sense of functional in the ontology. So

we interpreted ASIL assignment on components as ASIL propagation from the functions to the components. In the ontology, a component is assigned with the highest ASIL of the leaf functions it realizes. For example, *Sub_INT_9* realizes *FUN_INT_13* (and *FUN_INT_18*) but also realizes the following function:

FUN_INT_86: control base brake (BFD, ABS).

FUN_INT_86 has been assigned with ASIL D so *Sub_INT_9* is also assigned ASIL D through ASIL propagation even though *FUN_INT_13* is ASIL B. Figure 3.27 is a view of these informations in the ontology.

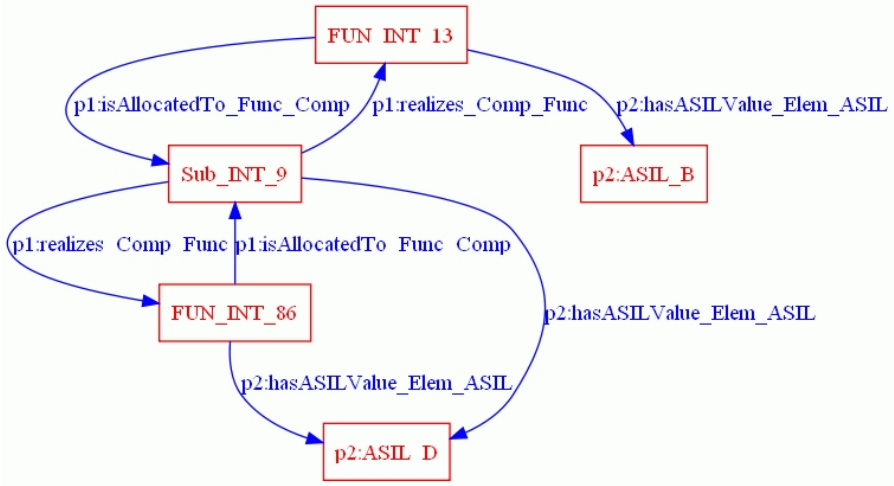

Figure 3.27: ASIL propagation on the physical architecture example – Protégé individuals visualization

Similarly to the functional architecture, ASIL propagation is defined in the ontology and explains how ASIL assignment is done on the components for the physical architecture. ASIL assignment activity is done by functional safety engineers so ontology engineers role is to record these informations in the ontology and verify that consistency still holds even after executing ASIL propagation. For instance, if component *Sub_INT_9* was assigned with ASIL C from functional safety, recording this information in the ontology and then performing ASIL propagation will assign ASIL D to the component and will result in an inconsistent ontology. ASIL propagation conclusions can still be invalidated due to criticality analysis but this does not reduce the produced information value.

Finally, ontology engineers can take advantage of the information structure defined in

the ontology and use it as a knowledge base by defining and executing different queries to bring important information to the surface (traceability coverage analyses, multi-model informations such as the complete set of functions allocated to a controller, *etc.*).

3.3.6 Conclusion

In this section the design approach presented in the previous chapter has been applied on the RCB project. The ontology centric design process has been developed so as not to completely change the current design process at Renault but to be based on the latter by explaining previously imprecise activities and by inserting the new role of ontology engineer. The ontology engineer verifies consistency issues that result mostly from parallel development branches. In particular, systems engineering and functional safety domains are two development branches that manipulate similar concepts that needed to be integrated for better synergy. The results presented in this application comes from the shared conceptualization and are threefold:

First: the integration of systems engineering and functional safety domains gives a precise and unique interpretation to previously similar notions with incompatible definitions (*e.g.*, functional system requirements and functional safety requirements) and therefore the information recorded in the ontology is not subject to implicit interpretations, gives an ontology compliant (conceptually) description of the system and is potentially more reusable.

Second: all the important system level (design) concepts of ISO 26262 have been defined and integrated with Renault own concepts. This integration provides the very precise foundations for the ontology design process for safety critical systems that is an answer for Renault to a development process that complies with ISO 26262.

Finally: the ontology based design process answers the consistency challenge caused by information loss at the processes interfaces (especially when the interface is a third party) as consistency can now be verified by ontology engineers.

3.4 Chapter Conclusion

In this chapter we presented the history of electric braking at Renault and the application of *ontology centric design process* for safety critical systems on the RCB project. The most important conclusion learned from precedent electric braking projects was that safety critical systems development requires the automobile constructor to position itself as a system developer rather than an integrator of commercial solutions (e.g., COTS) that are not completely mastered and that the company is liable for. Systems engineering is currently undergoing definition and deployment at Renault to address the system dimension shortcoming. In addition, international standard ISO 26262 is also a big item of concern and this work is part of multiple initiatives to adapt the development process in order to be compliant with this standard. Our proposition, the ontology centric design process, is structured around an ontology that enables to record very precisely all informations relative to one system design. The design process is requirement driven so the information records the top-down nature of the process. Even though it was actually possible to record most of the study case information, this has been done with relative difficulty as the project information has been produced without the formalized conceptualization. Nonetheless, the central ontology (at the conceptual level) was sufficiently general and precise so that most of the produced information from systems engineering and functional safety could be recorded precisely in the ontology. This study has been developed only to demonstrate the process capabilities and is also an example of project capitalization that is more appropriate for reuse than documents and models (that are hardly integrated). Let us note that to really be able to evaluate our process requires that it is applied to more projects. The construction of the ontology at the conceptual level has been fundamental to this work and has involved several changes that were problematic for the instantiation of project information. Moreover, not only some information was missing and therefore could not be added to the ontology, we have also identified and defined concepts that were not part of the project conceptualization, such as ASIL assignment from the main system functionalities. At the instance level, we recorded provided information that amounts to 66 requirements, 79 functions, 83 flows and 71 components. Traceability information was missing and has been only partially defined to verify the implemented concepts. As we

have defined a different conceptualization for ASIL assignment that uses traceability information and ASIL decomposition (information not provided) we performed risk analysis on two functional requirements and only verified that ASIL propagation was working on small examples. Ultimately, missing information mainly concerns relations between different concepts. Queries can be defined to reveal when information is missing. It would amount up to hundreds which reveals that tools are essential for industrial size projects that are generally even more complex.

The process and the ontology have proved satisfactory even though an ontology being incomplete by nature impacts appreciation of *general* and *precise* characteristics. The ontology defines the general concepts of design (*e.g.*, requirement, function, flow, component, ASIL) that enable to record *imprecisely* the information of any automotive safety critical system. Naturally, we want this information to be precise which is done with additional concepts, sub-concepts and constraints. Additional axioms can also be defined to complete the ontology for more preciseness. For instance, other requirements attributes such as priority (requirements are considered according to their priority) and flexibility (requirements are more or less imperative) could be defined. To conclude, the work presented in this thesis is an improved development process at Renault answering the rigor required when developing safety critical systems, it defines all the principles that enables to improve design models quality and consistency through an ontology, and it has enriched the reflexion with further investigations presented in the next chapter that concludes the thesis.

Conclusions and Future Work

Conclusions

The goal of this thesis is to bring a contribution to adapt and improve the development process for safety critical mechatronics systems at Renault. We observed that the process could benefit from more formalization as it was defined imprecisely and changes were required because of international standard ISO 26262 on functional safety. In order to achieve this goal, we have investigated the following research questions:

Q1 How can domain knowledge (*i.e.*, automobile domain knowledge) be formalized ?

Q2 How can expert knowledge about the development process be formalized ?

Q3 How can conformance to standard ISO 26262 be verified ?

Chapter 1 presents systems engineering and functional safety which are the two domains formalized and integrated in this work. It then goes through ontologies in general to focus on formal ontologies with OWL, SWRL and SQWRL languages. Formal ontologies were proposed to formalize a conceptualization of a domain with a formal semantics. The use of OWL and SWRL enables to formally define a domain therefore answering question Q1. Moreover, we chose to formalize process activities *inputs* and *outputs* so that expert knowledge coincides with domain knowledge and therefore the formalization also answers question Q2. Finally, formal languages consistency checking is an automatic analysis that enables to verify the consistency of an ontology and answers question Q3 as we formalized part of ISO 26262.

Chapter 2 presents the contributions. We chose to focus only on the design part of the system development process. We first identified that the documents and models produced

during the design process suffer from many semantic inconsistencies. In general, design is a collaborative process involving different actors that can have different *implicit* domain conceptualizations.

Our first contribution is the proposal to use ontologies to formalize an explicit conceptualization of a domain at Renault. This idea is presently accepted completely.

The second contribution consists in two ontologies on systems engineering that is relatively new and functional safety that is subject to change due to the publication of standard ISO 26262.

The third contribution is the integration of these two ontologies into one ontology for safety critical mechatronics systems design. We identified that these two domains presented some incompatibilities, in particular on the requirement notion, and solved these incompatibilities. The integration of these two domains enables to limit loss of knowledge at the process interfaces which allows to work with more complete information that resulted in a semi-automatic analysis for ASIL assignment called ASIL propagation because it is based on the traceability defined in the ontology.

Our fourth contribution that achieves the goal of this thesis is an improved design process that relies entirely on ontologies: the ontology centric design process.

The fifth contribution is the definition of the new role: ontology engineer.

We first presented the general principles of the approach. The ontology at the conceptual level corresponds to the system data-model. At the instance level, the ontology corresponds to the system consistency reference model. It is the global system model. Working at the conceptual level enables to define mappings between documents, models and the ontology. This enables to enrich the ontology with the information contained in the documents and models, to propagate the changes from one source to their different recipients and to check the consistency of the whole process (*i.e.*, the consistency of all documents and models). Then the approach is presented in details. The design process at Renault is in a transitional phase oriented towards full model-based systems engineering. For now, each project decides how process inputs and outputs are produced using documents or models. In anticipation of a future Renault specific seamless process (with

adequate tools), ontology centric design process introduces the new role of ontology engineer to support this transition. Ontology engineers mission will be to guarantee the design process consistency, ensuring that Renault needs are taken into account in accordance with systems engineering in general and ISO 26262 in particular.

Finally, in chapter 3, we applied the process defined in this work to the Regenerative Combi-Brake study case. The design process is improved in three ways.

First, it integrates formal ontologies which enables to ensure that consistency (also at the semantic level) is attained in a project. The ontology at the instance level is the global system model.

Second, it is better adapted for safety critical systems development as safety concepts have been integrated with systems engineering in a precise unambiguous way whereas this integration was previously implicit.

Third, it also complies with ISO 26262 as concepts of the standards are defined in the ontology and used by the process.

We conclude on other prior results that are part of this thesis. The work described in this thesis has been extensively presented at Renault. Two presentations have been given to DELTA ("Direction de l'Électronique et des Technologies Avancées") director. The work has also been presented during three internal meetings between different departments and several times for the team. Two six months internships were supervised on model transformation and formal verification. Three internal reports were produced. This work has also been made available to the public with presentations in two research laboratories, meetings between Renault and some partners and with three published papers two of which are international. Finally, this work continues to be exploited as presented in the next section.

Future Work

Sharing a conceptualization is invaluable. This actually addresses a peculiarity of the human condition revealed by this quote from Peter Benary: *"Misunderstanding is the most frequent form of communication between people"*. The concepts presented in this thesis

bring many benefits in real world applications where systems are developed by humans. Systems engineering discipline is defined by the INCOSE as *"an interdisciplinary approach and means to enable the realization of successful systems"*. Similarly, the extent of this work exceeds disciplinary boundaries and there are still many aspects that can be improved and need further research.

Related Work

The work presented in this thesis comes from Renault's decision to deploy systems engineering and adds to the discussion with an additional *ontology* dimension. Related works are performed based partially on our work. Lets us mention some of them.

Even though the ontology manipulates the requirement concept, it does not address syntactic nor grammatical structure of natural language. Presently, a thesis is in progress within the project RAMP (Requirement Analysis and Modeling Process). Its objective is to improve the efficiency and quality of requirements expressed in natural language throughout the system life cycle. Expected outcomes are mainly but are not restricted to better requirements authoring, better written requirements and better requirements management and traceability. Identified requirements parts would be associated with their manifestation as concepts in the ontology.

Similarly, variability and optimization in the design process are the subject of respective theses. The first one brings the idea of variability back to the design process as first class element. Given the time constraints relative to development in the automotive, the production of variability information (and therefore the evaluation of different solutions for the same system) remains very specific and most often very late in the design process. The ontology is an input for conceptual elements upon which variability can be expressed (*e.g.*, requirements, functions, flows, components, *etc.*). The second thesis is interested in multi-objective optimization. The considered approach defines an optimization function for a system, which depends on multiple parameters derived from needs and constraints on the system. Following systems engineering these parameters should ultimately correspond to attributes of concepts (*e.g.*, components cost) or to flows (*e.g.*, the road slope parameter impacts energy consumption during acceleration phases but the road is not a concept with

a slope attribute therefore it is recorded as a flow) in the ontology.

Renault is also involved in many collaborative projects where the ontology and the approach have a role to play. For instance, IMOFIS[4] has studied possible modeling tools specifically for safety considerations. Renault language has been implemented in the tools enabling to produce models in Renault dialect. Modeling tools rely on modeling languages (meta-models) and the ontology can actually also be considered as a Renault DSL (Domain Specific Language). The ontology can either be viewed as a modeling tool meta-model specification (the ontology is therefore implemented as a meta-model) or mapped to a tool meta-model giving more confidence that Renault needs are taken into account. More recently, Renault and the CEA[5] have signed an R&D agreement on the green vehicle theme. One research area concerns MBSSE (Model Based Systems and Software Engineering) where system and software models are produced using Renault profiles of SysML and UML and will ultimately give some answers to systems engineering deployment at Renault. The ontology centric process principles applies to this research area. More generally, our work serves as the foundation of Renault reference framework for mechatronics system development. In particular, the ontology is used as the base of the architecture framework in this reference framework. This work can be seen as the means by which Renault will be able to share its needs even more both internally and externally.

Finally, the work realized in this thesis is a theoretical work applied at Renault. It results in a general approach that can be adapted to any environment of any company.

Personal Perspectives

Ontologies cannot be considered as static as there are many occasions that can lead to changes such as changes in the conceptualized domain which leads to the fact that the ontology does not reflect the reality anymore. Ontology evolution is naturally a big concern in the semantic web community as ontologies, central to semantic web systems, are updated along the functionalities of ontology based systems.

In our context, we envision the addition of new concepts to the ontologies defined in

[4]IMOFIS (Ingénierie des MOdèles de FonctIons Sécuritaires), `http://www.imofis.org`
[5]CEA (Commissariat à l'Énergie Atomique et aux Énergies Alternatives), `http://www.cea.fr`

this thesis. Four main areas for evolution are identified.

First, the concept of requirement needs even more precisions. The ontology provides basic modeling pattern for functional and non functional requirements. Specifically, functional requirements are very precisely decomposed between one another and traced to functions and components. Other concepts are required for the hard part that concerns the non functional requirements. The non functional requirements considered in the ontology correspond only to requirements that have an interpretation relative to systems engineering elements (*i.e.*, functions, flows and components). We have given some examples for weight, cost and execution time but this is relatively imprecise and needs further investigation. The different types of non functional requirements need to be identified and defined with their semantic relations and related concepts such as process requirements that will be related to process concepts (activity, technique, *etc.*) for example.

Second, enabling to explicitly represent design choices and variability will benefit systems engineering in general but also functional safety that takes a logic view of the architectures when they study causes of feared customer events. For instance, the physical consideration of a data acquisition requirement can involve different types of sensors with specific characteristics.

This leads to the third area for improvement: optimization analyses that enable to compare two different solutions to the same problem. For instance, system customer cost could be used to differentiate between two solutions.

Fourth, as we only considered ISO 26262 parts that are relative to system level and design, the entire V cycle remains to be addressed completely. In particular, the V cycle ascending branch, *i.e.*, the integration phase, that relates to verification and validation activities.

In this work, we integrated systems engineering and functional safety. The next logical step is to integrate other domains or professional fields involved in the interdisciplinary realization of mechatronics systems. It is possible to emulate the approach presented in this thesis by defining new ontologies and integrating them using imports and specific axioms. In particular, software engineering is relatively close to systems engineering which is very

favorable for formalization and integration. Moreover ISO 26262 considers software as a specific part which formalization could build upon this domain ontology. More generally, the approach to define domain ontologies and then integrate them is our answer to better consider safety as a domain integrated to systems engineering and to make different domains work together by identifying incompatibilities and resolving them. The approach contributes to the improvement of current practices for assignment, argumentation and demonstration of system ASIL. We believe this approach can be adapted or implemented to other normative contexts such as the other standards derived from IEC 61508 bringing a solution to semantic consistency problems that exist in all industrial domains.

From the theoretical point of view, interesting perspectives would be to study alternative ontology languages. Nonmonotonic logics are characterized by a nonmonotonic consequence relation, meaning that adding a formula to a theory can result in reducing its set of consequences. This can be used to express integrity constraints for example. Even though we have seen that it was possible to formalize the universe of discourse with OWL, SWRL and SQWRL, a theory where the Open World Assumption and the Closed World Assumption coexist and its applicability to the industry needs further research.

From the practical point of view, first, more results are needed which will require more applications of the approach. On the one hand, the approach scalability when confronted to the amount of information contained in industrial systems in production has to be assessed. And on the other hand, the central ontology quality also needs to be evaluated to guide ontology evolution.

Second, a quantitative evaluation of the communication between actors that uses our approach and those that do not would characterizes ontologies as assets that are essential or that contributes during development.

Given the diversity of tools that can be used during the design process and their relative volatility as it is being constructed, mappings are not yet tool supported. They are however required by our approach: tooled mappings would guarantee ontology enrichment and model consistent synchronization. New concepts will most certainly emerge from these tools development which will lead to ontology evolution. Moreover, the ontology as the global system model will most certainly appeal for more features, such as ASIL propaga-

tion defined in this thesis, therefore new tools that support additional features are also considered for development.

Bibliography

[AUT] The AUTOSAR Consortium. URL `www.autosar.org`.

[EDO] Edona project. URL `http://www.edona.fr`.

[MEM] MeMVaTEx poroject. URL `www.memvatex.org`.

[ISO 1999] ISO/IEC Guide 51 Safety aspects – Guidelines for their Inclusion in Standards, 1999.

[IEC 2000] IEC 61508: Functional Safety of Electrical/Electronic/Programmable Electronic Safety-Related Systems, 2000.

[ISO 2000] ISO 14971 Medical Devices – Application of Risk Management to Medical Devices, 2000.

[ISO 2002a] ISO/IEC 15288 Systems Engineering – System Life Cycle Processes, 2002a.

[ISO 2002b] ISO/IEC Guide 73 Risk management – Vocabulary – Guidelines for use in standards, 2002b.

[ISO 2007] ISO/IEC 24707 Information Technology – Common Logic (CL): A Framework for a Family of Logic-Based Languages, 2007.

[Sys 2007] Systems Modeling Language (OMG SysML), 2007.

[UML 2007] Unified Modeling Language Specification, Version 2.1.1, 2007.

[ATT 2008] The ATESST Consortium: EAST-ADL 2.0 Specification, 2008. URL `www.atesst.org`.

[INC 2009] Survey of Model-Based System engineering (MBSE) Methodologies. INCOSE, 2009.

[INC 2011] *INCOSE Systems Engineering Handbook. Version 3.2.2.* INCOSE, 2011.

[ISO 2011] International Standard ISO 26262: Road Vehicles – Functional Safety, 2011.

[AAAM 1998] AAAM, editor. *The Abbreviated Injury Scale (1990) – Revision Update 1998.* Des Plaines/IL, 1998. Association for the Advancement of Automotive Medicine (AAAM).

[Albinet et al. 2007] A. Albinet, J.L. Boulanger, H. Dubois, M.A. Peraldi-Frati, Y. Sorel, and Q-D. Van. Model-Based Methodology for Requirements Traceability in Embedded Systems. *3rd ECMDA workshop on traceability*, 2007.

[Albinet et al. 2010] A. Albinet, L. Quéran, B. Sanchez, and Y. Tanguy. Requirement management from System modeling to AUTOSAR SW Components. *Embedded Real Time Software and Systems (ERTS^{2}2010)*, 2010.

[Anneke et al. 2003] K. Anneke, W. Jos, and B. Wim. *MDA Explained: The Model Driven Architectur: Practice and Promise.* Addison Wesley, 2003.

[Avizienis et al. 2004] A. Avizienis, J.-C. Laprie, B. Randell, and C. Landwehr. Basic Concepts and Taxonomy of Dependable and Secure Computing. *IEEE Transactions on Dependable and Secure Computing*, 1(1):11–33, Jan.-March 2004.

[Berners-Lee et al. 2001] Tim Berners-Lee, James Hendler, and Ora Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[Bishop 2008] Robert H. Bishop. *Mechatronic Systems, Sensors, and Actuators: Fundamentals and Modeling.* The Electrical Engineering Handbook Series, The Mechatronics Handbook, Second Edition Series, Volume 1. 2008.

[Booch et al. 2005] G. Booch, J. Rumbaugh, and I. Jacobson. *The Unified Modeling Language User Guide.* Addison-Wesley, second edition, 2005.

[Boulanger 2006] Jean-Louis Boulanger. *Expression et validation des propriétés de sécurité logique et physique pour les systèmes informatiques critiques.* PhD thesis, Université de Technologie de Compiègne, 2006.

[Brinkkemper et al. 2008] Sjaak Brinkkemper, Inge van de Weerd, Motoshi Saeki, and Johan Versendaal. Process Improvement in Requirements Management: A Method Engineering Approach. In *Requirements Engineering: Foundation for Software Quality*, volume 5025 of *Lecture Notes in Computer Science*, pages 6–22. Springer Berlin / Heidelberg, 2008. ISBN 978-3-540-69060-3.

[Brucker et al. 2006] Achim D. Brucker, Jürgen Doser, and Burkhart Wolff. Semantic Issues of OCL: Past, Present and Future. *ECEASST*, pages –1–1, 2006.

[Burr et al. 2005] H. Burr, T. Deubel, M. Vielhaber, S. Haasis, and C. Weber. CAx/Engineering Data Management Integration: Enabler for Methodical Benefits in the Design Process. *Journal of engineering design*, 16(4):385–398, August 2005.

[Bussler 2003] Christoph Bussler. The Role of Semantic Web Technology in Enterprise Application Integration. *Data Engineering*, 51(4):1–7, 2003.

[Chalé Góngora et al. 2010] H. G. Chalé Góngora, O. Taofifenua, and T. Gaudré. A Process and Data Model for Automotive Safety-Critical Systems Design. In 20^{th} *International Symposium of the INCOSE*, 2010.

[Chalé Góngora et al. 2011] H. G. Chalé Góngora, O. Taofifenua, T. Gaudré, N. Lévy, J.L. Boulanger, and A. Topa. Reducing the Gap Between Formal and Informal Worlds in Automotive Safety-Critical Systems. 21^{st} *International Symposium of the INCOSE*, 2011.

[Chalé Góngora et al. 2009] Hugo Guillermo Chalé Góngora, Ofaina Taofifenua, and Thiery Gaudré. Conception de Systèmes Critiques Automobiles - Étude de Mise en Oeuvre de la Norme ISO26262. In *Proceedings of 5ème Conférence Annuelle AFIS (Paris, France)*. AFIS, 2009.

[Driouche et al. 2007] R. Driouche, Z. Boufaida, and F. Kordon. An Enterprise Application Integration Architecture Supporting Ontology Based Approach for B2B Collaboration. *International Journal of Interoperability in Business Information Systems*, 2(2):39–64, September 2007.

[Estefan 2008] Jeff A Estefan. Survey of Model-Based Systems Engineering Methodologies (MBSE) Rev B. Technical report, INCOSE-TD-2007-003-01, June 10 2008.

[Evans et al. 1998] Andy Evans, Robert France, Kevin Lano, and Bernhard Rumpe. The UML as a Formal Modeling Notation. In *Computer Standards & Interfaces*, pages 336–348. Springer, 1998. URL `http://www.cs.york.ac.uk/puml/papers/evansuml.pdf`.

[Friedenthal et al. 2008] Sanford Friedenthal, Alan Moore, and Rick Steiner. *A practical guide to SysML: Systems Model Language*. OMG Press, 2008.

[Gao et al. 2007] David Wenzhong Gao, Chris Mi, and Ali Emadi. Modeling and Simulation of Electric and Hybrid Vehicles. In IEEE, editor, *Proceedings of the IEEE*, volume 95, pages 729–745, 2007.

[Gasevic et al. 2009] Dragan Gasevic, Dragan Djuric, and Vladan Devedzic. *Model Driven Engineering and Ontology Development*. Springer, 2 edition, 2009.

[Green et al. 2001] P. Green, M. Flynn, G. Vanderhagen, J. Ziiomek, E. Ullman, and K. Mayer. Automotive Industry Trends in Electronics: Year 2000 Survey of Senior Executives. Technical Report UMTRI-2001-15, Ann Arbor, MI: University of Michigan Transportation Research Institute., 2001.

[Gruber 1993] Thomas R. Gruber. A Translation Approach to Portable Ontology Specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.

[Gruber 2009] Tom Gruber. Ontology. In *Encyclopedia of Database Systems*, pages 1963–1965. Springer US, 2009.

[Guarino 1998] N. Guarino. Formal Ontology and Information Systems. In *Proceedings of the 1st Internationall Conference on Formal Ontologies in Information Systems (FOIS98) (Trento, Italy)*, pages 3–15. IOS Press, 1998.

[Harmelen and Fensel 1995] Frank Van Harmelen and Dieter Fensel. Formal Methods in Knowledge Engineering. *The Knowledge Engineering Review*, 10:345–360, 1995. URL `http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.14.5746&rep=rep1&type=pdf`.

[Herrmann 1999] Debra S. Herrmann. *Software Safety and Reliability: Techniques, Approaches, and Standards of Key Industrial Sectors.* IEEE Computer Society, International Organization for Standardization (ISO) / International Electrotechnical Commission (IEC), Washington, DC, USA, 1999.

[Horrocks et al. 2004] Ian Horrocks, Peter F. Patel-Schneider, Harold Boley, Said Tabet, Benjamin Grosof, and Mike Dean. SWRL: A Semantic Web Rule Language Combining OWL and RuleML. W3c member submission, World Wide Web Consortium, 2004. URL `http://www.w3.org/Submission/SWRL`.

[Hull et al. 2004] Elizabeth Hull, Ken Jackson, and Jeremy Dick. *Requirements Engineering.* SpringerVerlag, 2004.

[Huth and Ryan 2004] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and Reasoning About Systems.* Cambridge University Press, Second edition, 2004. ISBN 052154310X.

[Kergosien et al. 2010] Eric Kergosien, Mouna Kamel, Christian Sallaberry, Marie-Noëlle Bessagnet, Nathalie Aussenac-Gilles, and Mauro Gaio. Construction et enrichissement automatique d'ontologie à partir de ressources externes. *CoRR*, (abs/1002.0239), 2010.

[Levendovszky et al. 2002] T. Levendovszky, G. Karsai, M. Maroti, A. Lede-czi, and H. Charaf. Model Reuse with Metamodel-Based Transformations. *Lecture Notes In Computer Science*, 2319:166–178, 2002.

[Leveson 1995] Nancy G. Leveson. *Safeware - System Safety and Computers.* Addison-Wesley, 1995.

[Lykins et al. 2000] H. Lykins, S. Friedenthal, and A. Meilich. Adapting UML for an Object Oriented Systems Engineering Method (OOSEM). In *International Council on Systems Engineering* 10^{th} *Annual Symposium*, 2000.

[Maiden et al. 2008] Neil Maiden, Cornelius Ncube, and James Lockerbie. Inventing Requirements: Experiences with an Airport Operations System. In *Requirements Engineering: Foundation for Software Quality*, volume 5025 of *Lecture Notes in Computer Science*, pages 58–72. Springer Berlin / Heidelberg, 2008.

[McDermid 2001] John A McDermid. Software Safety: Where's the Evidence ? In *Proceedings of the Sixth Australian workshop on Safety critical systems and software (SCS'01)*, pages 1–6, Darlinghurst, Australia, 2001. Australian Computer Society, Inc.

[Micskei and Waeselynck 2010] Z. Micskei and H. Waeselynck. The many meanings of UML 2 Sequence Diagrams: a survey. *Software and Systems Modeling*, 2010.

[Motik et al. 2009] B. Motik, P.F. Patel-Schneider, and B. Parsia. OWL 2 Web Ontology Language: Structural Specification and Functional-Style Syntax. W3C Recommendation, October 2009. URL `http://www.w3.org/TR/owl2-syntax/`.

[Motik et al. 2006] Boris Motik, Ian Horrocks, Riccardo Rosati, and Ulrike Sattler. Can OWL and Logic Programming Live Together Happily Ever After ? In *International Semantic Web Conference*, pages 501–514, 2006.

[O'Connor and Das 2009] Martin O'Connor and Amar Das. SQWRL: a Query Language for OWL. *OWL: Experiences and Directions (OWLED), 6th International Workshop*, 2009.

[Papadopoulos et al. 2001] Y. Papadopoulos, J. Mcdermid, R. Sasse, and G. Heiner. Analysis and synthesis of the behaviour of complex programmable electronic systems in conditions of failure. *Reliability Engineering & System Safety*, 71(3):229–247, March 2001.

[Patel-Schneider et al. 2004] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. OWL Web Ontology Language Semantics and Abstract Syntax. W3C Recommendation, 10 February 2004. URL `http://www.w3.org/TR/owl-semantics/`.

[Roser and Bauer 2005] S. Roser and B. Bauer. Ontology-Based Model Transformation. *MoDELS 2005, Satellite Events*, pages 355–356, 2005.

[Söderström et al. 2001] Eva Söderström, Birger Andersson, Paul Johannesson, Erik Perjons, and Benkt Wangler. Towards a framework for comparing process modelling languages. *Lecture Notes In Computer Science*, 2348:600–611, 2001. Proceedings of the 14th International Conference on Advanced Information Systems Engineering.

[Sebastian et al. 2008] A. Sebastian, N. F. Noy, T. Tudorache, and M. A. Musen. A Generic Ontology For Collaborative Ontology-Development Workflows. In *16th International Con-*

ference on Knowledge Engineering and Knowledge Management (EKAW'08). Springer, 2008.

[Stevens et al. 1998] Richard Stevens, Peter Brook, Ken Jackson, and Stuart Arnold. *Systems Engineering: Coping with Complexity*. Prentice Hall PTR, 1998.

[Struss and Price 2004] Peter Struss and Chris Price. Model-Based Systems in the Automotive Industry. *AI Mag*, 24(4):17–34, Winter 2004.

[Stuckenschmidt and Harmelen 2005] H. Stuckenschmidt and F.V. Harmelen. *Information Sharing on the Semantic Web*. Advanced information and knowledge processing. Springer, 2005.

[Studer et al. 1998] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge Engineering: Principles and Methods. *Data and Knowledge Engineering*, 25(1-2):161–197, March 1998.

[Sure et al. 2002] York Sure, Steffen Staab, and Rudi Studer. Methodology for Development and Employment of Ontology Based Knowledge Management Applications. *SIGMOD Rec.*, 31(4):18–23, 2002. ISSN 0163-5808. doi: http://doi.acm.org/10.1145/637411.637414.

[Suwanmanee et al. 2005] S. Suwanmanee, D. Benslimane, and P. Thiran. OWL-based approach for semantic interoperability. 19^{th} *International Conference on Advanced Information Networking and Applications (AINA 2005)*, 1:145–150, 2005.

[Tudorache 2006] Tania Tudorache. *Employing Ontologies for an Improved Development Process in Collaborative Engineering*. PhD thesis, Technical University of Berlin, Shcool of Electrical Engineering and Computer Science, 2006.

[Ungermann 2009] Jörn Ungermann. On Clock Precision Of FlexRay Communication Clusters, 2009. URL `http://flexray.com/publications/On_Clock_Precision_Of_FlexRay_Communication_Clusters.pdf`.

[Webers et al. 2008] Wolfram Webers, Christer Thörn, and Kurt Sandkuhl. Connecting feature models and AUTOSAR: An approach supporting requirements engineering in automotive

industries. In *Requirements Engineering: Foundation for Software Quality*, volume 5025 of *Lecture Notes in Computer Science*, pages 95–108. Springer Berlin / Heidelberg, 2008.

[Wieringa 2004] R.J. Wieringa. Requirements Engineering: Problem Analysis and Solution Specification (Extended Abstract). In *Web Engineering*, volume 3140 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2004.

[Yu et al. 2006] Changrui Yu, Hongwei Wang, and Yan Luo. Extended Ontology Model and Ontology Checking Based on Description Logics. In *FSKD*, volume 4223 of *Lecture Notes in Computer Science*, pages 607–610, 2006.

Annexes

Appendix A

First Order Logic Axiomatization

The following is a partial axiomatization of the constructs used when defining an ontology with OWL.

Class(cls) - cls is a class.
Instance(inst) - $inst$ is an instance.
isInstanceOf(inst,cls) - $inst$ is an instance of the class cls.
isSubclassOf(subcls,cls) - $subcls$ is a subclass of the class cls.

Classes are sets and instances are objects. Predicates $isInstanceOf$ and $isSubClassOf$ correspond to membership and inclusion relations in set theory: $isInstanceOf(inst, cls)$ is noted $inst \in cls$; $isSubclassOf(subcls, cls)$ is noted $subcls \subseteq cls$. A class is a subclass of another class if all its instances are also instances of the other class (referred to as a *superclass*).

$$isSubclassOf(subcls, cls) \Leftrightarrow Class(subcls) \wedge Class(cls) \wedge (inst \in subcls \Rightarrow inst \in cls) \quad \text{(A.1)}$$

Usual relations *union*, *intersection*, *complement* and *disjoint* are defined for classes.

Union(cls1,cls2) - the class which instances are instance of $cls1$ or $cls2$ denoted $cls1 \cup cls2$.
Intersection(cls1,cls2) - the class which instances are instances of $cls1$ and $cls2$ denoted $cls1 \cap cls2$.
Complement(cls) - the class which instances are not instances of cls denoted $\neg cls$.
disjoint(cls1,cls2) - the classes $cls1$ and $cls2$ have no common instance.

For now we have the basic objects class (that are sets) and individuals (that are mem-

bers or elements of classes). Predicates are added to structure the whole. Informally, a relation links a domain to a co-domain.

Relation(rel) - *rel* is a relation.
hasDomain(rel,cls) - the domain of relation *rel* is the class *cls*.
hasCoDomain(rel,cls) - the co-domain of relation *rel* is the class *cls*.

A class is further described with predicates on the relations that it involves.

hasRelation(cls,rel) - the class *cls* implements the relation *rel*.
inherits(cls,rel) - the class *cls* inherits the relation *rel* of its superclass.
Image(cls,rel) - the class that is the image of class *cls* through relation *rel*.
minCardinality(cls,rel,card) - the minimum positive integer *card* of instances in relation with one instance of *cls* through relation *rel*.
maxCardinality(cls,rel,card) - the maximum positive integer *card* of instances in relation with one instance of *cls* through relation *rel*.

Defining a relation for a class is equivalent to adding the class to the domain of the relation (A.2). The defined relation(s) of a class are inherited to the subclasses of the class (A.3).

$$hasRelation(cls, rel) \Leftrightarrow Class(cls) \wedge Relation(Rel) \wedge hasDomain(rel, cls) \tag{A.2}$$

$$\begin{aligned} isSubclassOf(subcls, cls) \wedge (hasRelation(cls, rel) \vee \\ inherits(cls, rel)) \Rightarrow inherits(subcls, rel) \end{aligned} \tag{A.3}$$

The image of a class through a relation is defined at class level with the restriction that the image is a subclass of the co-domain of the relation (A.4). Also, the image of a class is inherited to its subclasses. It can be further customized with the restriction that the image of the subclass is a subset of the image of the superclass (A.5).

The same applies for minimum and maximum cardinalities predicates. They can be defined at class level for a relation. More formally, those predicates define an interval in $\mathbb{N}^+$.
Asserting $minCardinality(cls, rel, mincard)$ defines that the cardinal number of image(s), of any instance of the class *cls* under *rel*, is in the interval $[mincard\,; +\infty[$.

Asserting $maxCardinality(cls, rel, maxcard)$ defines that the cardinal number of image(s), of any instance of the class cls under rel, is in the interval $[0\,; maxcard]$.
If both minimum and maximum cardinalities are asserted, then the number of image(s) is in the interval $[mincard\,; +\infty[\cap [0\,;\ maxcard]$ with $mincard \leq maxcard$.
If nothing is asserted on cardinality, then the number of image(s) is in $\mathbb{N}^+$ by default.

Defined minimum and maximum cardinality relations for a class are inherited to the subclasses of the class (A.6) and A.7. They can be further customized at subclass level with the restriction that the set that corresponds to the defined cardinality interval of the subclass is included or equal to the set that corresponds to the cardinality interval of the superclass (A.8) and (A.9).

$$(hasRelation(cls1, rel) \vee inherits(cls1, rel)) \wedge hasCoDomain(rel, cls2) \Rightarrow Image(cls1, rel) \subseteq cls2 \tag{A.4}$$

$$(hasRelation(cls, rel) \vee inherits(cls, rel)) \wedge isSubclassOf(subcls, cls) \Rightarrow Image(subcls, rel) \subseteq Image(cls, rel) \tag{A.5}$$

$$\forall\, card \in \mathbb{N}^+ \quad (hasRelation(cls, rel) \vee inherits(cls, rel)) \wedge isSubclassOf(subcls, cls) \wedge minCardinality(cls, rel, card) \Rightarrow minCardinality(subcls, rel, card) \tag{A.6}$$

$$\forall\, card \in \mathbb{N}^+ \quad (hasRelation(cls, rel) \vee inherits(cls, rel)) \wedge isSubclassOf(subcls, cls) \wedge maxCardinality(cls, rel, card) \Rightarrow maxCardinality(subcls, rel, card) \tag{A.7}$$

$$\forall\, card, subcard \in \mathbb{N}^+ \quad (hasRelation(cls, rel) \vee inherits(cls, rel)) \wedge isSubclassOf(subcls, cls) \wedge minCardinality(cls, rel, card) \Rightarrow minCardinality(subcls, rel, subcard) \wedge card \leq subcard \tag{A.8}$$

$$\forall\, card, subcard \in \mathbb{N}^+ \quad (hasRelation(cls, rel) \vee inherits(cls, rel)) \wedge isSubclassOf(subcls, cls) \wedge maxCardinality(cls, rel, card) \Rightarrow maxCardinality(subcls, rel, subcard) \wedge card \leq subcard \tag{A.9}$$

Appendix B

Axioms for ASIL propagation

Propagation of the ASIL from the Safety Goals throughout the Functional System Requirements Hierarchy

$$
\begin{aligned}
&let \\
&\quad Z = \{z \mid p2{:}hasASILValue\_Req\_ASIL(x,z)\} \\
&\quad a = maxASIL(Z) \\
&in \quad p2{:}FunctionalRequirement(y) \wedge \\
&\quad [(p1{:}hasPart\_Req(x,y) \wedge \\
&\quad \neg p2{:}ASILDecompositionRelation\_FuncSafReq\_DecSche(x,link) \wedge \\
&\quad p2{:}hasASILValue\_Req\_ASIL(x,z)) \\
&\quad \vee \\
&\quad (p2{:}SafetyGoal(x) \wedge p1{:}derives\_Req(x,y) \wedge \\
&\quad p2{:}hasASILValue\_Req\_ASIL(x,z)) \\
&\quad \vee \\
&\quad (p1{:}hasPart\_Req(x,y) \wedge \\
&\quad p2{:}ASILDecompositionRelation\_FuncSafReq\_DecSche(x,link) \wedge \\
&\quad p2{:}isDecomposedIntoSubRequirement1\_ASILDecSch\_Req(link,y) \wedge \\
&\quad p2{:}ASILIsDecomposedInto1\_ASILDecSche\_ASIL(link,z)) \\
&\quad \vee \\
&\quad (p1{:}hasPart\_Req(x,y) \wedge \\
&\quad p2{:}ASILDecompositionRelation\_FuncSafReq\_DecSche(x,link) \wedge \\
&\quad p2{:}isDecomposedIntoSubRequirement2\_ASILDecSch\_Req(link,y) \wedge \\
&\quad p2{:}ASILIsDecomposedInto2\_ASILDecSche\_ASIL(link,z))] \\
&\qquad \Rightarrow p2{:}hasASILValue\_Req\_ASIL(y,a)
\end{aligned}
\tag{B.1}
$$

$$
\begin{aligned}
&let \\
&\quad Z = \{z \mid p2{:}hasASILValue\_Req\_ASIL(x,z) \vee p2{:}hasASILObjective\text{-} \\
&\qquad ValueBeforeDecomposition\_Elem\_ASIL(x,z)\} \\
&\quad a = maxASIL(Z) \\
&in \quad p2{:}FunctionalRequirement(y) \wedge \\
&\qquad [(p2{:}ASILDecompositionRelation\_FuncSafReq\_DecSche(x,link) \wedge \\
&\qquad p2{:}isDecomposedIntoSubRequirement1\_ASILDecSch\_Req(link,y) \vee \\
&\qquad (p2{:}isDecomposedIntoSubRequirement2\_ASILDecSch\_\text{-} \\
&\qquad Req(link,y)) \wedge p2{:}hasASILValue\_Req\_ASIL(x,z)) \\
&\qquad \vee \\
&\qquad (p1{:}hasPart\_Req(x,y) \wedge p2{:}hasASILObjective\text{-} \\
&\qquad ValueBeforeDecomposition\_Elem\_ASIL(x,z))] \\
&\qquad\quad \Rightarrow p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_\text{-} \\
&\qquad\quad\ ASIL(y,a)
\end{aligned}
\tag{B.2}
$$

Propagation of the ASIL from the Low Level Functional System Requirements throughout the System Functions Hierarchy

$$
\begin{aligned}
&let \\
&\quad Z = \{z \mid p2{:}hasASILValue\_Req\_ASIL(x,z) \vee \\
&\qquad p2{:}hasASILValue\_Elem\_ASIL(x,z)\} \\
&\quad a = maxASIL(Z) \\
&in \quad p1{:}SystemFunction(y) \wedge \neg p2{:}hasASILValue\_Elem\_\text{-} \\
&\qquad ASIL(y,nonexist) \wedge \\
&\qquad [(p1{:}LowLevelFunctionalSystemRequirement(x) \wedge \\
&\qquad p1{:}deduces\_Req\_Func(x,y) \wedge \\
&\qquad p2{:}hasASILValue\_Req\_ASIL(x,z)) \\
&\qquad \vee \\
&\qquad (p1{:}hasPart\_Func(x,y) \wedge p2{:}hasASILValue\_Elem\_ASIL(x,z))] \\
&\qquad\quad \Rightarrow p2{:}hasASILValue\_Elem\_ASIL(y,a)
\end{aligned}
\tag{B.3}
$$

$$
\begin{array}{l}
let \\
\quad Z = \{z \mid p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_\text{-} \\
\qquad ASIL(x,z)\} \\
\quad a = maxASIL(Z) \\
in \quad p1{:}SystemFunction(y) \wedge \\
\qquad [(p1{:}LowLevelFunctionalSystemRequirement(x) \wedge \\
\qquad p1{:}deduces\_Req\_Func(x,y) \wedge p2{:}hasASILObjectiveValue\text{-} \\
\qquad BeforeDecomposition\_Elem\_ASIL(x,z)) \\
\qquad \vee \\
\qquad (p1{:}hasPart\_Func(x,y) \wedge p2{:}hasASILObjectiveValue\text{-} \\
\qquad BeforeDecomposition\_Elem\_ASIL(x,z))] \\
\qquad\quad \Rightarrow p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_\text{-} \\
\qquad\qquad ASIL(y,a)
\end{array}
\tag{B.4}
$$

Propagation of the ASIL from the Low Level System Functions throughout the System Components Hierarchy

$$
\begin{array}{l}
let \\
\quad Z = \{z \mid p2{:}hasASILValue\_Elem\_ASIL(x,z)\} \\
\quad a = maxASIL(Z) \\
in \quad (p1{:}SystemComponent(y) \vee p1{:}Interface(y)) \wedge \\
\qquad \neg p2{:}hasASILValue\_Elem\_ASIL(y, nonexist) \wedge \\
\qquad [p1{:}LowLevelSystemFunction(x) \wedge \\
\qquad p1{:}isAllocatedTo\_Func\_Comp(x,y) \wedge \\
\qquad p2{:}hasASILValue\_Elem\_ASIL(x,z)] \\
\qquad \vee \\
\qquad [p1{:}hasPart\_Comp(x,y) \wedge p2{:}hasASILValue\_Elem\_ASIL(x,z)] \\
\qquad\quad \Rightarrow p2{:}hasASILValue\_Elem\_ASIL(y,a)
\end{array}
\tag{B.5}
$$

$$
\begin{array}{l}
let \\
\quad Z = \{z \mid p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_\text{-} \\
\qquad ASIL(x,z)\} \\
\quad a = maxASIL(Z) \\
in \quad (p1{:}SystemComponent(y) \vee p1{:}Interface(y)) \wedge \\
\qquad [(p1{:}LowLevelSystemFunction(x) \wedge \\
\qquad p1{:}isAllocatedTo\_Func\_Comp(x,y) \wedge \\
\qquad p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_ASIL(x,z)) \\
\qquad \vee \\
\qquad (p1{:}hasPart\_Comp(x,y) \wedge p2{:}hasASILObjectiveValue\text{-} \\
\qquad BeforeDecomposition\_Elem\_ASIL(x,z))] \\
\qquad\quad \Rightarrow p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_\text{-} \\
\qquad\qquad ASIL(y,a)
\end{array}
\tag{B.6}
$$

$$
\begin{aligned}
&let \\
&\quad Z = \{z \mid p2{:}hasASILValue\_Elem\_ASIL(y, z)\} \\
&\quad a = maxASIL(Z) \\
&in \quad p1{:}SystemComponent(x)) \wedge \\
&\qquad \neg p2{:}hasASILValue\_Elem\_ASIL(y, nonexist) \wedge \\
&\qquad p1{:}hasPart\_Comp(x, y) \wedge p2{:}hasASILValue\_Elem\_ASIL(y, z) \\
&\qquad\quad \Rightarrow p2{:}hasASILValue\_Elem\_ASIL(x, a)
\end{aligned}
\tag{B.7}
$$

$$
\begin{aligned}
&let \\
&\quad Z = \{z \mid p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_- \\
&\qquad ASIL(y, z)\} \\
&\quad a = maxASIL(Z) \\
&in \quad p1{:}SystemComponent(y) \wedge \\
&\qquad (p1{:}hasPart\_Comp(x, y) \wedge p2{:}hasASILObjectiveValue- \\
&\qquad BeforeDecomposition\_Elem\_ASIL(y, z))] \\
&\qquad\quad \Rightarrow p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_- \\
&\qquad\qquad ASIL(x, a)
\end{aligned}
\tag{B.8}
$$

Propagation of the ASIL from the Functional System Requirements to the Functional External Requirements

$$
\begin{aligned}
&let \\
&\quad Z = \{z \mid p2{:}hasASILValue\_Req\_ASIL(x, z)\} \\
&\quad a = maxASIL(Z) \\
&in \quad p2{:}FunctionalRequirement(x) \wedge p1{:}derives\_Req(x, y) \wedge \\
&\qquad p2{:}hasASILValue\_Req\_ASIL(x, z) \\
&\qquad\quad \Rightarrow p2{:}hasASILValue\_Req\_ASIL(y, a)
\end{aligned}
\tag{B.9}
$$

$$
\begin{aligned}
&let \\
&\quad Z = \{z \mid p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_- \\
&\qquad ASIL(x, z)\} \\
&\quad a = maxASIL(Z) \\
&in \quad p2{:}FunctionalRequirement(x) \wedge p1{:}derives\_Req(x, y) \wedge \\
&\qquad p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_ASIL(x, z) \\
&\qquad\quad \Rightarrow p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_- \\
&\qquad\qquad ASIL(y, a)
\end{aligned}
\tag{B.10}
$$

Propagation of the ASIL from the Functional External Requirements to the Functions of the External Elements

$$
\begin{aligned}
&let \\
&\quad Z = \{z \mid p2{:}hasASILValue\_Elem\_ASIL(x, z)\} \\
&\quad a = maxASIL(Z) \\
&in \quad p1{:}FunctionalExternalRequirement(x) \wedge \\
&\qquad p1{:}deduces\_Req\_Func(x, y) \wedge \\
&\qquad p2{:}hasASILValue\_Req\_ASIL(x, z) \\
&\qquad\quad \Rightarrow p2{:}hasASILValue\_Elem\_ASIL(y, a)
\end{aligned}
\tag{B.11}
$$

$$
\begin{aligned}
&let \\
&\quad Z = \{z \mid p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_ASIL(x, z)\} \\
&\quad a = maxASIL(Z) \\
&in \quad p1{:}FunctionalExternalRequirement(x) \wedge \\
&\qquad p1{:}deduces\_Req\_Func(x, y) \wedge p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_ASIL(x, z) \\
&\qquad\quad \Rightarrow p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_ASIL(y, a)
\end{aligned}
\tag{B.12}
$$

Propagation of the ASIL from the Functions of the External Elements to External Elements of the System

$$
\begin{aligned}
&let \\
&\quad Z = \{z \mid p2{:}hasASILValue\_Elem\_ASIL(x, z)\} \\
&\quad a = maxASIL(Z) \\
&in \quad p1{:}ExternalElementFunction(x) \wedge \\
&\qquad p1{:}isAllocatedTo\_Func\_Comp(x, y) \wedge \\
&\qquad p2{:}hasASILValue\_Elem\_ASIL(x, z) \\
&\qquad\quad \Rightarrow p2{:}hasASILValue\_Elem\_ASIL(y, a)
\end{aligned}
\tag{B.13}
$$

$$
\begin{aligned}
&let \\
&\quad Z = \{z \mid p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_ASIL(x, z)\} \\
&\quad a = maxASIL(Z) \\
&in \quad p1{:}ExternalElementFunction(x) \wedge \\
&\qquad p1{:}isAllocatedTo\_Func\_Comp(x, y) \wedge p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_ASIL(x, z) \\
&\qquad\quad \Rightarrow p2{:}hasASILObjectiveValueBeforeDecomposition\_Elem\_ASIL(y, a)
\end{aligned}
\tag{B.14}
$$

le cnam

Ofaina TAOFIFENUA

ONTOLOGY CENTRIC DESIGN PROCESS

Sharing a Conceptualization

le cnam

Résumé :
Dans le marché mondial fortement concurrentiel, un constructeur automobile doit offrir à ses clients des services innovants, respectueux de l'environnement et sûrs de fonctionnement. Tout cela doit être fait à des coûts très compétitifs tout en respectant des réglementations et des délais de plus en plus stricts. Ces travaux répondent à ces défis et visent à améliorer le processus de conception des systèmes mécatroniques critiques automobile. Ils montrent que l'utilisation de modèles formels et informels peuvent se rapporter à un modèle sémantique commun, *i.e.*, une ontologie système et sécurité, qui permet d'assurer la cohérence du processus de conception tout en respectant la norme ISO 26262. Les concepts de ces travaux ont été appliquées sur un système de freinage régénératif hybride intégré dans un véhicule électrique. L'application a démontré que l'ontologie réalisée permet d'enregistrer l'information produite lors de la conception et que l'utilisation d'ontologies permet effectivement de détecter les incohérences sémantiques ce qui améliore la qualité des informations de conception, favorise la réutilisation et assure la conformité à l'ISO 26262.

Mots clés :
Ingénierie des sytèmes, sécurité fonctionnelle, ISO 26262, ontologie, processus de conception

Abstract :
In the strongly competitive worldwide market of today, a car manufacturer has to offer to its customers relevant, innovative, reliable, environment friendly and safe services. All this must be done at very competitive costs while complying with more and more stringent regulations and tighter deadlines. This work addresses these challenges and aims at improving the design process for automotive safety critical mechatronics systems. It shows that the use of formal and informal models can commit to a common semantic model, *i.e.*, a system and safety ontology, that enables to ensure the consistency of the whole design process and compliance with standard ISO 26262. The concepts in this work have been applied on a regenerative hybrid braking system integrated into an electrical vehicle. It demonstrated that the realized ontology enables to record the information produced during design and that using ontologies effectively enables to detect semantic inconsistencies which improves design information quality, promotes reuse and ensures ISO 26262 compliance.

Keywords :
Systems engineering, functional safety, ISO 26262, ontology, design process