Vers un langage de haut niveau pour une ingénierie des exigences agile dans le domaine des systèmes embarqués avioniques

Benoit Lebeaupin

NNT : 2017SACLC078

Thèse de doctorat
de l'Université Paris-Saclay
préparée à CentraleSupélec

Ecole doctorale n°573
Interfaces : Approches Interdisciplinaires / Fondements, Applications et Innovation
Spécialité de doctorat : Sciences et Technologies Industrielles

par

Benoît Lebeaupin

Vers un langage de haut niveau pour une ingénierie des exigences agile dans le domaine des systèmes embarqués avioniques

Thèse présentée et soutenue à Gif-sur-Yvette, le 18 décembre 2017.

Composition du Jury :

Mme.	Pascale Le Gall	Professeur CentraleSupélec	(Présidente)
M.	Pierre de Saqui-Sannes	Professeur ISAE SUPEARO	(Rapporteur)
M.	Jean-Yves Choley	Maître de Conférences HDR Supméca	(Rapporteur)
M.	Antoine Rauzy	Professeur NTNU	(Directeur de thèse)
M.	Jean-Marc Roussel	Maître de Conférences HDR ENS Paris-Saclay	(Encadrant)

Remerciements

Je remercie tout d'abord mes encadrants Antoine Rauzy et Jean-Marc Roussel. Merci de m'avoir aidé quand j'en avais besoin et de m'avoir laissé tranquille à d'autres moments. Antoine, j'ai trouvé nos échanges (discussions, conseils, propositions...) très enrichissants, parce qu'ils étaient toujours pertinents bien que sur des sujets très variés. Jean-Marc, nos discussions, bien que parfois moins philosophiques et plus pratico-pratiques, ont été tout aussi importantes.

Je remercie également les personnes avec qui j'ai pu échanger à Safran, notamment Yannick Laplume, Marc Malot, Maurice Theobald, Luca Palladino et François Desnoyer. Merci d'avoir pris de votre temps pour moi, cette thèse n'aurait pas été très loin sans votre aide.

Un grand merci aux personnes que j'ai cotoyées au LGI :

— L'équipe 5 pour son support indéfectible, je ne peux même pas imaginer ce que serait le LGI sans vous.

— Les permanents avec qui j'ai pu discuter de tout, y compris parfois des exigences, autour de la table de la cuisine ou dans le coin café.

— Les doctorants, qu'ils soient déjà partis, arrivés en même temps que moi ou plus jeunes, pour avoir fait de ce lieu de travail un lieu de vie également. En particulier, à toutes les personnes avec qui j'ai pu partager un bureau, Hichem, Pietro, Jing, Tasneem, Fabio et Bruna : merci, c'était génial !

Je souhaite également remercier les autres doctorants d'Antoine et l'"équipe AltaRica" en général. Benjamin, Anthony, Melissa, Abraham, Loic, Michel, Pierre-Antoine, Leila, Tatiana, Huixing, merci pour tous les échanges, scientifiques ou non, que j'ai pu avoir avec vous.

Je tiens à remercier les gens que j'ai rencontrés à l'ENS, et dont certains sont devenus des amis (j'espère pour longtemps). S'ils n'ont (en général) pas participé à ma thèse directement, la vie n'est pas que le travail, et la plupart du temps que je n'ai pas passé au labo pendant ces trois dernières années l'a été avec eux. Parmi eux, merci aux habitants passés et présents de la bikok. La vie en colocation peut avoir quelques inconvénients, mais ils sont négligeables devant le plaisir que j'ai trouvé à vivre avec vous.

Même s'il fait aussi partie des deux groupes mentionnés juste au-dessus, je remercie Luc Pelissier pour avoir répondu à mes questions sur des sujets qu'il maîtrise mieux que moi, et pour les discussions qui en ont suivi.

Finalement, un grand merci à ma famille, que j'ai vue beaucoup moins que je ne l'aurais souhaité ces trois dernières années.

Chapitre 1

Introduction

1.1 Contexte

Actuellement, la plupart des spécifications de systèmes sont écrites en utilisant le langage naturel. Cependant, à cause d'un contexte industriel en évolution, les entreprises cherchent à avoir d'autres moyens de spécifier leurs systèmes, avec des méthodes adaptées à leur contexte et aux systèmes traités.

1.1.1 Un contexte industriel changeant

1.1.1.1 Des systèmes plus complexes

Afin d'avoir un avantage sur le marché, les industriels cherchent à concevoir des systèmes ayant de plus en plus de fonctionnalités, et donc des systèmes de plus en plus complexes. Une complexité plus importante des systèmes implique une complexité plus importante des spécifications. Le fait que la plupart des spécifications systèmes actuelles soient rédigées en langue naturelle limite les possibilités de rendre ces spécifications plus longues et plus complexes.

Les exigences doivent notamment être vérifiées, validées et gérées et il faut vérifier que l'ensemble d'exigences soit complet, cohérent, etc. La langue naturelle est difficile à traiter par ordinateur, et toutes ces étapes de gestion doivent être réalisée par des humains, par exemple lors d'inspections. De plus, l'ambiguïté inhérente des textes en langue naturelle rend ce traitement humain encore plus complexe. Ces éléments vont poser une limite supérieure à la taille des spécifications "raisonnablement" traitables.

1.1.1.2 Demande de réactivité plus importante

Une remarque et un objectif souvent rencontrés dans l'industrie est qu'il faut que le développement de nouveaux produits puisse être plus réactif et adaptable, afin de pouvoir suivre un marché toujours changeant. Cet objectif de réactivité a des implications importantes sur les processus de conception. Si les processus de conception n'ont jamais été

parfaitement séquentiels et qu'il existe toujours des parties itératives dans ces processus, l'objectif de réactivité va donner une importance beaucoup plus grande aux processus itératifs.

On ne souhaite pas recommencer le processus de conception depuis le départ lorsque l'on modifie les besoins. Il faut donc trouver des moyens de garder les éléments déjà développés et de pouvoir tracer les impacts des modifications. La langue naturelle est cependant peu adaptée à cette analyse d'impact.

1.1.1.3 Développement de l'ingénierie basée sur les modèles

La conception de systèmes s'appuie de plus en plus sur l'utilisation de modèles, justement pour répondre aux problèmes mentionnés ci-dessus. L'ingénierie basée sur les modèles vise à éviter l'approche "centrée sur les documents", en particulier les documents textuels, afin de permettre une communication plus efficace entre et à l'intérieur des équipes de conception.

Si l'ingénierie basée sur les modèles est de plus en plus présente dans le reste des processus de conception, les industriels identifient un manque dans l'étape de spécification. On peut considérer que cette étape de spécification constitue alors un goulot d'étranglement, où il est nécessaire d'écrire en langue naturelle des concepts qui jusqu'ici étaient adaptés à une formulation sous forme de modèles. Un des buts de ce travail est donc d'explorer les moyens d'articuler modèles et spécifications.

1.1.1.4 Mais une inertie importante...

Malgré les difficultés causées par l'utilisation de la langue naturelle, il n'est tout d'abord pas possible de s'en passer, et elle reste un outil très pratique. Notamment, c'est un outil que tout le monde maîtrise et qui ne nécessite pas ou peu de formation particulière en entreprise. En revanche, d'autres outils de communication peuvent nécessiter une formation et une adaptation, non seulement dans l'entreprise ou le groupe qui rédige les spécifications, mais aussi pour les lecteurs de la spécification, que ce soient d'autres entreprises ou même les agences de régulations qui revoient les spécifications. Il n'est donc pas possible de proposer de nouvelles méthodes pour les spécifications systèmes sans prendre en compte le contexte industriel, et il faut donc proposer des solutions qui non seulement répondent aux problèmes actuels mais soient également viables d'un point de vue économique.

1.1.2 Quels types de systèmes ?

Même si elles sont regroupées sous le même terme d'"ingénierie des exigences", les spécifications sont sensiblement différentes selon le type de système (ou de logiciel) spécifié. Nous détaillons ici les systèmes sur lesquels nous avons travaillé, qui sont donc le cadre de notre travail.

1.1.2.1 Des systèmes complexes

On l'a mentionné plus haut, les systèmes en général deviennent de plus en plus complexes. Dans notre cas, on se focalise sur des systèmes qui contiennent des éléments physiques et des éléments logiciels et qui peuvent être considérés comme étant constitués de systèmes plus petits. Ces systèmes se distinguent des systèmes purement logiciels, pour lesquels il existe d'autres méthodes d'ingénierie des exigences. Nous ne cherchons pas non plus à spécifier des pièces mécaniques d'un seul tenant par exemple.

Le fait que les systèmes spécifiés soient complexes est important car cela signifie que ces systèmes, ainsi que leurs interfaces, les besoins des parties prenantes, les exigences, etc. sont profondément hétérogènes. Cette hétérogénéité rend difficile une description unifiée du système. C'est un problème dans le domaine de l'ingénierie basée sur les modèles, puisqu'il faut rendre cohérents (ou au moins faire communiquer) des modèles décrivant différentes parties du système, qui peuvent n'avoir que très peu de choses en commun. Mais une spécification est aussi un type de description d'un système, et il faut utiliser un seul type d'élément (les exigences) pour définir des propriétés très variées.

1.1.2.2 Des systèmes critiques

Les systèmes que nous avons étudiés sont des systèmes avioniques, qui de plus sont impliqués dans les fonctions principales de l'avion (déplacement en l'air ou au sol). Ces systèmes, en cas de défaillance, peuvent causer des accidents allant jusqu'à la mort des occupants de l'avion et sont donc des systèmes critiques. Les agences de régulation pour les systèmes critiques (pour les domaines aéronautique et nucléaire par exemple) ont des critères sévères portant sur la qualité des exigences, et demandent que ces exigences fassent l'objet d'une traçabilité complète. L'ingénierie des exigences est donc particulièrement importante pour ces types de systèmes.

Parce que ces systèmes sont critiques, et qu'ils contiennent des parties physiques, il est difficile de modifier des éléments lors du processus de conception : par rapport à, par exemple, une application pour smartphone développée avec une méthode agile, les boucles de rétroaction sont beaucoup plus longues. Cela signifie que les exigences évoluent relativement lentement, mais en revanche, il est nécessaire de tracer plus finement l'impact de ces évolutions sur les autres exigences, et sur le système final.

1.1.2.3 Des systèmes entourés d'autres systèmes

En plus d'être constitués de sous-systèmes, les systèmes que nous étudions s'interfacent principalement avec d'autres systèmes techniques. C'est plutôt une bonne nouvelle, parce qu'a priori, les interfaces entre ces systèmes vont être plus facilement formalisables que les interfaces avec des humains. Un élément critique ici sera donc la définition des limites entre les systèmes, et de savoir ce qu'ils échangent ou n'échangent pas à leurs interfaces.

Comme on le verra dans la suite de ce rapport, nous basons pratiquement tout notre travail sur les interfaces entre le système et son environnement. Plus ces interfaces sont clairement définies et formalisables, plus la rédaction des exigences en sera facilitée.

1.2 Objectifs de la thèse

Des études telles que celle de Schmidt et collab. (2001) montrent qu'une partie importante des projets informatiques échouent ou dépassent leur limite de temps et/ou de budget à cause d'exigences incomplètes, irréalistes ou variables. Cette observation est probablement également vraie pour les types de projets autre que purement logiciels, comme le développement de systèmes physiques. Pour cette raison, l'ingénierie des exigences reste une étape critique du processus d'ingénierie système.

Dans le cadre de la chaire Blériot-Fabre entre Safran et CentralSupélec, nos interlocuteurs à Safran ont identifié l'ingénierie des exigences comme un point qu'il était possible d'améliorer. À partir de ce contexte, des objectifs ont été définis pour la thèse, en collaboration entre Safran et mes encadrants académiques.

1.2.1 Réduire l'ambiguïté

L'ambiguïté est un problème central pour les exigences : on cherche à ce que l'émetteur et le récepteur d'un "message" (la spécification) comprennent bien la même chose.

1.2.1.1 L'ambiguïté est un concept important, mais souvent mal défini

Chaque fois qu'on dresse une liste des qualités que doivent avoir les exigences, il est admis que les exigences doivent être non ambiguës. Cependant l'ambiguïté recouvre des concepts qui peuvent être assez différents : le fait qu'un mot ait plusieurs sens possibles et le fait qu'une même phrase puisse avoir deux structures distinctes sont deux cas d'ambiguïté, mais qui doivent être traités de différentes manières. L'ambiguïté recouvre également d'autres éléments : on demande à ce que les exigences soient "claires" et "précises", mais est-ce qu'une exigence peut être non-ambiguë si elle n'est pas claire ou si elle n'est pas précise ? On va donc chercher à explorer dans ce travail de quelles manières une exigence peut être ambiguë et comment l'éviter.

En plus de l'ambiguïté, d'autres caractéristiques des exigences influent sur la facilité avec laquelle ces exigences seront lues, comprises et gérées. Par exemple, plus les exigences seront longues et/ou complexes, plus il sera difficile de les comprendre. On va chercher à identifier et à quantifier ces différentes caractéristiques afin de pouvoir mieux les gérer.

1.2.1.2 Des moyens concrets pour réduire l'ambiguïté

On cherche donc à proposer des moyens concrets pour réduire l'ambiguïté et faciliter l'ingénierie des exigences dans le cadre posé dans la section précédente. "Concret" a ici deux sens.

Tout d'abord il faut proposer des méthodes qui soient effectivement applicables dans le contexte industriel. L'utilisation de langages formels pourrait effectivement rendre (certaines) exigences non ambiguës, mais il n'est pas faisable en pratique de demander à ce que les spécifications soient entièrement rédigées en langage formel.

Ensuite, nous cherchons à proposer, en plus des idées, des principes et des définitions, des moyens de mettre en œuvre ces idées et ces principes. Si nous n'avons pas développé un logiciel complet permettant de rédiger et de gérer des exigences car c'est un projet trop important pour une thèse, nous avons en revanche écrit un prototype permettant de réaliser des tests et d'illustrer, voire de montrer, l'utilité des principes proposés. De même, nous n'avons pas défini une méthodologie complète avec, par exemple, une check-list d'étapes à réaliser. Mais nous espérons que les éléments de méthode, les divers exemples ainsi que les cas d'utilisation présentés dans ce document dessinent un tableau suffisamment clair de ce que nous cherchons à réaliser.

1.2.2 Faciliter les traitements automatiques

L'un des principaux inconvénients de la langue naturelle est qu'il est difficile de la traiter par ordinateur. Si l'on souhaite faciliter l'ingénierie des exigences, il est nécessaire qu'une partie (de préférence la plus grande possible) des opérations soient faites automatiquement par des programmes informatiques.

1.2.2.1 Proposer des critères testables

Pour les diverses qualités, ou critères, sur les exigences que l'on a évoqués plus haut, on va chercher à définir des propriétés correspondantes pouvant être testées automatiquement. Pris tels quels, certains de ces critères sont non formels et ne peuvent pas, ou très difficilement, être testés automatiquement. Il va donc falloir proposer des critères qui soient suffisamment formels pour pouvoir les tester automatiquement, mais qui soient pertinents au regard du but final, i.e. faciliter la rédaction et la gestion des exigences.

Certains critères sur les exigences ne pourront pas facilement être testés automatiquement, même après les avoir reformulés ou décomposés en critères plus simples. On ne se focalisera pas sur ces critères dans ce travail.

1.2.2.2 Tester ces critères

Cet effort de formalisation ne s'applique pas qu'aux critères : il faut que la façon dont on écrit les exigences soit pensée de manière à pouvoir réaliser ces tests. En plus d'écrire des exigences qui soient non ambiguës pour les humains, il faut que leur structure, leur vocabulaire, leur format, soit aussi compréhensibles par des ordinateurs.

Après avoir défini les formats nécessaires pour les exigences (ainsi que pour le reste des éléments de la spécification tels que des modèles, des définitions), il faut écrire les programmes de tests et les appliquer sur des cas réels. Il est également nécessaire de faire un lien entre les résultats des tests et les qualités des exigences. Par exemple, la profondeur de l'arbre syntaxique d'une exigence est un élément permettant de quantifier sa complexité, mais sans autre contexte, savoir que cette profondeur est 7 pour une exigence donnée n'est pas très utile.

1.2.3 Dépasser les spécifications purement textuelles

Tous ces objectifs conduisent à un constat : les textes en langue naturelle seule ne permettent pas de réaliser ces objectifs. Si l'on ne peut se passer complètement de la langue naturelle, il est nécessaire de la compléter avec d'autres constructions.

1.2.3.1 La langue naturelle a des défauts...

L'utilisation de la langue naturelle pour l'écriture d'exigences présente un certain nombre d'inconvénients, e.g. :

— Les exigences sont un moyen de communication entre parties prenantes. On cherche à minimiser le risque d'incompréhensions. Or, les langues naturelles non contraintes sont intrinsèquement ambiguës.
— En pratique, les langues naturelles ne peuvent que difficilement être utilisées comme support pour du raisonnement automatisé : par exemple, il est difficile de détecter automatiquement qu'une exigence fait référence à un composant ou une fonction non définis.
— Comment articuler des exigences en langue naturelle et une ingénierie basée sur les modèles en un tout cohérent n'est pas clair. Or les modèles et l'ingénierie basée sur les modèles jouent un rôle de plus en plus important dans les processus d'ingénierie système.

Vu ces défauts, il pourrait sembler pertinent de remplacer la langue naturelle par d'autres moyens plus adaptés, cependant, ce n'est pas aussi simple.

1.2.3.2 ...mais elle reste incontournable

On ne peut pas se passer en pratique de la langue naturelle car c'est le seul moyen de communication que pratiquement tous les humains ont en commun. Les autres possibilités, telles que des langues formelles créées spécifiquement pour qu'elles soient non-ambiguës et traitables par ordinateur, ne sont connues que par une petite minorité de la population, et sont relativement difficiles à apprendre. De plus, la langue naturelle a l'avantage de permettre d'exprimer des concepts très variés, et souvent de manière très succincte, ce qui peut être beaucoup plus complexe avec des langues formelles.

On ne peut donc pas se passer de la langue naturelle pour rédiger des spécifications.

1.2.3.3 Compléter la langue naturelle avec d'autres éléments

Nous proposons de compléter les exigences en langue naturelle avec :

— des éléments portant sur les termes (mots ou groupes de mots) utilisés dans les exigences,
— des éléments portant sur la syntaxe de l'exigence.

Que sont ces éléments ?

Pour les termes utilisés dans les exigences, on va essentiellement chercher à fournir des définitions à ces termes. De plus, un système de liens explicites va permettre d'associer les

termes dans une spécification à des définitions de manière non-ambiguë et compréhensible par un ordinateur. Ces liens vont permettre aux lecteurs humains d'être sûrs de quoi parle une exigence, et vont également permettre de réaliser un ensemble de traitements sur les exigences de manière automatique.

Pour les éléments de syntaxe, on va associer à chaque exigence une représentation de sa syntaxe. Cette représentation est créée selon des règles développées spécifiquement pour ce travail et adaptées au problème qui nous intéresse, tout en étant relativement simples. Cette syntaxe va permettre aux lecteurs humains de facilement choisir le sens correct d'un texte, tout en offrant un support pour le traitement automatique.

Nous pensons que ces éléments complémentaires vont permettre d'écrire de meilleures exigences :

— parce que les exigences seront moins ambiguës,
— parce que les exigences seront testables rapidement et automatiquement, permettant de trouver des problèmes pas ou difficilement détectables auparavant,
— parce que les rédacteurs vont devoir faire attention aux éléments qui sont importants lors de la spécification, ce qui va améliorer la qualité des exigences "par construction". Par exemple, il va falloir se poser la question de savoir à quoi, précisément, fait référence telle exigence.

1.2.4 Minimiser le travail

Le sujet de cette thèse n'est pas seulement de résoudre le problème posé, mais aussi de le résoudre en prenant en compte les contraintes industrielles, notamment que, si une méthode proposée demande trop de travail et/ou de formation, elle ne sera pas utilisée en pratique.

1.2.4.1 Réutilisation

Une façon de réduire la charge de travail demandée est de réutiliser des choses déjà faites ou de mieux utiliser des choses qui seront faites de toute façon. Par exemple, on va utiliser des modèles d'architecture système, qui sont déjà au moins partiellement inclus dans les spécifications actuelles, pour définir les termes utilisés dans les exigences. En fait, dans les spécifications actuelles, on trouve des exigences et des modèles, et il ne reste "que" à relier ces éléments. Nous pensons que cette meilleure intégration des modèles et des exigences, qui peut être relativement simple à réaliser, présente des avantages non négligeables.

1.2.4.2 Équilibrer contrainte et permissivité du langage

Dans *Prolog : a logical approach*, Dodd écrit "Somewhere between ridiculous pedantry and erroneous formulation there presumably exists a reasonably precise way of specifying a problem in English" (Dodd et Dodd (1990)). On a vu les problèmes que peuvent causer la langue naturelle non contrôlée pour les spécifications, mais un autre extrême qui serait de demander à ce que les spécifications soient entièrement rédigées dans une langue formelle

présente aussi de sérieux inconvénients. Il faut donc contraindre suffisamment la formulation des exigences de manière à réduire les possibilités d'ambiguïtés, tout en laissant aussi des possibilités au rédacteur d'écrire ce qu'il souhaite si cela ne rentre pas dans les "cases" qui ont été prévues.

De plus, le "point optimal" de cet équilibre va probablement changer selon la spécification et le contexte. Il faut donc fournir aux rédacteurs d'exigences une méthode qui soit adaptable relativement facilement à leur cas particulier.

1.2.4.3 Un outil pour rédiger et tester les exigences

Toutes les idées que nous proposons ne peuvent pas être réalisées avec des documents écrits à la main ou avec un simple logiciel de traitement de texte. Un outil logiciel est donc nécessaire pour la rédaction des exigences telle que nous l'envisageons et il est également nécessaire d'avoir un outil (qui peut faire partie du même logiciel) pour réaliser des tests sur les exigences. Comme nous l'avons écrit plus haut, nous n'avons pas développé un tel outil de manière complète, mais nous avons écrit un prototype permettant de montrer que les principes proposés sont cohérents, et qu'un tel outil est possible.

1.3 Plan détaillé du manuscrit

Dans le chapitre 2, nous présentons une revue des travaux relatifs à l'ingénierie des exigences et à l'ingénierie basée sur les modèles, et une attention particulière est donnée aux travaux intégrant ces deux domaines.

Dans le chapitre 3, nous définissons le cadre dans lequel nous avons développé notre travail, notamment en définissant certains des concepts que nous utilisons, et donnons les idées et principes généraux qui sous-tendent notre travail : il n'est pas possible de se passer complètement de la langue naturelle, et on va donc chercher à réduire l'ambiguïté de cette dernière en la complétant avec des modèles et avec des éléments de syntaxe. Ces compléments vont également permettre de réaliser des tests sur les exigences.

Dans le chapitre 4, nous donnons un exemple de système, un ABS, et détaillons et illustrons les idées du chapitre précédent grâce à cet exemple. Nous allons chercher à construire les exigences à partir d'éléments de base, qui correspondent aux interfaces du système spécifié. On peut également utiliser des éléments plus complexes dans les exigences, mais ils seront toujours construits à partir de ces éléments de base. La syntaxe proposée permet de structurer ces éléments auquel l'exigence fait référence, en quelque chose qui soit effectivement une exigence : une propriété demandée au système spécifié. On peut ensuite réaliser des tests sur ces exigences améliorées, pour vérifier leurs qualités.

Le chapitre 5 consiste en une étude de cas : nous avons étudié une spécification industrielle pour un système réel, l'Electric Green Taxiing System. Nous avons transformé cette spécification industrielle en une nouvelle spécification, en s'appuyant sur les principes développés pendant cette thèse.

Le chapitre 6 présente le prototype logiciel que nous avons développé et comment les exigences, les modèles, les tests, etc. sont effectivement implémentés.

Table des matières

Liste des tableaux

Table des figures

Chapitre 2

Ingénierie des exigences et ingénierie basée sur les modèles

2.1 Ingénierie des exigences

Notre travail fait partie du domaine de l'ingénierie des exigences (IE). Nous présentons ici le processus "classique" d'ingénierie des exigences tel qu'il est pratiqué aujourd'hui, en général et plus particulièrement pour les systèmes complexes. Nous détaillons également un certain nombre de manques qui ont été identifiés dans la littérature concernant l'ingénierie des exigences et plus particulièrement la rédaction d'exigences. Un certain nombre de solutions ne faisant pas appel à des modèles sont proposées pour résoudre ces problèmes.

2.1.1 Processus d'ingénierie des exigences

2.1.1.1 Déroulement "classique" du processus d'IE

On peut séparer les processus d'IE en deux catégories (comme cela est fait dans certains manuels d'IE Pohl (2010); Badreau et Boulanger (2014)) : le développement des exigences et la gestion des exigences. Le développement des exigences consiste à créer les exigences, ce qui nécessite tout d'abord d'avoir une idée de l'environnement du système et de ses parties prenantes. Le développement des exigences peut être séparé en quatre étapes :

- L'élucidation, dont le but est de rassembler un ensemble de "pré-exigences" en étudiant toutes les sources d'exigences possibles.
- L'analyse vise à raffiner ces "pré-exigences" en les complétant, en résolvant les conflits, en essayant de trouver des exigences implicites, etc.
- La spécification (ou rédaction) d'exigences est l'étape où l'on "met au propre" les exigences obtenues dans les étapes précédentes. Le but est d'obtenir des exigences qui aient bien les qualités que l'on attend d'elles : non-ambiguïté, clarté, atomicité, cohérence, etc.
- Finalement, lors de l'étape de validation, on va vérifier que l'on a écrit de bonnes exigences (qualités intrinsèques) et que l'on a écrit des exigences pour le bon système

(est-ce qu'elles correspondent au besoin du client).

La gestion des exigences permet de garder à jour un référentiel d'exigences lors de son (inévitable) évolution et d'obtenir des métriques permettant le pilotage du projet. La gestion des exigences s'appuie également sur un ensemble de sous-activités, telles que les gestions des versions, de la configuration ou de la traçabilité des exigences, la priorisation, l'utilisation d'attributs, etc.

Toutes ces activités, de gestion comme de développement, ne sont généralement pas purement séquentielles et il y a toujours des itérations. Par exemple, dans la méthode Volere (Robertson et Robertson (2012)), il n'y a pas de séparation particulière entre développement et gestion des exigences, et les activités de gestion identifiées ici sont plutôt intégrées dans le développement.

2.1.1.2 Méthodes, outils et moyens

Beaucoup d'articles sur l'ingénierie des exigences ont été publiés, au point qu'il existe une revue systématique sur les revues systématiques traitant d'ingénierie des exigences (Bano et collab. (2014)). Il existe donc beaucoup de méthodes traitant des différentes activités du processus d'IE. Il y a également un certain nombre d'outils qui peuvent être associés à des méthodes particulières ou être plus généralistes. Cheng et Atlee (2007) identifient une partie de ces méthodes, mais ne mentionnent pas l'activité de rédaction des exigences, et ne mentionnent la langue naturelle que comme préalable à une formalisation/modélisation.

De la même manière qu'un système peut être vu de différents points de vue (du point de vue architectural, de la sûreté, de la maintenance, contrôle commande...), on peut également voir les exigences de différents points de vue. Par exemple, la méthodologie Goal-Oriented Requirement Engineering (GORE, van Lamsweerde (2001)) est focalisée sur les buts qui sont à l'origine des exigences. D'autres méthodes s'intéressent aux cas d'utilisation et aux scénarios, et sont donc a priori davantage centrées sur les utilisateurs, voir section 2.2.3.1. Comme pour l'ingénierie basée sur les modèles, des travaux (Sabetzadeh et Easterbrook (2005); Perrouin et collab. (2009)) s'intéressent à comment rassembler ces différents points de vue.

Concernant la gestion des exigences, les méthodes que nous avons observées comme étant les plus implantées dans l'industrie concernent la traçabilité des exigences. Torkar et collab. (2012) font une revue des nombreux efforts concernant la traçabilité des exigences. La traçabilité permet de remonter à l'origine des exigences, qu'elles soient sous forme de buts, de cas d'utilisation ou autre. Tracer les exigences permet aussi de déterminer les relations entre une exigence et la façon dont elle est testée, implémentée, validée, etc. Divers outils de traçabilité sont identifiés par Torkar et collab., à Safran, la suite DOORS d'IBM est utilisée.

2.1.1.3 La rédaction des exigences

Nous nous intéressons principalement dans ce travail à la rédaction (ou spécification) des exigences, même si nous allons aussi évoquer d'autres activités parmi celles mentionnées au-dessus. Un certain nombre de travaux se focalisent sur la rédaction d'exigences en utilisant

des modèles (cas d'utilisation Cockburn (2001), modèles de buts van Lamsweerde (2001), statecharts UML Glinz (2002), etc.). Ces travaux portent généralement sur des exigences spécifiant des systèmes logiciels, parce qu'ils ont tendance à être plus facilement modélisable. De plus, même si nous cherchons à explorer comment compléter, voire éventuellement remplacer, les exigences en langue naturelle, nous partons du principe que les exigences, sont, à la base, rédigées en langue naturelle.

Nous nous intéressons donc principalement à la rédaction d'exigences en langue naturelle, et en particulier comment écrire de "bonnes" exigences. Nous allons aussi étudier en priorité le texte de ces exigences, plus que les divers attributs (version, source, criticité, priorité, etc.) qu'elles peuvent avoir.

2.1.1.4 Écrire des exigences non ambiguës

Parmi les qualités que les exigences doivent posséder, il y a l'inambiguïté. L'ambiguïté peut, ironiquement, être définie de manière plus ou moins générale et peut inclure des concepts tels que la clarté ou la précision. Kamsties et Peach (2000) donnent des définitions plus précise de l'ambiguïté et des différentes sortes d'ambiguïtés rencontrées dans le cadre de l'IE.

On peut identifier deux approches pour obtenir des exigences non ambiguës :
- les méthodes qui visent à écrire des exigences non ambiguës (ou au moins le moins possible) dès le départ,
- les méthodes où, en partant d'exigences en texte libre non contraint, on cherche à détecter et résoudre les ambiguïtés.

La première catégorie implique de contraindre la langue utilisable d'une manière ou d'une autre, et n'est donc pas très adaptée dans le cas où les exigences sont rédigées par diverses parties prenantes, puisque cela forcerait toutes ces parties prenantes à apprendre les règles de rédaction. En revanche, dans le cas où les exigences proviennent principalement d'autres exigences (telles que celles spécifiant un sur-système), il y a potentiellement moins de personnes qui rédigent les exigences et il est donc plus facile d'imposer des règles et/ou un style particulier.

Shah et Jinwala (2015) font une revue des travaux et des outils visant à diminuer l'ambiguïté des exigences en langue naturelle. Nous détaillons certain de ces travaux et méthodes dans les sections 2.1.4 et 2.3.

Bijan et collab. (2013) notent que malgré les "nombreuses listes qui définissent ce qu'est une bonne exigence [...], comment formuler une exigence semble flou". Selon eux, le problème de l'ambiguïté n'est pas assez traité ou bien traité dans la littérature, et nécessite "plus qu'un simple contrôle des mots utilisés [dans les exigences]" pour être résolu.

Les méthodes cherchant à obtenir des exigences non ambiguës directement lors de la rédaction peuvent essentiellement être regroupées dans le cadre des langues contrôlées. On va mentionner ces différentes méthodes dans la suite, notamment avec les langues contrôlées "techniques", les langues contrôlées "basées sur la logique" et les gabarits.

2.1.2 Contexte industriel

2.1.2.1 Systèmes incluant des parties physiques et logicielles

L'ingénierie des exigences a été développée en premier lieu pour les systèmes logiciels. Une grande partie de la littérature scientifique d'IE concerne donc ce type de systèmes. Cependant, notre travail se focalise sur les systèmes physiques, ce qui amène des différences significatives. En particulier, les approches complètement formelles, telles que la "transition axiom method" Lamport (1989), la méthode "Event-B" Abrial (2010) ou RSML Whalen (2000), ne sont pas bien adaptées aux systèmes physiques. Le comportement de ces systèmes est moins prédictible et moins formalisable que le comportement des systèmes logiciels.

Ceci dit, l'IE des systèmes logiciels est une importante source d'inspiration. Par exemple, l'outil Stimulus Jeannet et Gaucher (2015), développé par la société Argosim, vise à améliorer les exigences temps réel pour les logiciels embarqués en simulant ces exigences. L'approche Stimulus suppose que ces exigences puissent être exprimées de manière complètement formelle. Cette hypothèse n'est pas vérifiée pour le type de système sur lesquels nous travaillons. Néanmoins, cette approche et celle que nous proposons se basent sur des principe assez similaires. En particulier, l'une et l'autre visent à développer les exigences et les modèles (tels que les modèles d'architecture ou les machines à états) en parallèle.

2.1.2.2 IE des systèmes critiques

Quelles sont les spécificités de l'IE pour les systèmes critiques, en comparaison avec d'autres systèmes ? Selon Guillerm et collab. (2010), la mauvaise compréhension des exigences ou de mauvaises exigences ont été la cause de catastrophes importantes. Ce fait impose de donner une attention particulière aux exigences des systèmes critiques, et cette attention est imposée par les autorités de certification dans les industries pertinentes (telles que le nucléaire ou l'aéronautique). Ces autorités de certification s'intéressent notamment à la qualité des exigences et à leur traçabilité.

Les spécifications de systèmes critiques sont également dotées d'un type particulier d'exigence, les exigences de sûreté. Ces exigences définissent généralement une probabilité maximale qu'un événement indésirable survienne. Ces exigences de sûreté découlent d'études de sûreté au niveau supérieur et vont également nourrir des études de sûreté de plus bas niveau. Elles vont avoir une influence importante sur l'architecture du système, par exemple parce qu'il peut être nécessaire d'inclure des redondances dans le système pour atteindre les probabilités fixées.

Exigences et études de sûreté dépendent les unes des autres, et selon Vilela et collab. (2017), une meilleure intégration de l'IE et des analyses de sûreté est désirable, que ce soit aux niveaux des termes, des notations, des modèles, des méthodes ou des outils utilisés. Ils notent aussi que l'utilisation de la langue naturelle rend complexe et long le traitement des exigences.

2.1.2.3 Des exigences non statiques

Les praticiens d'IE reconnaissent que l'évolution des exigences, après qu'elles aient été spécifiées, est inévitable (Pohl (2010); Badreau et Boulanger (2014); Nuseibeh et Easterbrook (2000)). Cela peut être dû au fait que les besoins des parties prenantes évoluent (changent et/ou se précisent), mais aussi à cause de facteurs internes à la spécification : si l'on trouve des erreurs, des contradictions, etc. dans les exigences, il est nécessaire de les corriger et donc de les changer. Li et collab. (2012) définissent l'évolution des exigences de la manière suivante : "Requirements evolution is a process of continuous change of requirements in a certain direction". (On note que la revue systématique de Li et collab. traite principalement des exigences logicielles, nous n'avons pas trouvé d'articles se focalisant spécifiquement sur l'évolution des systèmes non logiciels.)

Puisque l'évolution des exigences est inévitable, il est nécessaire de s'y préparer et d'avoir à disposition des outils permettant d'analyser les impacts des modifications, de décider si une modification est acceptée, d'implémenter ces modifications et d'obtenir des métriques relatives à l'évolution. La présence de liens de traçabilité est centrale pour permettre l'analyse d'impact et l'implémentation des modifications, mais n'est pas suffisante pour assurer ces activités. Toutes ces activités peuvent être réalisées lorsque les exigences sont rédigées en langue naturelle, mais cela les rend d'autant plus longues et complexes.

2.1.3 Problèmes identifiés par les industriels

Nous avons discuté avec des ingénieurs de Safran dans le cadre de cette thèse, et de ces discussions est ressorti un ensemble de problèmes que ces ingénieurs ont avec l'IE telle qu'elle est pratiquée aujourd'hui à Safran. Il est intéressant de noter que ces lacunes ne sont pas nécessairement les mêmes que celles identifiées dans la littérature scientifique.

2.1.3.1 Nombre d'exigences et niveau d'abstraction

Un certain nombre de qualités que l'on demande aux exigences sont contradictoires. Par exemple, il est facile de trouver dans une spécification des exigences qui ne sont pas "atomiques" et de les séparer chacune en plusieurs exigences pour qu'elles le soient. Cependant, cela augmente le nombre d'exigences, ce qui crée une surcharge de travail importante. Une des préoccupations des ingénieurs est donc de savoir comment limiter le nombre d'exigences, tout en écrivant des exigences qui soient facilement lisibles et traçables, et d'un bon niveau d'abstraction.

Un autre problème est qu'il n'est pratiquement pas possible de spécifier complètement et formellement les limites de l'espace de design pour un système, le nombre de dimensions de cet espace étant trop important. Des systèmes continuent d'être conçus et de fonctionner dans le monde réel parce que ce qui n'est pas précisé dans la spécification est complété, éventuellement de manière inconsciente, par les concepteurs. Au delà du problème d'interprétation et d'ambiguïté que cela crée, il y a aussi la question de quand s'arrêter pour le rédacteur d'une spécification : puisqu'il ne peut et ne doit pas spécifier de manière complète et complètement formelle, il faut se demander jusqu'où aller dans la formalisation, et quels

sont les endroits où il est possible d'écrire des exigences qui ne soient pas formelles tout en n'étant pas (trop) ambiguës.

2.1.3.2 Comment écrire des exigences non ambiguës ?

Liée au point précédent est la question de ce qu'est l'ambiguïté. Si les langages formels ne sont généralement pas ambigus, on ne peut pas restreindre l'absence d'ambiguïté à l'utilisation de langage formels. Il doit y avoir des moyens de communiquer qui soient plus pratiques que l'utilisation de langues formelles tout en étant raisonnablement confiant que émetteur et récepteur comprennent la même chose. Il est donc nécessaire d'explorer en détail ce qu'est l'ambiguïté et quand est-ce qu'elle est nocive (Chantree (2006)).

À partir de cette étude sur l'ambiguïté, on peut se demander comment écrire des exigences qui soient non ambiguës. Est-ce que l'utilisation de gabarits permet de réaliser cela ? Si on standardise les exigences, comment le faire, et jusqu'à quel point ?

2.1.3.3 Comment vérifier des ensembles d'exigences ?

Un problème qui ne touche évidemment pas seulement Safran est la question de comment vérifier des critères de cohérence et de complétude sur un ensemble d'exigences. Au delà de la vérification en elle-même, ce problème va fortement dépendre de la façon dont les exigences sont rédigées, en particulier, ce sera beaucoup plus complexe avec des exigences rédigées en langue naturelle.

2.1.3.4 Comment intégrer IE et ingénierie basée sur les modèles ?

L'ingénierie basée sur les modèles est de plus en plus présente à Safran, mais son utilité n'est pas toujours claire dans certains cas. Pour des exigences rédigées uniquement en langue naturelle, l'ingénierie basée sur les modèles ne facilite pas particulièrement les processus d'IE : par exemple, s'il est nécessaire de traduire des modèles en exigences textuelles, cela limite l'utilité de l'IE et de l'ingénierie basée sur les modèles. Une question posée est donc comment mieux intégrer ces deux domaines.

2.1.3.5 Quantifier la qualité des exigences

Si l'on souhaite avoir de "meilleures" exigences, il est nécessaire de définir des métriques permettant de, par exemple, noter la qualité des exigences ou des groupes d'exigences. Ce problème, également identifié par Bijan et collab. (2013), est a priori difficile à résoudre complètement et automatiquement (personne n'imagine que l'on puisse donner deux exigences à un logiciels, et qu'il produise une note unique pour chaque exigence, permettant de dire que l'une est absolument meilleure que l'autre). Cependant, nous pensons qu'il est possible de faire mieux que la situation actuelle, où ces métriques n'existent (pratiquement) pas. Safran et d'autres industriels ont lancé des groupes de travail tels que CESAR (Rajan et Wahl (2013)), afin (entre autres choses) de pouvoir définir des critères pertinent pour les exigences, ainsi que des moyens de vérifier et de quantifier ces critères.

2.1.4 Solutions non basées sur les modèles

2.1.4.1 Listes de critères

On trouve, dans différentes sources, un certain nombre de listes de qualités que devraient avoir les exigences pour être de bonnes exigences (par exemple, Pohl (2010); Badreau et Boulanger (2014); ISO/IEC/IEEE (2011); Rajan et Wahl (2013); Saavedra et collab. (2013)). Chaque qualité de ces listes se présente sous forme d'une sorte de conseil, tel que :

— Les exigences doivent être relativement courtes pour pouvoir être facilement compréhensible.
— Le vocabulaire et les constructions syntaxiques potentiellement ambigus doivent être évités.
— Une exigence ne doit pas contredire une autre exigence.

Ces critères de qualité peuvent avoir pour but de faciliter la communication (en écrivant des exigences moins ambiguës, plus claires, plus courtes, etc.), et d'autres découlent de la nécessité de de gérer les exigences (par exemple, les exigences doivent avoir un identifiant unique). De manière générale, nous n'avons que très rarement trouvé des justifications pour ces listes de critères.

2.1.4.2 Évaluation de ces critères

Saavedra et collab. (2013) proposent une analyse des différents critères et des méthodes pour les évaluer. Dans cette approche de correction des exigences une fois rédigées, on trouve des méthodes d'inspections, basées sur des "checklists" (avec des items comme "l'exigence est-elle ambiguë/atomique/etc. ?"), qui sont relativement simples à mettre en œuvre mais qui peuvent être longues et ne pas donner de très bons résultats. Des méthodes de Traitement Automatique des Langues (TAL) peuvent être utilisées pour automatiser une partie des évaluations. Assez rarement, ces méthodes sont uniquement basées sur des méthodes de TAL "classiques" et n'intègrent pas de connaissances du domaine et de l'IE a priori, par exemple en se basant sur des indicateurs de lisibilité tels que la formule de Flesch–Kincaid (Kincaid et collab. (1975)).

Kamsties et Peach (2000) avancent que la plupart des ambiguïtés nocives dépendent du cadre de l'IE et du domaine, et donc que ces méthodes généralistes vont souvent relever des fausses alertes, i.e. des textes qui paraissent ambigus pour un ordinateur mais qui ne le seront pas pour des lecteurs humains connaissant le contexte. D'autres travaux cherchent justement à intégrer ce contexte dans les méthodes de TAL, par exemple au moyen d'ontologies (Körner et Brumm (2009); Kamsties et Peach (2000); Kaiya et Saeki (2005)). Les résultats sont effectivement meilleurs, mais nécessitent, comme tous les systèmes experts, d'obtenir et de gérer une base de connaissance pertinente. Nous revenons sur les ontologies dans la section 2.3.4.

Il n'existe pas a priori de contre-indications théoriques à l'application directes de méthodes de TAL et/ou d'apprentissage automatique sur des corpus d'exigences (voir Parra et collab. (2015); Ormandjieva et collab. (2007) par exemple). Il est cependant difficile en pratique d'obtenir un nombre suffisant d'exigences ou de corpus d'exigences, et

encore davantage si l'apprentissage est supervisé (i.e. les exigences doivent être annotées).

2.1.4.3 Gabarits

Une piste pour améliorer la qualité des exigences dès la rédaction est l'utilisation de gabarits (appelés "templates" ou "boilerplates" en anglais) Mavin et collab. (2009); Fraga et collab. (2015). Un exemple typique de gabarit est "Le <*système*> doit <*faire quelque chose*> avec <*un niveau de performance donné*> dans <*un contexte donné*>". Après avoir choisi un gabarit adéquat, le rédacteur de l'exigence rempli les blancs avec les informations pertinentes. Par exemple "L'<ABS> doit <relâcher la pression de freinage> en <moins de 100ms> lorsqu'il y a <un risque de blocage des roues>".

Les livres de fondamentaux de l'IE mentionnent souvent les gabarits (Pohl (2010); Badreau et Boulanger (2014)). Mais ils avertissent aussi le lecteur de leurs limitations. Ils conseillent généralement de ne pas rendre l'utilisation de gabarits obligatoire. Le principal problème de ces gabarits est qu'ils tendent à être trop "rigides" : ils empêchent le rédacteur d'exprimer ce qu'il souhaite réellement. Avoir beaucoup de gabarits à disposition permet de donner plus de possibilités d'expression, mais rend la gestion et l'utilisation de ces gabarits plus complexe. D'un autre coté, utiliser peu de gabarits mais qui couvrent plus de cas est moins efficace pour guider la rédaction et réduire l'ambiguïté des exigences.

2.1.4.4 Langues contrôlées

Les gabarits visent à restreindre la langue naturelle. On peut ainsi les considérer comme des Langues Naturelles Contrôlées (LNC) Kuhn (2014). On peut distinguer deux types de langues naturelles contrôlées : les LNC "techniques", dont le but est de faciliter la communication entre humains (par exemple le Simplified Technical English, STEMG (2017)), et les LNC "basées sur la logique", qui visent à améliorer la communication entre humains et ordinateurs (par exemple l'Attempto Controlled English Fuchs et Schwitter (1996)). Un des objectifs de notre travail est de définir un langage d'exigence qui soit :

— moins ambigu que la langue naturelle,
— mais suffisamment proche de celle-ci pour que son utilisation ne demande pas une formation trop importante.

C'est aussi le but des LNC. Nous n'avons cependant pas trouvé une LNC qui puisse être utilisée directement pour les spécifications que nous avons étudié. Les LNC "basées sur la logique" sont essentiellement des langues formelles cachées derrière des mots du langage naturel. Dans notre contexte, nous considérons que les exigences ne sont pas susceptibles d'être complètement formalisées. Nous voulons cependant être capable d'appliquer des traitements informatisés aux exigences. Cela signifie que ces exigences doivent avoir une structure qui soit facilement compréhensible par un ordinateur, ce qui n'est pas le cas de la langue naturelle et de la plupart des LNC "techniques".

2.1.4.5 LNC “techniques”

En général, les LNC “techniques” prennent le langage naturel comme base, et ajoutent des règles qui retirent des éléments du langage naturel. Comme exemples de règles pour ce type de CNL on peut trouver :

— Do not use slang or jargon words (ASD-STE, STEMG (2017)).
— Paragraph length is limited to 150 words (EasyEnglish, Betts (2003)).
— When equal things are compared, the adjective is preceded and followed by as (Basic English, Ogden (1930)).

La plupart restreignent également le vocabulaire utilisable, par exemple en n'autorisant que l'utilisation des mots appartenant à une liste blanche. De plus, chaque mot de cette liste blanche a un sens associé, et il est interdit d'utiliser ce mot avec un autre sens.

Même si ces CNL contraignent la langue naturelle, elles sont quand même basées sur cette dernière, et ne sont pas descriptible de manière complète et exacte, contrairement à, par exemple, la logique du premier ordre. Ce fait implique qu'il est difficile d'interpréter automatiquement ces langages.

2.2 Ingénierie basée sur les modèles

Comme pour l'ingénierie des exigences, l'ingénierie basée sur les modèles, comme discipline d'ingénierie système, s'est beaucoup appuyée sur les travaux réalisés en ingénierie logicielle. Il est intéressant de se demander pourquoi, par exemple, SysML est inspiré d'UML, alors que l'on construit des systèmes depuis plus longtemps que l'on écrit des logiciels...

Toujours est-il que le but de l'ingénierie basée sur les modèles est de placer les modèles dans un rôle central des processus de conception, ce rôle étant “traditionellement” occupé par des documents rédigés en langue naturelle.

2.2.1 Principaux concepts

Nous présentons dans cette section quelques concepts centraux de l'ingénierie basée sur les modèles (MBSE).

2.2.1.1 Qu'est-ce qu'un système ?

De manière générale, un système est un objet composé de plusieurs parts qui interagissent (on peut par exemple donner la définition de Hall et Fagen (1956) : “a system is a set of objects together with relationships between the objects and between their attributes”). Cette définition est valide pour des systèmes pouvant être très variés et hors du cadre de notre étude (systèmes biologiques, sociaux, économiques, etc.). Nous nous intéressons au cadre plus restreint de l'ingénierie système, où les systèmes d'intérêt sont conçus dans un but particulier.

Dans ce cadre, les systèmes sont précisés par l'existence de ce but : les systèmes sont

conçus afin d'obtenir une fonction qui ne peut pas être obtenue par les composants du système individuellement. Les définitions pour l'ingénierie système (Cameron et collab. (2016); Micouin (2014); INCOSE)) reflètent ce fait. Une autre distinction avec les systèmes qui n'ont pas été conçus (systèmes naturels ou émergents) est qu'il est généralement plus facile d'identifier les composants des systèmes conçus. L'agencement de ces composants, l'*architecture* du système, va être un élément central des processus d'ingénierie système.

Ingénierie système L'ingénierie système est l'étude de la conception des systèmes. L'ingénierie système est une discipline scientifique, appliquée de manière très répandue en milieu industriel. On peut identifier un certain nombre d'étapes à réaliser lors de la conception d'un système, étapes qui peuvent être organisées selon différents processus (modèle en cascade, cycle en V, méthodes agiles, etc.). Cameron et collab. (2016) identifient quatre groupes d'activités lors du developpement d'un système générique :

— "Conceive", avec des activités se focalisant sur la définition de la mission du système et la définition de haut niveau du concept du système ("Business strategy", "Customer needs", "Regulation"...).
— "Design", où les activités visent à obtenir un design plus détaillé du système ("Requirements definition", "Interface control", "Failure and contingency analysis"...).
— "Implement", rassemble les activités où l'on crée effectivement les éléments du système, où on les teste et les intègre ("Element implementation", "Product integration", "Delivery"...).
— Les activités "Operate" se concentrent sur la suite du cycle de vie et sur l'évolution du système ("Operations", "Maintenance, repair, overhaul", "Upgrades"...).

La rédaction des exigences fait partie du groupe "Design".

2.2.1.2 Qu'est-ce qu'un modèle ?

Les modèles dont on va parler dans le cadre de l'ingénierie système peuvent être de plusieurs sortes :

— Des modèles qui servent à faciliter la communication entre humains.
— Des modèles qui permettent de faire des calculs, par exemple en simulant ces modèles.
— Éventuellement, des modèles qui permettent de générer automatiquement du code logiciel.

Les modèles du premier type n'ont pas besoin d'être formels, même s'ils sont écrits avec des notations standardisées. Les autres en revanche nécessitent l'utilisation de langages de modélisation formels.

Tous ces modèles vont être des abstractions de la réalité. Vu la complexité de la réalité, ces abstractions constituent généralement un point de vue (ou une partie d'un point de vue) du réel.

2.2.1.3 Qu'est-ce que l'ingénierie basée sur les modèles (MBSE) ?

On utilise des modèles depuis très longtemps lors de la conception de systèmes : par exemple, les lois physiques telles que les lois du mouvement de Newton, les lois du contact de

Hertz ou la loi d'Ohm sont des modèles. Le problème est d'utiliser ces modèles de manière cohérente et homogène lors de la conception de systèmes complexes, sans nécessairement avoir à se baser sur des documents textuels comme colonne vertébrale de ce processus de conception.

L'utilisation de ces documents textuels rédigés en langue naturelle présente un certain nombre d'inconvénients, similaires à ceux que l'on détaille dans ce rapport pour l'ingénierie des exigences en particulier. Le but du MBSE est d'éviter ces inconvénients, mais évidemment, ces méthodologies ont aussi leurs défauts. Vogelsang et collab. (2017) donnent certaines des raisons qui freinent ou soutiennent l'adoption du MBSE.

Selon Friedenthal et collab. (2007), "Model-based systems engineering (MBSE) is the formalized application of modeling to support system requirements, design, analysis, verification and validation, beginning in the conceptual design phase and continuing throughout development and later life cycle phases". "L'ingénierie basée sur les modèles est l'application formalisée de la modélisation pour supporter les exigences, le design, l'analyse, la vérification et la validation des systèmes, en commençant lors de la phase de design conceptuel et en continuant tout le long du développement et des phases de vies suivantes."

2.2.1.4 Processus de MBSE

Estefan et collab. (2007) identifient des méthodologies de MBSE utilisées en industrie. De la même manière qu'il existe différents points de vue sur un système, on peut voir le processus de conception de différents points de vue. Par exemple, la méthodologie "State Analysis" (SA, Ingham et collab. (2004)) est focalisée sur les états du système à concevoir. Le principe n'est pas de réduire le système à seulement ses états et de ne se préoccuper que des états lors de la conception, mais plutôt de s'intéresser d'abord aux états, et de s'en servir comme fil rouge lors de la conception.

L'Object-Process Methodology (OPM, Dori (2011)) utilise des concepts d'*objets* (quelque chose qui existe physiquement), pouvant avoir différents *états* et transformés par des *processus*, pour décrire des systèmes très variés. À partir de ces trois entités élémentaires, dotées de représentations graphiques, un ensemble d'opérations sont disponibles pour raffiner ou abstraire les modèles représentant le système à concevoir. Une particularité de cette méthodologie est que l'on cherche à garder un seul modèle intégré, unique, pour conserver les informations.

L'approche alternative consiste à développer (de manière plus ou moins indépendente) les vues du systèmes dans des modèles distincts. Les méthodologies utilisant SysML (Friedenthal et collab. (2014)) vont plutôt utiliser cette approche. L'inconvénient d'avoir des modèles distincts pour représenter les vues du système est qu'il n'est pas garanti que ces modèles coïncident quand on cherche à les rassembler. D'un autre coté, les points de vue pertinents lors de la conception d'un système peuvent être très différents et essayer d'obtenir un modèle unique faisant appel à des principes, des modèles, des domaines très hétérogènes est complexe.

2.2.2 MBSE : pourquoi et pourquoi pas ?

Il est difficile de réaliser des études expérimentales où l'on pourrait comparer formellement, par exemple avec un groupe témoin, les apports des méthodes de MBSE. Heureusement, les retours d'expériences, les entretiens avec les utilisateurs, et simplement l'avancement actuel des méthodologies MBSE dans l'industrie (Vogelsang et collab. (2017); Bone et Cloutier (2010); Hutchinson et collab. (2014)), permettent d'avoir des indications sur les avantages et les inconvénients du MBSE.

2.2.2.1 Inconvénients

Un certain nombre d'inconvénients ne sont pas spécifiquement liés à la nature de l'ingénierie basée sur les modèles, mais découlent plutôt du fait qu'un changement rencontre toujours une certaine inertie (Vogelsang et collab. (2017)). Par exemple, malgré les défauts des méthodes "document-centric" on continue de développer des systèmes de cette manière, et elles sont vues comme "marchant suffisamment bien" alors que l'adoption du MBSE peut être vue comme un saut dans l'inconnu. L'introduction de nouvelles méthodologies implique nécessairement un changement de point de vue et d'organisation, ce qui a tendance a être résisté.

Il y a également des inconvénients spécifiques du MBSE, tel que le besoin d'outils spécifiques, et généralement incompatibles avec les outils déjà utilisés. L'utilisation de modèles demande également une façon de penser particulière, qui n'est pas forcément naturelle pour les utilisateurs, même après une formation. Les méthodologies MBSE peuvent également être vues comme étant immatures.

Une autre difficulté de l'ingénierie basée sur les modèles concerne la cohabitation entre les différents types de modèles : il existe différents types de modèles pour chaque discipline impliquée dans le processus de conception (mécanique, informatique, sûreté de fonctionnement, etc.). Ces différents modèles sont souvent développés indépendamment, ce qui rend leur intégration difficile : par exemple, il n'est pas garanti qu'un élément d'un modèle corresponde à un et un seul élément dans un autre modèle.

Différentes approches sont développées pour essayer de résoudre ce problème, comme faciliter la comparaison entre modèles (Legendre et collab. (2017) par ex.) ou encourager le développement et le maintien d'un modèle unique (Dori (2011)). Les interactions entre modèles et avec d'autres éléments restent une question complexe.

2.2.2.2 Avantages du MBSE

Évidemment, puisque c'est le but, ou au moins un des buts, de l'ingénierie basée sur les modèles, la gestion de la complexité grandissante des systèmes est un facteur déterminant pour l'adoption de ces méthodologies. L'ingénierie basée sur les modèles peut améliorer la qualité des produits et des spécifications (Hutchinson et collab. (2014)). Cependant, et surtout pour les premiers projets réalisés en MBSE, il n'y a pas nécessairement de gains de temps et d'efforts. L'abstraction réalisée en modélisant permet aux ingénieurs d'avoir une meilleure vision des systèmes et des produits, et ainsi facilite la réutilisation

et la modularisation de leur travail. Le MBSE va permettre et/ou nécessiter de détecter des potentiels problèmes plus tôt dans le processus de conception, ce qui facilite cette conception.

2.2.2.3 État actuel dans l'industrie

L'utilisation de l'ingénierie basée sur les modèles est très inégale selon les industries. Selon le sondage réalisé par Bone et Cloutier (2010), SysML est principalement utilisé dans les industries aéronautique, spatiale et de la défense. Une raison possible est que le MBSE est nécessaire pour répondre aux demandes des autorités de certification (Vogelsang et collab. (2017)). Toujours selon Bone et Cloutier (2010), SysML est principalement utilisé dans des projets de grande taille, ou dans des projets plus petits, mais qui ont vocation à être intégrés dans des projets plus importants. La répartition de l'utilisation de méthodologies MBSE parmi les différentes disciplines d'un même projet est encore une fois inégale : si les ingénieurs systèmes font beaucoup d'ingénierie basée sur les modèles, ce n'est pas nécessairement le cas des ingénieurs de test.

2.2.3 Modèles, d'informel à formel

Nous présentons quelques types de modèles utilisés en MBSE et qui sont en rapport avec l'ingénierie des exigences.

2.2.3.1 Modèles pour communiquer

Un certain nombre de modèles sont utilisés uniquement pour communiquer entre humains et ne permettent pas de faire des calculs, d'être simulés ou de générer automatiquement du code. Une partie des diagrammes de SysML, en tout cas dans leur version basique, sont de ce type.

Cas d'utilisation Les cas d'utilisation (Jacobson (1993)) sont un type de modèle très répandu, y compris en dehors de toute méthodologie formelle d'ingénierie basée sur les modèles ou d'ingénierie des exigences. Ces cas d'utilisation ne sont d'ailleurs pas nécessairement sous forme de modèles, et peuvent simplement être écrit en texte. Leur principe est de décrire les interactions d'un système avec des acteurs externes, afin de réaliser un but.

Ces cas d'utilisation permettent d'imaginer le système tel qu'il sera une fois implémenté, ses interactions avec les acteurs externes, comme les utilisateurs du système par exemple, et de déduire de ces interactions des besoins sur le système. Une fois le système fini, on peut ensuite utiliser un cas d'utilisation comme test, afin de vérifier que les besoins sont bien remplis par le système.

Les cas d'utilisation nous intéressent car ils vont être à l'origine d'exigences. De nombreux articles s'intéressent aux liens entre les cas d'utilisation/scénarios et l'ingénierie des exigences (Somé (2006); Regnell et collab. (1995); Issad et collab. (2015)). Les cas d'utilisation ne permettent cependant pas de trouver toutes les exigences pertinentes pour

le système (comme les exigences non-fonctionnelles par exemple), et d'autres méthodes complémentaires doivent être utilisées.

Diagramme d'exigence Un des diagrammes de SysML est le diagramme d'exigence. Ce diagramme permet de représenter des exigences et leurs relations avec d'autres exigences et avec d'autres éléments. Les exigences sont représentée par leur texte, inclus dans un rectangle. Par exemple, si une exigence est dérivée d'une autre exigence, on trace une relation "derive" de la première exigence à la seconde. De même, si un composant satisfait une exigence, on trace une relation "satisfy" entre l'exigence et le composant. Selon Scanniello et collab. (2014), l'utilisation d'un diagramme d'exigence permet de simplifier la compréhension des lecteurs.

Les diagrammes d'exigences ne se focalisent pas sur le texte des exigences et tendent à considérer les exigences comme des blocs atomiques. Cependant, puisque nous nous intéressons à la rédaction des exigences, nous cherchons à étudier en détail ce qui apparaît dans le texte. Nous n'avons pas observé d'utilisations de diagrammes des exigences à Safran, peut-être parce que la taille des spécifications traitées (>100 exigences) rend l'utilisation de diagrammes peu pratique (dans Scanniello et collab. (2014), les spécifications utilisées comprennent une vingtaine d'exigences).

IBD et BDD Dans SysML, les diagrames de blocs internes (IBD) et de définition des blocs (BDD) sont les principaux diagrammes permettant de définir la structure d'un système. Le BDD décrit les relations logiques (composition, association, spécialisation) entre blocs, alors que l'IBD décrit la structure d'un système en donnant les blocs qui le composent et en représentant les connexions entre ces blocs. On ne va pas beaucoup se préoccuper d'où viennent ces blocs (association, spécialisation), par contre, la partie structurelle (composition, connections) des systèmes, l'architecture, nous intéresse plus.

Un intérêt de tels diagrammes de blocs est la possibilité de raffiner de plus en plus ces blocs : on peut par exemple partir d'un système, définir et délimiter des sous-systèmes dans un IBD, puis changer de point de vue, considérer un sous-système comme l'objet d'intérêt, et définir ses propres sous-systèmes dans un nouvel IBD, et ainsi de suite. Cette décomposition petit à petit suit assez bien le processus de conception tel qu'il est réalisé dans l'industrie. De plus, si les modèles d'architecture consistent simplement en des boîtes et des flêches, alors ils sont très simples à comprendre et n'ont pas besoin de beaucoup de formation [1]. Probablement pour ces raisons, les diagrammes de types IBD sont très utilisés dans l'industrie.

2.2.3.2 Formalisations de modèles SysML

De façon relativement similaire à la langue naturelle, les diagrammes SysML sont assez répandus, mais ne sont pas nécessairement formels. Des travaux proposent donc d'étendre, de modifier ou d'adapter certains des diagrammes SysML afin qu'ils puissent être utilisés

1. Selon Bone et Cloutier (2010), le sens des "ports" est cependant peu clair pour une partie des utilisateurs.

lors d'opérations informatiques. Par exemple, Knorreck et collab. (2011) présentent un langage permettant de représenter des propriétés logiques et temps réel avec des diagrammes paramétriques. Dans le domaine de la sûreté de fonctionnement, Mhenni et collab. (2014) proposent une méthode pour générer des artéfacts utilisés en sûreté, comme des arbres de défaillances, à partir de diagrammes de blocs internes.

À cause du succés, au moins dans la communauté scientifique d'ingénierie basée sur les modèles, de SysML, de nombreux travaux visant à utiliser des diagrammes SysML pour réaliser des calculs dans des domaines spécialisés (sûreté, mécanique, électrique, etc.) sont réalisés. Puisque les diagrammes SysML n'ont généralement pas été conçus pour ces besoins, il est nécessaire de les adapter. D'autres travaux (par ex. Prosvirnova et collab. (2013)) visent au contraire à créer leurs propres formalismes afin d'utiliser les méthodes de MBSE sans être dépendant des diagrammes SysML.

2.2.3.3 Ontologies

Une ontologie, dans le contexte des sciences de l'information, représente des concepts et les relations entre ces concepts. Une ontologie concerne généralement une partie spécifique du monde réel (un domaine). Dans un langage d'ontologie tel que OWL (Bechhofer et collab. (2009)), on peut définir des individus, des classes et des propriétés permettant de lier les individus entre eux. À partir de ces éléments relativement simples, il est possible de représenter des connaissances, de n'importe quel type, dans n'importe quel domaine.

Cette approche se rapproche de l'approche symbolique de l'intelligence artificielle, i.e. les méthodes visant à créer une intelligence artificielle à partir de règles de haut niveau, lisibles par des humains. Si les espoirs de créer une intelligence générale à partir de ces types de règles ont été pratiquement abandonnés, des ontologies dans des domaines plus restreints peuvent être créées et utilisées dans le monde réel. Il reste que la création et le maintien de ces ontologies, s'ils sont fait par des humains, est assez complexe (le problème de la "knowledge acquisition").

2.2.3.4 Langages formels

Un langage formel possède un alphabet, qui est un ensemble de symboles, et un ensemble de règles qui régissent la concaténation de ces symboles en mots. Le langage formel lui-même est l'ensemble des mots, formés des symboles de l'alphabet, qui respectent les règles de formation (la grammaire). Les langages formels sont généralement créés de manière à ce qu'ils soient non ambigus : par exemple, deux éléments qui ont le même symbole sont effectivement le même élément, et un mot peut être décomposé de manière unique en suivant les règles de grammaire. Cependant, ces langages ne sont généralement pas créés pour communiquer entre humains, sauf certains cas spécifiques tel que Lojban (Cowan (1997)).

Certains langages formels peuvent être rapprochés des ontologies. Par exemple, un programme Prolog (Clocksin et Mellish (2003)) est "composé d'une suite de faits et de relations entre ces faits" (Gal et collab. (1989)).

Beaucoup de choses peuvent être considérées comme des langages formels, comme les

expressions arithmétiques. Nous nous intéressons aux langages formels qui permettraient de faciliter l'ingénierie des exigences : les caractéristiques de non ambiguïté et le fait que ces langages peuvent être "compris" par des ordinateurs sont intéressants dans notre cadre. La méthode B (Abrial (2010)) ou RSML (Whalen (2000)) sont des exemples de ces langages formels de spécification. Ces langages se focalisent cependant sur les spécifications de systèmes logiciels.

2.3 Liens entre IE et MBSE

2.3.1 LNC "basées sur la logique"

Un défaut des langages formels est que peu de gens savent les lire, et encore moins savent les écrire. Le principe des langues naturelles contrôlées "basées sur la logique" est de faciliter cette lecture/écriture en faisant correspondre des mots du langage naturel aux éléments du langage formel. L'Attempto Controlled English (ACE, Fuchs et Schwitter (1996)) est un exemple de ce type de langue contrôlée.

Si les langues contrôlées comme l'ACE sont parfaitement lisibles pour des personnes sachant lire la langue naturelle correspondante (l'anglais pour l'ACE par exemple), ce n'est pas aussi facile pour l'écriture. Comme l'indiquent Fuchs et Schwitter eux-mêmes, l'ACE est un langage qu'il faut apprendre. Nous avons trouvé peu d'articles donnant des résultats expérimentaux sur la durée et la difficulté pour apprendre des langues contrôlées (certaines évaluations dans le cadre de la rédaction d'ontologies sont présentées par Safwat et Davis (2014)).

Même une fois apprises, puisque ces LNC sont basées sur un langage formel, elles ont les mêmes contraintes que ce langage. L'expressivité des utilisateurs peut donc être assez limitée en comparaison du langage naturel.

2.3.2 De la langue naturelle aux modèles

De manière générale, la génération automatique de modèles en partant de textes en langue naturelle fait partie de la discipline du Traitement Automatique des Langues (TAL). Dans le cadre plus restreint de l'ingénierie des exigences, des travaux de Almeida Ferreira et da Silva (2009); Gervasi et Nuseibeh (2002); Cabral et Sampaio (2008) visent à traduire automatiquement des exigences en langue naturelle en modèles pertinent en utilisant des méthodes de TAL.

Ces travaux ne sont pas applicables dans tous les contextes, notamment parce que les textes d'origine ne sont pas en langue naturelle libre. Par exemple, de Almeida Ferreira et da Silva (2009); Cabral et Sampaio (2008) se basent sur des langues naturelles contrôlées, et les exigences de Gervasi et Nuseibeh (2002) étaient "remarquablement structurées et régulières, et utilisaient un langage très précis sur un vocabulaire bien défini".

2.3.3 Des modèles à la langue naturelle

A l'inverse, on peut vouloir utiliser directement des modèles lors de la conception, et, quand c'est nécessaire, générer automatiquement des exigences en langue naturelle à partir de ces modèles. Nicolás et Toval (2009) présentent une revue de ces approches. La génération automatique d'exigences a un certain nombre d'avantages :

— Les concepteurs peuvent travailler directement sur les modèles qu'ils connaissent, et générer automatiquement des exigences en texte pour les parties prenantes ne connaissant pas ces modèles. Cela facilite une ingénierie basée sur les modèles, tout en n'aliénant pas les diverses parties prenantes.

— La qualité des exigences est améliorée : par exemple, une exigence générée automatiquement ne contiendra a priori pas de fautes d'ortographes. De plus les modèles ont des avantages sur les descriptions en langue naturelle (sinon on ne les utiliserait pas), par exemple, il va être plus facile de vérifier qu'une interface manque dans un modèle d'architecture que dans une description textuelle. Les ensembles d'exigences générés vont donc a priori être plus complets, cohérents et corrects.

— Ces exigences peuvent éventuellement être générée à la demande, ce qui permet d'avoir des exigences à jour par rapport aux modèles utilisés. Il pourrait également être possible de générer des exigences spécifiques pour les différentes parties prenantes, en n'incluant que les éléments pertinents pour ces parties prenantes.

Un point complexe est que les modifications doivent être réalisées directement sur les modèles, et pas sur les exigences textuelles, car les méthodes pour repasser du texte aux modèles sont peu développées pour la langue naturelle non-contrôlée.

Les travaux mentionnés dans la revue de Nicolás et Toval se focalisent principalement sur les systèmes logiciels. Les modèles utilisés sont relativement différents de ceux qui nous intéressent dans notre travail : par exemple, beaucoup de ces travaux partent de modèles de cas d'utilisation et de scénarios. Les exigences qui peuvent être générées automatiquement à partir de ces modèles sont trop peu précises pour les spécifications que nous avons étudiées. Bernard (2012) présente une approche spécifique à l'ingénierie système, utilisant UML et SysML.

De plus, nous essayons également de prendre en compte les exigences qui peuvent difficilement être exprimées sous forme (d'élément) de modèle. Ceci dit, nous sommes convaincus que si une exigence peut être exprimée de manière claire et non-ambiguë en la remplaçant par un élément de modèle, il serait préférable de l'exprimer de cette manière plutôt qu'en utilisant du texte en langue naturelle.

2.3.4 Ontologies pour les exigences

Kaiya et Saeki (2005) cherchent à analyser des spécifications en faisant correspondre les termes des exigences avec les éléments d'une ontologie. Fraga et collab. (2015) proposent de générer des gabarits à partir d'une ontologie, afin de permettre plus de flexibilité. Dans l'étude de cas présentée par Kaindl (2005), même si le mot "ontologie" n'est pas mentionné, l'auteur utilise un "modèle du domaine" pour définir les concepts apparaissant dans les exigences. Un des inconvénients de cette approche est que la création et la gestion de ces ontologies est un travail complexe.

Les ontologies sont le sujet de recherches récentes. Parmi d'autres travaux, nous pouvons citer l'effort de standardisation mené par l'"Object Management Group" (OMG), visant à lier les modèles UML avec le "Web Ontology Language" OWL) Colomb et collab. (2006). Pour éviter les problèmes liés à la création et à la gestion d'ontologies, Körner et Brumm (2009) proposent un outil permettant de faire des requêtes, basées sur un certain nombre de règles, à des ontologies déjà existantes telles que ResearchCyc Cycorp (2006) ou WordNet Miller (1995).

2.3.5 Liens entre exigences et modèles

2.3.5.1 En considérant l'exigence comme une "boîte noire"

D'autres travaux étudient comment lier les exigences avec des artefacts externes tels que des modèles. Dans SysML (Friedenthal et collab. (2014)), il est possible de créer des liens entre une exigence et d'autres éléments. Cependant, SysML traite le texte des exigences comme une boîte noire et ne permet pas de lier une partie précise du texte de l'exigence à un modèle. Par exemple, lorsqu'on écrit "l'ABS devra relâcher la pression de freinage", on souhaite être capable de savoir que les mots "pression de freinage" font référence à un concept spécifique défini hors de l'exigence.

2.3.5.2 En étudiant le texte des exigences

Le Requirement Interchange Format (ReqIF) OMG (2016) et des outils commerciaux d'IE tel que DOORS permettent l'ajout de balises définissant des hyperliens au texte des exigences. Cependant, cette fonction n'est pas forcément beaucoup utilisée dans l'industrie. Nous proposons des scripts utilisant ces liens hypertextes pour permettre des analyses plus efficaces que ce qu'il est possible de faire avec un texte pur.

Dans Kaindl (2005), une méthode appelée RETH (Requirements Engineering Through Hypertext) est appliquée à un projet industriel. Deux aspects principaux sont illustrés : créer des liens entre des scénarios (ou plus précisément entre des actions dans les scénarios) avec les exigences, et lier explicitement des mots dans les exigences à leur définitions. Le premier aspect est similaire aux liens de traçabilité, qui sont maintenant relativement courants dans la recherche académique et l'industrie. Nous sous sommes plutôt intéressés au second aspect. Dans cette étude de cas, les objets du domaine et les concepts sont principalement définis par des paragraphes de texte en langue naturelle. Un de nos buts est de remplacer ces textes en langue naturelle par des modèles.

Chapitre 3

Cadre conceptuel et méthodologique

3.1 Définitions

3.1.1 Ingénierie des exigences

La raison pour laquelle les systèmes sont décomposés en sous-systèmes est que cela permet de garder une complexité raisonnable lors de la conception, différentes équipes pouvant se concentrer sur différents sous-systèmes. Ce but définit les limites entre sous-systèmes : on va décomposer les systèmes selon un point de vue client/fournisseur (qu'ils soient dans la même entreprise ou non). Ce point de vue n'est pas forcément le même qu'un point de vue "fonctionnel" ou même qu'un point de vue "organique".

L'ingénierie des exigences (IE) est, selon la norme IEEE 29148-2011 ISO/IEC/IEEE (2011), "an interdisciplinary function that mediates between the domains of the acquirer and supplier to establish and maintain the requirements to be met by the system, software or service of interest" ("une activité interdisciplinaire qui sert d'intermédiaire entre les domaines du client et du fournisseur afin d'établir et de gérer les exigences qui doivent être respectées par le système, le logiciel ou le service auquel on s'intéresse")

3.1.1.1 Exigence

Selon la même norme, une exigence est "[a] statement which translates or expresses a need and its associated constraints and conditions", traduit par "un énoncé qui traduit ou exprime un besoin et ses conditions et contraintes associées".

Les exigences sont un outil de communication : leur but est que le fournisseur comprenne ce que le client demande. Comme tous les outils de communication, il existe une distorsion de l'information, lors de l'écriture des exigences comme lors de la lecture.

On souhaite que ces distorsions soient les plus petites possibles : idéalement, le rédacteur de l'exigence veut pouvoir exprimer exactement ses besoins, et veut que le lecteur comprenne exactement ce qu'il a voulu dire. Dans la définition de la norme 29148, les mots "contraintes"

a)

ID	Type	Texte	Auteur	Catégorie	Justification	Version	Liens entrants	Liens sortants
Req004	Req	La probabilité d'une défaillance causant X doit être inférieure à p	Paul Martin	Sûreté	Étude de sûreté Y	2.1	SurSyst/ R002	SubSyst/ R005, Testcase/ T003

b)

Req004 : La probabilité d'une défaillance causant X doit être inférieure à p

(Éventuellement :)
Note : Le fournisseur proposera une méthode de calcul respectant le standard S

FIGURE 3.1 – Exigence telle qu'elle apparaît a) dans un programme de gestion b) dans une spécification

et "conditions" impliquent une quantification précise, nécessaire pour que l'exigence ne soit pas ambiguë.

Un élément important de l'IE est qu'une exigence doit définir ce que le système doit faire, et non pas comment le faire. En d'autres termes, une exigence ne devrait pas contenir d'informations sur le système qu'elle spécifie, mais uniquement sur l'environnement de ce système. Une exigence est une propriété que doit avoir l'environnement du système une fois le système installé. Un exemple de mauvaise exigence, qui spécifie comment le système doit être, est "La partie commande du système doit être triplement redondante". On préfère une exigence du type "La probabilité que le système envoie une mauvaise commande doit être inférieure à 10^{-8} par heure". La seconde exigence définit une propriété que le client veut effectivement voir vérifiée, alors que la première définit un moyen parmi d'autres de réaliser cette propriété.

Concrètement, si elle apparaît dans un ensemble d'exigences imprimé ou sous format informatique, une exigence se présente sous la forme d'une (parfois plusieurs) phrase, accompagnée d'un identifiant unique. On appellera cette ou ces phrases le *texte* de l'exigence. Ce texte peut être complétée par une phrase visant à justifier/préciser l'exigence, et/ou un schéma ou un tableau.

Dans un programme de gestion des exigences tel que DOORS, le texte de l'exigence est généralement accompagné d'un ensemble d'*attributs*, tels que : l'identifiant unique déjà mentionné, un auteur, une catégorie, une justification, un numéro de version, etc. (cf fig 3.1). Le programme de gestion permet aussi de montrer les liens de traçabilité entrants (quel exigence de niveau supérieur/besoin est à l'origine de cette exigence) et sortants (quelle exigence a cette exigence comme origine, comment l'exigence sera-t-elle testée/validée). Par exemple, l'exigence "La probabilité que le système envoie une mauvaise commande doit être inférieure

à 10^{-8} par heure" peut contenir un lien vers une exigence de niveau supérieur, comme "La probabilité d'une défaillance du système doit être inférieure à 10^{-9} par heure".

3.1.1.2 Hypothèse

Dans une spécification, le client a besoin de mentionner les propriétés de l'environnement qui sont indépendantes de la la présence du système. Par exemple, le client va donner les conditions environnementales, comme la température ou l'humidité, dans laquelle le système va pouvoir se trouver. Ces propriétés ne sont pas dépendantes du système, et ne sont donc a priori pas des exigences : on les appelle des hypothèses.

3.1.1.3 Supposition

Malgré le principe "les exigences ne parlent que de l'environnement", il arrive qu'un client définisse des éléments de la conception du système, ne serait-ce que pour pouvoir parler de ces éléments. Par exemple, pour un système mécatronique, il est tout a fait possible que le client demande que le contrôleur du système respecte certaines normes, même si ce contrôleur est un sous-système du système complet, et donc a priori "hors limite" du client. Le client va partir du principe qu'il y aura quelque part dans le système un contrôleur électronique, ce qu'on appellera dans ce travail une "supposition". Le fait de demander à ce que ce contrôleur respecte une norme est en revanche une exigence sur (une sous partie du) système.

3.1.1.4 Spécification

Toujours selon la norme IEEE 29148-2011, une spécification système est "[a] structured collection of the requirements (functions, performance, design constraints, and attributes) of the system and its operational environments and external interfaces" ("un recueil structuré des exigences (fonctions, performances attendues, contraintes de conception et attributs) du système ainsi que ses environnements opérationnels et ses interfaces externes")

Dans ce rapport, j'utiliserai aussi "corpus d'exigences" ou "ensemble d'exigences" avec un sens proche de spécification, à la nuance près que ces deux termes mettent moins l'accent sur les éléments annexes, non-exigences, de la spécification (les "environnements opérationnels et les interfaces externes" du système).

Une spécification système comporte donc toutes les exigences du système, de préférences classées en différentes catégories, ainsi que :

- Les éléments annexes qui sont directement référencés dans les exigences. Par exemple, un ensemble de schémas techniques visant à définir l'enveloppe physique où le système doit être placé, ou un tableau définissant les différents cas de chargement du système.
- Des éléments de contexte qui décrivent le but, les fonctions, la portée et les interfaces du système.
- Des définitions de termes, d'abréviations.
- Finalement, la spécification comporte aussi des éléments servant à la gestion du document (table des matières, noms des auteurs, historiques des modifications).

3.1.1.5 Rédaction des exigences

Dans le processus d'ingénierie des exigences mentionné en section 2.1.1.1, l'étape de rédaction consiste à formuler les exigences sous la forme décrite plus haut. On peut aussi parler de spécification des exigences, qui englobera plus généralement la formulation et la structuration des exigences ainsi que des ensembles d'exigences.

3.1.1.6 Critères sur les exigences

Aussi appelé "caractéristiques" des exigences, il existe un certain nombre de critères que les exigences devraient respecter pour être de "bonnes" exigences. Ces critères sont en quelque sorte des exigences sur les exigences elles-mêmes, des méta-exigences. On peut classer ces critères en différentes catégories (voir sous-section 3.4.2) : selon si le critère concerne une exigence isolée ou un ensemble d'exigences, selon si le critère concerne plutôt le lexique, la syntaxe ou le sens général (sémantique) de l'exigence, ou selon un but, par exemple, est-il possible de tester automatiquement le critère ?

3.1.2 Modèles

3.1.2.1 Modèle

Un modèle d'un système est une représentation simplifiée de ce système d'un certain point de vue. Un modèle peut être utilisé pour permettre des simulations ou des prédictions, comme les lois du mouvement de Newton, mais aussi pour faciliter les communications entre humains : certains types de diagrammes en ingénierie système ne peuvent pas être simulés mais sont utilisés comme une aide pour visualiser les systèmes. Indépendamment de leurs buts, les modèles sont maintenant très répandus dans la conception de systèmes.

De quels modèles parlons-nous ? Pour commencer, le mot "modèle" est lui-même polysémique. Les modèles que nous voulons utiliser comme références pour les exigences sont des modèles pour faciliter la communication entre humains. Ils n'ont pas besoin d'être simulables, mais ils doivent suivre une grammaire formelle. En particulier, nous ne parlons pas ici de modèles mentaux (les modèles de la réalité que nous développons dans notre cerveau).

Il est important de faire la distinction entre un modèle et sa (ou ses) représentation(s) graphique(s). Un modèle contient de l'information, formatée de manière à ce qu'elle respecte la syntaxe de ce type particulier de modèle. Sa ou ses représentation(s) graphique(s) sont un moyen de visualiser cette information. Même si la façon dont les informations sont présentées, i.e. la *forme*, est importante pour la communication, dans ce travail, nous nous intéressons au *fond*, c'est à dire à l'information contenue. Nous voulons être capables d'affirmer qu'un mot fait référence à un concept bien particulier défini dans un modèle, La question de savoir comment représenter ce concept et ce modèle peut être traitée plus tard.

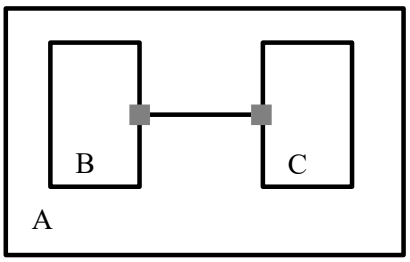


FIGURE 3.2 – Exemple basique de modèle d'architecture

3.1.2.2 Ingénierie basée sur les modèles

Le principe de base de l'ingénierie basée sur les modèles est de placer les modèles en tant qu'éléments centraux du processus d'ingénierie, à la place des documents textuels, plus statiques, de l'approche "traditionnelle". Dans ce rapport, les systèmes qui nous intéressent sont des systèmes complexes, qui peuvent comprendre des parties mécaniques, électriques, électroniques, logicielles, etc. Nous utiliserons le terme d'ingénierie basée sur les modèles comme traduction du terme anglais "Model-based systems engineering" (MBSE).

3.1.2.3 Modèle d'architecture

On va parler relativement souvent de modèles d'architecture dans ce rapport : Ces modèles sont très utilisés pour représenter les éléments qui composent un système et son environnement. On peut représenter des hiérarchies, comme la composition, avec ces modèles : dans la figure 3.2, l'élément A est composé des éléments B et C. On peut aussi représenter les interactions statiques entre éléments, par exemple, l'élément B a une interface avec l'élément C.

On peut représenter des éléments ayant des significations différentes en utilisant la même syntaxe : par exemple, on peut considérer que les éléments A, B et C représentent des objets physiques, tels qu'un groupe moteur, un moteur et un réducteur. On peut aussi considérer que A, B et C représentent des fonctions, telles que "fournir de l'énergie mécanique", "transformer l'énergie électrique en énergie mécanique" et "diminuer la vitesse de rotation". Dans ce travail, on ne s'intéresse pas vraiment au sens qui est donné à un modèle, on a juste besoin que ce sens existe et soit partagé par les lecteurs.

3.1.2.4 Automate à états

Alors que les modèles d'architecture (en tout cas d'après notre définition) sont plutôt statiques, les automates à états permettent de représenter des évolutions temporelles. Les automates à états principalement rencontrés en ingénierie système et logicielle sont des statecharts, popularisés par UML et basées sur HAREL (1987). Ces statecharts permettent de représenter des états inclus les uns dans les autres, ainsi que des décompositions ET et

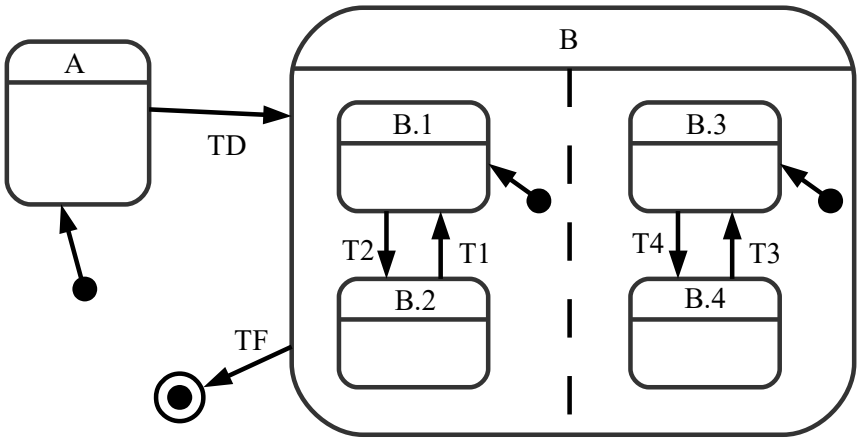


FIGURE 3.3 – Exemple d'automate à états, ici un statechart

OU à l'intérieur de ces super-états.

Par exemple, dans la figure 3.3, l'état A est initialement actif. Lorsque la transition "TD" est *tirée*, l'état A est désactivé et l'état B est activé, ainsi que les états B.1 et B.3. Les deux sous-automates B.1/B.2 et B.3/B.4 peuvent évoluer indépendamment tant que l'état B est actif, i.e. que la transition TF n'est pas tirée.

De manière générale, une transition a une condition booléenne associée (appelée une garde) ainsi qu'un événement associé. Quand l'événement survient, si la garde est vraie et que l'état de départ de la transition est actif, alors la transition est tirée. Il est possible d'associer des actions lorsqu'une transition est tirée, ou lorsqu'un état est activé ou désactivé.

En général, dans les statecharts que nous avons rencontré dans les spécifications industrielles, il n'y a pas ou peu d'actions associées à des transitions et à des entrées/sorties d'états. On s'intéresse plutôt dans ces cas au statut des états, i.e. s'ils sont actifs ou pas. De la même manière il n'y a souvent pas d'événement associés aux transitions et celles-ci sont tirées dès que la garde est vraie.

Comme pour les modèles d'architecture, on peut donner différentes interprétations des états et des transitions. Encore une fois, on ne se préoccupe que de l'existence d'une interprétation partagée.

3.1.3 Autres concepts

3.1.3.1 Ambiguïté

On a vu que le but des exigences était la communication. Les malentendus liés à des exigences ambiguës sont une cause importante de problèmes. De plus, plus vite ces ambiguïtés sont levées, moins l'effet sur le projet (coût et retard) est important (STECKLEIN et collab. (2004)). Nous cherchons donc à aider les ingénieurs à éviter l'écriture de mauvaises exigences. Les exigences peuvent être mauvaises pour différentes raisons (voir section 2.1.1.3),

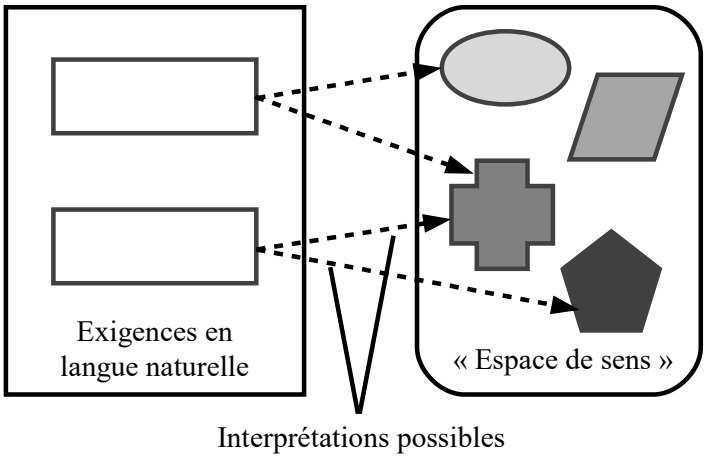


FIGURE 3.4 – Une représentation de l'ambiguïté

mais dans ce travail, nous nous concentrons principalement sur le texte en langue naturelle des exigences et sur comment réduire l'ambiguïté de ce texte.

Imaginons un espace de "sens" comme indiqué dans la figure 3.4. Quand quelqu'un lit un texte, cette personne associe le texte avec un élément de l'"espace de sens" (elle "interprète"). Un texte ambigu est un texte qui peut être associé avec différentes parties de l'espace de sens par différents lecteurs. Notre but est d'avoir une bijection entre texte et sens. Bien sûr, cet "espace de sens" n'existe pas physiquement ou même comme des bits dans un ordinateur, et on ne peut pas y faire directement référence. On peut cependant restreindre les interprétations possibles en fournissant des informations supplémentaires qui vont guider le lecteur vers le sens attendu.

Pour le langage écrit, on peut considérer au moins trois différents types d'ambiguïté : les ambiguïtés lexicale, syntaxique et sémantique.

L'ambiguïté lexicale concerne les mots isolés : il y a une ambiguïté lexicale si un mot a plusieurs sens possibles. Par exemple, un bar peut être un poisson, une unité de pression, un établissement où l'on sert de l'alcool, etc. Ce type d'ambiguïté peut souvent être levé grâce au contexte dans lequel on trouve le mot. Dans les exigences, c'est en général le cas pour les mots du langage courant : on se préoccupe souvent de pressions dans les spécifications aéronautiques et assez peu de poissons. Par contre, pour des concepts qui n'existent que dans la spécification, lever les ambiguïtés peut être plus difficile : est-ce que "les équipements du système montés sur le train d'atterrissage" et "les composants installés sur le train d'atterrissage" font exactement référence à la même chose ?

Il existe une ambiguïté syntaxique s'il existe plusieurs structures différentes pour une même phrase. Un cas classique concerne la portée des conjonctions (et, ou) : la condition "quand X est vrai et que Y est faux ou que Z est vrai" peut être lue "$(X \wedge \neg Y) \vee Z$" ou "$X \wedge (\neg Y \vee Z)$".

L'ambiguïté sémantique est plus générale et est associée à une phrase voire plusieurs :

un texte est sémantiquement ambigu si l'on peut lui associer plusieurs sens. Les ambiguïtés syntaxiques et lexicales peuvent mener à des ambiguïtés sémantiques.

3.1.3.2 Lexique

Parmi les définitions de "lexique" du dictionnaire Larousse, on trouve : "Dictionnaire spécialisé et généralement succinct concernant un domaine particulier de la connaissance." Dans notre travail, le lexique des exigences sera l'ensemble des mots que l'on peut utiliser dans les exigences. On considère que relève du lexique tout ce qui concerne soit des mots individuels, soit des groupes de mots pris comme un tout. Par exemple, "ABS" est un mot qui renvoie à un concept précis, tout comme "système anti-blocage des roues" : on considère que "système anti-blocage des roues" est une expression atomique, qu'on ne cherche pas à décomposer. Pour que les exigences ne soient pas ambiguës, il faut d'abord que les mots utilisés ne soient pas ambigus.

3.1.3.3 Syntaxe

Toujours selon le Larousse, la syntaxe est la "[p]artie de la grammaire qui décrit les règles par lesquelles les unités linguistiques se combinent en phrases." On va considérer comme "unités linguistiques" les mots du lexique mentionnés au dessus. On appellera syntaxe tout ce qui a trait à rassembler des mots pour en créer une exigence. La syntaxe qui nous intéresse pour les exigences est une syntaxe plus proche de celle des langues formelles que de celle des langues naturelles.

3.1.3.4 Sémantique

Dans les définitions de "sémantique" par le Larousse, on trouve "Étude du sens des unités linguistiques et de leurs combinaisons", mais aussi "Aspect de la logique qui traite de l'interprétation et de la signification des systèmes formels, par opposition à la syntaxe, entendue comme l'étude des relations formelles entre formules de tels systèmes."

La première définition concerne principalement les langues naturelles, alors que la seconde est valable pour les langues formelles. Le point commun de ces deux définitions est la référence au sens, à la signification. De manière générale, nous ne cherchons pas dans ce travail à étudier le sens des exigences : nous considérons que c'est trop complexe, et que l'étude du lexique et de la syntaxe peut déjà offrir des résultats intéressants.

Ceci dit, nous pensons que réduire l'ambiguïté liée au lexique et à la syntaxe d'une exigence réduira nécessairement l'ambiguïté sur le sens de cette exigence.

3.2 Analyse de la situation actuelle

3.2.1 Langues naturelles

Le but des exigences est que tout le monde devrait comprendre la même chose lors de la lecture, en d'autres termes, les exigences ne devraient pas être ambiguës.

3.2.1.1 Ambiguïté des langues naturelles

Les langues naturelles sont intrinsèquement ambiguës, de différentes manières : ambiguïtés lexicales, syntaxiques et sémantiques que l'on a décrites plus haut. Qu'est-ce qui rend les langues naturelles ambiguës ?

Tout d'abord, la plupart des mots ont plusieurs définitions (voir l'exemple de "bar" ci-dessus). De plus, certains mots peuvent avoir des définitions qui appartiennent à des catégories grammaticales (nom, verbe, adjectif...) différentes : le mot "contrôle" peut être un verbe ou un nom. En plus d'avoir plusieurs définitions distinctes, un mot peut manquer de précision, comme le mot "rapidement", dont le sens précis va dépendre du contexte et du lecteur.

Concernant la syntaxe, de nombreux cas d'ambiguïtés sont dus à la difficulté de savoir à quel termes s'applique une préposition ou une coordination. Par exemple, dans l'exigence "Il doit être possible de réparer les systèmes sur l'avion", on peut considérer que "sur l'avion" complète soit "réparer", soit "les systèmes". Ces deux possibilités ont des implications différentes en pratique : soit l'exigence demande qu'il soit possible de réparer les systèmes sans les enlever de l'avion, soit elle demande qu'il soit possible de réparer les systèmes qui sont installés sur l'avion (par opposition à d'autres systèmes au sol par exemple).

Ces ambiguïtés vont nécessiter que le lecteur fasse des inférences pour comprendre le texte, inférences qui ne seront pas nécessairement les mêmes que celles faites par le rédacteur ou par d'autres lecteurs. Un problème est que ces inférences sont nécessaires : il serait très difficile d'expliciter de manière non-ambiguë chaque petit détail d'une spécification, en utilisant la langue naturelle ou non. Le contexte et les formations des parties prenantes aident à faire des inférences correctes, mais pas de manière parfaite.

Un autre désavantage de cette ambiguïté est qu'il est complexe de réaliser des traitements automatiques sur les langues naturelles (TAL). Il existe des méthodes relativement bien développées de TAL, telles que l'étiquetage grammatical (i.e. donner automatiquement les catégories grammaticales des mots d'un texte, "part-of-speech tagging" en anglais). Cependant ces méthodes sont souvent des méthodes statistiques et nécessitent des corpus importants de textes (de préférence déjà étiquetés), et nous n'avons pas beaucoup d'ensembles d'exigences à disposition. De plus, vu que l'ingénierie des exigences est un domaine concernant un nombre de personnes relativement restreint, les méthodes de TAL les plus avancées n'ont pas nécessairement les mêmes buts que ceux que nous recherchons.

Si les langues naturelles sont ambiguës et difficilement traitables par ordinateur, pourquoi les utilise-t-on pour écrire des exigences ?

3.2.1.2 Mais les langues naturelles sont pratiques

Un avantage des langues naturelles est que tout le monde en connaît au moins une. De plus, les organisations qui conçoivent des systèmes ont généralement une langue naturelle commune. En fait, les langues naturelles sont le seul moyen de communication commun dès que l'on considère une population importante. Ces faits montrent que l'utilisation de la langue naturelle est pratique, puisque les besoins de formations sont moindres voire nuls, et que l'utilisation d'un autre langage est difficile, puisque cela nécessiterait de former les parties prenantes à ce langage.

Un autre avantage de la langue naturelle est qu'elle permet d'exprimer des choses extrêmement variées, alors que les langues formelles peuvent être sévèrement restreintes. Un texte en langue naturelle, une fois placé dans un contexte, permet d'inférer beaucoup plus d'informations que si l'on réduisait ce texte à sa taille en bits par exemple. Le but des langues formelles est justement que rien ne soit laissé à l'interprétation.

Ces avantages de la langue naturelle font que la quasi-totalité des exigences industrielles que nous avons rencontrées étaient écrites en langue naturelle. On peut se demander s'il est possible de l'éviter, et d'utiliser des langues formelles pour spécifier les systèmes.

3.2.2 Langues formelles

3.2.2.1 Les langues formelles ne sont pas ambiguës

La plupart des langues formelles sont construites pour éviter l'ambiguïté. Une exigence écrite dans une langue formelle permet que tous les lecteurs qui connaissent cette langue formelle comprennent la même chose. De plus, il est possible de réaliser des traitements informatiques sur la plupart des langues formelles. Par exemple, on peut imaginer de vérifier automatiquement des propriétés comme l'absence de contradiction entre les exigences. Ces avantages ont conduit à l'utilisation de langages formels pour la spécification de certaines applications logicielles (e.g. le logiciel critique embarqué de la ligne de métro 14 à Paris, Behm et collab. (1999)).

Les inconvénients de l'utilisation de langages formels rendent cependant leur utilisation très peu pratique pour la spécification de systèmes physiques.

3.2.2.2 Difficultés d'utilisation des langues formelles

Premièrement, les langues formelles ne sont maîtrisées que par une petite minorité de personnes. S'il n'est pas nécessaire que n'importe qui puisse lire une spécification, en dehors de cas assez particuliers, une spécification doit pouvoir être lue par des personnes non formées aux langues formelles. De plus, la rédaction de modèles formels demande une maîtrise plus grande de ces langues, qui peut être difficile à maintenir dans les entreprises.

Même si les spécifications sont seulement des abstractions des systèmes réels, les systèmes que l'on cherche à spécifier existent dans le monde réel. Nous sommes persuadés que, pour une langue formelle donnée arbitraire, une proportion non-négligeable des exigences de n'importe quelle spécification industrielle est pratiquement impossible à écrire en utilisant

ce langage.

Comme on l'a mentionné plus haut, l'utilisation de la langue naturelle permet des nuances qui apportent beaucoup d'informations aux lecteurs, même sans écrire "explicitement" ces informations. S'il est possible de faire des inférences à partir d'éléments de langage formel (par exemple, qu'une variable soit appelée "S12" ou "LIGHTS_ON" dans un code source est rigoureusement identique pour le compilateur, mais pas pour quelqu'un qui lit ce code source), un langage formel va nécessairement limiter les possibilités d'expression d'un rédacteur d'exigences.

3.2.3 L'utilisation actuelle des modèles

Les modèles sont actuellement sous-représentés dans les spécifications de systèmes (voir section 2.1.3.4). Premièrement, la plupart des exigences sont écrites en langue naturelle seulement. Deuxièmement, même si les spécifications actuelles incluent des modèles, nous pensons que ces modèles peuvent être utilisés plus efficacement. Nous proposons de lier ces modèles avec les exigences de manière explicite et systématique.

3.2.3.1 Progression de l'ingénierie basée sur les modèles

On a mentionné quelques éléments sur l'ingénierie basée sur les modèles dans le chapitre précédent (section 2.2.1.3). L'ingénierie basée sur les modèles est un concept relativement récent. Tout d'abord, même si notre travail concerne l'ingénierie système (et non uniquement logicielle), les concepts de l'ingénierie basée sur les modèles sont beaucoup inspirés du travail réalisé pour l'ingénierie logicielle, en particulier le "Model-driven engineering" (MDE). S'il est difficile de donner des dates précises, on peut citer le travail séminal (et assez théorique) de Wymore (1993), ainsi que, plus récemment, le début de l'initiative "Model-based systems engineering" (MBSE) par l'INCOSE en 2007.

Parce que ces méthodes sont relativement récentes, les pratiques industrielles ne sont généralement pas encore finalisées. De plus, il est assez complexe d'étudier l'impact de ces méthodologies sur l'efficacité du processus d'ingénierie. Ces facteurs rendent difficile une étude objective et générale de l'utilité de l'ingénierie basée sur les modèles. Les difficultés liées tant à l'approche traditionnelle "document-centric" qu'à l'ingénierie basée sur les modèles sont cependant bien connues.

3.2.3.2 Limites des documents textuels

La volonté de moins se focaliser sur les documents textuels a plusieurs origines :
— Les systèmes sont de plus en plus complexes.
— Les industriels veulent pouvoir être plus réactifs, et veulent donc que les systèmes puissent évoluer entre le début et la fin du processus de conception.
— Les textes en langue naturelle peuvent être ambigus.
— L'ubiquité des ordinateurs et des réseaux de communication font que le support papier n'est plus indispensable partout.

Des systèmes plus complexes impliquent des documents plus importants. Des évolutions plus fréquentes de ces systèmes impliquent des modifications, vérifications et validations plus fréquentes des documents. Ces éléments font ressortir les inconvénients des documents textuels, en particulier que seuls les humains peuvent les lire et les modifier. La charge de travail nécessitée par ces revues humaines devient donc très importante.

L'utilisation de modèles vise à permettre qu'une partie de cette charge de travail soit effectuée automatiquement, ou au moins plus facilement.

3.2.3.3 Des freins à l'adoption de l'ingénierie basée sur les modèles

Il y a, évidemment, un certain nombre d'inconvénients qui contrebalancent les avantages de l'ingénierie basée sur les modèles. Tout d'abord, l'utilisation de modèles n'est pas naturelle et nécessite une formation. Les entreprises doivent donc payer ce coût de formation, pour rédiger mais aussi pour pouvoir comprendre des modèles. Ces coûts, ainsi que la difficulté de prouver de manière claire la rentabilité des méthodes d'ingénierie basée sur les modèles, tendent à freiner leur adoption.

Au delà de l'inévitable résistance au changement, il y a aussi d'autres inconvénients. Si un texte en langue naturelle, une fois exporté dans un fichier PDF, est imprimable et est lisible par virtuellement n'importe quel ordinateur, ce n'est pas le cas des modèles en général. Il existe différents types de modèles et différents formats et outils pour un même type de modèle. Il est possible d'imprimer ces modèles comme on le ferait pour du texte, mais cela limite grandement leur utilité. Il faut donc s'assurer que les différents acteurs utilisent les mêmes modèles, avec les mêmes outils, ou au moins qu'ils soient traduisibles.

3.2.3.4 Qu'est-ce qu'un modèle dans une spécification ?

Une question importante concerne le sens des modèles dans une spécification : différents modèles peuvent être utilisés de différentes manières dans une spécification, mais leur signification par rapport aux exigences n'est pas toujours claire. Par exemple, on peut trouver dans des spécifications des exigences du type "Le système doit respecter l'automate à états en annexe A". Quel que soit le contenu de l'automate à états dans l'annexe A, le sens du mot "respecter" n'est sans ambiguïté ici. De la même manière, on trouve des exigences telles que "Le système doit s'interfacer avec les systèmes externes X, Y et Z comme spécifié dans le tableau 3.5"

Un certain nombre de questions, dont nous n'avons pas nécessairement les réponses, se posent ici. Clairement, de tels modèles peuvent servir à remplacer des exigences : il serait possible de remplacer le second exemple par un groupe d'exigences, une pour chaque entrée/sortie, du type "le système doit acquérir le signal booléen S1 du système X, le signal est soit ouvert, soit connecté à la terre, envoyé par un fil, et connecté à la terre quand S1 est vrai". Faire cela rendrait la spécification beaucoup plus longue, et donc plus difficile à lire, vérifier, valider, etc. Cependant, cela pourrait permettre au fournisseur de tracer des liens de traçabilités vers ces exigences pour justifier un choix de conception ou une exigence de plus bas niveau, ce qui peut être plus complexe quand ces exigences sont toutes rassemblées en un tableau.

On peut aussi se demander si ces éléments sont bien des exigences : ici le tableau 3.5

Nom	Type	Type connexion	Entrée/ sortie	Sys- tème	Connec -teur	Correspondance
S1	Discret	Terre/ouvert	Entrée	X	fil	Connecté à la terre quand S1 vrai
S2	Discret	28VDC/ouvert	Sortie	Y	fil	Connecté à 28VDC quand S2 vrai
S3	Analo- gique	0-28V	Entrée	Z	COM/ MON	Linéaire, 0V pour 0m/s, 28V pour 10m/s
S4	Discret	Bus type W	Entrée	Bus B	Label 223	Vrai quand S4 vrai

FIGURE 3.5 – Exemple de tableau définissant des interfaces

décrit l'environnement du système tel qu'il est, que le système soit présent ou pas, ce qui tendrait à impliquer que ce sont des hypothèses (cf section 3.1.1.2) plutôt que des exigences. Cependant, puisque ces informations vont avoir un impact sur le design du système, est-ce que la distinction entre exigences et hypothèses est pertinente pour le fournisseur ? Encore une fois, le fournisseur peut vouloir écrire des liens de traçabilités vers ces hypothèses. Cela implique soit de considérer ces hypothèses comme des exigences (est-ce grave en pratique ?), soit de créer un objet "hypothèse" pour les spécifications, doté de caractéristiques similaires aux exigences.

Pour l'instant, nous n'avons pas réellement défini quels types de modèles nous allons utiliser. Pour savoir quels modèles pourraient être utiles comme références pour les exigences, nous devons d'abord étudier quels concepts apparaissent dans les exigences.

3.3 Objectifs détaillés

3.3.1 Compléter le texte en langue naturelle

Nous considérons que, dans notre contexte, la langue naturelle doit faire partie de toute méthode réaliste d'IE. Cependant, nous pensons aussi que la langue naturelle seule ne devrait pas être l'unique moyen d'exprimer des exigences.

3.3.1.1 Compléter la langue naturelle

Plutôt que de n'utiliser que la langue naturelle pour spécifier, nous proposons de la compléter avec tout élément qui pourra être utile pour réduire l'ambiguïté et pour faciliter les traitements automatiques. De plus, nous pensons que l'utilisation de modèles parmi ces éléments additionnels va permettre de mieux intégrer ces modèles dans le processus d'ingénierie des exigences, et donc de mieux intégrer l'ingénierie des exigences à une

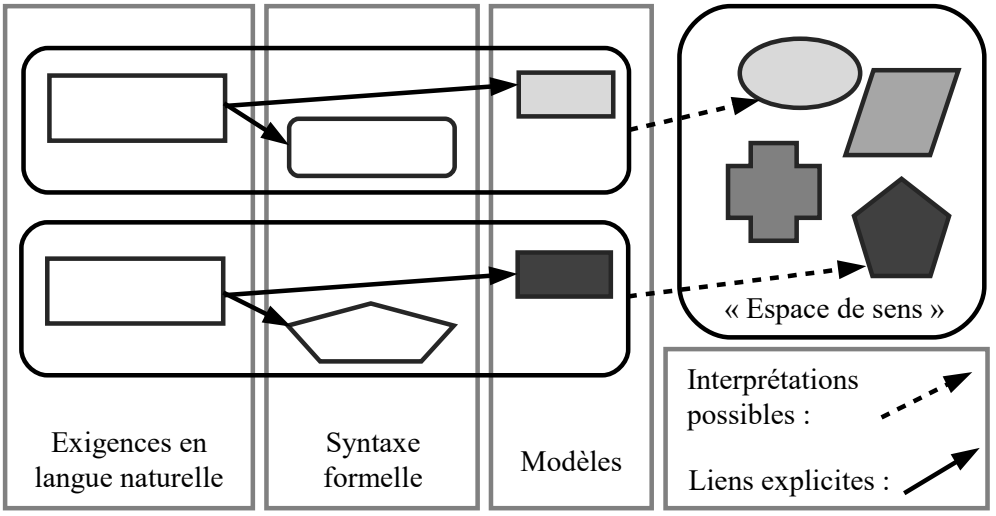


FIGURE 3.6 – Des modèles et une syntaxe formelle comme outils pour combattre l'ambiguïté

démarche d'ingénierie basée sur les modèles. Idéalement, on peut espérer un processus où l'on construit en parallèle les modèles et les spécifications, ces deux activités se nourrissant l'une l'autre.

Comme illustré dans le schéma 3.6, on lie les exigences à des informations syntaxiques et à des éléments de modèles. Cette information additionnelle précise le texte des exigences. Si le texte, l'information syntaxique et les modèles sont cohérents et correct, l'information supplémentaire devrait permettre aux lecteurs de choisir le sens correct lors de la lecture des exigences. On cherche ainsi à éliminer les potentielles ambiguïtés, telle qu'un texte avec plusieurs sens possibles ou plusieurs textes différents partageant le même sens.

Concrètement, on cherche à pouvoir lier un texte avec un sens préféré afin qu'il soit interprété correctement. Pour commencer, nous réalisons cela au niveau lexical, en liant des mots ou groupes de mots des exigences à des éléments servant de définitions. Cela constitue une première étape qui sera utile pour éviter l'ambiguïté lexicale. On donnera plus de détail dans les chapitres suivants, mais la figure 3.7 donne un exemple de ces liens. Le modèle d'architecture définit les éléments utilisés dans l'exigence. Les textes de l'exigence entre crochets ("la Puissance Électrique", "le Système", etc.) sont dotés de liens hypertextes pointant vers les éléments du modèle donnant leur définition.

Dans un second temps, nous explorons comment désambiguïser la syntaxe des exigences en utilisant le même principe : on complète un texte en langue naturelle avec des informations hypertextes, cette fois concernant la structure du texte. Dans la figure 3.8, la même exigence est associée à une syntaxe formelle. Toutes les exigences ne peuvent pas être complètement formalisées comme celle-ci, mais la syntaxe proposée en section 4.2.3 accepte aussi des fragments d'exigence non formalisés.

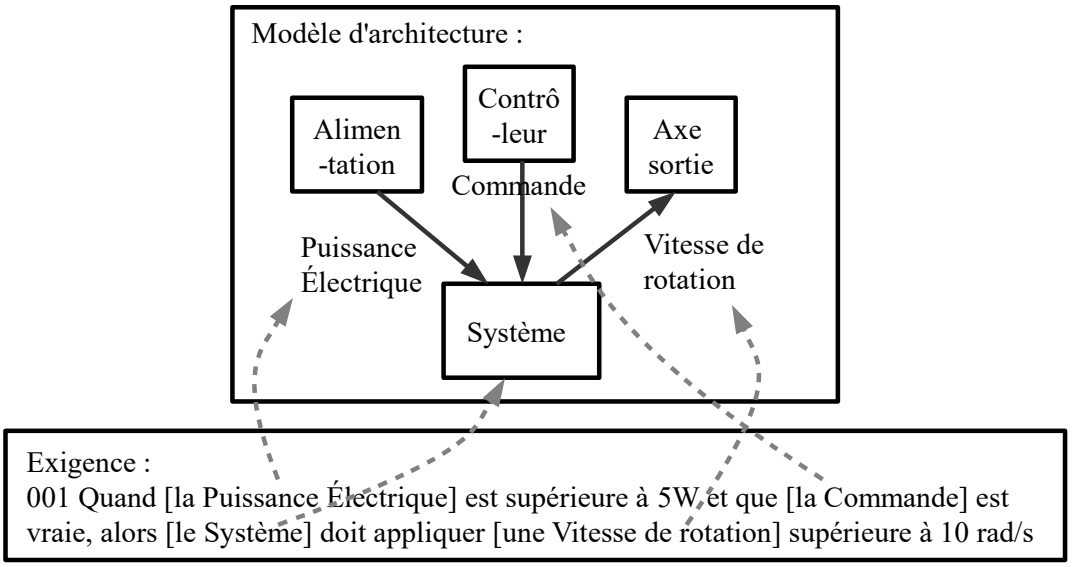


FIGURE 3.7 – Exemple d'exigence avec des liens tracés vers des éléments d'un modèle

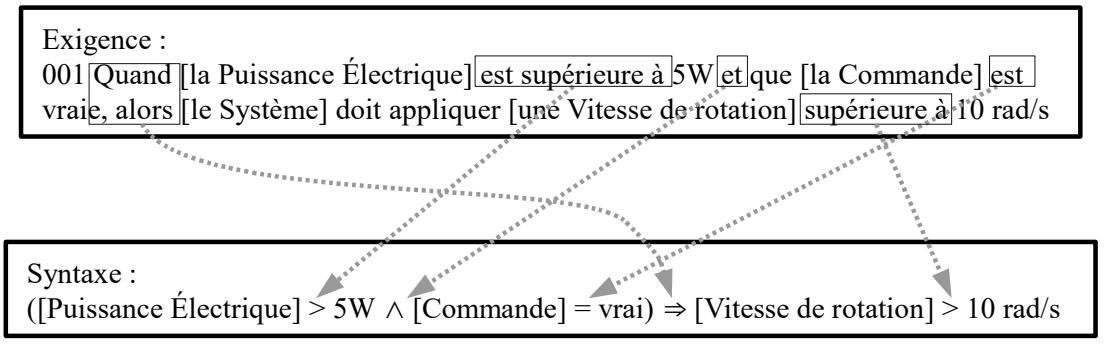


FIGURE 3.8 – Exemple d'exigence dotée d'une syntaxe formelle

3.3.1.2 Réalisation de traitements automatiques

La seconde utilisation de ces éléments additionnels est de faciliter les traitements automatiques. Une fois que l'on a ajouté un lien explicite à un fragment d'exigence, un script informatique peut déduire des choses de ce lien, même si l'on ne cherche à comprendre ni la cible, ni le point de départ du lien. Par exemple, on peut vérifier qu'il existe de tels liens pour chaque exigence, et donc qu'elles sont un minimum fixées dans la réalité, ou au moins dans les modèles qu'on se fait de la réalité. On peut aussi étudier quelles sont les exigences qui font référence à tel élément.

En plus de ces liens, des éléments de syntaxe formelle permettent d'obtenir des informations sur la structure des exigences. Ces informations permettent d'autres traitements, comme vérifier que cette structure est correcte ou sélectionner des exigences avec une structure similaire.

Nous pensons que ces traitements automatiques vont faciliter la gestion des exigences, en permettant de faire des tests rapides pour vérifier des propriétés qui seraient sinon faites par une lente et coûteuse inspection humaine des exigences. On présente le principe de ces tests en section 3.4 et on les détaille en pratique en section 4.3.

3.3.2 Utilisation d'une syntaxe simple

Notre but est que les éléments de syntaxe formelle soient simples, afin que la méthode soit utilisable en pratique. Pour la même raison, la syntaxe doit être permissive et adaptable. Cette syntaxe est détaillée dans la section 4.2.3

3.3.2.1 Complexité des langues formelles existantes

Les langues formelles existantes ne sont en général pas adaptées à ce que nous voulons faire. Elles sont maîtrisées par très peu de personnes et même pour ceux qui les maîtrisent, ont d'importantes limitations sur ce qu'il est possible d'exprimer en les utilisant. L'usage même du terme "langage formel" a tendance à rendre méfiants une partie des ingénieurs.

Ceci dit, nous ne cherchons pas à inventer quelque chose de complètement nouveau, toujours pour éviter que la syntaxe soit trop difficile à comprendre/apprendre. Nous avons donc cherché à définir une syntaxe adaptée à nos besoins, en réutilisant des concepts relativement courants.

3.3.2.2 Syntaxe minimale et adaptée

Nous définissons une syntaxe minimale, ainsi que des moyens de la modifier ou de l'agrandir. Cette syntaxe doit permettre de décrire la structure des exigences de façon utile : on ne se préoccupe pas vraiment de savoir si tel mot est un verbe ou un nom, parce qu'on va considérer que les lecteurs vont faire cet étiquetage grammatical facilement et de manière suffisamment fiable. Par contre, on va davantage s'intéresser à savoir si un élément est une condition ou une conséquence. En plus de décrire la structure des exigences, cette syntaxe doit prendre en compte l'autre partie de ce travail, les liens explicites vers

des modèles. Une autre caractéristique est que cette syntaxe n'est pas nécessairement un élément central et auto-suffisant mais plutôt un complément au texte en langue naturelle.

Pour obtenir cette syntaxe "minimale" nous avons étudié des spécifications industrielles. Lors de ces études, quand nous trouvions un élément qui semblait formalisable et qui ne l'était pas encore, nous ajoutions une construction correspondante à la syntaxe. Par exemple, les éléments "et", "plus grand que", "égal"... de l'exigence dans la figure 3.8 sont des éléments de syntaxe classique, susceptibles d'être trouvés dans n'importe quelle exigence. Nous avons essayé de faire que ces constructions soient relativement génériques, afin d'éviter d'écrire une syntaxe qui ne soit adapté qu'à nos cas d'études.

3.3.2.3 Une syntaxe permissive et flexible

Premièrement, définir une syntaxe permet plus de flexibilité dans les exigences qu'en utilisant des gabarits simples : un ensemble de gabarits simples ne va pouvoir accepter qu'un nombre fini de structures de phrases, alors qu'en définissant des règles de compositions ce nombre de structures possibles devient infini. Par exemple, on peut vouloir rajouter une condition "et qu'il n'y a pas de défaillance" dans l'exigence de la figure 3.8 : 'Quand [la Puissance Électrique] est supérieure à 5W, que [la Commande] est vraie et qu'il n'y a pas de défaillance, alors [le Système] doit..."

La syntaxe doit être permissive, et en particulier permettre la présence de fragments de texte non formalisé, parce qu'il est impossible de prévoir toutes les constructions dont vont avoir besoin les rédacteurs d'exigences. Dans l'exemple du paragraphe précédent, il est possible qu'on ne sache pas formellement ce que signifie "il n'y a pas de défaillance". Ce n'est pas un problème, tant que l'on est capable de donner un type à ce fragment de texte, pour permettre de l'inclure dans la syntaxe. Ici ce type est un booléen : soit il y a une défaillance, soit non.

Il doit aussi être possible de rajouter facilement de nouveaux éléments à cette syntaxe, afin que ce soient les utilisateurs, et pas nous-mêmes, qui puissent l'améliorer, la modifier ou l'adapter à leur contexte particulier. Toujours en reprenant l'exemple précédent, on peut vouloir introduire une fonction syntaxique "est-ce que le système X est défaillant ?". La définition de cette nouvelle fonction est assez simple : il suffit de préciser le nombre d'arguments, leur types et le type de sortie. Cette fonction prend un système comme argument et renvoie un booléen (est-ce que le système est défaillant ou pas).

3.3.3 Utilisation de modèles comme définitions et lexique

L'utilisation de modèles peut être vue comme un but en soi, afin de mieux intégrer l'ingénierie des exigences avec une démarche d'ingénierie basée sur les modèles. Nous pensons aussi que l'utilisation de modèles avec les exigences aura des avantages significatifs sur le processus d'ingénierie des exigences, tout en étant relativement simple. On détaille cette utilisation des modèles dans les sections 4.2.1 et 4.2.2.

3.3.3.1 Des modèles comme définitions

Un moyen de diminuer l'ambiguïté est de préciser ce que l'on essaye de dire. Concernant le lexique, c'est à dire les mots utilisés, une façon de faire est de rajouter des définitions à ces mots. Notre idée est d'utiliser les divers modèles du processus de conception pour servir de définitions pour les exigences. Ces modèles sont en général plus pratiques comme définitions que des paragraphes de texte. Pour les modèles d'architecture par exemple, beaucoup d'informations, telles que la structure hiérarchique des éléments ou leurs interfaces, sont visibles facilement et rapidement. Ces modèles d'architecture vont avoir un rôle particulier : les interfaces qu'ils définissent constituent l'environnement pertinent du système (par exemple "Puissance Électrique", "Commande" et "Vitesse de rotation" dans l'exemple de la figure 3.7).

Les autres éléments qu'on va utiliser dans les exigences vont être construits à partir de ces éléments de base de l'environnement (voir section 4.2.2). Ces contructions peuvent également être réalisées à l'aide de modèles, par exemple, pour spécifier des comportements, définir des états en utilisant uniquement du texte semble assez peu pratique. D'autres éléments, comme des définitions d'enveloppes physiques où installer le système, ne peuvent raisonnablement pas être définis sans modèles, comme par exemple des schémas, voire des modèles CAO.

3.3.3.2 Réutilisation de modèles existants

Avant de pouvoir utiliser des modèles comme références pour les exigences, il faut d'abord écrire ces modèles. Cela pourrait sembler ne pas être rentable, si les modèles étaient écrits en partant de zéro. Cependant, nous avons observé que cette étape de modélisation était généralement réalisée quoi qu'il arrive : toutes les spécifications industrielles que nous avons étudié incluaient des modèles qui auraient pu être utilisés comme références pour les exigences. Mais aucune de ces spécifications ne comportaient les liens explicites entre exigences et modèles que nous envisageons dans ce travail.

De plus, si l'on peut utiliser un modèle tel quel et éviter de le traduire pour le mettre sous une forme textuelle, cela diminue d'autant le travail nécessaire.

3.3.3.3 Liens précis entre modèles et exigences

Enfin, il est important de faire des liens précis entre modèles et exigences. Tout d'abord ces liens doivent être précis sur leur point de départ, i.e. dans les exigences : on souhaite pouvoir dire que tel mot fait référence à tel élément, pas juste que, quelque part dans l'exigence, il y a une référence. Ces liens doivent aussi avoir une cible précise, on veut savoir qu'une référence pointe vers un élément précis, pas vers le modèle complet. Par exemple, même si elle peut quand même être utile, on peut déduire beaucoup moins de choses de la figure 3.9 que de la figure 3.7.

Cette précision permet de réaliser les tests mentionnés plus haut, sans que l'on s'intéresse forcément à ce que sont les éléments au bout de ces liens.

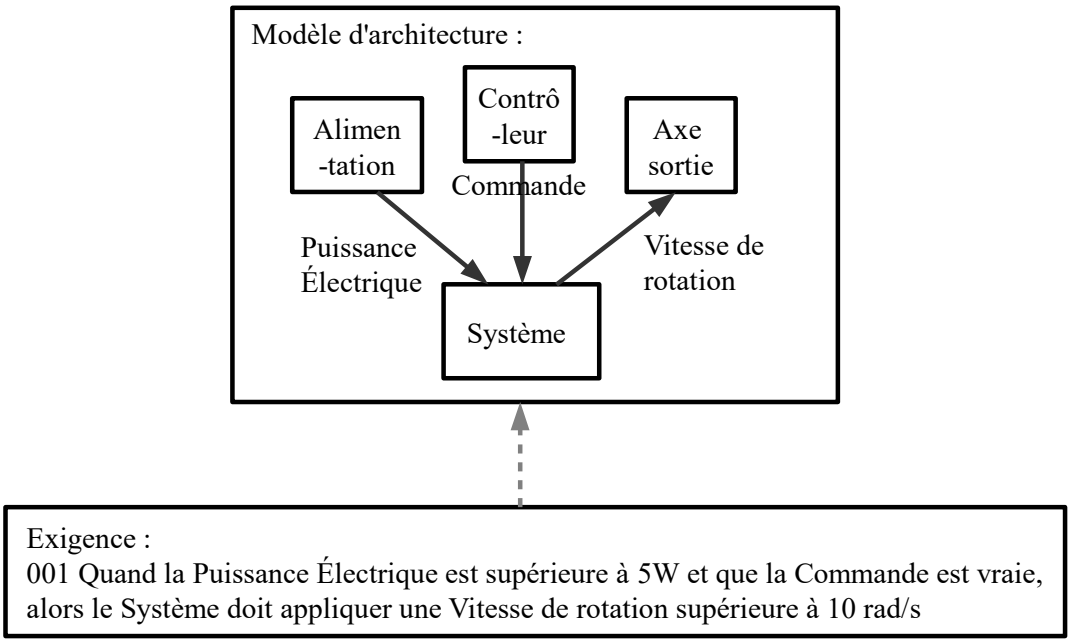


FIGURE 3.9 – Sans informations précises pour les liens, on peut seulement dire qu'une exigence fait référence à un modèle

3.4 Vérifications automatiques

3.4.1 Pourquoi tester ?

Nous proposons deux idées principales : ajouter des liens explicites entre des modèles et le texte des exigences, et ajouter une structure formelle aux exigences. Le but est de réduire l'ambiguïté en proposant un moyen sûr d'interpréter le texte : par exemple, si deux exigences font chacune référence à un concept, savoir que ce concept est le même est trivial si les références sont explicites. Cependant, vérifier manuellement ce type de propriété devient rapidement difficile quand le nombre d'exigences augmente. Le but suivant est donc naturellement d'automatiser ce genre de tests.

3.4.1.1 Vérification humaine lente et complexe

Un certain nombre de propriétés que l'on souhaite vérifier sur les exigences n'ont pas besoin d'être vérifiées par une intelligence humaine. En général, c'est le cas lorsque l'on s'intéresse uniquement au lexique ou à la syntaxe de l'exigence, sans chercher à en comprendre le sens. L'intérêt de vérifier ces propriétés rapidement, automatiquement et de manière fiable est assez évident.

Le temps nécessaire pour vérifier des propriétés qui ne concernent qu'une seule exigence (par exemple, vérifier que la syntaxe de l'exigence est correcte) est simplement proportionnel

au nombre d'exigences. Les propriétés qui demandent à s'intéresser à plusieurs exigences (par exemple, vérifier que, pour chaque concept utilisé dans les exigences, on utilise toujours le même terme) ont une complexité plus importante. Dans le cas de vérifications automatiques, vu les tailles des spécifications que l'on étudie (maximum ~1000 exigences) le temps de traitement est suffisamment court pour que les deux cas ne soient pas sensiblement différents. Ce n'est par contre pas le cas quand ce sont des humains qui vérifient ces propriétés. De plus, un algorithme de test "voit" l'ensemble de la spécification, ce qui pas nécessairement le cas pour des humains, si par exemple différentes parties de la spécification sont rédigées par différentes personnes.

La vérification de propriétés par tests informatiques sera donc très intéressante pour les propriétés concernant un ensemble d'exigences, mais ces tests peuvent aussi être très utiles pour vérifier des propriétés sur des exigences unitaires.

3.4.1.2 Une philosophie agile appliquée au exigences ?

La philosophie agile, formalisée en 2001 dans le *Manifesto for Agile Software Development* Beck et collab. (2001), prône notamment l'adaptation au changement et l'acceptation de ces changements plutôt que le suivi d'un plan prédéfini. Il parait difficile d'appliquer les principes agiles aux *systèmes* que l'on étudie : tout d'abord, ceux-ci sont des systèmes physiques, qu'il est difficile de prototyper et de livrer rapidement. De plus, pour des systèmes critiques, avoir des systèmes en évolution constante tout en garantissant leur sécurité parait très complexe.

Comme on l'a noté précédemment, les entreprises souhaitent pouvoir être plus réactives. Il est peut-être possible d'améliorer cette réactivité en appliquant les (ou certains) principes de la philosophie agile aux *exigences* spécifiant les systèmes plutôt qu'aux systèmes eux-mêmes. Une des choses qui permettrait de faire évoluer plus rapidement les exigences serait de pouvoir les tester souvent et rapidement.

3.4.1.3 Limites des tests automatiques

Ces tests ne sont évidemment pas une panacée. Un certain nombre de propriétés sur les exigences ne peuvent pas ou difficilement être vérifiées automatiquement. Au delà de tests qui renvoient juste une réponse "l'exigence est correcte/incorrecte (et pourquoi)", comme par exemple vérifier qu'une exigence a une syntaxe correcte, nous proposons aussi des outils pouvant faciliter la vérification de certaines propriétés, par exemple en sélectionnant uniquement les exigences susceptibles d'être pertinentes.

Il faut également faire attention à ne pas considérer les tests, ou les règles dont ils proviennent, comme des marqueurs absolus de la qualité des exigences. Certains tests existent plutôt pour soulever des potentiels problèmes, qui seront ensuite examinés par des ingénieurs, plutôt que pour rejeter toutes les exigences qui ne respectent pas la règle. Par exemple, on souhaite généralement que les exigences soient "atomiques", c'est à dire qu'elle n'expriment qu'un seul fait. De cette règle on peut créer un test qui détecte les exigences de la forme "$X \wedge Y$" ou "$Z \implies (X \wedge Y)$". Ce n'est cependant pas une règle absolue, et des exigences de la forme "$Z \implies (X \wedge Y)$" peuvent être des exigences acceptables, il ne

faut donc pas les rejeter automatiquement.

3.4.2 Que peut-on tester ?

Nous avons mentionné, notamment en section 2.1.4.1, des “propriétés” ou “règles”, aussi appelés “critères” que les exigences devraient respecter (en anglais, on parle de “quality attributes”, formant un “quality model”). Quels sont ces critères et comment les classer et en déduire des tests automatiques sur les exigences ?

3.4.2.1 Listes de critères

Des exigences sur les exigences Ces critères traduisent des besoins sur les exigences elles-mêmes, ce sont en quelque sorte des méta-exigences. Comme les exigences, ces critères devraient être non ambigus et relativement précis. Cependant, pour des exigences écrites uniquement en langue naturelle, il est difficile d’écrire des critères formels : on demande par exemple que les exigences soient “claires” (“Une lecture de l’exigence suffit pour la comprendre, la structure de la phrase est simple et n’utilise pas les subtilités littéraires.”) Badreau et Boulanger (2014), mais il est difficile de définir formellement cette clarté.

On ne va pas chercher à appliquer les méthodes proposées dans ce travail à ces “méta-exigences”, la différence entre les éléments qu’elles spécifient (un système et ses exigences) est trop importante pour que cela ait un sens. Cependant, puisque l’on va essayer de transformer ces critères en tests informatiques, il peut être utile de réfléchir, par exemple, à l’ambiguïté de ces critères.

Origines de ces critères Il y a de nombreux critères sur les exigences et les corpus d’exigences que nous aimerions tester. Des listes de critères de qualité sont définies dans des normes (comme le standard IEEE 29148-2011 ISO/IEC/IEEE (2011)), des articles scientifiques (Saavedra et collab. (2013) par exemple), des standards industriels (comme ceux rassemblés dans le projet CESAR Rajan et Wahl (2013)) ou des livres de fondamentaux de l’ingénierie des exigences.

Comme on l’a mentionné dans la section 2.1.4.1, l’origine de ces critères est rarement explicitée (pour le projet CESAR, les critères proviennent de standards industriels, mais comment ces standards ont été écrits n’est pas précisé). On peut supposer que ces critères sont généralement dus au “bon sens” et aux “bonnes pratiques” plutôt qu’à des études scientifiques systématiques (ce qui peut s’expliquer par la difficulté que de telles études scientifiques nécessiteraient).

Remarques Ces différentes listes de critères sur les exigences sont relativement proches, et l’on retrouve souvent les mêmes critères d’une liste à l’autre. Ce qui ne veut pas nécessairement dire que ces critères sont très clairs. Par exemple, on a dans la plupart de ces listes un critère “les exigences ne doivent pas être ambiguës”, avec plus ou moins d’explications sur ce critère. Ce critère, comme d’autres, est assez large, flou et difficile à vérifier simplement.

Comme le notent Saavedra et collab. (2013), ces critères peuvent être (et sont souvent) des rassemblements de critères plus élémentaires. Dans l'exemple de la clarté cité plus haut ("Une lecture de l'exigence suffit pour la comprendre, la structure de la phrase est simple et n'utilise pas les subtilités littéraires."), on peut identifier trois éléments :

— "Une lecture de l'exigence suffit pour la comprendre"
— "la structure de la phrase est simple"
— "n'utilise pas les subtilités littéraires."

Ces différents sous-critères peuvent être plus ou moins "traduisibles" en tests automatiques. Par exemple, si l'on a formalisé la syntaxe de l'exigence, on peut dire des choses sur la "simplicité" de cette structure. Un critère pertinent pourrait être la profondeur de l'arbre syntaxique.

Certains des critères sont couverts par d'autres méthodes (comme la traçabilité des exigences) ou ne peuvent que difficilement être vérifiés automatiquement, mais notre méthode est appropriée pour d'autres.

3.4.2.2 Proposition de classifications

Pour y voir un peu plus clair dans tous ces critères, nous proposons divers systèmes de classification. Ces différentes classifications ne sont pas nécessairement indépendantes : par exemple, les critères portant sur le sens des exigences vont généralement être plus difficiles à traiter automatiquement.

Exigences seules ou corpus Un premier moyen de classer les critères est selon s'ils portent sur des exigences unitaires ou sur des ensembles d'exigences. Cette distinction est généralement identifiée dans les diverses listes de critères. Un exemple de critère portant sur des exigences unitaires est le critère d'atomicité ("L'exigence est un élément identifiable et non décomposable ; elle n'exprime qu'un seul fait.") Badreau et Boulanger (2014). Un exemple de critère sur les ensembles d'exigences est la cohérence ("Il n'y a pas d'ambiguïté, ni d'inconsistance interne du référentiel des exigences ; il n'y a pas de contradiction entre les exigences, il y a une identification unique du document.") Badreau et Boulanger (2014).

La limite n'est pas nécessairement nette entre ces deux types de critères. Par exemple, dans la définition de Badreau et Boulanger, le critère de complétude ("Aucune exigence ne manque et chaque exigence est complète.") porte à la fois sur les exigences unitaires ("chaque exigence est complète") et sur les ensembles ("Aucune exigence ne manque"). La distinction unitaire/ensemble reste un élément important des critères d'exigences, pour les raisons citées plus haut : vérifier des propriétés sur les ensembles d'exigences va demander une vision globale du corpus, alors que les critères sur des exigences unitaires peuvent être vérifiés localement.

Lexique, syntaxe, sémantique ? Un autre moyen de classification, qui n'est pas identifié dans les listes de critères que nous avons étudiés, est de s'intéresser aux trois niveaux lexique/syntaxe/sémantique. Il y a des critères qui portent plutôt sur les mots, sur la structure, ou sur le sens de l'exigence (ou de l'ensemble d'exigences). Lorsqu'on demande

qu'une exigence soit complète ("Il faut que l'ensemble des concepts utilisés dans l'exigence soient définis et qu'aucune information ne manque."), la partie "Il faut que l'ensemble des concepts utilisés dans l'exigence soient définis" concerne le niveau lexical. Le critère de clarté comprend le fait que "la structure de la phrase est simple", ce qui pointe vers le niveau syntaxique. Un exemple de critère concernant le sens, la sémantique, est que l'exigence doit être correcte ("L'exigence correspond à un besoin réel d'une partie prenante (cohérence externe).").

La plupart des critères que nous avons étudiés sont de plus d'un type. C'est encore une fois généralement dû au fait que ces critères rassemblent plus d'un sous-critère. (En reprenant le principe des exigences sur les exigences, on peut dire que ces critères ne sont pas "atomiques".)

Cette classification est pertinente pour les tests automatiques que nous souhaitons faire, puisqu'elle va correspondre assez bien avec les principes de base de notre travail.

Complétude, cohérence, justesse Un autre type de classification, introduit par Zowghi et Gervasi (2002) et appliqué en pratique dans le projet CESAR Rajan et Wahl (2013), sépare les critères selon s'ils portent sur la cohérence, la complétude ou la justesse ("consistency", "completeness", "correctness" en anglais). Dans le cadre du projet CESAR, la complétude consiste à vérifier qu'il ne manque pas d'exigences selon différents point de vues (environnemental, process, sûreté, traçabilité...). La cohérence porte sur les liens des exigences avec d'autres exigences :

— d'autres exigences dans la même spécification,
— ou des exigences de plus haut ou plus bas niveau.

Enfin, la justesse est une classe "pouvant être considérée comme arbitraire" qui rassemble différentes propriétés se focalisant plutôt sur des exigences unitaires (atomicité, vérifiabilité, réalisabilité...).

Des critères plus ou moins testables De manière générale, puisqu'on ne cherche pas à "comprendre" automatiquement le sens d'une exigence, les critères qui vont porter sur la sémantique vont être difficile à tester. Par exemple, on ne va pas chercher à trouver des incohérences au niveau du sens des exigences. Par contre, les critères qui portent sur le lexique et sur la syntaxe vont être plus faciles à tester.

On va classer cette "testabilité" des critères en trois catégories :

— Les critères testables automatiquement : si un programme de test peut répondre directement si oui on non le critère est respecté. On utilisera l'abréviation VA (pour Vérification Automatique).
— Les critères qui ne peuvent être vérifiés que par une intervention humaine, mais dont la vérification peut être simplifiée par un ou plusieurs programmes de test. On utilisera l'abréviation SO (pour Support Outil).
— Les critères qui sont complètement hors de notre contexte et que notre méthode ne peut pas traiter. On utilisera l'abréviation HC (pour Hors Contexte).

Cela ne concerne pas directement la testabilité, mais on peut aussi identifier des critères qui seront respectés "par construction" (PC) si l'exigence est écrite de manière correcte selon les principes que nous avons détaillés précédemment. Par exemple, une exigence ne peut

pas avoir une syntaxe ambiguë si celle si respecte la syntaxe formelle que nous avons définie. Un autre exemple est que, dans une exigence dotée de la syntaxe proposée et de liens explicites, les mots utilisés ne peuvent pas être ambigus : soit ils sont liés explicitement à leur définition, écrite dans un modèle, soit ils représentent une fonction syntaxique autorisée (donc a priori non ambiguë).

Un exemple de critère testable automatiquement est l'homogénéité des termes utilisés, c'est à dire "est-ce que le même terme est utilisé à chaque fois que l'on fait référence à la même chose ?".

Un exemple de critère "SO" est la complétude de l'ensemble d'exigences du point de vue architectural : est-ce que toutes les interfaces du systèmes sont spécifiées, et est-ce que pour chaque interface, elle est complètement spécifiée ?

Vérifier que les exigences soient correctes, c'est à dire qu'elles "correspondent à un besoin réel d'une partie prenante", est un exemple de critère HC.

Nous donnons des tests réalisables avec notre approche et des scénarios où on utilise ces tests en section 4.3.

3.5 Codéveloppement entre exigences et modèles

Nous ne voyons pas nécessairement les étapes de modélisation et de spécification comme des étapes purement séquentielles, comme dans un processus "waterfall" idéal. Il est logique et préférable que, lors de l'étape de spécification, non seulement on se base sur des modèles construits avant, mais on affine également ces modèles. On peut détailler ces échanges entre modélisation et spécification.

3.5.1 Modèles nourrissant les exigences

3.5.1.1 Modèles précisant les exigences

Comme nous l'avons présenté précédemment, nous cherchons à utiliser des modèles afin de préciser les exigences. En plus de les utiliser comme définitions, ces modèles peuvent aussi pointer vers des éventuels manques ou problèmes dans les exigences. Par exemple, s'il est précisé dans un modèle d'architecture que le système interagit avec son environnement par un certain nombre d'interfaces, mais que certaines de ces interfaces ne sont pas mentionnées dans les exigences, alors on peut supposer que les exigences sont incomplètes.

3.5.1.2 Traduction des modèles vers les exigences ?

On peut aussi imaginer d'avoir un programme qui traduirait automatiquement des modèles en exigences : on a mentionné précédemment dans la section 3.2.3.4 que dans une certaine mesure, certaines propriétés peuvent être écrites indépendamment dans une exigence ou dans un modèle. S'il est nécessaire pour une partie prenante que les spécifications soient rédigée en langue naturelle, alors que d'autres parties prenantes utilisent des modèles,

un script relativement simple permettrait de générer des exigences à partir de certains modèles.

Par exemple, si l'on reprend l'exemple de la section 3.2.3.4, chaque ligne du tableau peut être remplacée par un gabarit à trou : "le système doit <acquérir/envoyer> le signal <Type> <Nom> <du/vers le> système <Systeme>, le signal est <TypeConnexion>, envoyé par <Connecteur>, <Correspondance>", où les trous sont remplis par les cases du tableau correspondantes. On peut réaliser la même chose avec un automate à états.

La question de savoir si ces exigences sont vraiment des exigences reste entière, mais cela n'empêche pas de le faire en pratique.

3.5.2 Exigences nourrissant les modèles

3.5.2.1 De meilleures exigences demandent de meilleurs modèles

Nous avons remarqué qu'essayer d'écrire de meilleures exigences avait aussi tendance à nécessiter l'amélioration ou la création de modèles. Si l'on écrit une exigence mentionnant un concept qui ne peut pas être relié à un élément de modèle pertinent, on peut se poser plusieurs questions :

- Est-ce qu'il existe un concept proche mais pas équivalent/pas assez défini dans un modèle déjà intégré à la spécification ?
- Est-ce que ce concept apparaît dans d'autre exigences ?
- Est-ce qu'il serait utile d'avoir un modèle pour définir ce concept ?
- Est-ce qu'un modèle déjà existant pourrait servir de référence ?

Selon la réponse à ces questions, il pourrait être utile de modifier/créer un modèle afin de servir de référence à ce concept. Évidemment, si ce concept n'est pas central dans la spécification, n'est utilisé qu'une seule fois dans les exigences et qu'il faudrait créer un modèle à partir de rien pour servir de référence, cela ne semble pas rentable.

En cherchant à formaliser la syntaxe des exigences, on a besoin d'information sur les concepts qui apparaissent dans les exigences. Par exemple, si l'on a dans une exigence "Quand X, la sortie S6 doit être supérieure à 10V", Cela implique que la sortie S6 est une tension, et les règles de notre syntaxe formelle vont demander que ce soit indiqué explicitement. Un bon endroit où ajouter cette indication est le modèle où est définie "S6", en ajoutant par exemple un attribut "tension" à l'élément "S6".

3.5.2.2 Remplacer des exigences textuelles par des modèles

Un des inconvénients des exigences textuelles est que leur nombre peut rapidement devenir important pour des systèmes complexes. Un certain nombre d'exigences ont généralement une structure similaire et un sens proche. Il est parfois possible de placer ces exigences dans un modèle non textuel, comme l'exemple de la section 3.2.3.4. On ne peut généralement pas mettre toutes les exigences dans des modèles. Pour remplacer une exigence par un (élément de) modèle, on peut se demander :

- Est-ce que cette exigence est une "vraie" exigence (pas une hypothèse ou une supposition par exemple) ?

— Est-ce qu'il existe d'autres exigences avec une structure et un sens proche et facilement modélisable ?
— Si l'exigence est centrale dans la spécification, ce n'est probablement pas une bonne idée de la mettre sous forme de modèle : cela risque de la "cacher".
— Est-ce que les questions de traçabilité risquent de compliquer le travail en aval ? Un modèle "contenant" beaucoup d'exigences va être la source de beaucoup d'exigences de niveau inférieur/de méthodes de tests/etc. ce qui peut poser un problème.
— Est-ce que l'exigence serait plus claire en utilisant un modèle ? Plus simple à écrire ?

3.6 Remarques méthodologiques

Nous rassemblons ici un certain nombre de remarques relatives au déroulement de la thèse, éventuellement exprimées ailleurs dans ce rapport de thèse, qui nous semblent pertinentes.

3.6.1 Spécificités de l'ingénierie des exigences

L'étude de l'ingénierie des exigences (IE) présente un certain nombre d'éléments particuliers.

Premièrement, ce n'est pas un sujet qui a été beaucoup abordé lors de mon éducation antérieure à la thèse (concrètement, quelques cours sur les "cahiers des charges" au niveau collège et des références à l'IE lors des cours d'ingénierie système au niveau post-bac). Je ne connaissais donc pratiquement pas le domaine de l'ingénierie des exigences au début de la thèse. Une réflexion sur la formation initiale à l'ingénierie des exigences pourrait être utile, notamment parce qu'une partie des solutions possibles au problème de l'ambiguïté de la langue naturelle sont difficilement applicables en principe, parce qu'elles nécessitent justement trop de formation.

Nous pensons que c'est aussi un sujet qui est bien mieux traité en connaissant comment l'ingénierie des exigences est utilisée et réalisée en pratique dans l'industrie. Si nous n'avons pas écrit de véritables spécifications pour un projet industriel, nous espèrons que les études de spécifications réelles et les discussions avec les ingénieurs des exigences à Safran nous ont donné une connaissance suffisante du problème. Cette connaissance et ce point de vue industriel sont nécessaires pour proposer des solutions qui soient applicables en pratique.

Les interactions avec le monde industriel sont intéressantes scientifiquement, mais présentent aussi des inconvénients. En particulier, si l'on se place dans le cadre d'une démarche hypothético-déductive, expérimenter une méthode d'ingénierie des exigences (ou d'ingénierie système en général) complète est très complexe :

— Premièrement, il est difficilement faisable de demander à un grand groupe industriel de tester une nouvelle méthode (si tant est que cette méthode soit finalisée, ce qui n'est pas le cas ici) pour le développement de ses systèmes réels.
— Deuxièmement, même si l'expérience était possible, isoler les effets de cette nouvelle méthode des autres effets pouvant influencer les résultats serait également complexe.

Il est bien sûr possible de tester des choses plus simples qu'une méthodologie complète,

par exemple en proposant des exigences à des lecteurs et en leur demandant quelles sont les exigences (ou les groupes d'exigences) les moins ambiguës. Ce type d'expériences, bien que plus simple, demande quand même un certain travail pour être réalisé correctement, et nous n'avons pas eu le temps de le faire.

Finalement, l'ingénierie des exigences est à l'intersection d'un monde non-formel, les besoins, et d'un monde formel, ce que doit effectivement faire le système. Cette position nous empêche d'utiliser des raisonnements complétement formels, tout en ouvrant la voie à des processus plus efficaces que s'ils étaient complètement non-formels.

3.6.2 Quelques étapes de la thèse

Nous avons commencé le travail de cette thèse par une étape assez classique d'étude bibliographique, tout en discutant avec nos interlocuteurs à Safran. La possibilité d'avoir accès à des spécifications industrielles réelles (comme celle de l'Electric Green Taxiing System (EGTS)) a été d'une aide inestimable. À partir de ces éléments, nous avons commencé à développer des solutions aux problèmes formulés par Safran.

Nous avons testé ces solutions de façon "artisanale" grâce aux spécifications à disposition : en particulier, nous avons essayé de réécrire les exigences de ces spécifications en exigences de meilleure qualité (moins ambiguës, plus facilement traitables informatiquement). Une fois les principes de notre méthode à peu près fixés, nous avons réalisé une "traduction" plus complète de la spécification EGTS, en écrivant les exigences et les modèles sous des formats lisibles par des programmes informatiques. En même temps, nous avons écrit ces programmes de lecture et de test, les deux opérations étant difficilement dissociables.

Nous avons également développé l'exemple de l'ABS, afin d'avoir un exemple un peu complexe pouvant servir d'illustration, mais aussi pour pouvoir développer un système et ses exigences en même temps, et ne pas se limiter à des traductions de spécifications déjà existantes. La spécification partielle de l'ABS et son processus de conception que nous avons réalisés ne visent pas à être parfaitement réalistes, mais ils ont bien servi les objectifs fixés. Finalement, nous avons écrit un prototype d'interface graphique, présenté dans le chapitre 6, pour permettre de réaliser les tests plus facilement.

Chapitre 4

Mise en oeuvre

4.1 Exemple d'illustration : l'ABS

4.1.1 Contexte

Dans cette section, nous présentons un exemple de système industriel, qui sera utilisé comme illustration dans la suite. Nous avons choisi l'antiblockiersystem (ABS) comme exemple parce qu'il peut être compris relativement simplement, tout en ayant des caractéristiques similaires aux autres systèmes que nous avons étudié (surtout des systèmes aéronautiques). Parmi ces caractéristiques, on remarque que :

— l'ABS comprend des parties physiques aussi bien que logicielles,
— la plupart de ses interactions sont avec d'autres systèmes techniques,
— il peut être décomposé en sous-systèmes,
— c'est un système critique.

Un ABS, ou système anti-blocage des roues, est un système mécatronique dont le but est d'empêcher les roues d'un véhicule de se bloquer pendant le freinage. Le blocage des roues doit être évité car il implique que les roues glissent sur la chaussée, ce qui pose plusieurs problèmes :

— véhicule incontrôlable,
— adhésion moins importante (et donc distance de freinage augmentée) sur la plupart des surfaces,
— risques d'éclatement ou de combustion des pneus (en particulier pour les avions).

Le principe de l'ABS est que, quand certaines conditions indiquant un blocage des roues imminent sont détectées, le système relâche les freins pour éviter ce blocage. Le freinage normal est repris quand les conditions indiquant un blocage ne sont plus vérifiées. Ce cycle de freinage/relâche peut être répété plusieurs fois par seconde.

Dans ce travail nous considérerons un ABS installé dans une automobile. Dans les voitures modernes, il y a généralement un système anti-blocage des roues installé pour chaque roue. Pour garder les explications simples, nous considérerons uniquement un seul ABS agissant sur une roue, indépendamment des autres roues et systèmes anti-blocage. De plus, nous considérerons que la relâche des freins est binaire : soit le système transmet le liquide de freinage jusqu'aux freins comme s'il n'était pas installé, soit la pression de

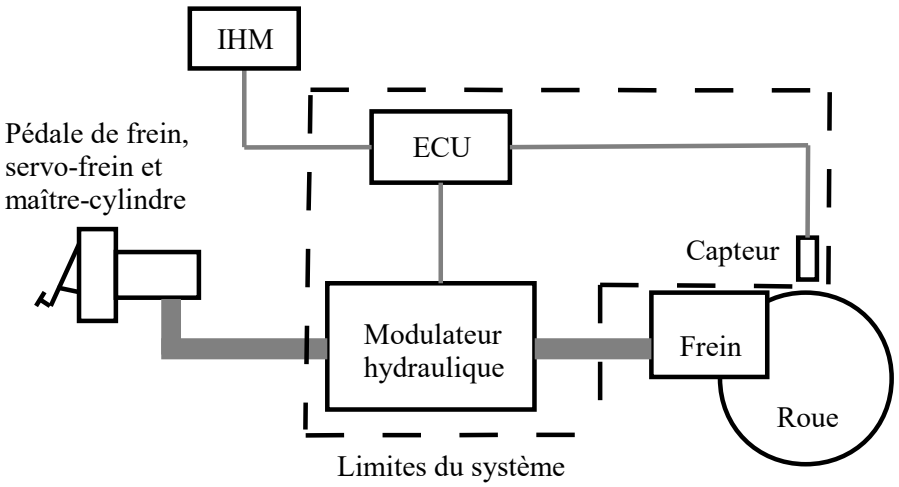


FIGURE 4.1 – Le système anti-blocage des roues et son contexte

freinage est mise à zéro.

4.1.2 Architecture et comportement

La figure 4.1 représente l'environnement de l'ABS et une simple décomposition de ce système. Quand le contrôleur (electronic control unit, ECU) détecte un risque de blocage des roues, le modulateur hydraulique isole le fluide hydraulique dans le maître cylindre du fluide hydraulique dans les freins et relâche la pression dans les freins. Une architecture possible du modulateur hydraulique est détaillée en figure 4.2. Le système détecte quand les freins doivent être relâchés en calculant le glissement de la roue contrôlée. Ce glissement est calculé en utilisant la vitesse de rotation de cette roue et la vitesse du véhicule. La vitesse du véhicule est approximée en se basant sur les vitesses de rotation des autres roues, elles-même mesurées par des capteurs. Le contrôleur communique également avec l'interface homme-machine et/ou un ordinateur de bord de la voiture. Ces éléments communiquent avec le conducteur (par exemple grâce à un avertisseur lumineux sur le tableau de bord) et avec d'autres systèmes.

Nous donnons des modèles plus détaillés et "formels" dans la suite, au fur et à mesure de l'étude des exigences et des éléments qu'elles contiennent.

4.1.3 Exemples d'exigences

Dans cette partie, nous donnons des exemples d'exigences pour ce système. Encore une fois, ils serviront d'illustration dans ce rapport. La forme et le contenu de ces exemples sont similaires à ceux des exigences réelles trouvées dans des spécifications industrielles.

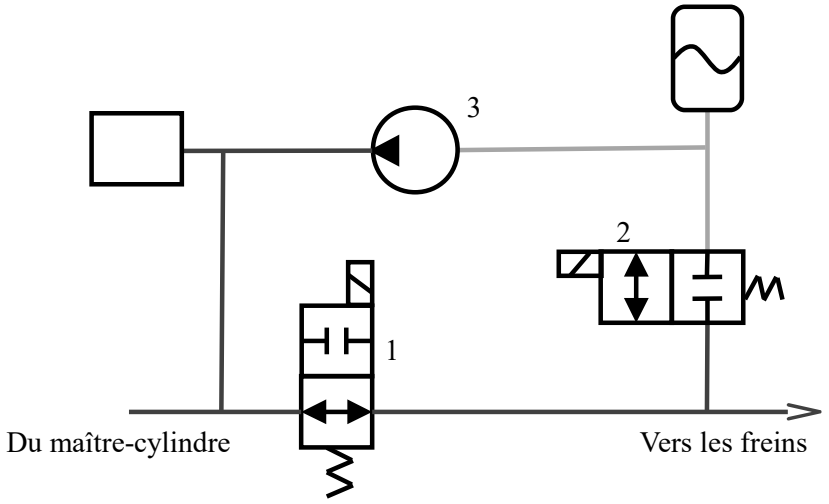


FIGURE 4.2 – Une décomposition possible du modulateur hydraulique (adapté de REIF (2015))

Exi. 1 *Quand la puissance d'alimentation est plus petite que 10mW, la pression dans les freins doit être égale à la pression dans le maître cylindre.*

Cette exigence spécifie que, quand il n'y a pas ou peu d'alimentation électrique, le système de freinage doit transmettre la pression du maître-cylindre aux freins comme si l'ABS n'existait pas.

Exi. 2 *Le système ne doit pas être endommagé par une température ambiante comprise entre -30°C et 60°C.*

Exi. 3 *Le système ne doit pas être endommagé par une pression dans le maître-cylindre inférieure à 60 bars.*

Une partie non-négligeable des spécifications que nous avons étudiées était constituée d'"exigences environnementales". Ces exigences décrivent les conditions environnementales que le système doit être capable de supporter et/ou les performances dans ces conditions.

Exi. 4 *Dans l'état Marche, la probabilité d'une défaillance empêchant le conducteur de diriger le véhicule doit être inférieure à* 10^{-8} *par heure.*

Exi. 5 *La probabilité d'une défaillance non-détectée empêchant le système d'appliquer une pression dans les freins de plus de 20 bars dans l'état Freinage doit être inférieure à* 10^{-8} *par heure.*

Une partie des exigences industrielles concerne des aspects de sûreté. Ces deux exemples spécifient la probabilité maximale qu'un événement indésirable survienne.

Exi. 6 *Dans l'état Arrêt, et quand la pression dans le maître-cylindre est plus faible que 50 bar, alors la pression dans les freins doit être égale à la pression dans le maître cylindre.*

Des automates à états sont souvent utilisés pour décrire le comportement des systèmes. Nous parlons plus en détail de ces états dans la section 4.2.2.2.

Exi. 7 *Le système ne doit pas demander une intensité de plus de 1A sur l'alimentation électrique.*

Cette exigence peut être nécessaire pour limiter la charge sur l'alimentation électrique.

Exi. 8 *Il doit être possible de remplacer les éléments hydrauliques de l'ABS en moins de 2 heures.*

Exi. 9 *Après une défaillance causant un détachement total ou partiel de pièce, les éléments du système ou de fixation doivent rester à l'intérieur de leur enveloppe de conception.*

Exi. 10 *Le système doit être conçu de manière à minimiser les possibilités d'erreurs humaines qui réduiraient significativement la sécurité pendant la maintenance et l'opération.*

On note que les exigences sont, en général, assez hétérogènes.

4.2 Construction d'une exigence

Dans cette section, nous présentons et détaillons des concepts qui pourraient selon nous être utiles pour les spécifications et illustrons ces concepts avec l'exemple de l'ABS.

L'ABS nous a aussi servi à expérimenter nos idées : nous avons écrit une spécification (partielle) de l'ABS et les modèles associés. Nous ne prétendons pas présenter un rapport parfaitement réaliste des processus de conception d'un ABS parfaitement réaliste. Nous avons essayé de partir d'un départ plausible, avec un système peu spécifié, et d'arriver à une fin plausible, avec un système davantage spécifié. Nous partageons ici les commentaires, problèmes et possibles solutions qui sont apparues dans ce processus.

4.2.1 Éléments de départ : interfaces du système

Nous avons besoin d'un point de départ : nous pourrions commencer avec un but de très haut niveau, par exemple "on souhaite empêcher le blocage des roues pendant le freinage tout en étant capable de freiner". Cependant nous voulons nous concentrer sur l'écriture des exigences. Partir de ce but nous demanderait d'étudier les activités évoquées en section 2.1.1.1, telles que :

— identifier l'environnement et les parties prenantes du système,
— identifier les besoins de ces derniers, qui peuvent être nombreux et incompatibles,
— analyser ces besoins, les négocier...

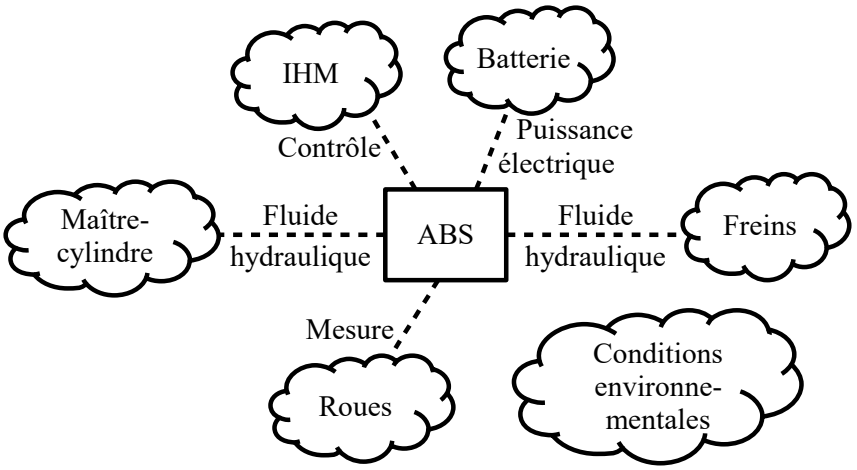


FIGURE 4.3 – Environnement de l'ABS

Même si ces problèmes sont intéressants et méritent l'attention que la communauté de l'IE leur porte, ils ne sont pas le sujet de notre travail.

De plus, nous avons observé que dans l'industrie, la conception partait rarement d'une page blanche, en particulier pour les systèmes physiques. Par exemple, si l'on souhaite spécifier un système B qui est inclus dans un plus gros système A, les exigences de B sont souvent construites à partir de celle de A.

Nous avons choisi un point de départ pour le processus où une partie de l'environnement est grossièrement définie comme montré dans la figure 4.3. Il est intéressant de noter que simplement en définissant une ébauche de l'environnement d'un système, le client guide déjà la solution. Par exemple, on pourrait imaginer un système (même s'il ne serait pas optimal) qui, au lieu de détourner le liquide de freinage, avertirait le conducteur qu'un blocage des roues est imminent pour qu'il relâche les freins lui-même. En écrivant que le système doit interagir directement avec le fluide hydraulique venant du maître-cylindre et allant aux freins, nous empêchons cette solution alternative. Cela tend à montrer qu'une spécification ne peut pas être une "boîte noire" parfaite, où le fournisseur est complètement en charge de la conception du système.

Le choix de l'environnement n'est pas définitif : le fournisseur peut négocier avec le client pour modifier l'environnement pris en compte. Par exemple, le fournisseur de l'ABS pourrait demander de pouvoir accéder à des données générées par le véhicule (vitesse, rapport de transmission...) qui ont été oubliées ou pas considérées nécessaire par le client.

4.2.1.1 Modèles d'architecture

Environnement Nous avons écrit en section 3.1.1.1 que les exigences devraient décrire l'environnement du système, pas le système lui-même. Plus précisément, nous pensons que les exigences devrait être focalisées sur les parties de l'environnement qui interagissent directement avec le système. Nous discuterons dans la section 4.2.2 ce que l'on peut faire

concernant les éléments de l'environnement qui n'interagissent pas directement avec le système.

Boîtes et flèches Nous nous intéressons donc aux interfaces entre le système et son environnement. Ce sont les éléments que nous devrions retrouver dans les exigences, et donc, par extension, dans les modèles auxquels ces exigences font référence. Une manière classique de décrire un système parmi son environnement est d'utiliser des modèles architecturaux, évoqués en section 3.1.2.3, tels que S2ML Batteux et collab. (2015) ou l'Internal Block Diagram de SysML Friedenthal et collab. (2014). Puisque ce type de modèle est très répandu dans l'industrie, il a au moins deux avantages :

— la plupart des ingénieurs connaissent le principe de ce type de modèle,
— quand les ingénieurs commencent à écrire la spécification d'un système, il est probable qu'il existe déjà un modèle décrivant les interactions de ce système avec son environnement, ou au moins que ce modèle soit en cours de construction.

Dans ce type de modèle, il y a généralement des boîtes représentant les systèmes et sous-systèmes et des flèches ou des lignes connectant ces systèmes qui représentent les interfaces. Nous allons utiliser un modèle de ce type comme définition des interfaces qui seront utilisées dans la spécification. Lorsque toutes les interfaces mentionnées dans les exigences sont (correctement) liées explicitement avec le modèle de définition, il est par exemple facile de dire si oui ou non deux exigences font référence au même concept. C'est quelque chose qui peut être difficile à faire dans les spécifications rédigées uniquement en texte libre.

Boîtes plus ou moins noires Nous avons écrit plus haut, comme d'autres auteurs le mentionnent(e.g. "une spécification ne devrait contenir que des informations sur l'environnement" Zave et Jackson (1997)), que les exigences ne devraient pas spécifier ce qui est à l'intérieur d'un système, mais devraient seulement décrire son environnement. Cependant une telle approche en "boîte noire" sans compromis peut créer des difficultés.

Par exemple, on pourrait vouloir exprimer l'Exi. 8 "Il doit être possible de remplacer les éléments hydrauliques de l'ABS en moins de 2 heures". A priori, c'est une exigence valable : s'il y a une fuite de liquide de frein par exemple, on peut vouloir remplacer uniquement les éléments du système qui peuvent être à l'origine de cette fuite. Cependant, sans décrire l'intérieur de l'ABS, il va être difficile de donner une définition des "éléments hydrauliques de l'ABS". Le schéma 4.4 donne un exemple du modèle d'architecture mis à jour, où "modulateur hydraulique" représente les "éléments hydrauliques de l'ABS". On peut utiliser le même type de modèle d'architecture pour décrire l'intérieur et l'environnement du système.

Quel point de vue ? Dans certaines méthodes d'ingénierie système, comme celle développée par CESAMES Krob (2017), on identifie 3 types de points de vue sur un système : les visions opérationnelle, fonctionnelle et organique. Ces visions cherchent à séparer les étapes de conceptions en partant du plus abstrait et "boîte noire" (opérationnel) au plus concret et "boîte blanche" (organique). Dans quelle vision nous plaçons nous pour écrire les modèles qui nous intéressent dans ce travail ?

Les modèles que l'on veut utiliser doivent correspondre aux exigences. Dans notre

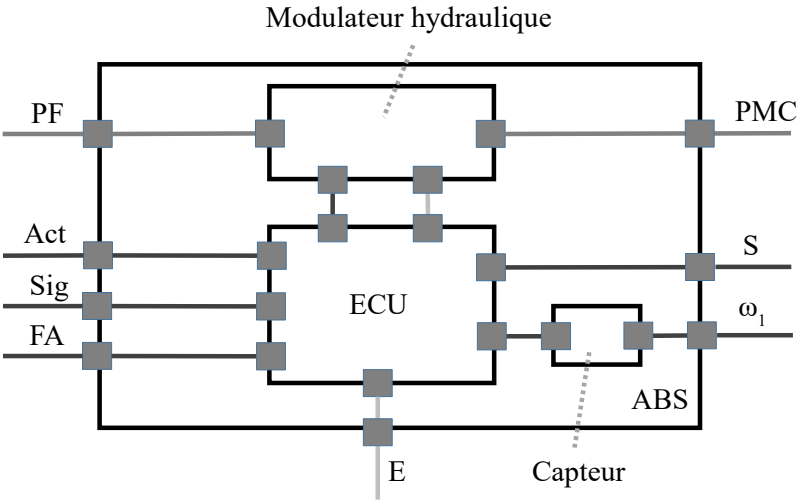


FIGURE 4.4 – Modèle d'architecture des composants de l'ABS

contexte, même si les exigences seront plutôt "organiques", elles ne peuvent pas être toutes classées comme ça. On cherche à éviter les exigences de trop haut niveau (fonctionnelles et opérationnelles) car elles ne seront pas assez précises et concrètes. Mais on cherche également à écrire des exigences qui ne rentrent pas trop dans les détails : on va davantage s'intéresser à la valeur booléenne d'une entrée/sortie qu'aux processus physiques qui vont régir cette valeur.

Une autre remarque est que, même si on leur assigne un sens (une sémantique) différent, les modèles d'architecture opérationnels, fonctionnels et organiques sont syntaxiquement très proches : ils sont composés de boîtes et de flèches, que ces boîtes soient des systèmes, des fonctions ou autres. Comme nous ne cherchons pas dans ce travail à comprendre le sens ni des exigences ni des système associés, au final, il importe peu que ces systèmes soient considérés comme fonctionnels ou organiques. Nous cherchons juste à savoir que tel mot de telle exigence fait référence à tel élément de tel modèle.

Grammaire formelle utilisée Nous avons utilisé une grammaire pour la définition des modèles d'architecture utilisés dans notre travail. Cette grammaire est minimale et peut être étendue (par exemple en introduisant des concepts de classes, d'extensions, de packages, etc.) ou modifiée pour mieux correspondre à des grammaire existantes. La grammaire est donnée dans la figure 4.5.

Concrètement, un modèle d'architecture commence par un bloc système englobant tous les éléments du modèle, ce bloc système étant éventuellement "fictif" (e.g. "univers" ou autre). Un bloc système a un nom, et peut avoir une liste d'attributs, des ports, d'autres blocs systèmes et des connections internes. Une liste d'attribut est une liste de variables, dont chacune a un nom et un type. Un port a un nom et un type. Une connection relie au moins deux adresses (ce n'est pas précisé ici, mais ces adresses doivent faire référence à des ports).

Symboles terminaux : {Name, Type, Address}
Symboles non-terminaux : {ModelArchitecture, Block, Attributes, Var, Port, Connection}
Axiome : ModelArchitecture

Règles de production ("*" est l'étoile de Kleene) :
- ModelArchitecture → Block
- Block → Name Attributes Port* Block* Connection*
- Attributes → Var*
- Var → Name Type
- Por → Name Type
- Connection → Address Address Address*

FIGURE 4.5 – Grammaire formelle pour les modèles d'architecture

4.2.1.2 Modélisation des interfaces

Abstraction des interfaces On cherche à abstraire l'environnement de manière à pouvoir l'appréhender facilement et exprimer des propriétés concrètes le concernant. Un besoin concernant un système pourrait être "le système doit résister à des conditions environnementales normales". C'est un besoin réel et nécessaire, mais les exigences doivent être plus concrètes et explicites que ça. Si on est capable d'écrire des abstractions représentant les propriétés de l'environnement qui nous intéressent, on peut écrire des exigences concrètes. Par exemple, si on suppose que la seule condition environnementale qui nous intéresse est la température ambiante, l'exigence correspondante pourrait ressembler à l'Exi. 2 : "Le système ne doit pas être endommagé par une température ambiante comprise entre -30°C et 60°C.".

Niveaux d'abstraction Le choix du niveau d'abstraction est important : on a considéré ici que la "température ambiante" pouvait être représentée par une simple valeur scalaire, mais on pourrait aussi la représenter par un champ scalaire 3D ou même en décrivant le mouvement brownien des particules. Évidemment, dans ce contexte particulier, le dernier choix est absurde. Cependant, il est important de noter que, puisque toute description du monde réel est une abstraction d'une façon ou d'une autre, le choix des abstractions à utiliser est un choix fait par l'auteur d'une spécification.

Puisque les exigences sont des outils de communication, en plus d'être capables de décrire de façon adéquate le monde réel, les abstractions choisies doivent être compréhensibles par les lecteurs de la spécification. Dans l'exemple précédent, le choix de considérer la température ambiante comme un champ scalaire ou comme une simple valeur scalaire est un choix entre deux modèles qui peuvent être utilisés pour la communication entre humains comme pour la prédiction. Ces modèles sont utilisés dans des contextes différents.

L'avantage d'utiliser une valeur scalaire est que tout le monde comprend ce que signifie ce modèle, puisqu'on l'apprend à l'école et qu'on l'utilise dans la vie courante. Les champs scalaires sont bien moins répandus. De plus, les abstractions plus complexes rendent les documents d'exigences plus difficiles à lire, puisqu'elles contiennent plus d'information.

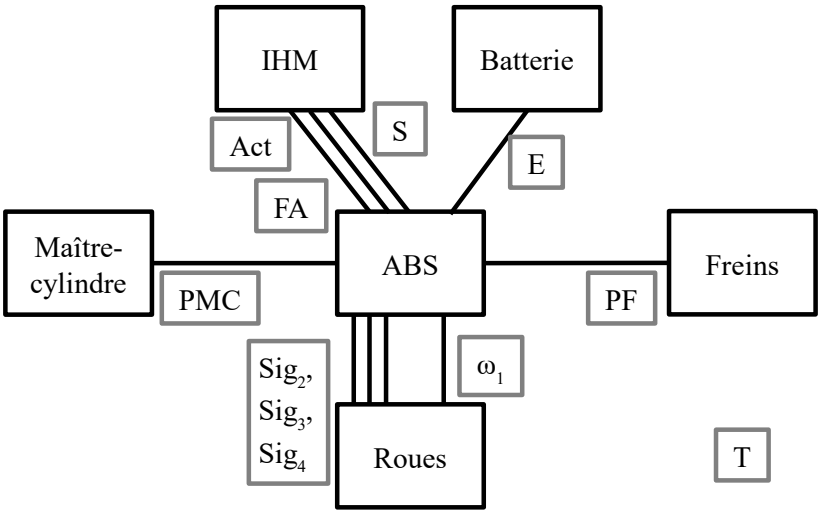


FIGURE 4.6 – Interfaces détaillées de l'ABS

Évidemment, c'est aussi l'avantage de ces représentations plus complexes : on peut décrire des phénomènes plus précis.

Pour l'exemple de l'ABS, nous avons essayé de modéliser le plus possible l'environnement, tout en utilisant des abstractions facilement compréhensibles. Des interfaces détaillées de l'ABS sont présentées en figure 4.6 et dans le tableau 4.1 L'environnement présenté n'est pas complet : nous nous sommes principalement intéressés aux systèmes techniques qui interagissaient avec l'ABS et aux choses qui pouvaient facilement être abstraites.

Références explicites Écrire des abstractions de l'environnement du système a un autre but : on est maintenant capable de faire référence à ces abstractions, nous les avons réifiées.

Supposons que l'on souhaite exprimer que "en cas de perte de l'alimentation, l'ABS ne devra pas s'opposer au passage du fluide hydraulique du maître-cylindre aux freins". Nous pouvons écrire l'exigence "Quand [la puissance d'alimentation] est plus petite que 10mW, [la pression dans les freins] doit être égale à [la pression dans le maître cylindre]". En écrivant une exigence de cette manière, nous savons maintenant exactement ce à quoi l'exigence fait référence : ici aux abstractions "Puissance d'alimentation" (E), "Pression dans les freins" (PF) et "Pression dans le maître-cylindre" (PMC). On a, comme présenté dans la section 3.3.3, utilisé des éléments de modèles comme définitions des termes dans les exigences.

En plus des apports sur le lexique comme l'absence d'ambiguïté des termes, cela permet également d'aider, voire d'automatiser, des vérifications de syntaxe : par exemple de vérifier que PMC est une pression et ne peut donc pas être comparée à une force, ou que E peut être comparée à une puissance électrique.

TABLEAU 4.1 – Interfaces du système

PMC	Pression dans le maître-cylindre
PF	Pression dans les freins
S	Statut du système
Act	Commande d'activation
FA	Frein appuyé
ω_1	Vitesse de rotation de la roue contrôlée
Sig_2, Sig_3, Sig_4	Images des vitesses de rotation des autres roues
E	Puissance d'alimentation
T	Température ambiante

Attributs Nous avons vu que PMC, PF et T sont des abstractions, et sont généralement relativement simples pour des questions de clarté et de compréhensibilité. Un moyen de rendre de l'information perdue lors de l'abstraction peut être d'ajouter des "attributs" à ces abstractions. Par exemple, l'ABS échange des données avec le reste du véhicule, comme le statut du système, ou des commandes de l'IHM. On peut voir ces données d'un point de vue purement fonctionnel : l'IHM envoie un booléen qui commande l'activation ou la désactivation du système. Cependant, ces données sont transmises par des phénomènes physiques et des composants physiques. Dans une exigence, on ne veut généralement pas voir le détail complet, par exemple, du dimensionnement des broches et fiches, mais ce type d'information va être nécessaire au fournisseur. Heureusement, ces genres de choses sont généralement normalisées : on peut spécifier que l'IHM enverra la commande marche/arrêt en utilisant, par exemple, un bus CAN avec un fiche mâle DE9. Nous pensons qu'il est plus simple de donner cette information dans un modèle définissant les interfaces que dans une exigence en langue naturelle.

On peut utiliser des attributs pour préciser les abstractions introduites plus haut, comme nous l'avons fait dans la table 4.2.

De cette manière, nous pouvons spécifier le contenu de ces abstractions (la commande booléenne) ainsi que la forme (quel protocole, utilisant quels composants). La signification de ces attributs peut être décrite explicitement, en créant par exemple un lien de "bus CAN" vers la norme qui définit ce standard.

Intervalles Quand on utilise des grandeurs physiques à valeurs réelles dans les exigences, il est nécessaire d'être attentif aux égalités. Par exemple, dans l'Exi. 1, on aurait pu écrire "Quand la puissance d'alimentation est nulle" plutôt que "quand la puissance d'alimentation est plus petite que 10mW". Cependant, la première condition n'est formellement jamais vraie,

TABLEAU 4.2 – Interfaces du système et leurs attributs

Abréviation	Nom complet	Unité	Valeurs possibles	Autre
PMC	Pression dans le maître-cylindre	Bar	Nombres réels	Tuyau diamètre D
PF	Pression dans les freins	Bar	Nombres réels	Tuyau diamètre D
S	Statut du système		{Marche, Arrêt, Défaillance}	Bus CAN
Act	Commande d'activation		Booléen	bus CAN
FA	Frein appuyé		Booléen	bus CAN
ω_1	Vitesse de rotation de la roue contrôlée	Rad/s	Nombres réels	
Sig_1, Sig_2, Sig_3	Images des vitesses de rotation des autres roues	V	Nombres réels	
E	Puissance d'alimentation	Watt	Nombres réels	ligne continue 12V
T	Température ambiante	℃	Nombres réels	

puisque la puissance d'alimentation n'est jamais exactement égale à zéro. C'est un problème, puisque le fournisseur doit alors deviner la condition pour laquelle il doit considérer la puissance comme "nulle".

Pour ce type de grandeurs à valeurs réelles, il vaudrait mieux utiliser des inégalités (ou des intervalles de tolérance) dans les exigences. Cela limite cependant la lisibilité : pour la fin de cette même exigence, écrire "la pression dans les freins doit être plus grande que 95% de la pression dans le maître-cylindre et plus petite que 105% de la pression dans le maître-cylindre" est moins clair qu'écrire "la pression dans les freins doit être égale à la pression dans le maître-cylindre".

Seuils Si nous utilisons des inégalités, comment choisir les seuils ? L'Exi. 5 pourrait être une exigence construite à partir du but "On souhaite éviter les défaillances qui empêchent le conducteur de freiner". La limite des "20 bars" est choisie comme un seuil : au dessous de 20 bars, on considère que le conducteur ne peut pas freiner. Ce seuil peut être vu comme arbitraire : a priori le freinage sera pratiquement le même avec une pression de 19,99 bars et avec une pression de 20,01 bars. On a cependant besoin d'une limite pour écrire une exigence complète et non-ambiguë. Un moyen pour éviter cet effet de seuil peut être d'ajouter plus d'exigences (avec 30 bars, 20 bars, 10 bars...) pour différents niveaux de défaillances.

De plus, on ne regarde ici qu'une seule dimension, si la pression est suffisante ou non, mais on pourrait aussi ajouter des conditions temporelles. Par exemple, on peut vouloir spécifier qu'une sortie retardée est moins dangereuse que pas de sortie du tout, ou que des dépassements transitoires d'un seuil sont acceptables.

Rajouter ces précisions à une spécification demande soit d'écrire (et donc de gérer) plus d'exigences, soit d'utiliser des moyens plus expressifs pour décrire les propriétés que l'on souhaite. Par exemple, au lieu d'avoir un ou plusieurs seuils et d'écrire une exigence pour chaque seuil, on pourrait définir une fonction continue qui mettrait en relation la gravité d'une défaillance avec le seuil de probabilité que cette défaillance survienne. Cette approche présente aussi des défauts. Premièrement, le but des exigences est la communication : il faut s'assurer que tous les éléments que l'on utilise soient compris de la même manière par tout le monde, de préférence sans nécessiter une formation importante. Deuxièmement, ajouter des détails très précis n'est pas utile si le client et/ou le fournisseur n'ont pas de méthodes pour tester, vérifier ou calculer ces détails.

4.2.1.3 Liens exigences-architecture

Quel sens pour ces liens ? Dans les spécifications industrielles actuelles, les liens explicites que nous proposons entre texte des exigences et modèles n'existent pas. Le sens associé à ces liens est assez simple : quand on a un lien entre un mot d'une exigence et un élément de modèle, cela signifie que ce mot représente l'élément de modèle.

Cela permet de dire si deux exigences font références au même concept ou pas. Par exemple, dans la spécification de l'EGTS que nous avons étudiée, dans une des exigences on trouvait l'expression "eTaxi HMI discrete ORDER_BKWD" et dans une autre "eTaxi HMI backward command". Ces exigences faisaient effectivement référence au même concept, mais ce n'est pas évident sans un moyen de vérification supplémentaire, comme les références

explicites que nous proposons. Ce problème de non homogénéité des termes est identifié par les industriels et dans les listes de critères de qualité des exigences.

Quand on a une information précise, il est possible d'en déduire des informations moins précises mais qui peuvent également être utilies : quand un mot d'une exigence fait référence à un élément de modèle, on en déduit que l'exigence "parle de" cet élément de modèle. Cette relation peut être utile afin de tracer les liens entre les exigences et les éléments de modèles, par exemple en réalisant des matrices de connectivité.

Co-construction Dans ce que l'on a présenté précédemment, nous avons utilisé les modèles pour rendre les exigences moins ambiguës, mais, comme noté dans la section 3.5.2.1, nous avons trouvé qu'encourager l'écriture d'exigences précises menait à des modèles plus détaillés du système et de son environnement.

Par exemple, nous avons considéré que la puissance d'alimentation était modélisée par une valeur appelée "E", exprimée en Watt. Mais supposons que l'on souhaite écrire des exigences qui font référence au courant dans l'alimentation, par exemple, l'exigence 7 : "Le système ne doit pas demander une intensité de plus de 1A sur l'alimentation électrique". Si l'on garde le modèle présenté précédemment, il n'y a pas d'interface "intensité" dans ce modèle : on ne peut donc pas préciser ce qu'est cette "intensité" dans l'exigence en créant une référence vers le modèle d'architecture.

Une solution est de considérer que la puissance électrique "E" est simplement l'intensité multipliée par la tension dans la ligne d'alimentation, supposée constante et égale à 12V. On modifie alors l'Exi. 7 en "Le système ne doit pas demander une puissance de plus de 12W sur l'alimentation électrique".

On peut aussi mettre à jour le modèle d'architecture et définir deux interfaces "I" et "U". Ces interfaces représentent l'intensité et la tension de l'alimentation et remplacent la puissance "E". En faisant cela, on va casser les liens vers "E" écrits dans les autres exigences. Cependant, puisque ces références sont explicites et peuvent être lues par un programme informatique, il sera facile de trouver les liens cassés.

Nous pensons que mettre l'accent sur l'écriture d'exigences précises, faisant références à des concepts bien identifiés, devrait aider à construire des modèles plus complets.

4.2.2 Constructions d'éléments plus complexes

4.2.2.1 Composition des interfaces

Propriétés plus complexes Pour le moment, nous avons utilisé des abstractions qui correspondaient directement avec des phénomènes physiques et des interfaces concrètes du système. On peut vouloir exprimer des propriétés plus complexes, qui sont calculées à partir d'autres variables, en utilisant par exemple des logiques temporelles. Ces propriétés peuvent représenter des propriétés de l'environnement du système, mais qui n'interagissent pas directement avec celui-ci.

Par exemple, pour détecter un risque de blocage de roue, le contrôleur de l'ABS doit connaître une estimation de la vitesse du véhicule. On pourrait écrire l'exigence "Quand la

vitesse du véhicule est plus petite que X, le système doit faire Z". Un problème de cette exigence est que le fournisseur de l'ABS peut ne pas savoir quelle est la définition exacte de "vitesse du véhicule". Si 'vitesse du véhicule" n'est pas une entrée de l'ABS et si la méthode pour la calculer n'est pas donnée explicitement, le fournisseur ne peut pas savoir comment il doit estimer cette vitesse. Cette situation est ambiguë et peut créer des non-conformités si les méthodes de calcul de la vitesse du fournisseur et du client sont différentes.

On considère ici que la vitesse du véhicule est calculée en utilisant la moyenne des vitesses de rotation des quatre roues [1] : $Vit = Rr * (\omega_1 + \omega_2 + \omega_3 + \omega_4)/4$. Où Vit est la vitesse du véhicule, Rr est le rayon des roues et les ω_i sont les vitesses de rotation des quatre roues.

On pourrait remplacer "vitesse du véhicule" dans les exigences par une version en langue naturelle de sa définition. L'exigence précédente donnerait "Quand la moyenne des vitesses de rotation des quatre roues multipliée par le rayon des roues est plus petite que X, le système doit faire Z". Cette solution présente plusieurs défauts :

— L'exigence est plus difficile à lire.
— L'exigence est plus difficile à écrire.
— Un lecteur ne comprendra pas nécessairement que "la moyenne des vitesses de rotation des quatre roues multipliée par le rayon des roues" représente la vitesse du véhicule, ce qui donne un contexte à l'exigence et qui pourrait servir à un test de "bon sens" s'il y avait une erreur dans l'exigence ou autre part.

Les rédacteurs de spécifications écrivent parfois les définitions telles que "$Vit = Rr * (\omega_1 + \omega_2 + \omega_3 + \omega_4)/4$" dans des exigences. Par exemple, on pourrait trouver "La vitesse du véhicule doit être la moyenne des vitesses de rotation des quatre roues multipliée par le rayon des roues" comme "exigence" d'une spécification. Mais cette approche présente aussi quelques problèmes :

— Une exigence est sensée être une demande au concepteur du système, mais qu'est-ce que l'on demande ici ?
— Cela ajoute une exigence au corpus, il va donc falloir gérer cette exigence (la valider, la justifier, la tracer, etc.). Ce travail supplémentaire va être difficile parce, comme énoncé dans le premier point, la raison d'être de cette exigence n'est pas claire.
— Cette approche nécessite que les exigences qui utilisent "vitesse du véhicule" fassent référence à l'exigence qui contient la définition. Ces liens entre exigences à l'intérieur d'un même corpus peuvent créer des problèmes pratiques et conceptuels (par exemple, il est considéré préférable que les exigences d'un même corpus soient indépendantes entre elles).

Utilisation de modèles Nous proposons que la méthode précise de calcul pour l'estimation de la vitesse du véhicule soit définie dans une annexe du corpus d'exigences. Les rédacteurs d'exigences peuvent ainsi utiliser "vitesse du véhicule" dans les exigences et créer un lien explicite de cette expression vers sa définition dans l'annexe. C'est le même principe que quand nous proposons de lier les mots "pression dans les freins" à leur définition dans un modèle d'architecture. Dans le cas de "vitesse du véhicule" cela ressemble à la définition, dans un code source, d'une nouvelle variable à partir d'autres variables définies

1. C'est une simplification.

TABLEAU 4.3 – "Abstractions composées" pour l'ABS

Abréviation	Nom complet et unité	Définition
$Freq_2$, $Freq_3$, $Freq_4$	Fréquences des signaux Sig_2, Sig_3, Sig_4 (Hz)	
ω'_2, ω'_3, ω'_4	Estimations des vitesses de rotation des 3 roues non contrôlées (rad/s)	$\omega'_i = (Freq_i/48) * 2\pi$
Rr	Rayon des roues (m)	
Vit	Estimation de la vitesse du véhicule (m/s)	$Vit = Rr * (\omega_1 + \omega'_2 + \omega'_3 + \omega'_4)/4$
$Gliss$	Estimation du glissement de la roue	$Gliss = (Vit - Rr * \omega_1)/Vit$

précédemment. Cela permet de garder les exigences relativement faciles à lire tout en évitant l'ambiguïté, et, comme pour les liens vers un modèle d'architecture, on peut utiliser ces liens dans des programmes informatiques.

Dans la table 4.3, nous avons défini ces "abstractions composées" que nous utilisons dans les exigences de l'ABS. Une remarque : puisque nous considérons seulement un ABS agissant sur une roue, nous considérons que ce système reçoit les images des vitesses de rotation des autres roues.

Où s'arrêter ? L'utilisation de modèles comme références permet de simplifier les exigences. Par exemple, prenons les exigences d'un système arbitraire : "Dans l'état [On], quand le frein est activé, le [couple de sortie] doit être nul" et "Dans l'état [On], quand le frein est désactivé, le [couple de sortie] doit être égal à l'entrée [Comm_Coupl]". On peut imaginer de définir une variable "couple commandé" qui soit nulle quand le frein est activé et égale à "Comm_Coupl" quand le frein n'est pas activé. Cela permet de simplifier les deux exigences en : "Dans l'état [On], le [couple de sortie] doit être égal au [couple commandé]" en faisant référence à la définition de "couple commandé".

Cela a l'avantage de diminuer le nombre d'exigences, tout en ayant des exigences plus simples. On peut aussi considérer que, puisque les deux exigences originales décrivent un comportement facilement traduisible en logique booléenne, les écrire sous forme de texte n'apporte rien. D'un autre coté, le sens de la nouvelle exigence est moins clair pour un lecteur humain, qui doit étudier la définition de "couple commandé" pour saisir ce sens.

La question "quand doit-on s'arrêter de formaliser/modéliser ?" n'a pas de réponse unique : cela va dépendre du contexte. C'est pour cela qu'il est utile d'en connaître les avantages et inconvénients.

4.2.2.2 Automate à états

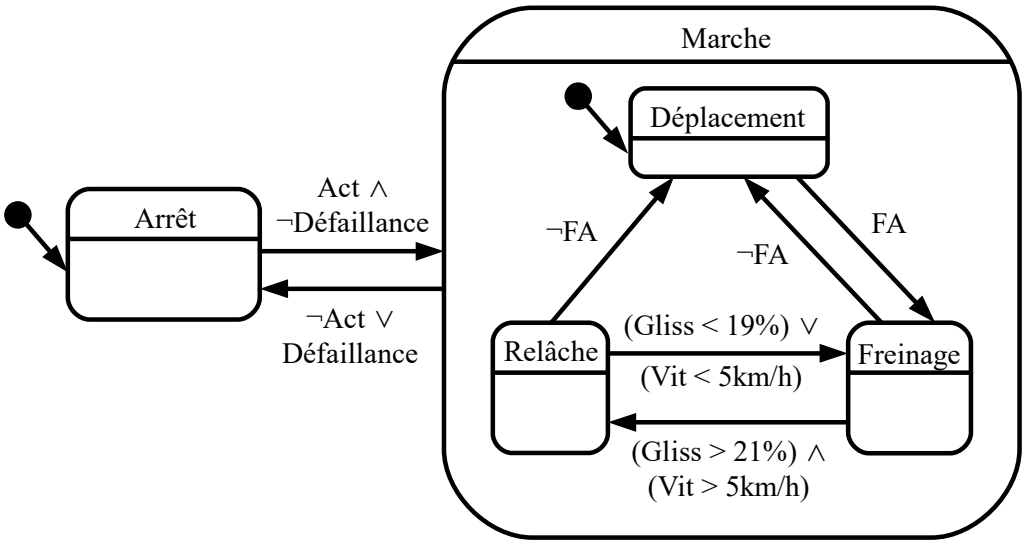


FIGURE 4.7 – Automate à états définissant les états utilisés dans les exigences de l'ABS

Un cas particulier de composition Nous avons utilisé "vitesse du véhicule" pour un exemple simple qui utilise de l'arithmétique basique, mais on peut aussi définir des variables plus complexes de cette manière. Un concept récurrent dans les spécifications est le concept d'"état" (aussi appelé "mode") : dans les corpus d'exigences réels, on trouve rarement des exigences avec une logique purement combinatoire comme l'Exi. 1 ("Quand la puissance d'alimentation est plus petite que 10mW, la pression dans les freins doit être égale à la pression dans le maître cylindre"). Les antécédents, ici "Quand la puissance d'alimentation est plus petite que 10mW" incluent souvent une expression telle que "dans l'état X".

Nous considérons que ces états sont essentiellement la même chose que les variables composées telles que "vitesse du véhicule". Une nuance est que ces états permettent d'exprimer des propriétés plus complexes, en particulier concernant les aspects temporels. La différence pratique étant qu'au lieu d'utiliser des équations arithmétiques, on utilise un automate à états (voir section 3.1.2.4) pour définir un état. L'automate à états que nous avons modélisé pour servir de référence aux exigences de l'ABS est présenté dans la figure 4.7.

On remarque que pour définir les transitions entre états, on utilise : des éléments directement en interface avec l'ABS comme "pédale de frein appuyée" (FA), ainsi que des "abstractions composées" telles que "vitesse du véhicule" (Vit).

Dans les exigences, nous ferons référence à un état précis de l'automate à états en écrivant des phrases telles que "Dans l'état Freinage, le système doit faire X".

De même que pour l'exemple de la vitesse du véhicule, dans les spécifications industrielles, ce type d'automate à états est parfois défini en utilisant des exigences textuelles, par exemple :

— "Le système doit avoir un état Arrêt"

— "Le système doit avoir un état Marche"
— "Le système doit passer de l'état Marche à l'état Arrêt quand Act est faux ou que Défaillance est vrai"
— Etc.

Nous avons trouvé que cette approche présentait les problèmes que nous avons illustrés avec l'exemple de "vitesse du véhicule" : davantage d'exigences, dont le sens n'est pas clair et des interactions entre exigences potentiellement non désirées.

Il est important que l'on puisse suivre les liens explicites depuis leurs origines, dans les exigences, vers les éléments de base, qui sont en principe les éléments du modèle d'architecture. Pour cela, il est nécessaire d'inclure les mêmes liens explicites que l'on a mentionné pour les exigences, dans les automates à états et les nouvelles variables telles que "vitesse du véhicule". Nous considérons que ces éléments sont des intermédiaires, servant à simplifier l'expression des exigences.

Un modèle classique de comportement Les automates à états sont un type de modèle assez répandu pour décrire les comportements. Comme pour les modèles d'architecture, on peut considérer les points de vues organiques, fonctionnels et opérationnels. Dans ces différents points de vue, les états sont respectivement appelés "configurations", "modes" et "contexte opérationnels", mais la syntaxe reste identique. Comme pour les modèles d'architecture, nous ne nous préoccupons pas vraiment du sens associé à ces modèles, juste qu'il existe un sens.

Durées des transitions Dans la définition d'un automate à états, les transitions ont toujours une durée nulle : c'est à dire que lorsqu'une transition est tirée, l'état de départ est immédiatement désactivé et l'état d'arrivée immédiatement activé. Cela pose un problème dans les spécifications, car on va demander au système de changer certaines valeurs de manière discontinue : par exemple, dans l'état "Relâche" on souhaite que la pression dans les freins soit nulle, et dans l'état "Freinage" que cette pression soit égale à la pression dans le maître cylindre (non-nulle, puisque l'on freine). Les processus physiques (et dans une moindre mesure, les processus logiciels également) ne peuvent pas être mis à jour instantanément, et ces exigences vont donc être forcément violées.

Dans les spécifications que nous avons étudiées, une manière d'éviter ce problème est de demander que la transition entre deux états ait une durée donnée. Dans ce cas, pendant la transition, ni l'état de départ ni l'état d'arrivée ne sont actifs, et donc les exigences pour lesquelles l'antécédent demande qu'un état soit actif sont vérifiées (si X est faux, $X \implies Y$ est vrai quel que soit Y). Cette solution pose problème car les formalismes d'automate à états ne permettent généralement pas ce genre de comportement.

Une solution est d'introduire un nouvel état pour chaque transition à laquelle on donne une "durée" et que ce soit cet état qui soit actif pendant la "transition". Après écoulement du temps spécifié, l'état-transition est désactivé et l'état d'arrivée de la transition originale est activé, voir figure 4.8. Les transitions ne peuvent pas avoir de durées dans les formalismes de machine à états classiques (pour de bonnes raisons). On peut cependant obtenir le comportement attendu en incluant un état virtuel et une transition qui est tirée au bout d'un temps donné entre les deux états reliés par une transition. Cette construction, elle, est

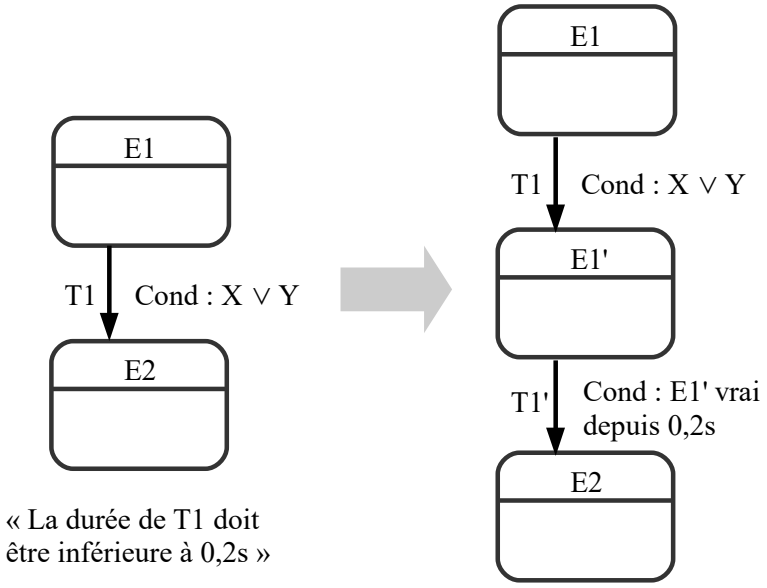


FIGURE 4.8 – Remplacement des "durées" des transitions par des états d'attente

valide dans le cadre des automates temporisés, une extension assez courante des automates à états.

On peut également ne pas commander directement la valeur physique mais plutôt une valeur demandée, qui elle commande la valeur physique : par exemple, au lieu d'écrire "Dans l'état Relâche, la pression dans les freins doit être nulle" et "Dans l'état Freinage, la pression dans les freins doit être égale à la pression dans le maître cylindre (PMC)", on va plutôt définir une variable composée "pression demandée dans les freins", qui sera nulle dans l'état Relâche, et égale à PMC dans l'état Freinage. On écrit ensuite une exigence qui lie la valeur réelle (la pression dans les freins) avec cette valeur demandée, par exemple au moyen d'une fonction de transfert : "La fonction de transfert de [la pression demandée dans les freins] à [la pression dans les freins] doit atteindre 95% de la valeur demandée en moins de 0.1s, avoir une précision statique de 1% et ne pas dépasser 120% de la valeur commandée."

Grammaire formelle pour les automates à états Comme pour les modèles d'architecture, nous avons écrit les automates à états que nous avons utilisés selon la grammaire suivante. De la même manière, cette grammaire a été construite de façon ad-hoc selon nos besoins et elle peut être modifiée pour la rapprocher d'autre formalismes similaires. La grammaire est donnée en figure 4.9.

Un automate à états peut être composé d'un ensemble d'états, reliés les uns aux autres par des transitions. Un état est doté d'un nom et peut contenir d'autres états, reliés également par des transitions. Automates à états et états peuvent aussi contenir des sous-automates évoluant en parallèle. Les transitions contiennent des références à un état de sortie et un état d'entrée, et une condition de franchissement. La condition de

Symboles terminaux : {Name, From, To, Condition}
Symboles non-terminaux : {ModelStatechart, State, Transition, Parallel}
Axiome : ModelStatechart

Règles de production ("*" est l'étoile de Kleene) :
- ModelStatechart → Parallel Parallel Parallel*
- ModelStatechart → State* Transition*
- State → Name Parallel Parallel Parallel*
- State → Name State* Transition*
- Parallel → Name State* Transition*
- Transition → From To Condition

FIGURE 4.9 – Grammaire formelle pour les automates à états

franchissement est une fonction à valeur booléenne, dont la grammaire est sensiblement égale à celle présentée en section 4.2.3.2.

4.2.2.3 Autres modèles possibles

Nous avons mentionné principalement les modèles d'architecture et les automates à états, mais d'autres types de modèles pourraient être utiles comme références pour des exigences en langue naturelle.

Puisque que nous n'avons pas fait beaucoup d'hypothèses sur les modèles que nous pourrions utiliser comme référence pour les exigences, il n'y a pas beaucoup de contraintes sur les types de modèles que nous pourrions utiliser. Si l'on veut préciser la signification de divers fragments de texte dans des exigences, beaucoup de types de modèles pourraient être utiles.

Comme mentionné dans la section 3.3.3.3, il serait préférable de pouvoir adresser des éléments du modèle plutôt que le modèle entier : par exemple, si l'on parle de "l'état Freinage", on veut être capable d'écrire une référence directement à l'état correspondant dans le modèle, plutôt qu'à l'intégralité de l'automate à états. C'est la même chose pour le modèle d'architecture : nous voulons faire un lien des mots "pression dans les freins" vers l'interface correspondante dans le modèle d'architecture, pas vers le modèle d'architecture complet. Il s'en suit qu'une des contraintes sur les modèles est qu'ils devraient posséder une structure permettant de :

— distinguer les éléments individuels,
— définir une adresse pour ces éléments.

Lorsque nous avons étudié des spécifications industrielles, en plus des modèles d'architecture et des automates à états, nous avons identifié quelques types de modèles qui pourraient être utiles comme compléments aux exigences textuelles :

— Des modèles 3D créés avec des outils de conception assistée par ordinateurs, utilisés pour définir par exemple la position d'un système à l'intérieur d'un système plus gros. Il y a généralement des schémas inclus dans les spécifications actuelles, mais,

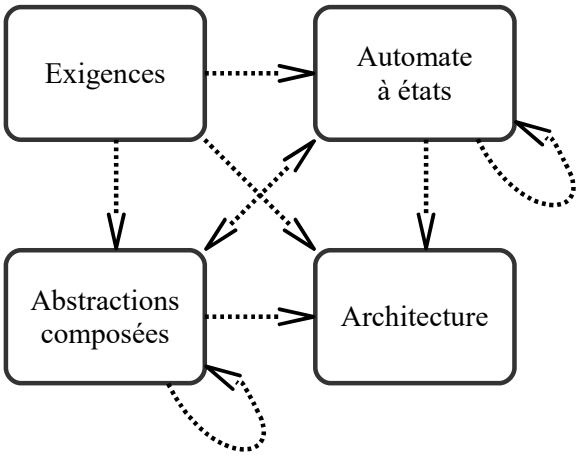


FIGURE 4.10 – Liens explicites possibles à l'interieur d'une spécification

comme pour les modèles d'architecture, ils ne sont pas nécessairement bien liés aux exigences.

— Les définitions de séquences, comme des cycles d'utilisation, utilisés par exemple dans les calculs de fatigue. Ces séquences pourraient être définies en utilisant des automates à états (éventuellement temporisés), mais ce formalisme est probablement trop complexe : un simple tableau est plus simple à écrire et à comprendre.

4.2.2.4 Vision globale des références explicites

Les exigences peuvent directement faire référence aux éléments de l'interface du système, mais aussi faire référence à des éléments intermédiaires qui sont construits à partir des interfaces du systèmes, comme les états d'un automate à états. Ces éléments intermédiaires ne font pas non plus nécéssairement des références directes aux interfaces du système. Par exemple, les automates à états peuvent faire référence à d'autres abstractions composées (comme l'exemple de "vitesse du véhicule", qui est utilisé dans la condition d'une transition de l'automate donné en figure 4.7). Comme on l'a mentionné à la fin de la section 4.2.2.2, on veut pouvoir suivre ces références explicites de leur point de départ (les exigences) à leur point d'arrivée (les interfaces).

Toutes les références forment un graphe orienté, dont les nœuds sont des éléments parmi les exigences, l'automate à états, les "abstractions composées" et le modèle d'architecture. La figure 4.10 donne les liens possibles : les éléments des exigences, des automates à états et des abstractions composées peuvent faire des références vers les éléments de l'automate à états, vers des abstractions composées et vers les éléments du modèle d'architecture. Les éléments de l'architecture ne font eux pas de références. On note que l'on ne fait pas de références explicites vers les exigences.

On détaille certaines opérations permettant d'explorer ces références explicites dans la section 6.

4.2.3 Syntaxe pour lier ces éléments

Dans les deux sections 4.2.1 et 4.2.2, nous avons surtout étudié les parties lexicales des exigences, en rendant explicite le sens de mots ou groupe de mots, tels que "pression dans les freins" ou "vitesse du véhicule". Dans cette partie, nous développons les concepts introduits en section 3.3.2 et nous nous penchons sur la syntaxe. Nous avons cherché les règles spécifiques qui gouvernent la structure des exigences. Les exigences devraient respecter les règles de la langue naturelle pour être facilement compréhensibles, mais nous pouvons aussi trouver d'autres règles qui devraient être respectées par les rédacteurs d'exigences.

4.2.3.1 Une syntaxe simple

Qu'est-ce qu'une exigence ? Une exigence sur un système est une propriété, une propriété qui doit être vraie quand le système est installé dans son environnement. Si la propriété est fausse, cela veut dire que le système ne respecte pas l'exigence. Les exigences, au moins dans le contexte dans lequel nous nous plaçons, doivent être suffisamment précises pour qu'elle soient, soit vraies, soit fausses. Cela signifie que les exigences sont, d'un point de vue informatique, des fonctions à valeur booléenne.

De plus, il est généralement possible de décomposer les exigences en plus petits fragments : par exemple, les exigences que nous avons étudiées sont souvent sous forme d'implications, comme l'Exi. 1.

"La puissance d'alimentation est plus petite que 10mW" est l'antécédent et "la pression dans les freins est égale à la pression dans le maître cylindre" est appelé le conséquent. Pour une équation booléenne $R = A \implies B$, où A et B sont des fonctions à valeur booléenne, R est faux si et seulement si A est vrai et que B est faux : $A \implies B$ est équivalent à $\neg A \vee B$.

Dans l'exemple précédent, on peut continuer la décomposition : "la puissance d'alimentation est plus petite que 10mW" est une fonction à valeur booléenne et est composée de "la puissance d'alimentation", "est plus petite que" et "10mW". On peut considérer qu'il n'est pas utile de décomposer 'la puissance d'alimentation" davantage : c'est une expression atomique. De plus, cette expression correspond à un élément que l'on a défini plus tôt : on sait ce que l'expression "la puissance d'alimentation" veut dire, puisque :

— nous avons défini le concept qu'elle représente (l'interface appelée "E" dans le modèle d'architecture),
— nous avons fait un lien explicite de cette expression vers la définition du concept.

Que doit-on restreindre ? Dans les spécifications, on peut trouver des expressions telles que "ne doit pas être endommagé par" (Exi. 2 et Exi. 3) ou "la probabilité d'une défaillance causant X" (Exi. 4 et Exi. 5). Ces expressions sont couramment utilisées et les ingénieurs ne semblent pas avoir de problèmes pour les comprendre, nous ne les considérons donc pas comme particulièrement ambiguës, même si en donner une définition formelle serait complexe. On peut les inclure directement dans le langage en créant des fonctions syntaxiques pour ces expressions (ce que sont ces fonctions syntaxiques est détaillé en section 4.2.3.2). Par exemple, "Non endommagé" prend un argument de type "système" et renvoie un booléen

(soit le système est endommagé, soit non). "La probabilité d'une défaillance causant X" prend un argument booléen et renvoie une probabilité.

Si l'on veut donner une définition plus précise de ce que signifie "Non endommagé" ou "La probabilité d'une défaillance causant X", on peut écrire cette définition dans l'introduction de la spécification ou dans un document externe tel qu'une norme, et créer des hyperliens des exigences vers ces définitions.

Une syntaxe modulable Nous cherchons à définir une syntaxe modulable, c'est à dire avec beaucoup (potentiellement une infinité) de structures d'exigence possibles, contrairement aux gabarits par exemple, qui sont limités à quelques structures possibles. De la même manière que, dans un texte en langue naturelle, on peut remplacer un nom dans un texte par un groupe nominal, on souhaite pouvoir remplacer une valeur booléenne par n'importe quelle autre expression ayant une valeur booléenne.

Cette syntaxe doit permettre d'écrire potentiellement une infinité de structures d'exigence différentes, mais elle doit contraindre quand même les exigences possibles (sinon elle serait inutile). Nous pensons que la syntaxe proposée constitue un compromis acceptable entre expressivité et contrainte. De plus, nous l'avons définie pour que les utilisateurs puissent être capables de rajouter des éléments de syntaxe afin de l'adapter à leur cas particulier.

4.2.3.2 Définition formelle de la syntaxe

Pour décrire la structure des exigences, nous proposons un système de type similaire à ceux existant dans certains langages de programmation. Une exigence est représentée par un couple (S,P). P est la propriété que le client veut voir vérifiée et S est le système auquel on demande cette propriété. Dans la spécification d'un système T, dans les exigences, S et T sont généralement le même système, mais pas toujours : on peut vouloir exprimer une propriété sur un sous-système de T plutôt que sur le système complet, comme dans le cas de l'Exi. 8.

Termes Nous définissons un ensemble de termes de manière récursive à partir d'un ensemble F de fonctions :

- Les fonctions d'arité nulle sont des termes.
- Si f est une fonction d'arité n, avec $n > 0$, et que $(t_1, \ldots, t_n)$ sont n termes ; alors $f(t_1, \ldots, t_n)$ est un terme.

Types Introduisons maintenant les types : chaque fonction d'arité n de F a un type de sortie et n types d'entrée. Pour qu'un terme soit bien typé, les types qu'il contient doivent respecter un système de satisfaction de contrainte.

Le type d'un terme est le type de sortie de sa fonction racine : le type d'un terme de forme $f(t_1, \ldots, t_n)$, où f est une fonction d'arité n, est le type de sortie de f. Les contraintes sur les types dépendent de la fonction f. De plus, si une exigence est un couple (S,P), S doit être de type "système" et P de type "booléen".

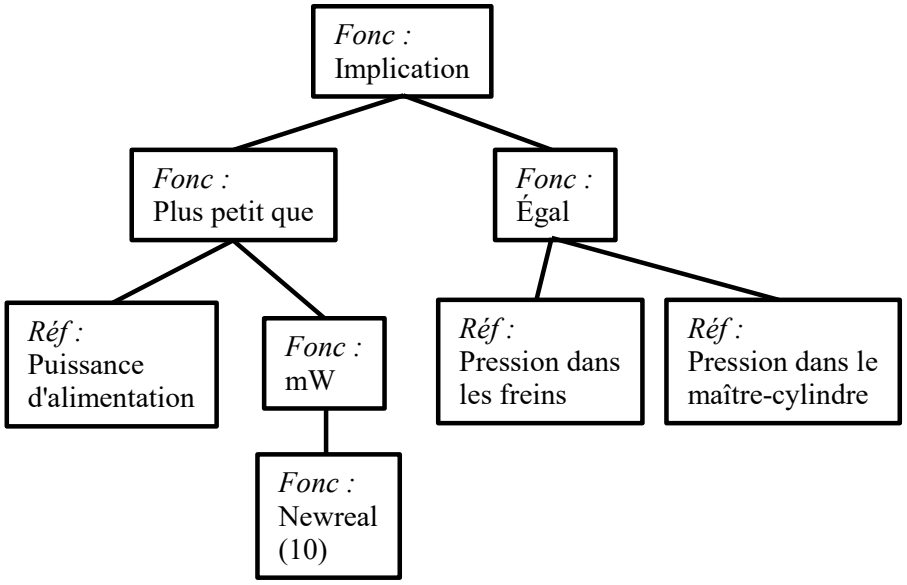


FIGURE 4.11 – Arbre syntaxique d'une exigence

Cette décomposition est relativement similaire à l'analyse grammaticale d'une phrase en langage naturel, qui donne des arbres syntaxiques : une phrase (“‘La poule traverse la route”) peut être composée d'un groupe nominal (“la poule”) et d'un groupe verbal (“traverse la route”). Le groupe verbal peut lui-même être composé d'un verbe (“traverse”) et d'un groupe nominal (“la route”), etc. Cette décomposition est également assez proche des Arbres Syntaxiques Abstraits utilisés en informatique.

Exemple basique et fonctions Pour notre exemple précédent, Exi. 1, nous pouvons définir l'arbre dans la figure 4.11.

Parmi les fonctions utilisées dans cet arbre :
— “Implication” est une fonction binaire, avec un type de sortie booléen et deux entrées de type booléen.
— “Plus petit que” est une fonction binaire, avec un type de sortie booléen et deux entrées de même type quelconque (on n'a pas défini une fonction “Plus petit que” pour chaque unité physique (N, m, W...))

Nous considérons qu'une mesure (un couple nombre-unité) est construite en utilisant une fonction, correspondant à l'unité en question, qui prend le nombre en argument. Par exemple, pour construire l'élément “10mW”, on utilise la fonction “mW” qui prend le nombre “10” en argument.

Les références à des modèles (ici “puissance d'alimentation”, “pression dans les freins”, “pression dans le maître cylindre”) sont des fonctions d'arité nulle, donc des termes. Ce sera le cas pour toutes les références. Les fonctions qui ne sont pas des références peuvent correspondre à des éléments pouvant être trouvés dans n'importe quelle spécification. C'est le cas de

"Implication", "Égal" ou 'Plus petit que". Certaines fonctions sont plus spécifiques à un domaine particulier : par exemple, la fonction correspondant à l'unité "FH" (Flight-hour, heure de vol) n'est pas pertinente hors du cadre aéronautique.

Traduire un besoin en exigence Considérons un but de haut niveau de l'ABS, tel que "Quand le système n'est pas activé, le système ne doit pas empêcher le freinage". Ce type de but est utile pour comprendre le contexte et le principe du système spécifié, mais il est trop peu précis et trop ambigu pour être une exigence. Une exigence plus explicite inspirée de ce but pourrait être l'Exi. 6 : "Dans l'état Arrêt, et quand la pression dans le maître-cylindre est plus faible que 50 bar, alors la pression dans les freins doit être égale à la pression dans le maître cylindre". (On suppose qu'une pression dans le maître cylindre inférieure à 50 bar constitue un comportement attendu de l'environnement. Quand cette condition est fausse, cela constitue un comportement anormal de l'environnement. Ce que le système doit faire dans ce cas peut être spécifié dans d'autres exigences.)

Puisque c'est le but de ce travail, l'Exi. 6 n'est pas limitée à cette composante textuelle.

On détaille la syntaxe de cette exigence dans le schéma 4.12. Ici, les éléments de l'arbre syntaxique sont pour la plupart des fonctions génériques : ils pourraient être trouvées dans n'importe quelle spécification, à part "Arrêt", "pression dans le maître-cylindre" et "pression dans les freins". On identifie ces fragments de texte comme des références :

— à l'architecture du système (pour "pression dans le maître-cylindre" et "pression dans les freins")
— ou à une description du comportement du système (pour l'état "Arrêt").

Liens entre syntaxe et modèles Cette syntaxe et l'utilisation des modèles introduite précédemment ne sont pas indépendantes. Premièrement, le type d'un fragment de texte faisant référence à un élément de modèle doit être défini dans ce modèle : par exemple, pour qu'un vérificateur syntaxique valide le fragment d'exigence "la pression dans le maître-cylindre est plus faible que 50 bar", il faut qu'il sache que "la pression dans le maître-cylindre" est bien une pression et peut être comparée à "50 bar". Pour pouvoir utiliser efficacement les éléments de syntaxe, il va donc falloir rajouter des types dans les attributs des éléments de modèles.

De plus, si l'on introduit de nouveaux types de modèles (comme ceux mentionnés en section 4.2.2.3), il est aussi nécessaire d'introduire de nouvelles fonctions syntaxiques. Par exemple, pour utiliser des états dans une exigence, nous avons besoin d'une fonction "Est-ce que l'état X est actif ?". Cette fonction prend un état comme argument et renvoie un booléen. Donc, lorsque l'on introduit de nouveaux modèles pour servir de référence dans les exigences, nous avons aussi besoin d'étendre le langage d'exigence avec de nouvelles fonctions permettant d'utiliser ces références.

Cette syntaxe peut également être utilisée à l'intérieur des modèles, pour définir les conditions de franchissement des transitions d'un automate à états ou pour définir une nouvelle abstraction composée telle que l'exemple "vitesse du véhicule" de la section 4.2.2.1. Dans le permier cas, le type de cette condition doit être un booléen, et dans le second cas, le type de l'expression donnera le type (et donc éventuellement l'unité) de cette abstraction.

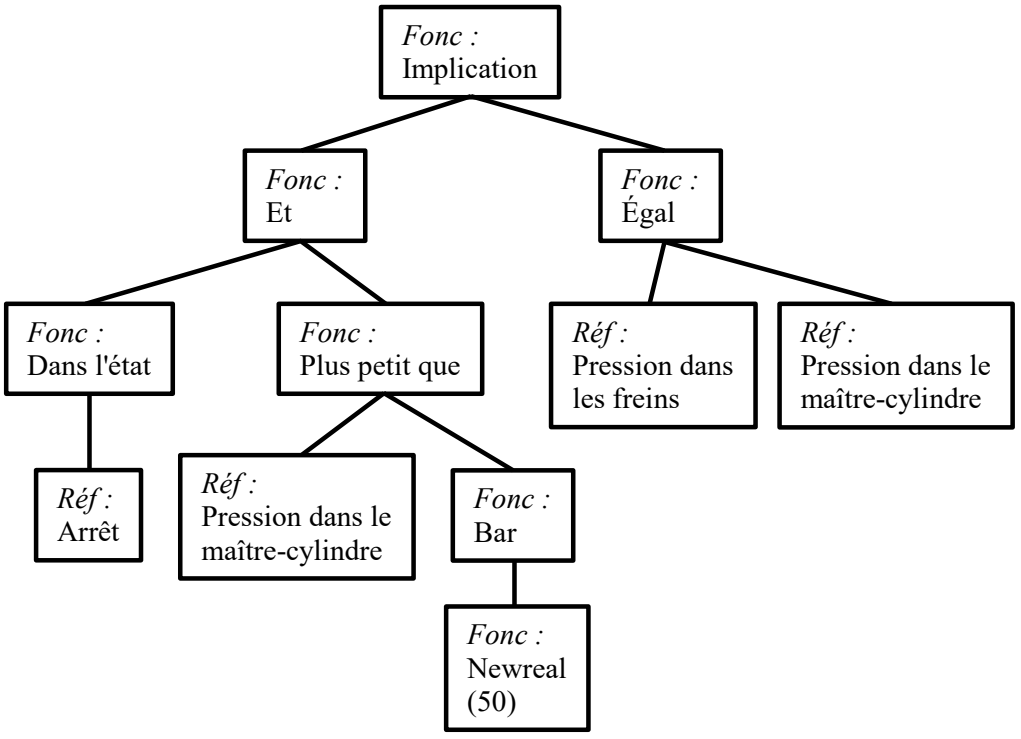


FIGURE 4.12 – Arbre syntaxique d'une autre exigence

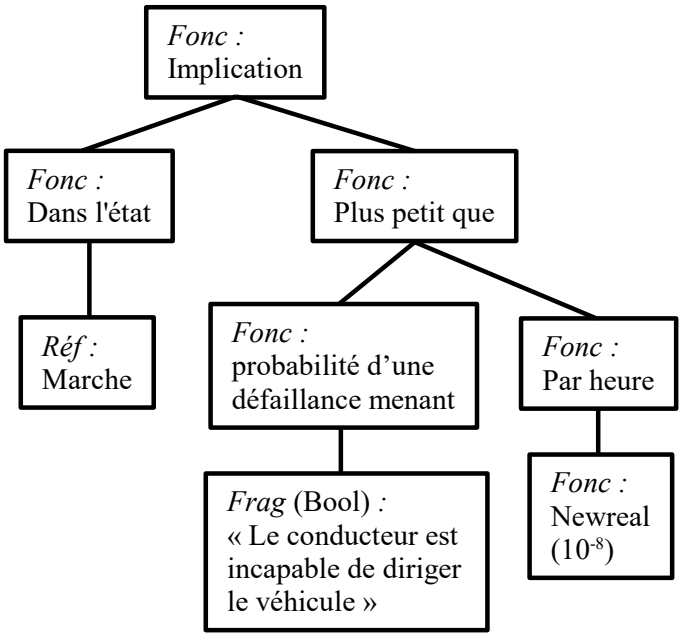


FIGURE 4.13 – Arbre syntaxique avec une feuille non formalisée

4.2.3.3 Support d'éléments non formels

Formaliser partiellement une exigence Ce langage typé permet de formaliser la syntaxe de certaines exigences, mais pas toutes. Pour éviter de limiter l'expressivité des rédacteurs, nous devons leur permettre d'écrire des exigences, ou des fragments d'exigences, en texte libre. Pour permettre cela tout en gardant la syntaxe que nous venons de décrire, nous proposons qu'un ingénieur puisse écrire le texte libre qu'il souhaite, mais il doit ajouter un type à ce fragment d'exigence.

Par exemple, considérons l'exigence 4 : "Dans l'état Marche, la probabilité d'une défaillance empêchant le conducteur de diriger le véhicule doit être inférieure à 10^{-8} par heure".

Si l'on ne veut pas, ou ne peut pas, formaliser "empêchant le conducteur de diriger le véhicule", on peut laisser ce texte libre dans l'exigence et lui donner le type "booléen" (on considère que soit le conducteur peut diriger le véhicule, soit il ne peut pas). Nous pouvons décomposer le reste de l'exigence comme indiqué dans la figure 4.13.

Il est aussi possible d'écrire une exigence complète en texte libre, sans aucune information de syntaxe. Évidemment, une telle exigence n'apparaîtra pas dans une requête qui dépend d'informations syntaxiques, par exemple une requête sélectionnant l'antécédent de chaque exigence s'il existe.

On ne peut pas tout formaliser Dans les exemples précédents il n'était pas trop complexe d'écrire des exigences formalisées, ce n'est cependant pas le cas pour toutes les exigences.

Par exemple, on peut identifier trois parties dans l'Exi. 9 ("Après une défaillance causant un détachement total ou partiel de pièce, les éléments du système ou de fixation doivent rester à l'intérieur de leur enveloppe de conception") :

— un antécédent : "Après une défaillance causant un détachement total ou partiel de pièce"
— un conséquent : "rester à l'intérieur de leur enveloppe de conception"
— les éléments concernés : "les éléments du système ou de fixation"

Si c'est possible, nous pourrions utiliser un modèle de Conception Assisté par Ordinateur comme référence pour la définition de "leur enveloppe de conception". À part cela, formaliser cette exigence semble difficile.

Dans l'Exi. 10 on peut identifier une condition "pendant la maintenance et l'opération" mais pas beaucoup plus. On peut noter que cette exigence n'est pas nécessairement une bonne exigence (elle n'est ni complète ni précise). Cependant, comme c'est le genre d'exigences que l'on trouve dans les spécifications industrielles, a priori elles ne sont pas inutiles et les ingénieurs veulent pouvoir exprimer de telles propriétés.

Où arrêter la formalisation ? Il pourrait être possible de formaliser davantage la syntaxe et le vocabulaire de ces deux dernières exigences. Le problème est plus de savoir si ce serait utile plutôt que de savoir si c'est faisable. Peut-être que l'on peut réécrire ces exigences pour qu'elles soient parfaitement formelles, mais si l'effort nécessaire pour cette formalisation est trop important comparé aux bénéfices retirés, ce ne serait pas rentable.

Pour l'Exi. 10, il serait peut-être possible de définir un automate à états où l'un des états serait "maintenance". Mais définir formellement les limites de cet état (les transitions d'entrée et de sortie) serait difficile. De plus, il n'est pas dit qu'avoir une définition formelle de "maintenance" facilite beaucoup la conception du système.

D'un point de vue général, nous ne pensons pas que des spécifications complètement formelles soient utiles et faisables en pratique pour les systèmes que nous avons étudiés. Cependant, nous pensons qu'écrire des spécifications davantage formalisées que celles actuellement rédigées en langue naturelle peut être utile. On peut imaginer une échelle de formalisation : différents contextes (systèmes, industries, etc.) auront différents niveaux de formalisation optimaux. On a besoin d'outils pour écrire ces différents niveaux, et donc des outils qui permettent de faire cohabiter formel et non formel.

Un avantage de notre approche est qu'elle est graduelle : contrairement à des méthodes comme Event-B Abrial (2010), il n'y a pas besoin d'avoir une spécification complètement formelle pour observer les premiers bénéfices. Des premières étapes relativement simples peuvent déjà être utiles : par exemple, on peut omettre toute la partie concernant la syntaxe que l'on a décrite dans ce travail et uniquement tracer des liens entre les mots utilisés dans les exigences et leur définitions. Une fois ces liens tracés, il est par exemple possible de vérifier qu'un concept est toujours nommé de la même manière dans les exigences, ou de savoir quelle exigence fait référence à quel élément de modèle.

Pour les exigences qu'on ne peut pas (ou qu'il n'est pas rentable d') écrire formellement, d'autres méthodes peuvent faciliter l'ingénierie des exigences. Par exemple, l'Exi. 9 n'est pas nécessairement spécifique au système anti-blocage des roues. Si toutes les autres spécifications écrites par l'entreprise incluent cette exigence, il pourrait être efficace d'avoir

un ensemble d'exigences constantes et de les inclure automatiquement dans toute nouvelle spécification, ou d'autres méthodes de réutilisation.

4.3 Programmes et scénarios de test

Nous avons présenté comment faire des liens entre les exigences et des modèles servant de définitions, et comment ajouter une structure formelle aux exigences. Nous nous intéressons maintenant à comment utiliser en pratique ces éléments pour réaliser des tests sur les exigences.

4.3.1 Programmes de test

Nous avons donc écrits des tests pour vérifier ou aider à vérifier les critères présentés en section 3.4.2.

4.3.1.1 Tests sur les liens

Voici un ensemble de tests utilisant les liens explicites afin de vérifier certains critères.

Homogénéité des termes On peut vérifier si chaque concept est toujours mentionné dans les exigences en utilisant les mêmes mots : par exemple, qu'il n'y ait pas dans la spécification une exigence qui parle de "pression de freinage" et une autre de "pression dans les freins". Tout d'abord, il faut que dans les exigences, on ait tracé les liens explicites entre les termes utilisés et la définition de ces concepts. Si c'est le cas, on a deux solutions :

- Soit on considère que le terme à utiliser pour faire référence à un concept est précisé dans la définition de ce concept (e.g. dans le modèle de la figure 4.6). Dans ce cas il suffit simplement de vérifier que les termes utilisés dans les exigences correspondent à ce terme.
- Soit on considère que ce n'est pas le cas. Dans ce cas on compare les mots utilisés dans les exigences, qui font référence à une même adresse, entre eux.

Référence aux interfaces On a noté précédemment que toutes les exigences, au final, faisaient référence à des éléments de l'interface du système, soit directement, en faisant directement référence à ces interfaces, soit en faisant référence à des éléments construits à partir de ces interfaces (automates à états par ex.). Cela doit se retrouver dans les concepts utilisés dans les exigences. Si l'on a tracé les liens explicites non seulement à partir des exigences, mais aussi à partir des modèles et autres éléments de la spécification, alors on peut partir des exigences, et en suivant les références, arriver aux "feuilles", qui ne font référence à rien d'autre et qui sont les éléments de base permettant de définir tout le reste. Ces "feuilles" doivent correspondre aux interfaces entre le système et son environnement, dans l'exemple de l'ABS ce sont les éléments du tableau 4.2.

Si ce n'est pas le cas, on a peut-être un problème de niveau d'abstraction. Si les exigences sont trop haut niveau, on va avoir tendance à faire référence à des éléments de

l'environnement qui ne sont pas en contact direct avec le système. Si les exigences sont trop bas niveau, on va plutôt faire référence à des éléments internes au système, et ainsi violer le principe de boîte noire.

Référence à toutes les interfaces Les exigences doivent être complètes. Une condition de cette complétude est que, pour toutes les interfaces du systèmes, ces interfaces soient complètement spécifiées. Il n'est pas vraiment possible de vérifier automatiquement qu'une interface est complètement spécifiée, mais on peut au moins vérifier que l'on n'a pas oublié d'interfaces. Pour faire cela, on vérifie que, pour toute interface définie dans un modèle d'architecture, il existe au moins une exigence qui y fait référence. Si ce n'est pas le cas, il y a probablement un problème.

Liens cassés Il est facile de vérifier que, si un lien explicite existe, sa cible existe bien. Ce genre de test permet de déterminer, lorsqu'on met à jour un modèle, quelles exigences y font référence et lesquelles sont potentiellement impactées. Cela va aider à garder les exigences à jour.

De manière générale les références vont faciliter les analyses d'impact, en facilitant la traçabilité des éléments des exigences.

Sélections On peut sélectionner toutes les exigences qui font référence à un ou plusieurs concepts précis, que ces concepts soient des interfaces (e.g. "Pression dans les freins"), des éléments d'automate à états (e.g. l'état "Arrêt"), des annexes, ou autre. Ces sélections ne sont pas exactement des tests, mais elles peuvent permettre de simplifier les vérifications de divers critères : par exemple, si l'on est intéressé par une interface particulière, et que l'on veut vérifier qu'elle est complètement spécifiée, les exigences qui n'y font pas référence ne sont pas utiles. On peut donc sélectionner uniquement les exigences qui font référence à cette interface afin de faciliter les vérifications.

4.3.1.2 Tests sur la syntaxe

D'autres tests utilisent plutôt la syntaxe formelle des exigences.

Atomicité On peut vérifier, en utilisant la syntaxe, que les exigences ont, ou au contraire n'ont pas, une structure particulière. Comme on l'a mentionné précédemment, les exigences de la forme "$X \wedge Y$" ou "$Z \implies (X \wedge Y)$" sont susceptibles d'être non-atomiques.

Complexité de la structure Les exigences dont la structure syntaxique est trop complexe risquent d'être difficiles à comprendre, même si cette structure n'est pas ambiguë. Un indicateur de cette complexité est la profondeur de l'arbre syntaxique associé à l'exigence. Par exemple, la profondeur de l'arbre syntaxique de l'exigence 1, donné en figure 4.11, est 4. On peut mettre en évidence les exigences dont la profondeur de l'arbre syntaxique dépasse un certain seuil.

On souhaite aussi que les exigences soient plutôt concises. Même si cela ne fait pas appel aux principes que nous avons introduit dans ce travail, il est possible de réaliser un simple comptage des mots sur les exigences et de mettre en évidence celles qui dépassent un certain seuil.

Unités On peut tester les exigences pour vérifier :
- tout d'abord que les unités utilisées ne rendent pas la syntaxe de l'exigence incorrecte (par exemple on n'essaye de comparer une force avec un couple), ce qui permettra aussi de détecter les cas où l'unité est manquante,
- ensuite qu'on ne mélange pas les unités du système international (SI) avec des unités d'autres systèmes.

On peut éventuellement imaginer de pouvoir remplacer automatiquement les unités d'autres systèmes par leur unité SI correspondante (ou l'inverse).

Tolérances Un problème potentiel identifié précédemment (4.2.1.2) est l'existence d'égalités entre des valeurs réelles. Comme pour l'atomicité, ce problème peut être détecté en vérifiant que les exigences n'ont pas une structure particulière, ici une structure où un élément à valeur réelle doit être égal à une valeur précise.

Exigences booléennes Lorsque des critères demandent à ce que les exigences soient testables, vérifiables, précises ou non ambiguës, une partie de ces critères concerne le fait, entre autres choses, que l'exigence doit pouvoir être soit vraie soit fausse. Pour être valide selon la syntaxe présentée précédemment, une exigence doit être un booléen, et un script le vérifie.

4.3.1.3 Sélection des exigences par gabarit

Nous avons également écrit une recherche par gabarit utilisant la syntaxe formelle et les liens explicites. Cette recherche permet de définir une structure arbitraire et de sélectionner toutes les exigences ayant cette forme. Par exemple, une recherche sélectionnant toutes les exigences de la forme "$(* \wedge *) \implies *$", où "*" peut être n'importe quel sous-arbre, va renvoyer l'exigence 6 parce qu'elle contient un "Et" booléen en antécédent, mais pas les exigences 1 ni 4.

4.3.2 Scénarios de test

Nous avons écrits quelques scénarios de tests utilisant ces programmes, et comment ils peuvent améliorer les exigences. On va suivre le classement présenté en section 3.4.2.2, avec tout d'abord des scénarios où des problèmes peuvent être détectés automatiquement, voire même résolus automatiquement. Ensuite nous présentons des scénarios dans lesquels des scripts peuvent aider, mais qui nécessitent une intervention humaine. Enfin nous donnons quelques exemples de scénarios avec des problèmes que nous considérons hors de notre contexte ou qui sont trop difficiles à traiter.

Pour chaque scénario, on considère un point de départ, puis une séquence d'événements, et éventuellement des remarques.

4.3.2.1 Problèmes automatisables

Scénario 01 : écrire des exigences atomiques Point de départ : un besoin du type "Quand l'ABS est désactivé, le freinage n'est pas empêché, et une lumière sur le tableau de bord avertit le conducteur"

Séquence :

- On formalise la partie lexicale de cette exigence : on utilise les interfaces du systèmes, ou les variables créées à partir de ces interfaces. "Quand l'état de de l'ABS est [Arrêt], [pression_freins] est égale à [pression_maître-cylindre], et [lampe_tableau] est vraie."
- On formalise la syntaxe : "Dans l'état [Arrêt], $\Longrightarrow$ ([pression_freins] = [pression_maître-cylindre] $\wedge$ [lampe_tableau] = Vrai)".
- On préfère que les exigences soient atomiques, et on peut les tester : on détecte automatiquement que cette exigence est de la forme "$Z \Longrightarrow (X \wedge Y)$".
- Ce critère n'est pas absolu, notamment parce qu'il est en conflit avec d'autres critères, comme le besoin d'avoir des exigences complètes, ou d'avoir le moins possible d'exigences.
- On considère ici que l'on sépare cette exigence en deux exigences ("$Z \Longrightarrow X$" et "$Z \Longrightarrow Y$)") : "Dans l'état [Arrêt], $\Longrightarrow$ [pression_freins] = [pression_maître-cylindre]" et "Dans l'état [Arrêt], $\Longrightarrow$ [lampe_tableau] = Vrai".

Scénario 02 : écrire des exigences avec les mêmes termes Point de départ : plusieurs exigences où un même concept (par exemple "la pression dans les freins") est désigné par différents termes ("la pression dans les freins", "la pression de freinage" ou "P_fr")

Séquence :

- Si ce n'est pas déjà le cas, on crée les liens explicites entre les termes et le modèle où est défini "la pression dans les freins".
- Un script compare le nom du concept, indiqué dans le modèle de définition, avec les termes utilisés dans les exigences.
- On peut imaginer plusieurs réponses pour les cas où il n'y a pas correspondance :
  - On peut simplement remplacer automatiquement les termes incorrects par le nom défini dans le modèle.
  - On peut mettre en évidence les exigences où il y a un problème pour une revue par un humain.
  - On pourrait aussi définir des synonymes autorisés. Il faudra de préférence donner explicitement ces synonymes dans la spécification finale.

Scénario 03 : écrire des exigences au bon niveau d'abstraction Point de départ : un besoin de haut niveau "Le freinage n'est pas empêché quand la vitesse du véhicule est plus petite que 5km/h", et "vitesse du véhicule" n'est pas défini explicitement à partir des interfaces.

Séquence :
— On formalise le besoin en une exigence "[Vitesse_véhicule] < 5km/h $\implies$ [pression_freins] = [pression_maître-cylindre]". On considère que "Vitesse_véhicule" est défini dans un modèle de l'environnement du système.
— On peut détecter automatiquement que "Vitesse_véhicule" ne fait pas directement ou indirectement référence aux interfaces du système. Cela implique que l'exigence est de trop haut niveau.
— On a au moins deux solutions :
  — On peut définir "Vitesse_véhicule" à partir des interfaces du système, comme on l'a fait dans la section 4.2.2.1.
  — On peut modifier l'exigence afin qu'elle ne fasse référence qu'à la vitesse de rotation d'une (ou des) roue(s).

4.3.2.2 Aide partielle des tests

Scénario 04 : écrire des ensembles d'exigences complets (à propos d'un élément ou d'un état) Point de départ : un ensemble d'exigences incomplet, où l'on a oublié une exigence telle que "Quand on est dans l'état [Marche], la [puissance_alimentation] doit être inférieure à 5W".

Séquence :
— On ne peut pas savoir automatiquement qu'une telle exigence manque, à moins d'avoir développé une ontologie assez poussée.
— On peut trier toutes les exigences et sélectionner uniquement celles qui font référence aux différents états.
— À partir de ces sélections plus réduites, il va être plus facile de trouver une éventuelle exigence manquante.
— On rajoute ensuite l'exigence dans la spécification.

Remarque : le même principe est valable pour vérifier l'absence de contradictions ou de redondances : en sélectionnant un sous-ensemble de toutes les exigences qui traitent d'un état ou d'un élément précis, il sera plus facile de détecter des problèmes.

Scénario 05 : écrire des exigences classées par priorité Point de départ : un ensemble d'exigences non classées.

Séquence :
— Un classement par priorité fin demande de comprendre le sens des exigences, et n'est donc pas possible automatiquement
— Par contre, de manière générale, on peut vouloir donner la priorité à des exigences portant sur certaines interfaces du système. Par exemple, pour un avion, les éléments portant sur le système de divertissement en vol sont potentiellement de priorité plus basse que les systèmes liés à la sûreté.
— On peut donner un rang particulier à toutes les exigences faisant référence à tel élément, ou contraindre les exigences portant sur l'élément X à avoir une priorité plus faible que les exigences portant sur l'élément Y.
— Cela peut permettre de commencer de classer les exigences par priorité, mais une

intervention humaine sera nécessaire pour obtenir un classement finalisé.

4.3.2.3 Problèmes hors de notre contexte ou trop difficiles

Scénario 06 : écrire des ensembles d'exigences faisables Point de départ : un ensemble d'exigences où deux exigences ou plus ne peuvent pas être respectées en même temps (à cause de contraintes de temps/ressources/physiques/...) comme : "La batterie doit avoir une charge de plus de 2 kWh" et "Le système doit peser moins de 2 kg".

Séquence :
— On ne peut pas détecter informatiquement de tel problèmes, car cela nécessiterait de comprendre le sens des exigences, ainsi que de connaître le contexte industriel.
— Nous considérons ce genre de problème comme hors de notre contexte : il faut les résoudre, mais les outils que nous proposons ne peuvent pas aider.

Scénario 07 : écrire des exigences traçables Point de départ : un besoin ou une exigence de haut-niveau tel que "Les freins ne doivent être appliqués que si le conducteur appuie sur la pédale de frein"

Séquence :
— On déduit de cette exigence haut niveau une exigence au niveau du système : "la [pression_freins] doit toujours être inférieure à la [pression_maître-cylindre]"
— Il faut tracer un lien explicite entre cette exigence et l'exigence de niveau supérieur.
— Cependant, ce n'est pas le sujet de notre travail, et ces liens de traçabilité sont hors de notre contexte.

Scénario 08 : écrire des exigences correctes Point de départ : une exigence telle que "Le système doit être violet à pois verts"

Séquence :
— Un programme ne peut pas savoir si l'exigence correspond à un besoin réel d'une partie prenante.
— Pour le vérifier, il est nécessaire d'analyser l'exigence et sa source.
— Encore une fois, c'est hors de notre contexte : les outils que nous proposons ne peuvent pas aider, mais il y a d'autres outils qui sont plus adaptés.

Chapitre 5

Étude de cas

5.1 Présentation du système

Nous avons étudié en détail une spécification industrielle, qui concerne un système aéronautique : l'Electric Green Taxiing System (EGTS). Ce système est composé de parties physiques et logicielles et est un système critique. La spécification n'est ni un premier jet ni complètement finalisée, ce qui la rend intéressante : il était toujours possible de trouver des problèmes, qui de plus n'avaient pas de solutions évidentes puisqu'ils étaient déjà passés au travers de plusieurs revues.

Une expérience a consisté à traduire cette spécification en une spécification "plus formelle". Dans cette spécification formalisée, nous avons ajouté des modèles, des liens entre exigences et modèles, et des éléments de syntaxe formelle à certaines exigences. Évidemment, cette "traduction" n'était pas la seule possible : pour commencer, il a fallu interpréter des éléments ambigus de la spécification originale. Ensuite, nous avons dû choisir quand arrêter la formalisation : comme on l'a noté dans la section 4.2.3.3, il aurait été possible de formaliser davantage, mais nous avons essayé de nous limiter à un effort "réaliste" de formalisation.

Nous présentons des résultats obtenus lors de cette "traduction".

5.1.1 Contexte

Le but de l'EGTS est de pouvoir déplacer un avion de ligne mono-couloir au sol sans avoir à utiliser ses réacteurs. La consommation de carburant lors des opérations de taxi (déplacement au sol) peut aller jusqu'à 6% de la consommation totale de carburant d'un avion, notamment parce que les moteurs d'avions ne sont pas optimisés pour l'utilisation au sol. Économiser ce carburant permet de diminuer les coûts d'opération et de limiter les émissions de gaz à effet de serre. De plus, ne pas utiliser les réacteurs au sol a d'autres avantages, tels que :

— Réduction des risques d'ingestion de débris par les moteurs.
— Réduction des risques pour les humains sur l'aire de trafic.
— Réduction du bruit dans l'aéroport.

L'EGTS utilise des moteurs électriques situés dans les trains d'atterrissage principaux, alimentés par le groupe auxiliaire de puissance, pour permettre le déplacement de l'avion. Il existe des coûts additionnels dus à l'EGTS, notamment une consommation accrue de carburant pour alimenter le groupe auxiliaire de puissance, les coûts de maintenance de l'EGTS, et une consommation de carburant supplémentaire à cause du poids additionnel du système.

Malgré un intérêt des constructeurs d'avions, des compagnies aériennes et des aéroports, le projet EGTS tel que développé par une coentreprise Honeywell-Safran a été abandonné, "à cause de l'importante diminution du prix du pétrole et de l'environnement économique actuel de industrie aéronautique" Gubisch (2016).

5.1.2 Fonctionnement général

5.1.2.1 Composants

L'EGTS est constitué de :

— Deux actionneurs, un pour chaque train d'atterrissage principal. Ces actionneurs transforment l'énergie électrique en énergie mécanique (rotation des roues). Chacun de ces actionneurs inclut un moteur électrique, des éléments de transmission, un système d'engagement/désengagement pour isoler le moteur des roues, ainsi que des capteurs.
— Des équipements d'électronique de puissance pour alimenter les actionneurs.
— Des composants hydrauliques utilisés pour le système d'engagement/désengagement des actionneurs.
— Un contrôleur.

5.1.2.2 Besoins

L'EGTS doit être capable de fournir un couple positif jusqu'à une vitesse de l'avion en marche avant de 20 nœuds, couple dépendant de la commande du pilote et de la puissance électrique disponible. Il faut également que le système soit capable de fournir une vitesse en marche arrière supérieure à 2 nœuds. Le système doit également fournir une protection contre le glissement des roues au sol. Il est nécessaire que le système puisse être désengagé pour les phases où il n'est pas utilisé. Finalement, le système doit fournir des informations à l'équipage, aux équipes de maintenance et aux enregistreurs de vol.

Le système doit également respecter un ensemble d'exigences portant notamment sur les conditions environnementales, la sûreté, la fiabilité, la maintenance, l'installation.

5.1.3 Interfaces

Il existe de nombreuses interfaces entre le système et le reste de l'appareil. Comme on l'a indiqué précédemment, l'EGTS obtient sa puissance électrique du système électrique de l'avion. Cette puissance électrique est transformée en puissance mécanique et transmise aux roues. Des information de contrôle sont envoyés à l'EGTS par l'IHM. Il y a également

des données provenant des moteurs de l'avion, du système de freinage, des systèmes hydrauliques et des trains d'atterrissage qui sont fournies à l'EGTS de manière à garantir une opération sûre. Le système fournit des informations de statut à un système qui peut alerter l'équipage en cas de problème et à un système se focalisant sur la maintenance. Pour finir, des interfaces physiques sont définies pour les différents emplacements où le système est installé (sur les trains d'atterrissage et à l'intérieur du fuselage de l'appareil).

Un modèle de l'environnement résumant ces interfaces avec le système est inclus dans la spécification. De plus, un certain nombre de ces interfaces sont détaillées dans les exigences, parfois à l'aide de tableaux similaires au tableau 3.5. Il y a aussi des annexes indiquant les emplacement physiques où les éléments du système peuvent être installés.

5.1.4 Spécification

La spécification compte 171 exigences au total. Elle se présente sous forme d'un fichier Portable Document Format (PDF) de 65 pages, en incluant les annexes et les pages d'introductions. Chaque exigence est sous la forme d'un texte d'une à quelques lignes, accompagné d'un identifiant. Les exigences sont parfois accompagnées de schémas ou de tableaux. Il y a parfois également des remarques précisant l'exigence.

Un ensemble d'exigences de la spécification font la description d'un automate à états, en utilisant du texte en langue naturelle. Il y a aussi un certain nombre d'exigences qui peuvent être considérées comme des définitions (comme dans l'exemple “vitesse du véhicule” de la section 4.2.2.1), des suppositions (section 3.1.1.3) ou des hypothèses (section 3.1.1.2).

5.2 Traduction vers une spécification “plus formelle”

Il est assez complexe de faire des expériences portant sur les processus de conception. Pour pouvoir tester les outils que nous proposons sans avoir à les intégrer à un véritable processus de conception d'un véritable système industriel, nous avons étudié une spécification réelle d'un système industriel (ici l'EGTS) et en avons déduit une spécification telle qu'elle aurait pu être si on avait appliqué les principes présentés dans ce travail.

5.2.1 Principe

En étudiant des spécifications industrielles et en discutant avec les personnes de l'industrie qui travaillent sur ces spécifications, nous avons identifié des problèmes généralement rencontrés dans les spécifications. Nous avons trouvé de tels problèmes dans la spécification de l'EGTS. Après avoir réfléchi à des solutions à ces problèmes, nous avons cherché à réécrire la spécification de l'EGTS en en éliminant le plus possible.

Pour faire cela, nous avons dû tout d'abord comprendre le principe de la spécification et les grandes lignes de ce qui y est demandé. Il a également fallu reprendre les modèles qui étaient inclus plus ou moins explicitement dans cette spécification. Nous avons aussi identifié les références implicites contenues dans les exigences pour les remplacer par des références explicites. Finalement, nous avons repris les exigences en introduisant la syntaxe

formelle détaillée dans ce rapport. (Ces trois dernières étapes ne se suivent pas de manière chronologique en pratique, et ont plutôt été réalisées en même temps.)

5.2.1.1 Compréhension de la spécification

Nous pensons que, comme pour toute communication, connaître le contexte est très important pour permettre une bonne compréhension. Une des premières tâches a donc été d'identifier le contexte de la spécification.

Les spécifications donnent généralement quelques éléments de contexte, soit directement dans le préambule, soit dans les exigences elles-mêmes. Par exemple, les exigences concernant la fiabilité et la maintenance vont insister sur le fait que le système doit être relativement facile à réparer ou à remplacer, et que les pannes n'aient généralement pas de gros effets sur l'appareil et sa mission. Cependant, ce contexte n'est pas suffisant, notamment parce que je ne connaissais pas particulièrement le contexte de l'industrie aéronautique au début de mon doctorat. Mes discussions avec les contacts à Safran ont donc été très intéressantes et utiles.

En plus du contexte du système spécifié lui-même, il y a aussi le contexte lié au processus de l'ingénierie des exigences :

— À quoi va servir la spécification de l'EGTS ?
— Pour qui et par qui est-elle écrite ?
— Quels sont les documents pertinents des niveaux supérieurs (sur lesquels est basée la spécification) et inférieurs (qui se basent sur la spécification) ?
— Sous quelle forme la spécification est-elle écrite à la base (par exemple, avec un programme de gestion des exigences tel que DOORS, ou juste un programme de traitement de texte) ?

Encore une fois, je n'avais que très peu entendu parler d'ingénierie des exigences auparavant, et les discussions avec les personnes concernées dans l'industrie ont été très formatrices.

5.2.1.2 Reprendre les modèles

Dans la spécification de l'EGTS, un modèle du système et de son environnement est fourni. Cependant, les interfaces entre ces deux parties ne sont pas définies de manière homogène : certaines étaient relativement précises et formelles, d'autres moins. De plus, ces informations sont réparties un peu partout dans la spécification. Comme mentionné précédemment, un ensemble d'exigences fait essentiellement la description d'un automate à états, en utilisant du texte en langue naturelle.

Nous avons écrit un modèle d'architecture qui rassemble toutes les informations pertinentes et relatives à l'architecture (systèmes, sous-systèmes, interfaces, attributs des systèmes et des interfaces...). Une partie de ce modèle est donné en figure 5.1.

Nous avons également modélisé un automate à états qui reprend les états et les transitions définis dans les exigences originales, présenté en figure 5.2. Nous avons également rajouté des états permettant de simplifier l'expression de certaines exigences. Les “durées des transitions” mentionnées dans la section 4.2.2.2 ne sont pas indiquées.

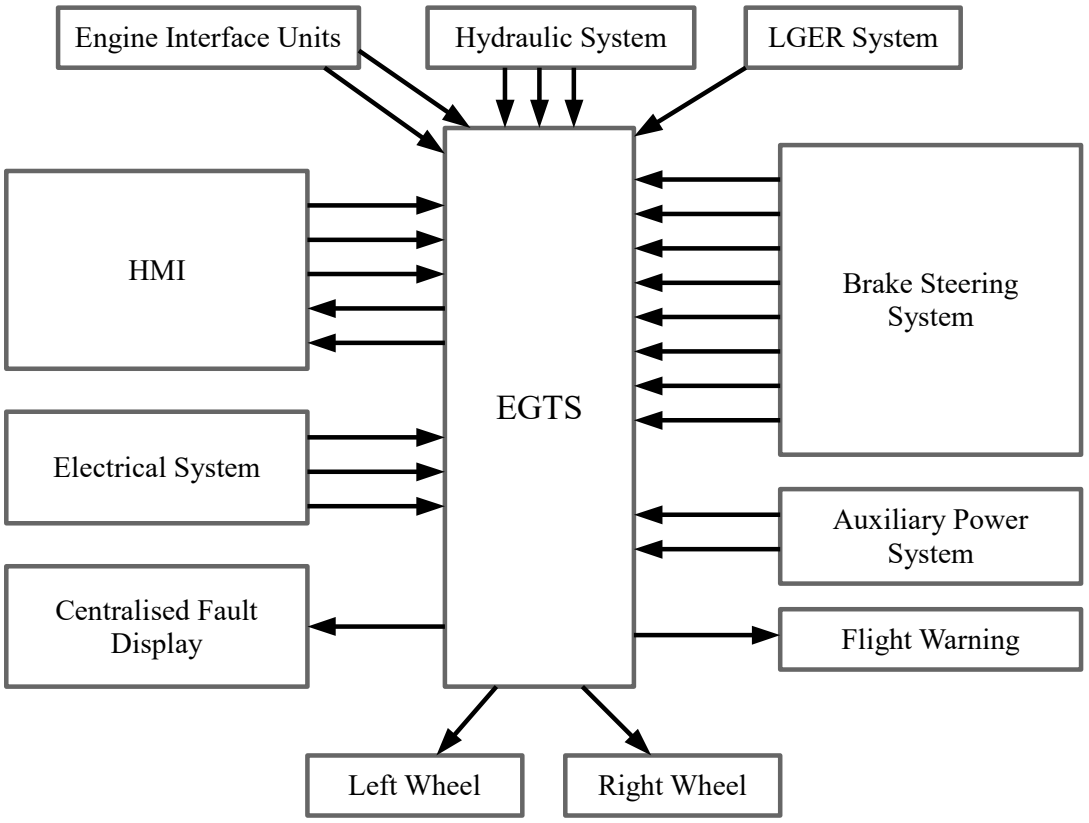


FIGURE 5.1 – Interfaces de l’EGTS avec les systèmes externes

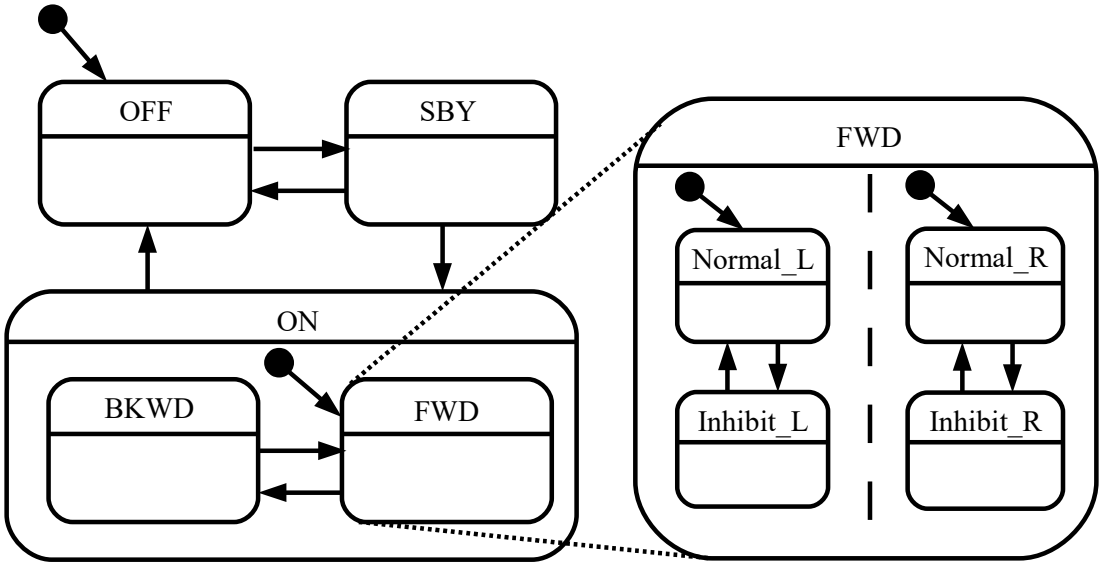


FIGURE 5.2 – Automate à états pour l’EGTS

Sur les 171 exigences de la spécification originale, on en a remplacé 30 par des modèles ou des précisions sur les modèles.

5.2.1.3 Tracer des liens

On cherche à identifier les concepts qui apparaissent dans les exigences et expliciter les références vers les définitions de ces concepts. On va répéter une routine :

— Identifier un concept dans une exigence que l’on a pas encore rencontré dans d’autres exigences.
— Identifier les endroits où l’on fait référence à ce concept dans d’autres exigences.
— Identifier ce concept parmi les éléments de modèles (ou autre).
— Écrire les liens explicites entre les exigences et l’élément de modèle.

Ces étapes ne sont pas forcément évidentes. Par exemple, il peut être difficile d’être sûr que plusieurs éléments font référence à des concepts différents, ou font référence au même concept. Ces concepts ne sont pas nécessairement déjà présent dans les modèles. Si c’est le cas, il faut compléter les modèles (on voit encore une fois le principe de “co-construction” entre modèles et exigences).

Une fois ces liens tracés, comme métrique simple, on peut compter le nombre de références. Par exemple, dans l’Exi. 1, “la puissance d’alimentation”, “la pression dans les freins” et “la pression dans le maître-cylindre” sont trois références au modèle présenté dans le schéma 4.6. Quand nous avons “traduit” la spécification de l’EGTS, nous avons rendu ces références explicites et permis de les compter facilement.

Pour les 141 exigences restantes, nous avons écrit 376 liens vers des éléments externes aux exigences. Ces éléments peuvent être des composants du modèle d’architecture, des

états d'une machine à états, ou des annexes telles que des tables ou des schémas. La répartition de ces liens parmi les exigences est assez hétérogène, avec certaines exigences incluant six ou sept liens, alors que d'autres n'en ont qu'un seul. Par exemple, l'exigence “The Taxiing System shall avoid design features that may generate intrusive noise in the cabin or cockpit” n'a qu'un seul lien, “The Taxiing System”, qui fait référence au bloc système correspondant dans le modèle d'architecture.

5.2.1.4 Reprendre les exigences

Nous avons vu qu'il était possible d'écrire des exigences avec une syntaxe formelle. Nous avons donc étudié les exigences de l'EGTS et, quand c'était possible et cohérent, nous avons écrit des exigences correspondantes dotées de cette syntaxe formelle. Comme indiqué dans la section 4.2.3.3, nous n'avons pas cherché à formaliser toutes les exigences.

Par exemple, prenons l'exigence mentionnée ci-dessus : “The Taxiing System shall avoid design features that may generate intrusive noise in the cabin or cockpit”. Il n'y a pas d'autres exigences qui font référence au “bruit dans la cabine ou le cockpit”, on peut donc se demander s'il est pertinent de créer un élément de modèle pour une seule exigence. L'exigence originale n'est pas très précise : à partir de quand un son devient-il gênant ? Il est possible de spécifier un volume limite acceptable, mais il y a probablement d'autres éléments qui peuvent rendre un son gênant (fréquence du son, s'il est répété…).

Est-ce un problème si cette exigence n'est pas précise ? Est-ce que l'effort nécessaire pour écrire une exigence formalisée est plus important que l'avantage d'avoir cette exigence sous forme formelle ? Nous avons considéré qu'il n'était ici pas “rentable” de traduire cette exigence en une exigence formelle. D'autres exigences étaient davantage susceptibles d'être formalisées.

Nous n'avons pas cherché à traduire “mot à mot” les exigences : certaines exigences ont été remplacées par des modèles ou des éléments de modèles, et nous avons également reformulé certaines exigences pour qu'elles aient le même sens mais soient plus compréhensible, ou écrites de manière homogène avec d'autres exigences similaires.

Quand nous avons “traduit” la spécification industrielle, nous avons formalisé la syntaxe de 55 des 141 exigences. Les autres exigences ont simplement été écrites en texte libre.

5.2.2 Différences avec la création d'une spécification

Nous avons réalisé cette traduction pour servir d'expérience, car réaliser une expérience directement sur un processus de conception réel était difficilement possible. Il faut donc se demander quelles sont les différences avec l'expérience réalisée et comment les principes proposés dans ce travail s'appliqueraient sur un cas où la spécification n'est pas déjà écrite.

5.2.2.1 Interprétation du texte

Tout d'abord, il a nécessairement fallu interpréter le texte de la spécification EGTS. Ce texte contient nécessairement des ambiguïtés potentielles vu qu'il est rédigé en langue naturelle. Nous avons donc dû faire des choix quand nous avons détecté ces ambiguïtés :

par exemple, nous avons choisi de considérer que “eTaxi HMI discrete ORDER_BKWD” et “eTaxi HMI backward command” faisaient référence au même concept.

Mais il y a probablement d’autres cas où nous avons supposé que telle exigence avait tel sens, sans nous rendre compte qu’un autre lecteur (voire même l’auteur) aurait pu l’interpréter d’une autre manière. Une fois les exigences écrites de manière formelle, il est très difficile d’y voir un autre sens que celui désiré par l’auteur (c’est après tout le but de notre travail). Cela signifie qu’il faut faire attention lorsque l’on transforme des éléments non-formels en éléments formels : puisque nous ne sommes pas les auteurs de la spécification originale, il est possible que nous ayons parfois modifié le sens voulu par ces auteurs. Si les auteurs d’une spécification sont les mêmes que ceux qui la formalisent, ce problème devrait être moins présent.

5.2.2.2 Jusqu’où formaliser ?

Lors de la traduction, même si l’on a adapté certains éléments (des exigences ou des modèles), il a fallu se baser sur l’existant. Lors de la création d’une spécification, même si l’on ne part généralement pas d’une feuille complètement blanche, on est plus libre dans la formulation des exigences et des modèles.

Ici, cela a une influence importante sur les éléments que l’on a formalisé ou pas pendant la traduction : il est relativement facile de formaliser une exigence déjà bien écrite, parce que l’on va pouvoir identifier facilement les éléments qui la composent. Pour une exigence qui n’est pas précise à l’origine, on risque de surinterpréter, et nous n’avons pas essayé de rajouter trop d’éléments nouveaux (comme des modèles par exemple) par rapport à ce qui était déjà présent dans la spécification. Nous avons essayé de “deviner” quels étaient les éléments qui demandaient trop d’effort à formaliser, mais nous n’avons pas réellement de preuves concrètes.

Dans le cas où il n’y a pas de spécification déjà écrite, on peut voir plus clairement quels éléments sont trop complexes à modéliser/formaliser lorsqu’on essaye de les écrire.

5.2.2.3 Co-construction

La relation entre modèles et exigences est également différente dans le cas de la traduction. Ici, nous avons essayé de faire communiquer des exigences et des modèles une fois ces éléments donnés. Le résultat serait probablement différent si la création de ces éléments coïncidait avec la création de liens explicites entre eux.

Évidemment, les modèles et les exigences de la spécification originale n’ont pas été développés de manière complètement indépendante, mais probablement pas non plus de manière complètement liée. Comme nous l’avons indiqué précédemment, nous pensons que faire des liens systématiques entre exigences et modèles va avoir des effets importants sur la qualité de ces éléments.

5.2.3 Exemples

Nous donnons quelques exemples d’exigences de la spécification EGTS et comment nous les avons traités.

5.2.3.1 Définition d’un état

Une des exigences servant à définir l’automate à états dans la spécification originale est la suivante :

“The Taxiing System shall have an OFF mode where :
— The power electronics are unpowered
— The Wheel actuators do not apply any retardation force or traction force to the aircraft wheels”

Dans la spécification originale, cette exigence est la première fois où l’on fait référence à un “OFF mode”. Il n’existe pas d’autres exigences ou de modèles qui serviraient à définir l’existence de ce mode. Cette exigence a donc un but double : définir l’existence d’un état “OFF”, et donner des propriétés requises pendant que cet état est actif. Comme nous l’avons développé plus haut, nous pensons que le premier but est mieux rempli en écrivant un modèle d’automate à états. Nous pouvons par contre écrire une exigence correspondant au second but, par exemple : “In state [OFF], the [Input Power] and the [Wheel Torque] shall be null”

On a également reformulé les propriétés demandées en les exprimant au moyen des interfaces du système.

5.2.3.2 Définition d’une transition

D’autres exigences servent à définir les transitions entre états :

“The Taxiing System shall transition into ACTIVE mode when all the following conditions are met :
— the STANDBY mode is True
— the A/C ground speed is anywhere between -2KT included (backward) and +20KT included (forward)”

Comme précédemment, nous avons considéré qu’il était plus pertinent de définir cette transition dans un automate à états. Ici, comme l’exigence ne fait que définir cette transition, elle a été complètement remplacée dans la traduction par la transition dans l’automate à états.

5.2.3.3 Une exigence plus classique

Il n’est pas demandé au système de fournir une vitesse au sol supérieure à 20 nœuds en marche avant, mais si il arrive que la vitesse de l’appareil soit supérieure à cette limite (peut-être à cause d’une pente, d’un tracteur d’aviation ou autre), un exigence demande que le système ne freine pas l’appareil.

“When in FWD control mode, if the A/C [1] ground speed exceeds 20KT forward, the Taxiing System shall :
— Not oppose the A/C acceleration

1. Aircraft

— Not be damaged"

L'antécédent de cette exigence est pratiquement déjà formalisé. Pour le conséquent, il pourrait être utile d'expliciter “Not oppose the A/C acceleration” en indiquant à quoi cela correspond sur les interfaces du système. Nous avons considéré que cette expression était équivalente à “le couple de sortie doit être positif” (i.e. ne pas être un couple de freinage). “Not be damaged” est une expression rencontrée assez fréquemment dans les spécifications. Nous avons défini une fonction syntaxique correspondant à cette expression, mais si ce n'est pas le cas, nous aurions pu la laisser telle quelle, comme texte libre, sans problème particulier.

5.2.3.4 Une formalisation plus complexe

Une exigence de la spécification EGTS est la suivante : “In case of wheel actuator disengagement failure leading to a wheel lock, it shall be possible to manually disengage the wheel actuator to release the blocked wheel without removing the wheel or jacking the MLG in less than 5min.”

Même si cette exigence a une structure et un sens relativement clairs, nous n'avons pas pu la formaliser entièrement. En effet, il aurait fallu définir de manière formelle un certain nombre d'éléments, tel que “wheel actuator disengagement”, “manually disengage the wheel actuator” ou “jacking the MLG”. Nous avons considéré que définir ces éléments dans un ou des modèles (architecture ou autres) aurait demandé trop d'efforts.

5.2.4 Remarques

Nous avons aussi étudié, moins en détail, d'autres spécifications industrielles qui concernaient des systèmes aéronautiques comparables mais qui étaient écrites par d'autres équipes. Nous avons trouvé que les principes présentés ici pouvaient être appliqués relativement bien à ces autres spécifications. Nous n'affirmons pas que les concepts développés dans ce travail peuvent être appliqués à n'importe quelle spécification de système, et davantage de travail est nécessaire pour déterminer la portée de notre travail.

5.2.4.1 Durée de la traduction

Il est difficile de donner des durées précises, qui permettent d'avoir une idée de l'effort nécessaire pour réaliser les différentes étapes présentées ici (acquisition du savoir, réécriture des exigences, création de modèles...). En effet, nous avons conçu ces activités en même temps que nous les avons réalisées. De plus, ces différentes activités ont aussi été réalisées de manière plus simultanée que séquentielle.

On peut cependant donner des ordres d'idées de ces durées. Concernant l'acquisition du savoir (contexte, vocabulaire spécifique, justifications sous-jacentes) sur une spécification, nous pensons que le temps nécessaire dépend fortement de la présence et de la disponibilité de spécialistes. Dans notre cas, avec des discussions au moins hebdomadaires avec un spécialiste du domaine, trois semaines nous ont permis de comprendre la plus grande partie de la spécification EGTS. Il y a toujours des détails qui ne se révèlent ambigus ou difficiles à comprendre que lorsque l'on étudie la spécification de manière beaucoup plus poussée.

Concernant la réécriture des exigences, nous estimons à environ un mois, pour une personne seule, le temps nécessaire pour réécrire les 171 exigences de la spécification EGTS. Une partie de ces exigences n'ont pratiquement pas été modifiées et ont donc été assez rapides à traiter, alors que d'autres ont demandé plus de réflexion et de travail. Il faut également remarquer que les exigences réécrites n'ont pas été vérifiées/validées, en tout cas pas au niveau de véritables exigences industrielles.

La création/réécriture des modèles a été assez rapide, notamment parce que ces modèles étaient au moins partiellement définis dans la spécification originale : cette étape de modélisation a duré environ deux semaines, toujours pour une personne seule.

De manière générale, cette traduction a donné une spécification cohérente, mais qui n'a pas été ensuite réintroduite dans un véritable processus industriel. On peut donc imaginer qu'il faille un temps supplémentaire pour obtenir une spécification vérifiée et validée. Ceci dit, nous pensons que l'effort demandé est suffisamment limité, surtout en comparaison de l'utilité de disposer d'une spécification moins ambiguë et plus pratique, pour que l'approche proposée soit viable économiquement.

5.3 Résultats des tests

5.3.1 Remarques préliminaires

Nous avons réalisé un certain nombre de tests sur la spécification "formalisée". Évidemment, puisque ces tests nécessitent les références explicites et la syntaxe présentées dans ce travail, nous n'avons pas pu tester directement la spécification originale. Nous avons modifié, parfois de manière relativement importante, les exigences originales. La relation entre les problèmes détectés par des tests sur la spécification "formalisée" et les problèmes qu'a effectivement la spécification originale n'est donc pas évidente. De plus, certains tests vérifient des propriétés qui sont intrinsèquement liées à ces nouveaux éléments : par exemple, le test vérifiant que les liens explicites ont bien une cible existante et valide n'a pas de sens si ces liens n'existent pas.

Nous avons cependant essayé de réaliser la traduction sans inventer de nouveaux problèmes, et sans trop en retirer de la spécification originale.

Cela a forcément nécessité des interprétations : par exemple, certaines exigences mentionnent le couple appliqué par le système sur les roues (en Nm) et d'autres mentionnent la force appliquée par le système sur les roues (en N). Une relation explicite entre ces valeurs n'est pas donnée dans la spécification originale et l'interface mécanique entre le système et les roues n'est pas suffisamment détaillée pour savoir si cette interface est modélisée une force ou un couple, voire les deux. Si, une fois que l'on a précisé les références explicites et la syntaxe, il est facile de déterminer si, oui ou non, un exigence fait bien référence aux interfaces (cf section 4.3.1.1), il est plus difficile de trancher pour la spécification originale.

5.3.2 Tests

5.3.2.1 Homogénéité des termes

Ce test, détaillé en section 4.3.1.1, consiste à vérifier qu'un même concept est toujours désigné avec le même texte dans les exigences. Ce test est assez simple à réaliser, de plus, la "formalisation" de la spécification n'a pas beaucoup affecté les résultats : dans les exigences formalisées, on a conservé tels quels les termes utilisés dans les exigences originales. Nous avons comparé directement les chaînes de caractères, il y a donc un certain nombre de différences très mineures dues à la présence ou non de majuscules ou d'articles. Certaines différences sont dues à l'utilisation inhomogène d'acronymes, par exemple "Main Landing Gear" dans une exigence et "MLG" dans une autre.

D'autres différences peuvent être plus problématiques : le système est parfois désigné "Taxiing System", parfois "eTaxi System" ; l'alimentation de puissance est parfois nommée "main input electrical power", parfois "electrical supply" parfois "power supply".

5.3.2.2 Référence aux interfaces

Comme indiqué plus haut, dans la spécification originale, il est difficile de dire si tel élément fait bien référence aux interfaces du système, puisque les références ne sont pas explicites. Nous avons construit la spécification formalisée de manière à ce qu'elle soit cohérente, ce qui nous a permis de nous apercevoir que certains éléments mentionnés dans les exigences n'avaient pas de définitions explicites. Par exemple, le "couple commandé" ou la "vitesse commandée" n'ont pas de définitions claires. Certaines exigences font références à un "couple maximal" ou une "puissance électrique maximale" qui ne sont pas définis (ou alors définis de manière très peu claire).

5.3.2.3 Référence à toutes les interfaces

On peut vérifier que toutes les interfaces mentionnées dans le modèle d'architecture sont liées à au moins une exigence (cf section 4.3.1.1). C'est le cas dans la spécification de l'EGTS.

5.3.2.4 Atomicité

On a cherché analyser si les exigences étaient atomiques en réalisant le test présenté en section 4.3.1.2. Aucune n'a une structure du type "$X \wedge Y$", par contre, sur les 55 exigences dotées d'une syntaxe formelle, 13 ont une structure du type "$Z \implies (X \wedge Y)$". Sur ces 13 exigences, certaines ne posent a priori pas de problèmes. Notamment, une partie de ces "non-atomicités" sont dues à la définitions d'intervalles de tolérance pour une sortie du système. Il pourrait être utile de raffiner le test pour limiter le nombre de faux positifs.

5.3.2.5 Complexité

On a réalisé un test donnant la profondeur maximale de l'arbre syntaxique pour chaque exigence. L'exigence avec la plus grande profondeur (9) est celle-ci :

"The probability of an unannunciated failure of the Taxiing System leading to retardation force greater than 700daN and lower than 2400 daN on one wheel or more when the A/C speed is <V1 shall be less than 1E-8/FH during the Take/Off roll."

Trois exigences ont une profondeur de 8, huit ont une profondeur de 7, quatorze de 6. La plupart ont une profondeur de 2, 3 ou 4. S'il est assez clair qu'une plus grande profondeur est liée à une plus grande complexité, davantage d'études seraient intéressantes pour mieux définir cette relation, et faire intervenir d'autres facteurs, comme une mesure de la largeur de l'arbre syntaxique.

5.3.2.6 Autres tests sur la syntaxe

Nous n'avons pas trouvé de cas de typage incohérent (comme essayer de comparer un couple et une force).

Toutes les unités utilisées ne font pas partie du système international, mais leur utilisation est au moins homogène (on n'utilise pas des mètres par seconde dans une exigence et des nœuds dans une autre).

Nous avons construit les exigences de la spécification formalisée de manière à ce que toutes les exigences soient de type booléen.

Chapitre 6

Prototype logiciel

6.1 Architecture et formats

Dans cette section, nous présentons un prototype de programme permettant de réaliser des tests sur les exigences. Ce prototype ne peut pas être utilisé comme outil de gestion des exigences et ce n'est pas son but, nous cherchons à montrer que les concepts proposés dans ce travail peuvent être utilisés en pratique. Idéalement, les programmes de test et les éléments logiciels les accompagnant pourraient être intégrés à un logiciel de gestion des exigences déjà existant.

6.1.1 Présentation des différents composants

Tout d'abord, nous détaillons les formats des principaux composants d'une spécification qui seront utilisés pour les tests : les exigences, les modèles d'architecture, les "abstractions composées" telles que "vitesse du véhicule" de la section 4.2.2.1, et les automates à états. Nous présentons également des éléments qui ne sont pas nécessairement attachés à une spécification en particulier, mais qui sont indispensables pour le bon fonctionnement des tests : les définitions de fonctions et le système d'unité. Tous ces éléments sont pour le moment écrits et conservés en format XML. Le format XML a l'avantage d'avoir une structure d'arbre intrinsèque, qui correspond bien aux différents éléments. De plus, il existe des "parsers" permettant d'importer et de manipuler facilement des éléments XML avec Python, qui est le langage de programmation que nous utilisons.

De façon très résumée, un fichier XML est un arbre enraciné (un graphe acyclique orienté, possédant une unique racine, dont chaque nœud ne peut avoir qu'un seul parent). Chaque nœud est un élément qui a une "étiquette" ("tag" en anglais), et qui peut avoir des éléments fils. Ces éléments peuvent aussi avoir des attributs.

6.1.1.1 Exigences

Un exigence est un élément avec une étiquette "Req", doté d'un identifiant en attribut (on peut envisager d'ajouter d'autres attributs de l'exigence directement comme attributs

XML, avec la limite que ces attributs doivent être des chaînes de caractères). Cet élément "Req" a trois éléments fils :

— Un élément doté d'une étiquette "Prop" contient la propriété demandée par l'exigence.
— Un élément (doté d'une étiquette) "Syst" fait référence au système auquel on demande cette propriété.
— Un élément "Demand" définit le niveau d'obligation de l'exigence ("shall", "should"). On peut considérer par défaut que toutes les exigences ont le niveau d'obligation "shall", et donc ignorer cet élément.

L'élément "Demand" est assez simple et doit seulement contenir une chaîne de caractère correspondant au niveau d'obligation ("shall", "should" ou autre). L'élément "Syst" doit contenir un arbre d'éléments de forme quelconque, dont le type final doit être "BlockSys", correspondant à un (ou plusieurs) système du modèle d'architecture.

L'élément "Prop" doit contenir un arbre d'éléments de forme quelconque, mais dont le type final doit être "Bool" (i.e. un booléen). L'arbre XML de cet élément a la même structure que la structure syntaxique de l'exigence : par exemple, l'arbre syntaxique de l'exigence 1 en figure 4.11 a la même forme que l'arbre XML contenu dans l'élément "Prop" de cette exigence.

Les éléments XML à l'intérieur de "Prop" peuvent avoir une étiquette parmi les trois suivantes :

— "func", pour "fonction", le choix de la fonction étant déterminé par l'attribut "funcdef". La valeur de cet attribut doit correspondre à une des fonctions du fichier "définitions des fonctions".
— "a", qui est un lien explicite, dont l'adresse ciblée est la valeur de l'attribut "href".
— "frag", pour les parties non formalisées de l'exigence. L'attribut "type" donne le type de ce fragment de texte libre.

Le fichier d'exigences est simplement une suite d'éléments "Req".

6.1.1.2 Architecture

Un modèle d'architecture est composé de systèmes inclus les uns dans les autres. Le fichier XML est composés d'éléments :

— "Block", représentant les systèmes et sous-systèmes.
— "Port", représentant les interfaces des systèmes.
— "Connection", représentant les connections entre ces interfaces.

Chaque élément "Block", "Port" et "Connection" a un nom. Ces éléments peuvent aussi avoir des "Attributes" qui permettent de donner des attributs plus complexes que des simples chaînes de caractères. Par exemple, si l'on veut définir une caractéristique du système telle que le poids, on va définir un élément, parmi les "Attributes" de ce système, donnant le nom et le type (au moins) de cette caractéristique. De même, on peut définir que telle interface utilise tel type de fiche ou tel type de bus.

Encore une fois, on va utiliser la structure d'arbre intrinsèque de XML pour représenter ces systèmes. Si un système B est directement inclus dans un autre système A, dans le fichier XML, l'élément représentant B sera un élément fils de A, voir figure 6.1. De même,

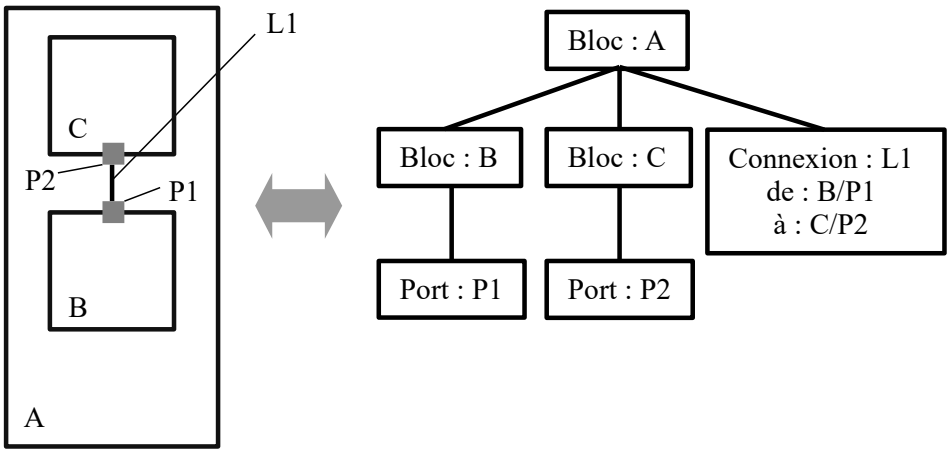


FIGURE 6.1 – Un modèle d'architecture "boîte et flèche" et l'arbre XML correspondant

si un port P1 est une interface du système B, l'élément P1 sera le fils de B. Les connections sont définies comme des fils directs du plus petit système qui les contient entièrement.

Ce format XML correspond à la définition formelle des modèles d'architecture donnée en section 4.2.1.1.

6.1.1.3 Abstractions composées

Pour la définition d'une abstraction composée, il est important de donner les éléments avec laquelle elle est construite. Ces éléments peuvent être des éléments du modèle d'architecture, de l'automate à états ou d'autres abstractions composées.

On va utiliser, comme pour les exigences, des éléments XML d'étiquette "a", dont l'adresse cible est donnée par la valeur de l'attribut "href". On peut aussi définir la façon dont on compose cette nouvelle variable, en utilisant la même syntaxe que pour les exigences : par exemple, si une variable est égale à $A * (B + C)$, on peut écrire un arbre syntaxique correspondant, cf figure 6.2.

6.1.1.4 Automates à états

On va utiliser des automates à états proches des "statecharts" d'UML/SysML, avec la possibilité de définir des états inclus les uns dans les autres. La structure de ces automates à états est, au final, relativement proche de celle des modèles d'architecture, comme on peut le voir dans la figure 6.3. Le fichier XML sera composé d'éléments "State" pour les états et "Transition", pour les transitions. Seuls les éléments "State" peuvent avoir d'autres éléments "State" et "Transition" comme enfants. Dans les éléments "Transition" on définit l'état de départ, d'arrivée, et la condition de la transition. Cette condition est un booléen, que l'on définit en utilisant la même syntaxe que pour les exigences.

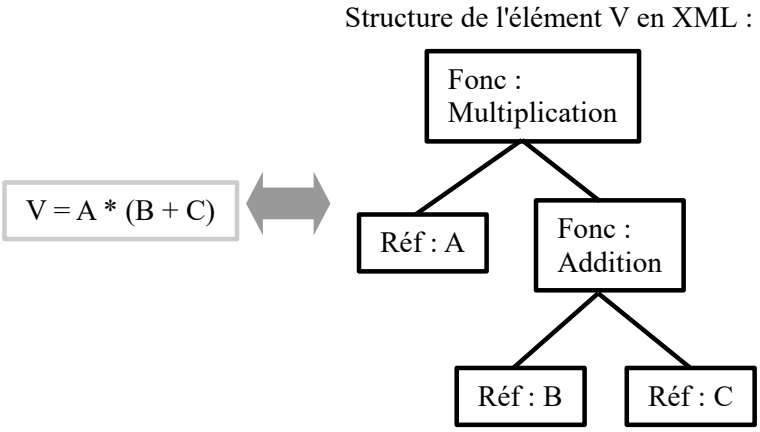


FIGURE 6.2 – Une entrée composée et l'arbre XML correspondant

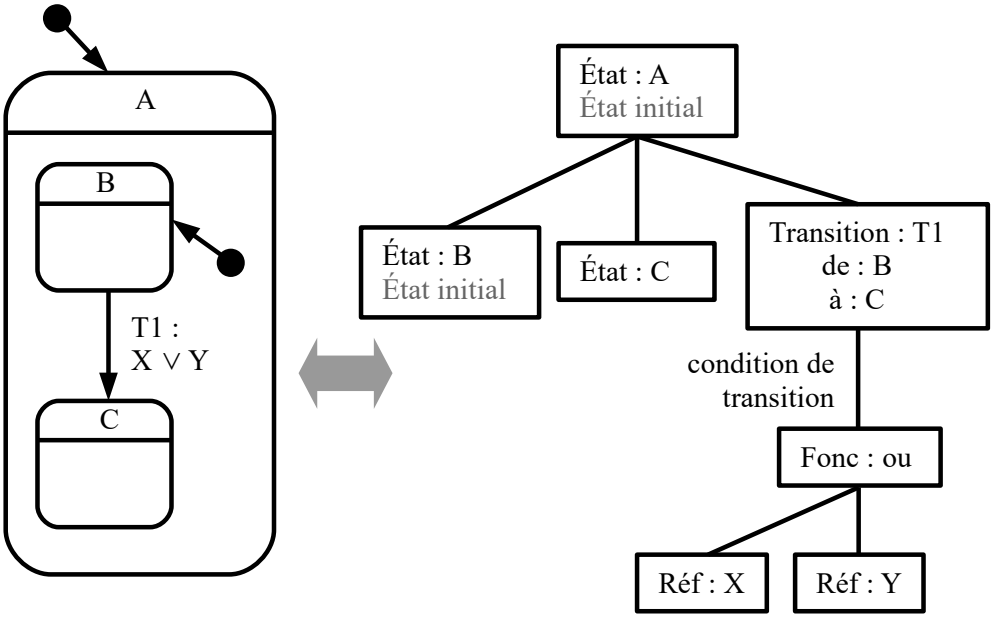


FIGURE 6.3 – Un automate à états et l'arbre XML correspondant

Encore une fois, ce format correspond à la grammaire formelle des automates à états en section 4.2.2.2.

6.1.1.5 Fonctions

Le fichier de définition des fonctions est une liste de fonctions dont l'utilisation est acceptée dans les exigences. Ce fichier est une liste d'éléments "Function". Pour chaque fonction, on définit :

— son nom dans un attribut "Name",
— son ou ses types accepté(s) en entrée, dans un ou plusieurs éléments "Input", fils de "Function",
— son type de sortie, dans un élément "Output", fils de "Function",
— comment écrire la fonction en langue naturelle, dans un attribut "NL".

Il y a quelques autres subtilités, par exemple, on considère que la fonction booléenne "And" peut recevoir un nombre d'entrées, supérieur à deux, non défini à l'avance. Un attribut de la fonction précise donc ce genre de caractéristiques pour le script de vérification (parser) de syntaxe. Les types acceptés en entrée d'une fonction ne sont pas non plus forcément fixés : par exemple, nous n'avons défini qu'une seule fonction "plus grand que", qui doit pouvoir accepter aussi bien des types "Force" que "Couple" en entrée, si les deux entrées sont de même type. Nous avons donc rajouté des attributs, lisibles par le script de vérification de syntaxe, à ces types de fonctions.

6.1.1.6 Types

Le fichier de types rassemble les types acceptés par le parser syntaxique. Ces types sont ordonnés selon une hiérarchie, qui signifie que certains types sont inclus les uns dans les autres : par exemple, "N" et "daN" sont des types distincts, qui sont inclus dans le type "Force", lui-même inclus dans le type général "Measure". Chaque type correspond à un élément XML, dont l'étiquette est le nom du type.

Le système de type utilisé est plutôt ad hoc, et il serait intéressant de pouvoir utiliser un système complet tel que le système international d'unités. Par exemple, le parser syntaxique ne "sait" pas que multiplier une distance par une force donne un couple.

6.1.2 Liens entre ces composants

6.1.2.1 Références explicites

Pour les références explicites, on utilise des balises hyperliens (les éléments "a") de la même manière qu'un lien est défini en HTML par exemple. Ces références, qu'elles soient écrites dans les exigences, dans un automate à états ou dans les abstractions composées, sont donc des liens unidirectionnels, de l'origine vers la cible. Puisque les ensembles d'exigences que l'on considère vont contenir au maximum un millier d'exigences (dû au fait qu'ils doivent être lisibles par des humains), obtenir automatiquement les liens dans l'autre sens est facile et rapide. Une autre solution serait de considérer les liens comme des objets

individuels distincts et de gérer un répertoire de liens. Cependant, cette solution ajouterait encore une chose à gérer et à garder cohérente avec le reste de la spécification.

6.1.2.2 Fonctions

Les fonctions syntaxiques utilisées dans les exigences, les automates à états et les abstractions composées doivent correspondre aux fonctions définies dans le fichier de définition de fonction. La vérification de la correspondance se fait par une comparaison directe des chaînes de caractères : si on a un élément "`<func funcdef='Imply'>`" dans une exigence, on doit avoir un élément dont l'attribut "Name" est exactement "Imply" dans le fichier de définition des fonctions. Les fonctions sont importées dans un dictionnaire Python afin de ne pas avoir à parcourir l'ensemble du fichier de définition à chaque fois que l'on souhaite accéder à une fonction. Il est tout à fait envisageable de remplacer cette correspondance de chaînes de caractère par des liens explicites utilisant un adressage potentiellement plus fiable, ou qui permettrait par exemple de renvoyer la définition de la fonction si on clique dessus dans une exigence.

6.1.2.3 Types

De la même manière, les types présents dans les exigences (et autres) doivent correspondre aux types définis dans le fichier de types. Comme pour les fonctions, la vérification utilise une comparaison directe des chaînes de caractères, et on importe le fichier de définition dans un dictionnaire.

Le processus est cependant un peu plus complexe puisque les types ne sont pas toujours donnés tels quels dans les exigences, mais peuvent être tirés des éléments de l'architecture, de l'automate à états, des abstractions composées, ainsi que des fonctions. Par exemple, lorsque l'on écrit "la pression dans le maître-cylindre est plus faible que 50 bar" (Exi. 6), le type de "50 bar" est "Bar", un sous-type de "Pressure", alors que le type de "pression dans le maître-cylindre", tel que défini dans le modèle d'architecture, est "Pressure". On a défini la fonction "est plus faible que" de manière à ce qu'elle puisse accepter ce cas de figure, même si les éléments "Bar" et "Pressure" ne sont pas exactement les mêmes, tout en rejetant les cas où, par exemple, on essaye de comparer un type "Force" avec un type "Pressure"

6.1.2.4 Remarque

Notre but n'est évidemment pas que les ingénieurs des exigences écrivent leurs exigences/modèles/etc. directement sous le format XML présenté ici. Idéalement, la rédaction de ces composants de spécification se ferait en utilisant un éditeur adapté (déjà existant ou créé pour cela), et les représentations XML seraient générées automatiquement à partir de ces éditeurs.

Le développement de tels éditeurs et de traducteurs automatiques sous le format XML proposé ici est une tâche assez importante qui n'était pas le cœur de notre travail. Nous pensons cependant que de tels éditeurs sont absolument nécessaires pour avoir une méthode d'ingénierie des exigences acceptable. Nous avons donc essayé de construire les concepts

que nous proposons de manière à ce que de tels éditeurs soient faisables.

Concernant les modèles d'architecture et les automates à états, à partir du moment où l'éditeur d'origine inclut quelques concepts de base tels que la hiérarchie (un système est composé de sous-systèmes, un état peut contenir des états fils), il paraît faisable de générer automatiquement les descriptions XML.

Pour les définitions des fonctions et des types, les concepts que nous avons proposés pour l'instant ne sont pas très complexes. Pour définir un nouveau type, il suffit d'avoir un nom et éventuellement de savoir si ce type peut être considéré comme inclus dans un autre type existant. Pour définir une fonction, nous avons essentiellement besoin d'un nombre d'entrées, du type de ces entrées, et du type de sortie. Il paraît faisable de définir un éditeur, éventuellement ad hoc, permettant de créer de nouvelles fonctions ou types. De plus, il n'est a priori pas nécessaire de recréer des dictionnaires de fonctions et de types complets pour chaque nouvelle spécification. Par contre, si l'on souhaite avoir un système de type plus raffiné et compatible avec le système international d'unités par exemple, cela peut être plus complexe.

Pour les exigences, construire un éditeur permettant de manipuler les concepts proposés (références explicites et syntaxe formelle) sans ajouter une charge de travail trop importante semble être une tâche conséquente. Nous ne pensons pas qu'il soit possible de traduire automatiquement un texte en langue naturelle en exigence formalisée. En revanche, nous pensons qu'il est possible d'aider le rédacteur au moment où il écrit une exigence (par exemple en proposant automatiquement des liens ou des fonctions syntaxiques pouvant correspondre au texte écrit), de manière à ce que la charge de travail/de formation ne soit pas trop importante.

Nous évoquons plus en détail ce à quoi ce type d'éditeur pourrait ressembler en section 7.2.3.2.

6.2 Implémentations des propositions

6.2.1 Architecture globale du programme

Nous donnons une représentation de l'architecture du programme en figure 6.4. Nous avons un fichier XML pour chaque partie de la spécification. Ces fichiers XML sont importés et rassemblés dans une instance de classe python "SpecObj". C'est sur cet objet spécification et ses différents composants que sont réalisés les tests.

Une interface graphique assez basique permet à l'utilisateur de lancer les importations et les tests. Il y a un certain nombre de traitements avant et après ces tests afin que les requêtes que l'utilisateur lance restent relativement simples et que les résultats retournés soient bien formatés et donnent des informations pertinentes.

Pour donner un ordre d'idée, le programme python complet comporte 1100 lignes de code. Pour l'EGTS, les définitions XML des exigences, du modèle d'architecture, de l'automate à états et des abstractions composées sont des fichiers longs de 2100, 600, 300 et 130 lignes respectivement. Les fichiers de définition des fonctions et des types font 300 et 100 lignes respectivement.

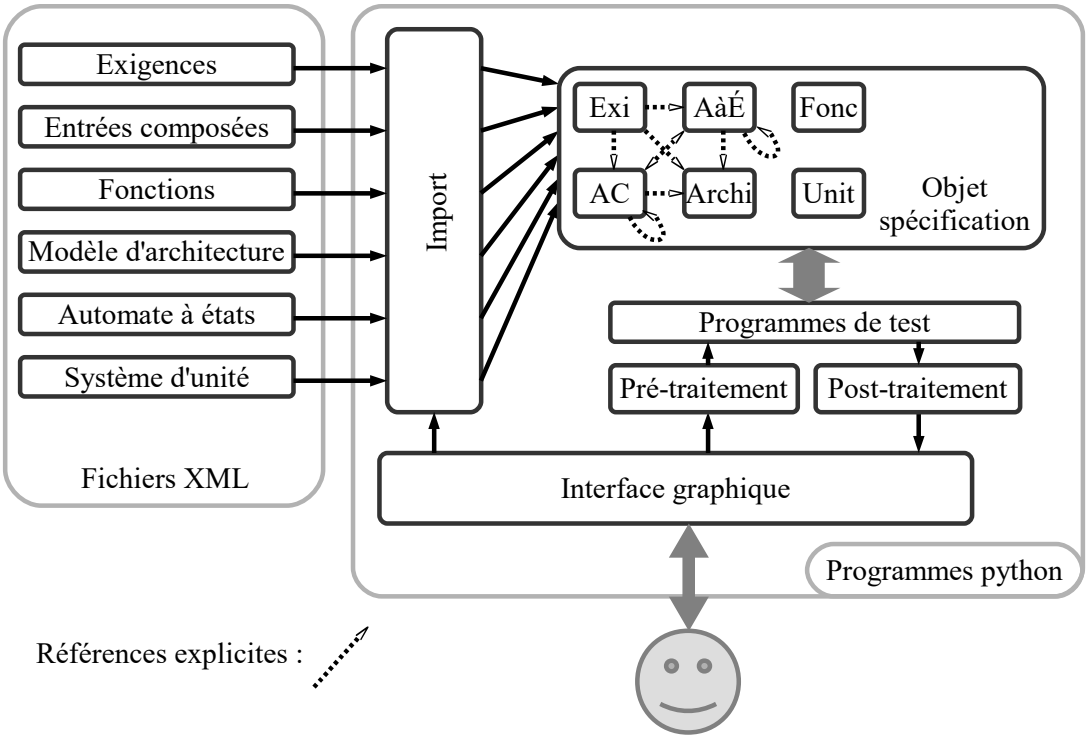


FIGURE 6.4 – Architecture du prototype logiciel

TABLEAU 6.1 – Classes d'équivalences pour l'automate à états de la figure 6.5

Classe d'équivalence	États contenus	Variables référencées
1	{A,B}	{S,T}
2	{A1,A2}	{S,T,U,V}
3	{B1,B2}	{S,T,W,X}
4	{B3,B4}	{S,T,Y,Z}

Une piste d'amélioration de ce prototype concerne l'ajout d'interfaces entre l'utilisateur et les fichiers XML : pour l'instant nous avons rédigé ces fichiers XML directement, avec un éditeur de texte, ce qui n'est évidemment pas optimal. Il est également possible d'améliorer et de rajouter des tests, ainsi que de compléter l'interface graphique.

6.2.2 Interprétation des éléments XML

Les divers fichiers XML sont tout d'abord parsés en utilisant la bibliothèque lxml (http ://lxml.de/), une bibliothèque permettant de manipuler des fichiers XML avec Python. Ensuite, ces fichiers sont généralement traités pour être plus facilement lisibles par les scripts de tests.

Par exemple, comme on l'a mentionné, les fonctions et les types sont importés dans des dictionnaires Python. Pour chaque fonction, un objet "SyntaxFunction" est créé. Cet objet contient le nom, le nombre d'entrées, le type de ces entrées, le type de sortie, etc. de la fonction. Lors de l'importation du fichier de type, pour chaque type, on crée une liste d'"ancêtres" (les types qui contiennent ce type).

Pour l'automate à états, on crée un ensemble de classes d'équivalence. Ces classes d'équivalence permettent de répondre à la question : quand on fait référence à un état de l'automate, quelles sont les variables qui définissent cet état ? En effet, le fait qu'un état soit actif dépend des variables utilisées dans la ou les transitions qui ciblent cet état : si aucune transition menant à l'état n'est tirée, il ne sera pas actif. Mais l'activité de l'état dépend aussi d'autres transitions : si aucune transition menant à un état qui peut mener à l'état considéré n'est tirée, alors l'état considéré ne sera jamais actif. La même chose est vraie pour les états encapsulés : si l'état père d'un état n'est pas actif (et donc que les transitions y menant n'ont pas été tirées), alors cet état ne peut pas être actif. Par contre, les transitions entre les états fils d'un état n'ont pas d'influence sur l'activité de cet état. On définit donc pour chaque état une classe d'équivalence, et pour chaque classe d'équivalence une liste de variables qui ont une influence sur l'activité des états de cette classe.

Par exemple, pour l'automate à états de la figure 6.5, les différentes classes d'équivalences, les états qu'elles contiennent et les variables auxquelles elles font références sont donnés dans le tableau 6.1. On va donc considérer que, quand on fait référence à l'état B1, on fait indirectement référence aux variables S, T, W et X.

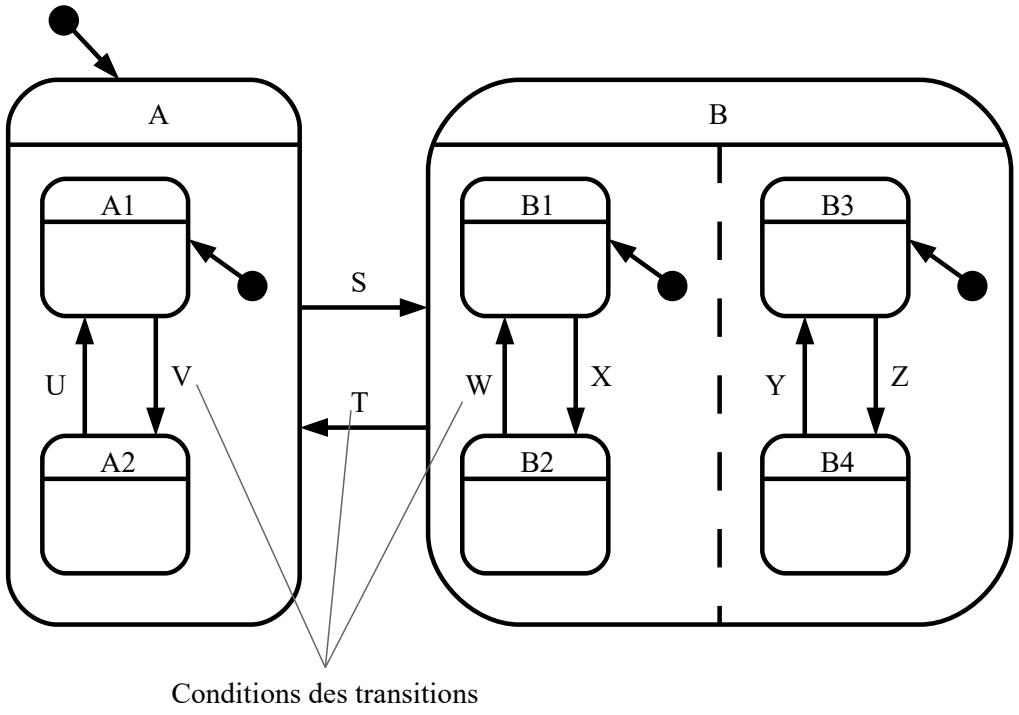


FIGURE 6.5 – Exemple d'automate à états

Pour chaque exigence, chaque abstraction composée, chaque élément de l'automate à états et du modèle d'architecture, on définit une adresse et on ajoute cette adresse dans les attributs de l'élément. Lorsque tous les fichiers nécessaires ont été importés, on peut créer, pour chaque élément référençable, une liste des liens qui ciblent cet élément.

6.2.3 Implémentation des tests

Une fois les importations des divers composants de la spécification terminées, on peut réaliser les opérations (tests et filtrages) qui nous intéressent. Pour explorer la syntaxe des exigences, les fonctions récursives vont être incontournables, vu que l'on ne peut pas savoir a priori quelle sera la structure d'une exigence.

Nous présentons les tests dans cette sous-section et les illustrons dans la sous-section suivante.

6.2.3.1 Trouver une exigence par identifiant

Un filtrage très simple consiste à renvoyer l'exigence correspondant à un identifiant donné, en comparant, pour chaque exigence, son identifiant avec la requête. On renvoie l'exigence sous sa forme XML, mais aussi, parce que la forme XML n'est pas très lisible, sous une forme plus proche de la langue naturelle.

Pour cela, on utilise une fonction récursive “recurWriteFunc” qui parcourt l'arbre syntaxique de l'exigence. Cette fonction récursive donne, pour chaque fonction syntaxique de l'exigence, le texte en langue naturelle correspondant à cette fonction. Par exemple, pour la fonction “Max”, avec deux arguments E1 et E2, le texte correspondant est “la valeur maximum entre (<E1> et <E2>)”, où “<E1>” et “<E2>” sont remplacés par les chaînes de caractères renvoyées par “recurWriteFunc” lorsqu'elle est appliquée aux sous-arbres E1 et E2.

Pour les liens explicites, le texte du lien explicite est directement donné dans l'exigence (à l'intérieur de l'élément étiqueté “a”), la fonction récursive renvoie donc ce texte. De la même manière, pour les éléments “frag”, non formalisés, le texte renvoyé est simplement celui situé à l'intérieur de l'élément.

6.2.3.2 Filtrage par motif

On cherche à trouver toutes les exigences qui se conforment à un motif donné. Il existe une syntaxe, appelée XPath, permettant de sélectionner des éléments dans un arbre XML, mais on ne peut pas directement l'utiliser pour réaliser ce type de filtrage. Nous allons cependant nous inspirer de XPath pour la définition de notre langage de recherche par motif, et aussi l'utiliser directement dans le script de filtrage.

En partant de la racine de l'exigence avec une fonction récursive, on va comparer un à un les éléments rencontrés avec le motif de recherche, en s'arrêtant (et en éliminant l'exigence) :

- Si l'étiquette du motif et de l'arbre ne correspondent pas.
- Si un attribut est spécifié dans le motif, et que cet attribut n'existe pas ou ne correspond pas dans l'élément.
- Si le nombre d'enfants précisé dans le motif et le nombre d'enfants de l'élément ne sont pas les mêmes.

Si l'arbre entier est parcouru sans problème, alors l'exigence correspond au motif et est incluse dans les sorties de la recherche.

Des expressions particulières dans le motif sont utilisées pour permettre une correspondance quel que soit les données (étiquette, attribut, nombre d'enfants) de l'élément. Ces expressions permettent de réaliser les “trous” du motif.

6.2.3.3 Suivi des références explicites

On a deux moyens d'explorer les références explicites.

Exploration des références directes Tout d'abord, on peut, pour chaque référence explicite (c'est à dire pour chaque élément étiqueté “a” des exigences, des abstractions composées et de l'automate à états), chercher l'élément ciblé par cette référence. On peut obtenir des informations plus ou moins détaillées de cette manière, comme le nombre de liens sortants dans l'ensemble des exigences, le nombre de liens dans une exigence particulière, ou le nombre de liens allant de l'automate à états vers le modèle d'architecture.

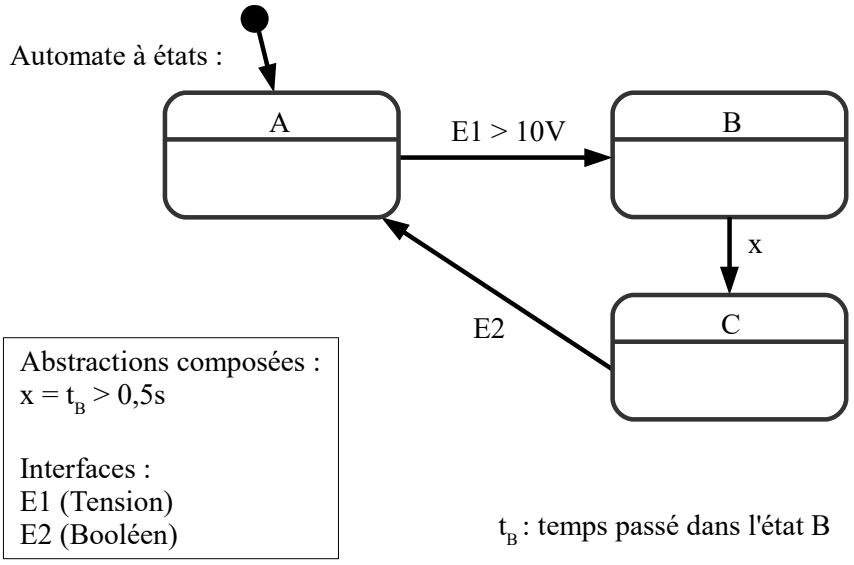


FIGURE 6.6 – Exemple de référence circulaire, mais tout de même cohérente

Exploration en partant d'une exigence On peut aussi, en partant seulement d'une exigence, suivre les liens sortants de cette exigence, puis, pour chaque élément sur lequel on arrive, suivre les liens auxquels cet élément fait référence, et continuer jusqu'à ne plus avoir de liens à explorer. On a noté en section 4.2.2.4 que l'ensemble des liens, des éléments dont ils partent et auxquels ils font référence forment un graphe orienté.

Ce graphe n'est pas nécessairement acyclique : prenons l'exemple donné en figure 6.6. L'automate à états a une seule classe d'équivalence telle que définie dans la section 6.2.2. Les variables référencées par cette classe d'équivalence sont "x", "E1" et "E2". Cela signifie que quand on fait référence à l'état "B", on va faire indirectement référence aux éléments "x", "E1" et "E2". Or l'élément "x" fait lui-même référence à l'état "B".

On a écrit un script pour réaliser cette exploration qui parcours les liens rencontrés tout en gardant en mémoire les éléments déjà rencontrés pour éviter les boucles infinies comme celles que pourrait causer l'exemple précédent. Ce script va permettre de connaître, pour chaque exigence, quels sont les éléments auxquels elle fait référence, directement ou indirectement.

6.2.3.4 Parser de type

Comme on l'a mentionné précédemment, le processus de vérification de type est relativement complexe, puisque l'on va utiliser les informations contenues dans tous les composants de la spécification (exigences, automate, architecture, abstractions composées, fonctions et types). Le principe est, encore une fois, d'utiliser un script récursif parcourant l'exigence à parser. Ce script, si l'élément actuel est une fonction syntaxique A (qui existe dans le fichier de définition des fonctions), va comparer les éléments de l'exigence (nombre d'enfants de

l'élément, leur types) avec ce qui est attendu dans la définition de la fonction. Si tout est correct, le script renvoie alors le type de sortie de la fonction. Si l'élément est une référence explicite, le script va alors suivre le lien pour déterminer le type à renvoyer en sortie. Si l'élément est un fragment de texte libre (i.e. a une étiquette "frag"), alors le script renvoie simplement le type qui est donné en attribut de l'élément.

On peut ainsi vérifier, en testant une exigence à la fois, que les exigences sont syntaxiquement correctes, selon les critères détaillés dans les chapitres précédents.

6.2.3.5 Autres

Vérification des références Afin de vérifier que la cible d'une référence explicite existe bien, on réalise une recherche XPath dans le fichier correspondant, si aucun élément n'est trouvé, c'est que la cible n'existe pas. La syntaxe de XPath et les adresses définies dans les références ne correspondent pas exactement, et on a donc écrit un script permettant de passer de l'une à l'autre.

Homogénéité des termes La vérification du texte utilisé dans les références explicites a déjà été mentionnée plus haut (section 4.3.1.1). On a choisi, plutôt que de comparer avec un texte donné par l'élément référencé, de créer une liste de termes pour chaque adresse distincte : on parcours les exigences, et pour chaque lien rencontré, on regarde si le texte de ce lien est déjà dans la liste de termes associée à l'adresse cible du lien, si ce n'est pas le cas, on rajoute le texte à la liste de termes.

On obtient en sortie un dictionnaire de "synonymes" pour toutes les adresses distinctes trouvées dans les exigences.

Profondeur d'une exigence On utilise une fonction récursive assez simple qui renvoie, pour un élément donné, un entier correspondant à la distance maximale entre cet élément et les feuilles descendant de cet élément. Cette fonction, appliquée à la racine d'une exigence, donne la profondeur de son arbre syntaxique.

Référence à toutes les interfaces Une fois que l'on a déterminé les liens entrants pour tous les éléments du modèle d'architecture, il est évident de déterminer, pour chaque interface du système, s'il existe au moins une référence à cette interface.

6.2.4 Remarques et limitations

On peut faire quelques remarques sur les implémentations de tests réalisées.

Tout d'abord, il est probablement possible d'optimiser les différents scripts de manière à les rendre plus rapides. Cependant, pour les tests réalisés sur la spécification EGTS, ces scripts ont tous des durées insignifiantes pour un humain (i.e. inférieures à 0.5s), et, a priori, c'est aussi le cas pour des spécifications de taille "raisonnable" (environ moins d'un millier d'exigences).

Traduction automatique en langue naturelle Le script permettant d'obtenir une exigence en langue naturelle à partir de l'exigence XML est un début, mais est loin d'être parfait. Notamment, pour l'instant, une fonction syntaxique ne peut avoir qu'une seule traduction en langue naturelle, ce qui conviendra bien pour certaines exigences, mais donnera des exigences peu claires pour d'autres.

Le problème inverse, la traduction de la langue naturelle générique vers l'exigence formalisée en XML est beaucoup plus vaste, mais une traduction restreinte, où le texte en langue naturelle de départ est déjà très contraint, pourrait être utile pour la rédaction ou la correction d'exigences.

Filtrage par motif La syntaxe utilisée pour l'instant pour le filtrage par motif est assez complexe, et il pourrait être utile de la simplifier, ou de prévoir une interface permettant de faire des recherches plus facilement.

Autres tests D'autres tests sont évidemment possibles, tout comme de raffiner ceux que l'on a présentés ici. Par exemple, il est envisageable de pouvoir détecter automatiquement certaines exigences contradictoires en utilisant la syntaxe présentée dans ce travail.

6.3 Démonstration

Nous présentons un exemple jouet permettant d'illustrer divers concepts et scripts introduits plus haut. Nous commençons avec seulement un modèle d'architecture et des exigences, et nous introduisons les autres concepts au fur et à mesure. Nous donnons des représentations graphiques des divers éléments (exigences, modèle d'architecture, automate à états...), ainsi que leur formulation en XML, et les résultats obtenus dans le prototype logiciel.

Nous présentons tout d'abord l'interface graphique du prototype (cf figure 6.7) : le menu "Import" permet de choisir les fichiers d'exigences, d'automate à états, d'architecture, etc. à importer. Les six labels colorés en haut de la fenêtre indiquent si les fichiers correspondants ont bien été importés, ici ce n'est pas encore le cas. Les boutons et champs sur la droite permettent à l'utilisateur de réaliser des tests sur la spécification. Le grand champ blanc est la fenêtre où le programme donne des informations à l'utilisateur, tels que les résultats des tests/requêtes/etc.

6.3.1 1er exemple basique

Présentation On considère un système très simple, présenté en figure 6.8, qui interagit avec trois autres systèmes :

— une alimentation électrique,
— un système de commande,
— un axe pouvant être mis en rotation.

On souhaite que le système mette l'axe en rotation lorsqu'il est alimenté et qu'on lui commande de le faire.

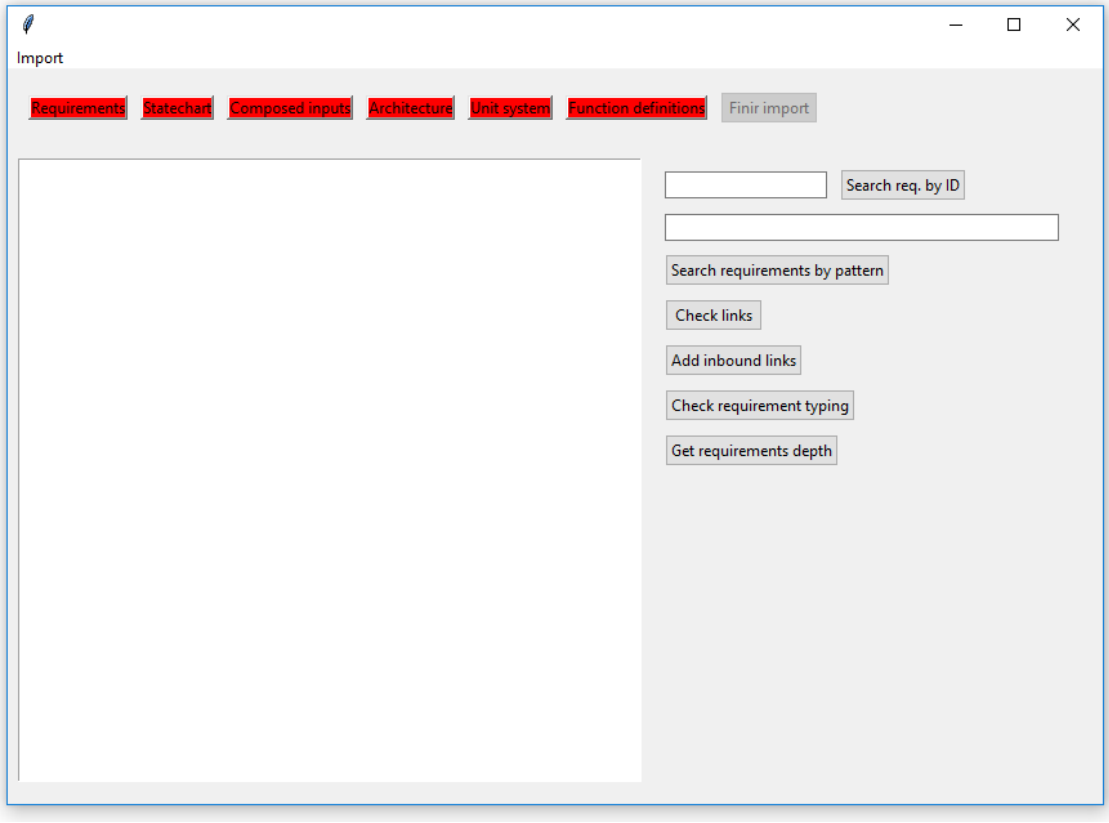
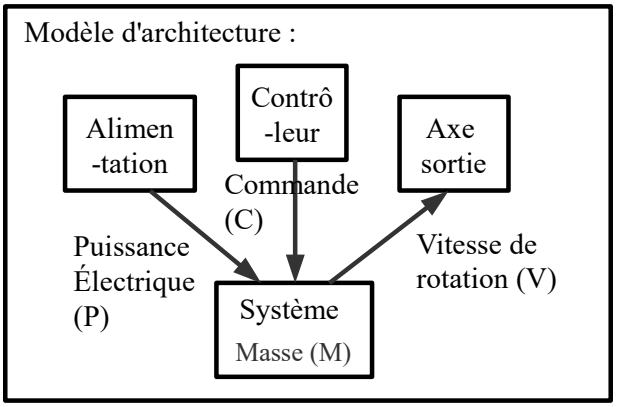


FIGURE 6.7 – Interface graphique du prototype logiciel



Exigences :
001 Quand P > 5W et C = vrai, alors le système doit appliquer V > 10 rad/s

002 Quand P < 5W ou C = faux, alors le système doit appliquer V < 0,1 rad/s

003 La masse du système doit être inférieure à 10kg

FIGURE 6.8 – Environnement et exigences du système jouet

```
<Req Flag="0" id="001">
  <Syst><a href="Archi/SurSysteme/Systeme">le systeme</a></Syst>
  <Demand>doit</Demand>
  <Prop>
    <func funcdef="Implication">
      <func funcdef="Et">
        <func funcdef="pGq">
          <a href="Archi/SurSysteme/Systeme/Port_reserv::Puissance electrique">la
          puissance d'alimentation</a>
          <func funcdef="Unit_Puissance_W">
            <func funcdef="NewReal">5</func>
          </func>
        </func>
        <func funcdef="Egal">
          <a href="Archi/SurSysteme/Systeme/Port_reserv::Commande">la commande</a>
          <func funcdef="NewBool">Vrai</func>
        </func>
      </func>
      <func funcdef="pGq">
        <a href="Archi/SurSysteme/Systeme/Port_reserv::Vitesse de rotation">la vitesse
        de rotation</a>
        <func funcdef="Unit_VitRot_radpars">
          <func funcdef="NewReal">10</func>
        </func>
      </func>
    </func>
  </Prop>
</Req>
```

FIGURE 6.9 – Exigence 001 du système jouet en format XML

La formulation de l'exigence 001 en XML est donnée en figure 6.9. L'arbre syntaxique correspondant à la propriété de l'exigence 001 est donné en figure 6.10. La formulation du modèle d'architecture est donnée en figure 6.11. Sans grande surprise, les descriptions XML sont assez longues et peu claires.

Tests sur cet exemple Lorsque l'on demande de générer une version en "langue naturelle" de l'exigence 001, on obtient le résultat suivant (cf figure 6.12) : "`Quand (<la puissance d'alimentation> plus grand que 5W et <la commande> egal a Vrai), <le systeme> doit appliquer <la vitesse de rotation> plus grand que 10rad/s`".

Clairement, cette exigence n'est pas écrite en français correct, mais elle reste compréhensible. Cette génération automatique de texte peut être améliorée.

On peut également sélectionner les exigences selon leur structure (cf figure 6.13) : la requête "`["Req", ["*", "*", ["Prop", [["func[@funcdef=Implication]", ["func[@funcdef=Et]", "*"]]]]]]`" renvoie toutes les exigences qui correspondent au motif donné en figure 6.14, ici l'exigence 001.

6.3.2 Fonctions et types

Présentation Toutes les fonctions utilisées dans les exigences doivent apparaître dans le fichier de définition des fonctions. De même, tous les types mentionnés dans les différents fichiers doivent être définis dans le fichier de définition des types.

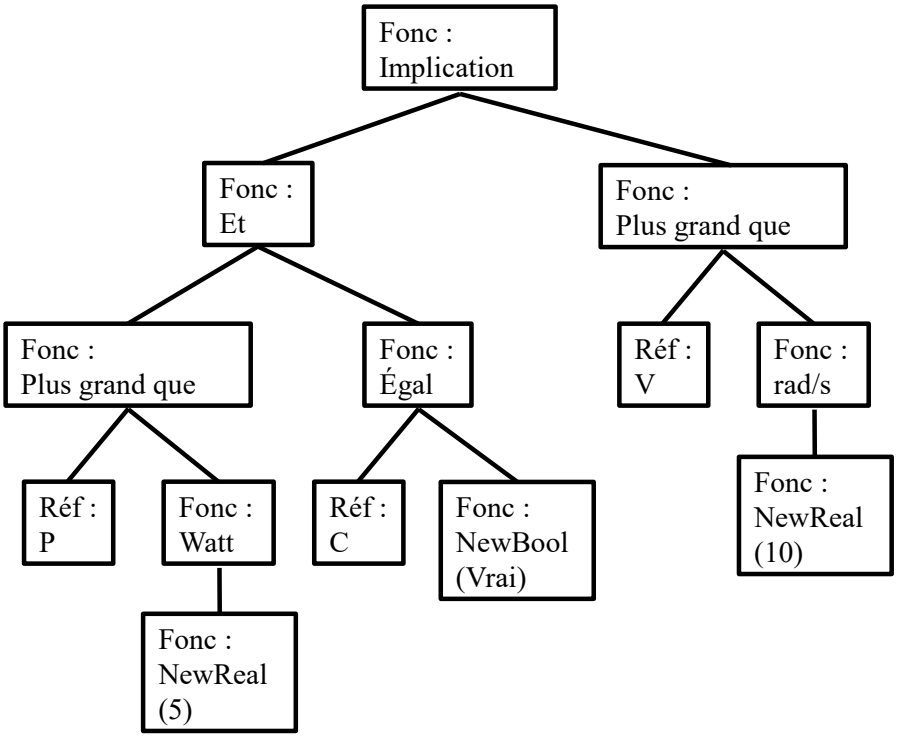


FIGURE 6.10 – Arbre syntaxique de la propriété de l'exigence 001

```
<ModelArchitecture>
  <Block Flag="0" Name="SurSysteme">

    <Block Flag="0" Name="Systeme">
      <Attributes>
        <Var Flag="0" Type="Masse">Poids du systeme</Var>
        <Var Flag="0" Type="Temperature">Temperature de surface</Var>
      </Attributes>
      <Port Flag="0" Name="Puissance electrique" E_ou_S="entree" Type="Puissance" ></Port>
      <Port Flag="0" Name="Commande" E_ou_S="entree" Type="Bool" ></Port>
      <Port Flag="0" Name="Vitesse de rotation" E_ou_S="sortie" Type="radpars" ></Port>
    </Block>

    <Block Flag="0" Name="Alimentation">
      <Port Flag="0" Name="Puissance electrique" E_ou_S="sortie" Type="Puissance" ></Port>
    </Block>

    <Block Flag="0" Name="Controleur">
      <Port Flag="0" Name="Commande" E_ou_S="sortie" Type="Bool" ></Port>
    </Block>

    <Block Flag="0" Name="Axe de sortie">
      <Port Flag="0" Name="Vitesse de rotation" E_ou_S="entree" Type="radpars" ></Port>
    </Block>

    <Connection Name="Alim_connection">
      <Link>Systeme/Port_reserv::Puissance electrique</Link>
      <Link>Alimentation/Port_reserv::Puissance electrique</Link>
    </Connection>

    <Connection Name="Comm_connection">
      <Link>Systeme/Port_reserv::Commande</Link>
      <Link>Controleur/Port_reserv::Commande</Link>
    </Connection>

    <Connection Name="Vitesse de rot_connection">
      <Link>Systeme/Port_reserv::Vitesse de rotation</Link>
      <Link>Axe de sortie/Port_reserv::Vitesse de rotation</Link>
    </Connection>

  </Block>
</ModelArchitecture>
```

FIGURE 6.11 – Modèle d'architecture de l'environnement en format XML

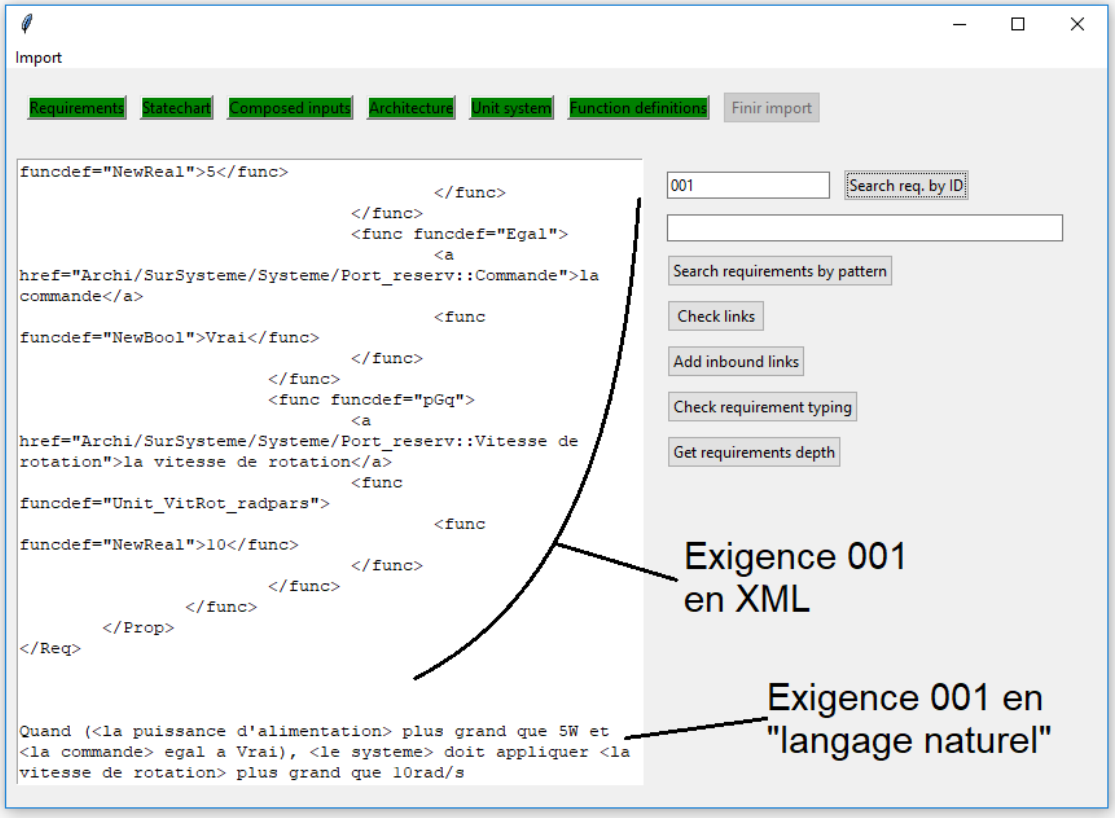


FIGURE 6.12 – Lors d'une recherche par ID, l'exigence est renvoyée sous forme XML et en texte généré automatiquement

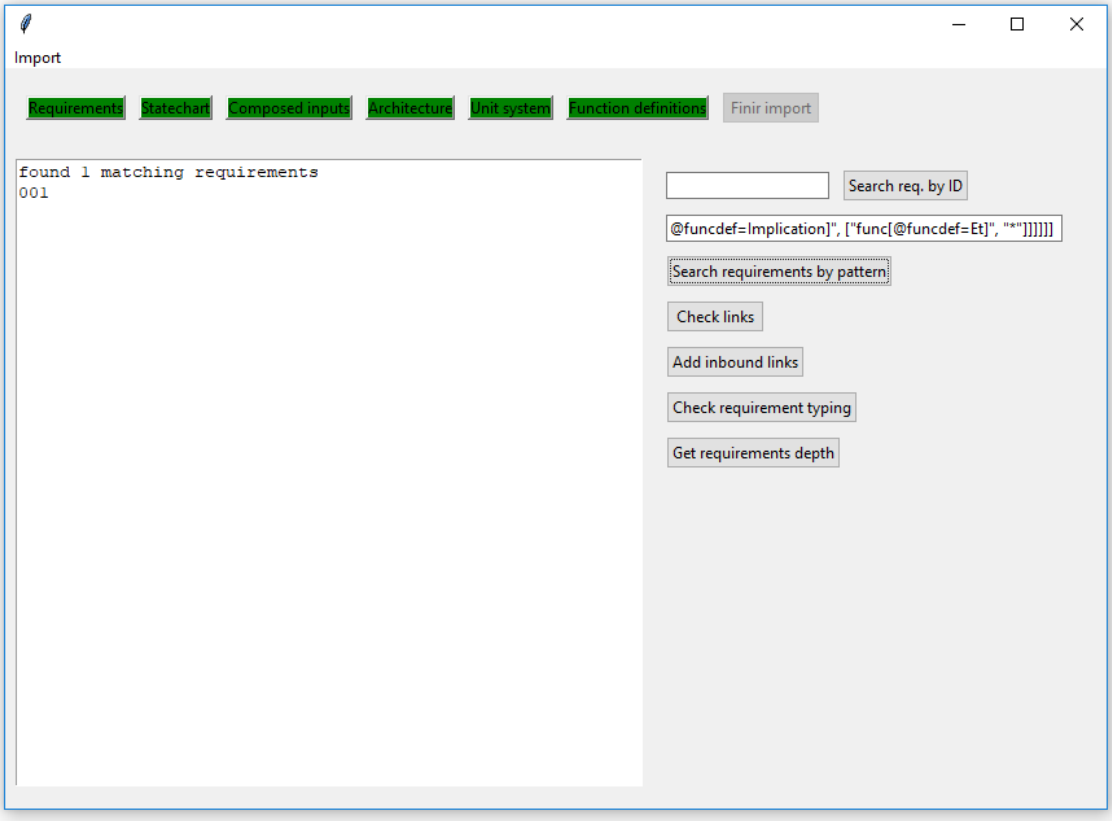


FIGURE 6.13 – On renvoie l'identifiant des exigences qui correspondent au motif de recherche

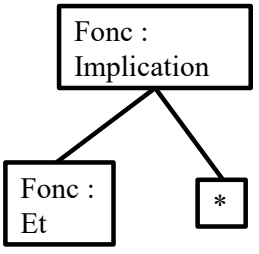


FIGURE 6.14 – Exemple de motif de recherche, "*" signifie que n'importe quel sous-arbre est accepté

```
20  <Function Name="Ou" Functype='MultiInputs' NL='(+$Input$1+ ou +$Input$2+%Repeat%( ou
    *$Input$i)+)'>
21    <Inputs>Bool</Inputs>
22    <Output>Bool</Output>
23  </Function>
24
25  <Function Name="Egal" Functype='Normal' NL='$Input$1+ egal a +$Input$2'>
26    <Input EqClass="X">Any</Input>
27    <Input EqClass="X">Any</Input>
28    <Output>Bool</Output>
29  </Function>
30
31  <Function Name="pGq" Functype='Normal' NL='$Input$1+ plus grand que +$Input$2'>
32    <Input EqClass="X">Measure</Input>
33    <Input EqClass="X">Measure</Input>
34    <Output>Bool</Output>
35  </Function>
36
37  <Function Name="pPq" Functype='Normal' NL='$Input$1+ plus petit que +$Input$2'>
38    <Input EqClass="X">Measure</Input>
39    <Input EqClass="X">Measure</Input>
40    <Output>Bool</Output>
41  </Function>
42
43  <Function Name="TON" Functype='Normal' NL='$Input$1+ est vrai pendant plus de +$Input$2'>
44    <Input>Bool</Input>
45    <Input>Duree</Input>
46    <Output>Bool</Output>
47  </Function>
48
49  <Function Name="Unit_Puissance_W" Functype='Normal' NL='$Input$1+W'>
50    <Input>Real</Input>
51    <Output>W</Output>
52  </Function>
```

FIGURE 6.15 – Extrait du fichier XML de définition des fonctions

Ici, on a des fonctions booléennes (et, ou, implication), des fonctions de comparaison (égalité, plus grand et plus petit que), ainsi que des fonctions permettant de définir des mesures (i.e. un nombre associé à une unité). On va retrouver les types associés à ces mesures (kg, A, V, etc.) dans le fichier de définition des types.

Une partie du fichier de définition des fonctions est donné en figure 6.15. Le fichier de définition des types est donné en figure 6.16.

Tests Une fois ces types et ces fonctions définis, on peut vérifier que nos exigences sont correctes syntaxiquement. Pour les exigences données, il n'y a pas de problèmes, mais si on introduit des erreurs, elles vont être détectées. Par exemple, si l'on change dans l'exigence 002 le nom de la fonction "pPq" (plus petit que) en "pP", une fonction qui n'existe pas, le parser de types ne va pas trouver cette fonction et le signaler comme indiqué en figure 6.17.

De même, si l'on essaye de remplacer "Unit_Puissance_W" (la fonction qui prend un réel en argument et renvoie une mesure en Watt) par une autre fonction telle que "Implication", le parser détecte que "Implication" attend deux arguments mais n'en a reçu qu'un seul, voir figure 6.18.

On peut aussi remplacer "Unit_Puissance_W" par "Unit_VitRot_radpars". Cette fois,

```
<UnitSystem>
  <Any>
    <BlockSys></BlockSys>
    <State></State>
    <Transition></Transition>
    <Bool></Bool>
    <Measure>

      <Real>
        <Integer></Integer>
      </Real>

      <Tension>
        <V></V>
      </Tension>
      <Intensite>
        <A></A>
      </Intensite>
      <Puissance>
        <W></W>
      </Puissance>

      <Masse>
        <kg></kg>
      </Masse>

      <VitRot>
        <radpars></radpars>
      </VitRot>

      <Duree>
        <s></s>
      </Duree>

    </Measure>
  </Any>
</UnitSystem>
```

FIGURE 6.16 – Fichier XML de définition des types

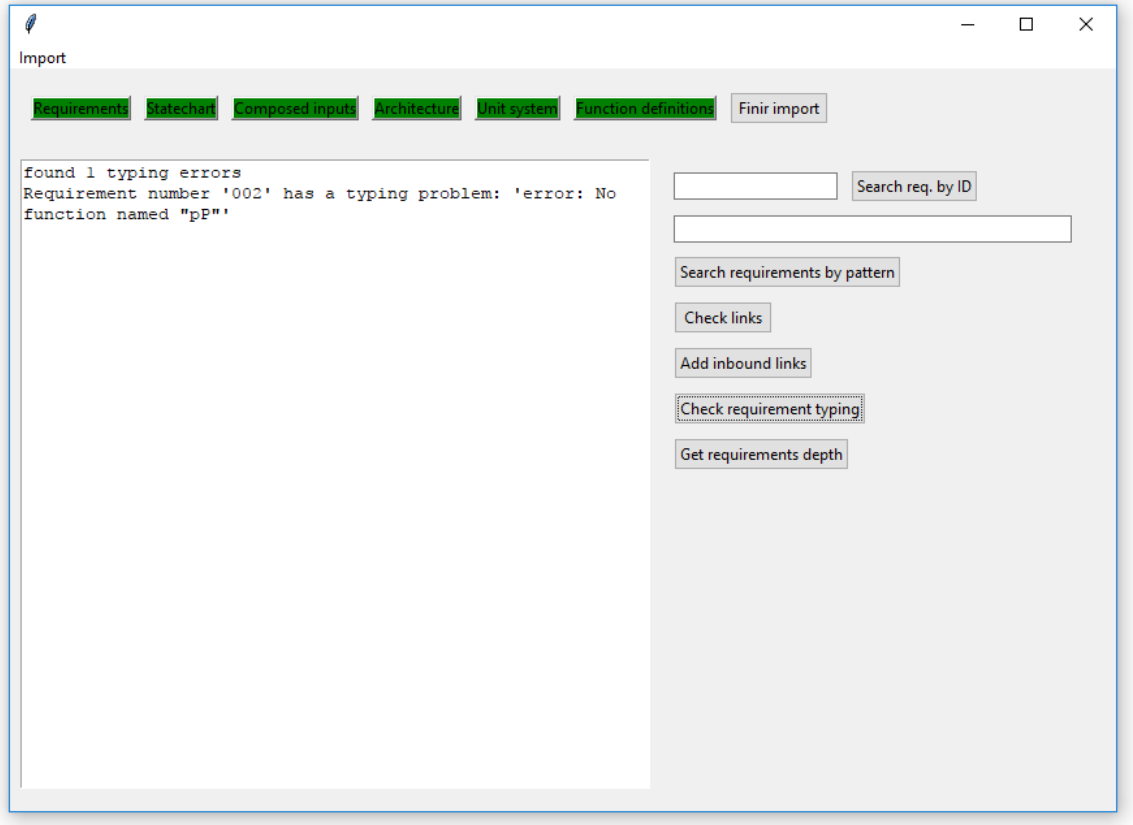


FIGURE 6.17 – Vérification du typage des exigences

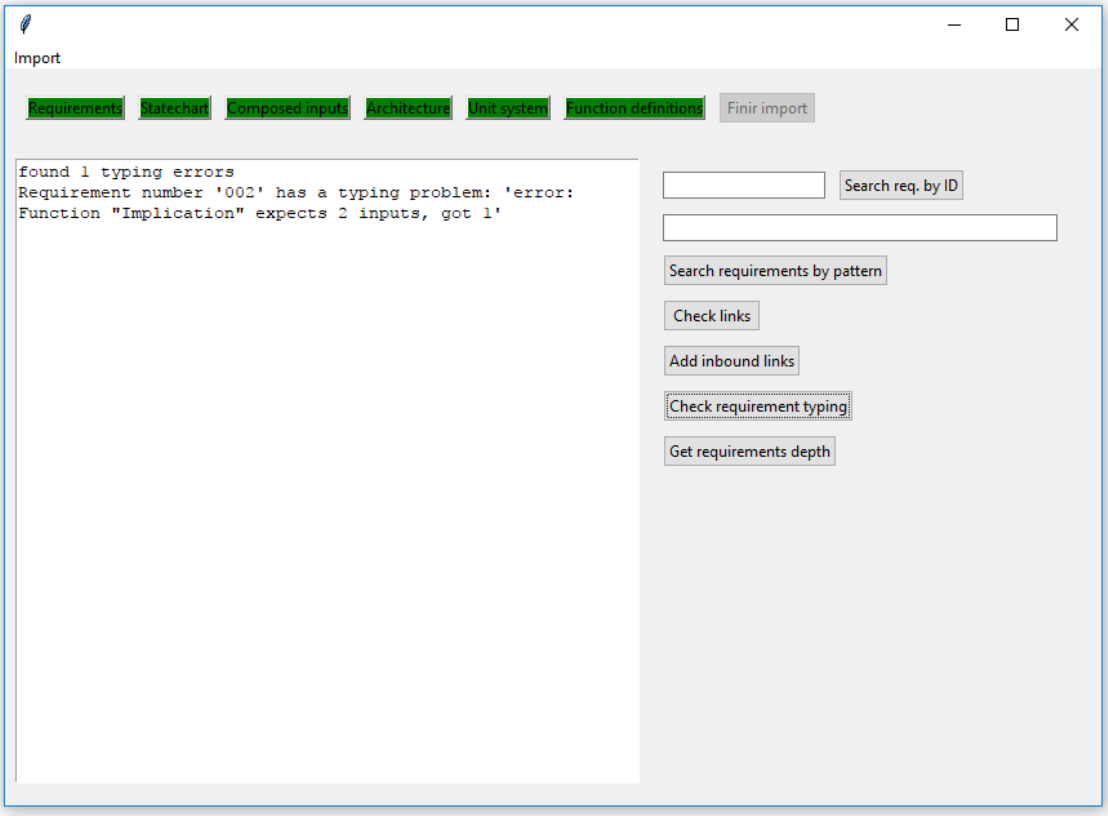


FIGURE 6.18 – Vérification du typage lorsque l'on introduit une autre erreur

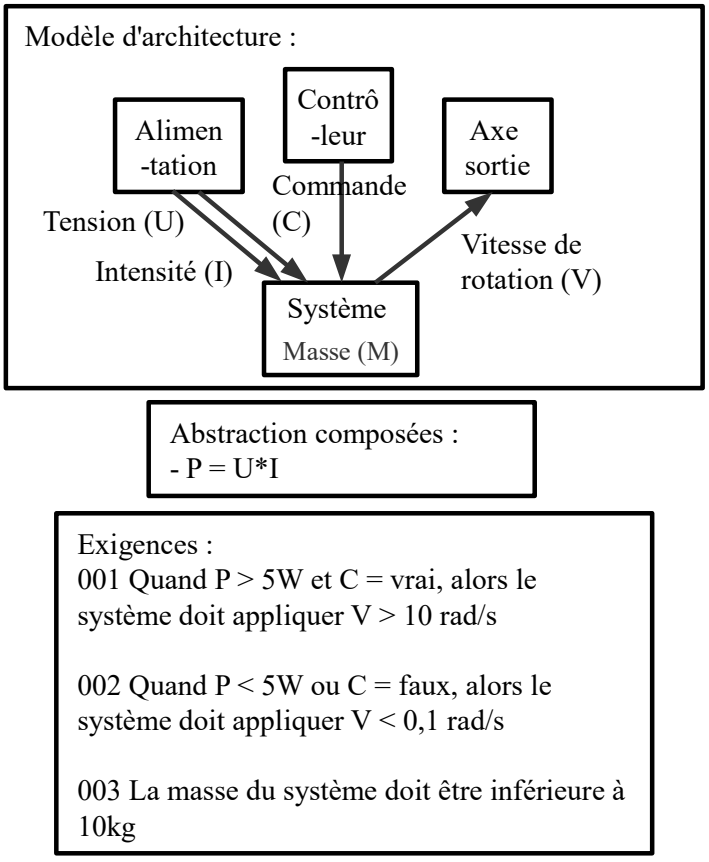


FIGURE 6.19 – Mise à jour du modèle d'architecture, on introduit une "abstraction composée" pour que les exigences ne changent pas

on a un problème parce que l'on va essayer de comparer, en utilisant la fonction "pPq", une mesure exprimée en radians par seconde avec un élément du modèle d'architecture ("Puissance électrique") qui est une puissance. Le parser renvoie le message "`found 1 typing errors Requirement number '002' has a typing problem: 'error: Units "Puissance" and "radpars" are not compatible'`.

6.3.3 Introduction de nouveaux éléments

Abstraction composées Supposons que l'on souhaite mettre à jour le modèle d'architecture et utiliser, à la place de l'interface "Puissance électrique", deux interfaces "Tension" et "Intensité". Si l'on ne souhaite pas modifier la formulation des exigences, on "recrée" la variable "Puissance électrique" dans le fichier d'abstractions composées, et on modifie la cible des références à cette variable vers la nouvelle définition. Cette opération est illustrée dans la figure 6.19.

La nouvelle définition de "Puissance électrique", des éléments modifiés du modèle

```
<ComposedInputs>

<CI Name='Puissance electrique' Flag="0" Type='Puissance'>
  <frag type="Puissance">
    <a href="Archi/SurSysteme/Systeme/Port_reserv::Intensite">le courant d'alimentation</a>
    multiplie par
    <a href="Archi/SurSysteme/Systeme/Port_reserv::Tension">la tension d'alimentation</a>
  </frag>
</CI>

</ComposedInputs>
```

FIGURE 6.20 – Nouvelle définition de “Puissance électrique” dans le fichier d’abstractions composées

```
<Block Flag="0" Name="Systeme">
  <Attributes>
  <Port Flag="0" Name="Puissance electrique" E_ou_S="entree" Type="Puissance" ></Port>
  <Port Flag="0" Name="Commande" E ou S="entree" Type="Bool" ></Port>
```

```
<Block Flag="0" Name="Systeme">
  <Attributes>
  <Port Flag="0" Name="Tension" E_ou_S="entree" Type="Tension" ></Port>
  <Port Flag="0" Name="Intensite" E_ou_S="entree" Type="Intensite" ></Port>
  <Port Flag="0" Name="Commande" E ou S="entree" Type="Bool" ></Port>
```

FIGURE 6.21 – Remplacement de l’interface “Puissance électrique” dans le modèle d’architecture par deux interfaces “Tension” et “Intensité”

```
<func funcdef="Et">
  <func funcdef="pGq">
    <a href="Archi/SurSysteme/Systeme/Port_reserv::Puissance electrique">la
    puissance d'alimentation</a>
    <func funcdef="Unit_Puissance_W">
```

```
<func funcdef="Et">
  <func funcdef="pGq">
    <a href="CI/Puissance electrique">la puissance d'alimentation</a>
    <func funcdef="Unit_Puissance_W">
```

FIGURE 6.22 – Modifications des adresses dans les exigences

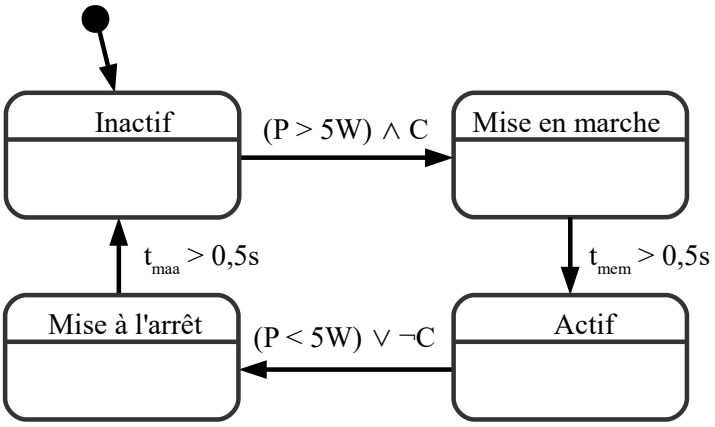


FIGURE 6.23 – Automate à états permettant de définir des délais entre l'application des exigences

d'architecture, ainsi que les adresses modifiées dans les exigences sont présentés dans les figures 6.20, 6.21 et 6.22.

On note que la définition de "Puissance électrique" en figure 6.20 n'est pas parfaitement formelle : en effet, comme mentionné précédemment, le système de type est assez basique et il n'est pour l'instant pas possible de déduire qu'une intensité multipliée par une tension donnent une puissance. Nous avons donc donné directement le type de "Puissance électrique" et laissé sa définition non formelle, tout en précisant explicitement à partir de quels éléments on construit cette nouvelle variable.

Automate à états Supposons que l'on souhaite un délai entre l'application des exigences 001 et 002, parce que, telles que les exigences sont écrites, elles vont forcément être violées, par exemple lorsque la commande passe de vrai à faux ou l'inverse. En effet, si l'on considère que l'axe de sortie est un élément physique, il est impossible que la vitesse de rotation passe de plus de 10 rad/s à moins de 0.1 rad/s instantanément.

On considère qu'un délai de 0,5 seconde entre l'application de ces deux exigences est suffisant. On introduit donc un automate à états, présenté en figure 6.23, dotés de quatre états : les états "Actif" et "Inactif" sont utilisés dans les exigences et vont correspondre presque complètement aux conditions "P $>$ 5W et C $=$ vrai" et "P $<$ 5W ou C $=$ faux" respectivement. La différence est introduite par les états "Mise en marche" et "Mise à l'arrêt", qui sont simplement des états d'attente permettant de poser un délai entre les applications des exigences 001 et 002. "t_{mem}" et "t_{maa}" représentent le temps écoulé depuis l'activation de l'état "Mise en marche" et "Mise à l'arrêt" respectivement. Les exigences modifiées sont les suivantes :

Exi. 1 *Quand l'état « Actif » est vrai, alors le système doit appliquer V $>$ 10 rad/s.*

Exi. 2 *Quand l'état « Inactif » est vrai, alors le système doit appliquer V $<$ 0,1 rad/s.*

Une partie de la description XML de cet automate à états est donnée en figure 6.24. La fonction “TON” représente un “Timer ON Delay” : une fonction à sortie booléenne S, avec deux arguments, un booléen E et une constante temporelle t, S est vraie si E est vraie depuis plus de t secondes.

Tests On peut réaliser des tests sur les références explicites des divers éléments de la spécification. Tout d’abord que les cibles de ces références existent bien et qu’elles sont bien des éléments du modèle d’architecture, de l’automate à états ou des abstractions composées. C’est bien le cas ici, et l’on peut avoir un récapitulatif des différents liens entrants et sortants (voir figure 6.25). Si l’on donne des adresses incorrectes, ces liens invalides seront détectés.

```
<ModelStatechart>

  <State Flag="0" Name="Inactif" InitialState="True">
  </State>

  <State Flag="0" Name="Mise en marche">
  </State>

  <State Flag="0" Name="Actif">
  </State>

  <State Flag="0" Name="Mise a l arret">
  </State>

  <Transition Flag="0">
    <From>Inactif</From>
    <To>Mise en marche</To>
    <Condition>
      <func funcdef="Et">
        <func funcdef="pGq">
          <a href="CI/Puissance electrique">la puissance d'alimentation</a>
          <func funcdef="Unit_Puissance_W">
            <func funcdef="NewReal">5</func>
          </func>
        </func>
        <func funcdef="Egal">
          <a href="Archi/SurSysteme/Systeme/Port_reserv::Commande">la commande</a>
          <func funcdef="NewBool">Vrai</func>
        </func>
      </func>
    </Condition>
  </Transition>

  <Transition Flag="0">
    <From>Mise en marche</From>
    <To>Actif</To>
    <Condition>
      <func funcdef="TON">
        <func funcdef="Dans etat">
          <a href="SM/Inactif">Mise en marche</a>
        </func>
        <func funcdef="Unit_Duree_s">
          <func funcdef="NewReal">0.5</func>
        </func>
      </func>
    </Condition>
  </Transition>
```

FIGURE 6.24 – Description en format XML des quatre états et de deux des quatre transitions

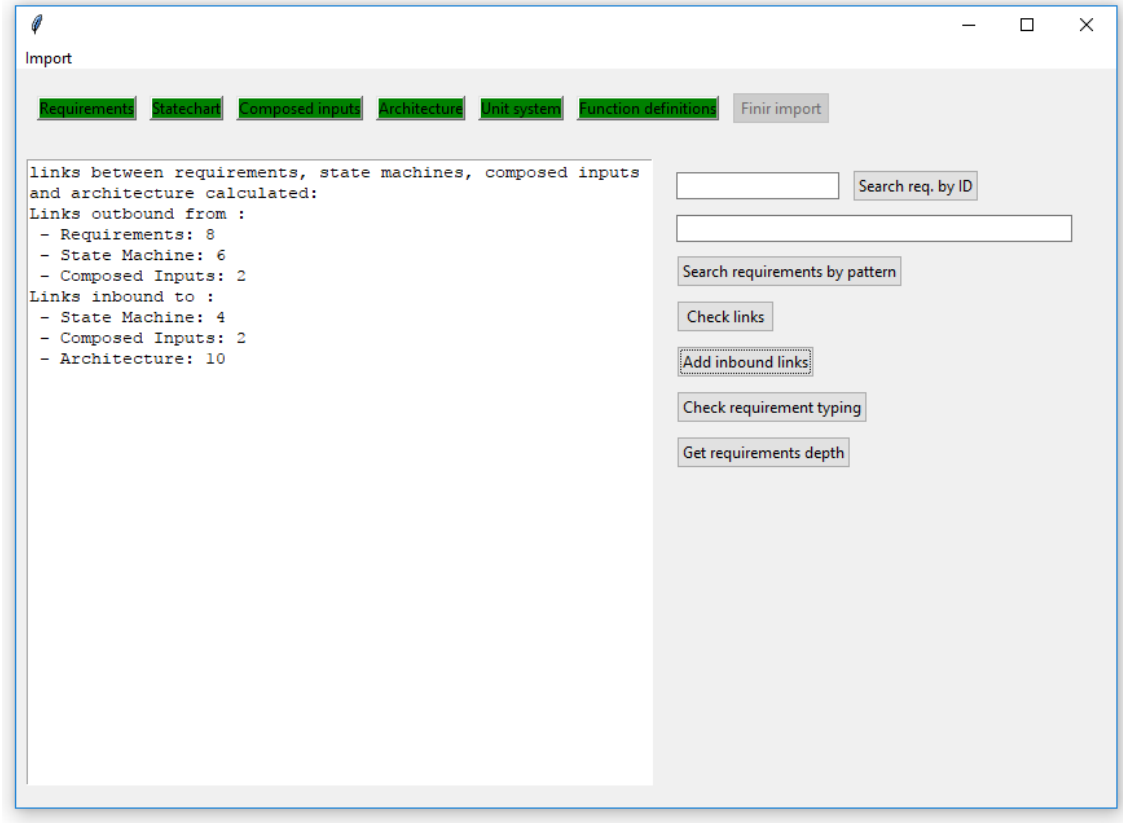


FIGURE 6.25 – On a vérifié la validité des références explicites, et on obtient un récapitulatif de ces liens

Chapitre 7

Conclusion et perspectives

7.1 Bilan

7.1.1 Résumé

Dans ce travail, nous avons présenté des idées pour rendre le texte des exigences moins ambigu, tout en évitant de rajouter une grosse charge de travail pour les rédacteurs et les lecteurs des spécifications. En plus de l'examen de la littérature scientifique pertinente, nous avons basé notre travail sur l'étude de spécifications industrielles réelles et sur des discussions avec des ingénieurs des exigences dans l'industrie. Nous pensons, et certains travaux (Ormandjieva et collab. (2007)) pointent dans la même direction, qu'il est possible d'améliorer le processus complet d'ingénierie des exigences en définissant plus précisément le vocabulaire et la syntaxe du texte des exigences. Cette information supplémentaire peut ensuite être utilisée pour réaliser des tests rapides et automatiques sur les exigences.

Le cœur de notre travail consiste tout d'abord à conceptualiser le rôle des modèles dans le contexte des spécifications systèmes : essentiellement, les modèles sont des définitions ou des informations additionnelles sur les interfaces entre le système spécifié et son environnement. Deuxièmement, nous suggérons de connecter explicitement ces modèles avec le texte des exigences. Nous proposons aussi un moyen de formaliser la syntaxe des exigences en les décomposant en fragments plus petits et en organisant ces fragments en un arbre syntaxique typé. Finalement, nous présentons comment utiliser cette information en pratique en réalisant des tests sur les exigences.

Nous avons expérimenté nos idées en "traduisant" une spécification industrielle existante et en écrivant une spécification (partielle) pour un système exemple.

7.1.2 Commentaires

De manière générale, étudier l'ingénierie des exigences est un travail assez complexe. Premièrement, c'est un travail qui se situe à la limite entre formel et non formel : on veut pouvoir dire si oui ou non un système respecte une spécification, mais le système existe dans le réel, qui est généralement trop difficile à formaliser entièrement, et la spécification

est écrite à partir de sources non formelles, comme les besoins des parties prenantes.

Il n'est pas réaliste d'ignorer, ni la partie informelle, par exemple en demandant que les exigences soient écrites en logique formelle, ni la partie formelle, en laissant les exigences en texte libre. Il faut donc arriver à concilier ces deux parties, en développant des méthodes qui s'appuient aussi bien sur des éléments formels que non-formels, et qui permettent d'articuler ces deux types d'éléments, potentiellement très différents.

Une autre difficulté est qu'il est difficile de faire des expériences et de valider les concepts proposés. Comme d'autres domaines touchant aux processus de conception dans les entreprises, certains éléments de la méthode scientifique classique sont difficilement applicables. Il est possible de faire des prédictions sur des concepts réduits (par exemple, que telle exigence a plus de chance d'être bien comprise que telle autre), mais faire des prédictions sur l'ensemble de la méthode et ensuite tester ces prédictions en conditions réelles est pratiquement impossible.

Nous pensons cependant qu'un regard scientifique sur les problèmes rencontrés en ingénierie des exigences (et en ingénierie système en général) dans les entreprises est utile et désirable. J'ai trouvé très intéressant d'avoir à prendre en compte les contraintes pratiques de l'environnement d'entreprise dans le développement de nouvelles méthodes d'ingénierie des exigences. Il serait assez facile de dire "Les exigences sont ambiguës ? il n'y a qu'à les écrire en logique du premier ordre/Coq/...", mais cela ne répondrait pas au problème.

7.2 Perspectives

De nombreuses questions relatives à ce travail restent ouvertes. Ces questions peuvent être plutôt conceptuelles (un élément de modèle peut-il et doit-il remplacer une exigence textuelle ?), comme très pratiques (quel éditeur utiliser pour les exigences et les modèles ?). Nous mentionnons ici les questions qui nous semblent les plus importantes.

7.2.1 La frontière floue entre exigences et modèles

Comme on l'a mentionné en section 3.2.3.4, la frontière entre exigences et éléments de modèle n'est pas absolue. Si l'on peut écrire une exigence en utilisant un élément de modèle et vice-versa, il est nécessaire de s'intéresser aux avantages et aux inconvénients de l'une ou l'autre formulation.

7.2.1.1 Quelles exigences sont concernées ?

C'est l'un des points sur lequel nous avons insisté : nous pensons qu'il n'est pas pratiquement faisable d'écrire des spécifications complètement formelles. Un certain nombre d'exigences ne sont simplement pas formalisables ou en tout cas sont plus simplement écrites en langue naturelle. Pour ces exigences, chercher à les exprimer sous forme de modèle ne semble pas pertinent.

Par contre, d'autres exigences ont déjà une forme qui peut être formalisée en un élément de modèle particulier. Par exemple, le tableau de la section 3.2.3.4, dont chaque ligne peut

être remplacé par une exigence. On peut aussi définir de nouvelles abstractions composées et/ou de nouveaux états afin de diminuer le nombre d'exigences ou la complexité des exigences.

Par exemple, prenons des exigences similaires à celles de la section 6.3 : "Quand P > 5W et C est vrai depuis plus de 0.5s, alors le système doit appliquer V = 10rad/s" et "Quand P < 5W ou C est faux depuis plus de 0.5s, alors le système doit appliquer V = 0.1rad/s". Si l'on définit les mêmes états que dans la section 6.3, et que l'on définit de plus une abstraction V_comm, égale à 10rad/s si l'état "Actif" est vrai et 0.1rad/s si l'état "Inactif" est vrai, alors on peut écrire une seule exigence : "Quand l'état "Actif" ou l'état "Inactif" est vrai, alors V doit être égal à V_comm".

7.2.1.2 Avantages et inconvénients

Remplacer des exigences de cette manière diminue le nombre d'exigences, ce qui est désirable pour les industriels, puisqu'il y a alors moins d'exigences à valider, vérifier, tracer, etc. Cependant, il est possible que la gestion des exigences devienne plus complexe.

Par exemple, il est très souvent nécessaire de réaliser une traçabilité des exigences, c'est à dire de donner explicitement les liens entre une exigence d'un système et l'exigence du sur-système qui en est à l'origine. (Ainsi que les liens entre une exigence du système et les exigences des sous-systèmes qui ont cette exigence comme origine). Si nous utilisons des éléments de modèles (tableaux, attributs dans un modèle d'architecture, états, etc.) pour remplacer les exigences, cela nécessite d'inclure les divers modèles dans cette traçabilité, ce qui n'est pas forcément faisable en pratique avec les outils actuels. De plus, si l'on exprime beaucoup de choses avec une seule exigence ou un seul modèle/élément de modèle, cela peut nécessiter beaucoup de liens de traçabilité sur cet élément, ce qui n'est généralement pas désirable.

L'ajout d'intermédiaires, comme dans l'exemple "Quand l'état "Actif" ou l'état "Inactif" est vrai, alors V doit être égal à V_comm" juste au dessus, va nécessairement cacher les relations entre exigences et interfaces du système. Au lieu d'avoir directement un lien entre les valeurs de "P", "C" et "V" comme dans les deux exigences de départ, dans l'exigence finale on va devoir explorer les définitions de "V_comm", "Actif" et "Inactif". Il est possible que cela rende les exigences plus difficiles à lire (et donc à corriger, valider, vérifier, etc.), surtout si les termes utilisés pour désigner les "intermédiaires" sont difficiles à relier à quelque chose de concret.

Ceci dit, il y a bien des cas où remplacer les exigences textuelles par des éléments de modèles est bénéfique. Par exemple, dans le cas de la spécification EGTS, où un automate à états est décrit avec des exigences textuelles, ces exigences ne sont de toute façon pas compréhensibles par quelqu'un ne sachant pas ce qu'est un automate à états et son fonctionnement. Il paraît donc plus pratique de remplacer ces exigences par un modèle explicite, parce que les exigences textuelles ne sont pas plus facile à comprendre que le modèle.

7.2.2 De nouveaux critères pour les exigences ?

il existe de nombreuses listes de critères que devraient respecter les exigences, mais nous pensons que ces listes peuvent être améliorées.

7.2.2.1 Qu'est-ce qu'un "bon" critère ?

Les critères de qualité que devraient respecter les exigences sont, en quelque sorte, des exigences sur les exigences. On ne peut pas demander à ce que tous les principes de l'ingénierie des exigences s'appliquent aussi à ces méta-exigences, car les éléments spécifiés (systèmes pour les exigences, exigences pour les méta-exigences) sont très différents. Cependant, on peut faire un certain nombre de parallèles.

Comme les exigences elles-mêmes, ces critères sont issus de buts plus généraux. Le principal étant que les exigences sont un outil de communication, et doivent donc permettre de communiquer avec le moins de distorsion possible entre l'émetteur et le receveur. Il y a d'autres buts, comme le besoin de gestion de ces exigences, ou qu'il faille qu'elles ne soient pas trop complexes à lire et à écrire.

Si l'on considère que les critères sur les exigences sont eux-mêmes des exigences, cela signifie que ces critères doivent s'appliquer à eux-mêmes. Nous avons constaté que ce n'est pas nécessairement le cas. Par exemple, beaucoup de ces critères sur les exigences ne sont pas atomiques : toutes les listes de critères incluent le fait que les exigences ne doivent pas être ambiguës, mais une exigence peut être ambiguë de différentes manières (ambiguïté lexicale, syntaxique ou autre), et il nous semblerait utile d'avoir des critères plus simples, plus atomiques. Il y a aussi des critères qui s'incluent les uns dans les autres. Par exemple, lorsque l'on demande qu'une exigence soit "claire", cela implique généralement qu'elle ne soit pas ambiguë. Comme le critère d'ambiguïté est déjà composé de plusieurs critères plus simples, c'est donc encore pire pour la "clarté".

Certains travaux (Saavedra et collab. (2013) par ex.) s'intéressent également aux contradictions entre critères.

7.2.2.2 Critères pertinents

Il faut des critères qui découlent des buts que l'on fixe aux exigences, sinon ces critères ne sont pas utiles. Toujours en faisant un parallèle avec les exigences, il pourrait peut-être être utile de définir une liste de critères en incluant explicitement la traçabilité entre ces critères et les buts dont ils découlent. Il pourrait aussi être utile de définir une hiérarchie de critères, avec des critères composés de sous-critères, qui sont vérifiés si tous les sous-critères sont vérifiés.

Dans les listes de critères actuelles, il existe généralement des classifications, ne serait-ce que de savoir si tel critère s'applique à une exigence unique ou à un ensemble d'exigences. L'effort de hiérarchisation mentionné dans le paragraphe précédent pourrait permettre de compléter ces classifications déjà existantes.

7.2.2.3 Critères testables

Vérifier que toutes les exigences respectent un ensemble de critères devient plus long si l'on rajoute de nouveaux critères, même si ces critères ne sont que des décompositions de critères plus généraux. Un moyen de limiter le travail nécessaire pour ces vérifications est de s'assurer que ces critères soient testables facilement.

Premièrement, il serait pratique que, comme pour les exigences, il soit possible de dire si oui ou non le critère est respecté. Ce n'est pas faisable pour tous les critères, mais a priori, surtout si l'on introduit des critères plus simples et plus atomiques, il doit être possible qu'une partie des critères soient précis et booléens. Par exemple une quantité pertinente pour savoir si une exigence est trop longue est son nombre de mots, en ajoutant un seuil sur le nombre de mots, on obtient un critère booléen.

Une fois que l'on a des critères booléens, il faut créer des procédures, informatisées ou non, pour réaliser les tests pertinents. De la même manière que pour les exigences, un critère n'est pas utile s'il n'est pas testable. Plus les critères sont testables facilement, moins la gestion des exigences en général sera difficile. On va donc chercher à avoir des critères qui, en plus d'être pertinents par rapport au but des exigences, soient testables et facilement testables.

7.2.3 Cohérence entre langue naturelle et d'autres médias

La coexistence de texte en langue naturelle et d'éléments de syntaxe formelle mène à un problème crucial : nous devons assurer que les informations contenues dans ces deux médias sont cohérentes. Comme pour tout cas où l'on a plus d'une description de la même chose, nous devons être sûr que ces descriptions ne se contredisent pas. Lorsqu'on modifie l'une d'elles, la modification devrait affecter les autres descriptions, ou au moins l'impact devrait être facilement analysable.

7.2.3.1 Un problème difficile

Un moyen d'assurer la cohérence est de considérer une des descriptions comme la source, et de l'utiliser pour générer l'autre. Cette génération devrait être faite de préférence automatiquement, pour éviter une charge de travail additionnelle et pour être assuré de la cohérence. C'est difficile à réaliser avec la langue naturelle, qu'elle soit la source ou la description générée automatiquement :

— Si la langue naturelle est la source, cela veut dire que l'on est capable de générer automatiquement du texte formel à partir de texte non formel. Certain travaux (de Almeida Ferreira et da Silva (2009) par ex.) se concentrent sur ce problème, mais le traitement automatique de la langue est difficile et n'est pas forcément bien adapté aux applications que l'on traite ici.

— Si le langage formel est la source, cela signifie que le rédacteur des exigences doit être capable de lire et d'écrire ce langage formel. C'est un défaut important si l'on souhaite avoir une méthode relativement facile à utiliser. Accessoirement, le texte généré peut sembler peu naturel aux lecteurs.

Le principe des langues naturelles contrôlées “basées sur la logique” est d'utiliser le langage formel comme source et de le “cacher” en utilisant des mots du langage naturel. Le rédacteur a toujours besoin de maîtriser, ou au moins de comprendre, la logique formelle sous-jacente, pour savoir ce qu'il peut ou ne peut pas écrire.

7.2.3.2 Atténuation possible ?

S'il n'est pas possible de transformer automatiquement un texte en langue naturelle en texte formel, nous pensons qu'il est possible de faciliter la rédaction d'exigences formelles.

Nous proposons dans ce travail tout d'abord de limiter les termes utilisables dans les exigences (en tout cas pour les parties formalisées des exigences), de la même manière qu'une langue contrôlée inclut généralement une liste noire/liste blanche de vocabulaire utilisable. Des routines d'autocomplétion ou de correction automatique, comme celles qui existent actuellement pour la langue naturelle, pourraient être utiles afin que les rédacteurs d'exigences soient aiguillés vers des formulations correctes (au sens du lexique et de la syntaxe proposés), sans que l'effort à fournir ne soit trop important.

Nous proposons aussi de créer des liens explicites entre termes de l'exigence et éléments externes (modèles, définitions, éventuellement fonctions syntaxiques). Si les routines d'autocomplétion/de correction automatique mentionnées au dessus existent, alors on peut, en plus d'utiliser les termes adaptés, créer les références explicites automatiquement.

Par exemple, supposons qu'un utilisateur rédige une exigence pour l'ABS présenté en section 4.1. Il cherche à écrire une condition mentionnant l'état “Arrêt”. Il commence à taper les lettres “arr” et la routine d'autocomplétion lui propose un ensemble de termes, qui sont :

— inclus dans l'ensemble des éléments des modèles associés à l'ABS ou dans les fonctions syntaxiques existantes,
— compatibles syntaxiquement avec ce qui a été écrit dans l'exigence jusqu'ici (par exemple, si juste avant, l'utilisateur a écrit une fonction “dans l'état X”, alors on va attendre un état ensuite),
— compatibles par une certaine mesure avec la chaîne de caractère “arr”.

Parmi ces termes, on va trouver “Arrêt”. L'utilisateur choisit ce terme, ce qui l'ajoute dans la formulation en langue naturelle de l'exigence et dans l'arbre syntaxique. De plus, un lien est automatiquement tracé depuis l'exigence vers l'adresse de l'état “Arrêt”.

On peut aussi imaginer un formalisme graphique où la création d'exigences consisterait à relier des boîtes les une aux autres, comme dans Simulink, de MathWorks, ou dans le langage de programmation visuel Scratch (https ://scratch.mit.edu/).

7.2.4 Éditeur pour exigences et modèles

Les concepts proposés dans la section 7.2.3.2 n'ont pas encore été implémentés. Nous pensons qu'ils sont réalisables, au moins en partie. Il sera nécessaire pour cela que les rédacteurs d'exigences utilisent un éditeur spécialisé. De manière générale, les concepts proposés dans ce travail sont difficilement compatibles avec l'utilisation de simples logiciels de traitement de texte/tableurs pour la rédaction des exigences. Un éditeur d'exigences

spécialisé reste donc à écrire pour pouvoir appliquer ce travail dans le monde réel.

Il y a également la question du ou des éditeur(s) pour les modèles. Nous n'avons pas défini précisément quels sont les modèles à utiliser dans notre méthode, il serait donc prématuré de parler d'outils précis à utiliser. Cependant, nous pouvons déjà définir quelques contraintes basées sur nos besoins. Par exemple, nous préférerions importer des modèles déjà existants plutôt que de les recréer à partir de rien. Il serait donc préférable d'utiliser des types de modèles et des outils proches de ceux qui existent actuellement et sont répandus dans l'industrie.

Toujours dans un soucis de réutilisation, notre but n'est pas de créer de nouveaux éditeurs d'exigences ou de modèles. Les éditeurs d'exigences tels que DOORS sont assez modulables, et plutôt que de les remplacer, il serait plus pertinent de les compléter avec des modules additionnels.

7.2.5 Autres spécifications

Une autre perspective concerne l'applicabilité de ce que nous proposons à d'autres types de spécifications : les spécifications industrielles que nous avons étudiées concernaient différents systèmes, mais qui étaient relativement similaires (contexte aéronautique, systèmes incluant des parties physiques et logicielles, systèmes critiques). On peut se demander si les idées proposées ici pourraient être appliqués dans des contextes différents. Par exemple, si l'on ne se place plus dans un contexte aéronautique, certains éléments doivent évidemment être adaptés (FH, pour Flight Hour, heure de vol, n'est plus une unité pertinente). Mais la vraie question est de savoir si les principes de bases que nous proposons sont toujours pertinents. Nous pensons que le processus de construction réalisé avec l'exemple de l'ABS est cohérent, mais nous n'avons pas pu étudier des spécifications de l'industrie automobile.

7.2.5.1 Spécifications logicielles

On a déjà mentionné précédemment les différences entre spécifications logicielles et les spécifications de systèmes tels que l'EGTS. On peut s'interroger sur ce qu'impliquent ces différences sur les principes que nous proposons dans ce travail.

Un programme informatique a des interfaces qui sont nécessairement formalisables, puisqu'elles sont effectivement formelles une fois le programme implémenté. Cependant, ces interfaces peuvent être très complexes, et il n'est pas forcément souhaitable de les formaliser lors de la spécification. Par exemple, si le logiciel est utilisé directement par des humains, formaliser complètement l'interface graphique dès l'étape de spécification paraît excessif. Les logiciels, comparés aux systèmes physiques, vont également manipuler beaucoup plus souvent des concepts discrets et des comportements séquentiels.

Si l'on souhaite appliquer les idées proposées dans ce travail aux spécifications logicielles, il faut donc prendre en compte ces différences. Par exemple on peut imaginer d'adapter la syntaxe des exigences en autorisant des exigences plus "séquentielles", comme "quand tel événement survient, ensuite il doit se passer ça". Il est probablement nécessaire d'avoir d'autres types de modèles à référencer dans les exigences, qui soient plus adaptés aux interfaces des systèmes logiciels.

7.2.5.2 Interfaces plus ou moins formalisables

Il existe beaucoup de cas où les interfaces d'un système sont trop complexes pour être formalisées dans une spécification.

Par exemple, on trouve l'exigence suivante dans la spécification de l'EGTS : "Tous les composants situés sur le train d'atterrissage doivent supporter ou être protégés contre leur utilisation par inadvertance comme une marche ou comme une prise." Cette exigence paraît utile, mais elle n'est pas exactement précise. Cependant, détailler exactement les efforts possibles dus à "l'utilisation par inadvertance comme une marche ou comme une prise" va être assez complexe, surtout que les rédacteurs de la spécification ne savent pas a priori la forme et la localisation exacte des composants.

Un autre exemple, lié à la question de savoir si les spécifications doivent être des "boîtes noires", est celui d'un ascenseur. Un ascenseur est constitué de divers éléments, mais, si l'on considère une spécification en "boîte noire", la spécification n'est pas sensée faire référence à ces divers sous-composants. Pourtant, les utilisateurs vont principalement interagir avec la cabine de l'ascenseur, et essayer de spécifier l'ascenseur sans faire référence à cette cabine (ne serait-ce que pour en définir les dimensions) est pratiquement impossible.

On a remarqué que, assez souvent, les interfaces difficiles à formaliser étaient les interfaces entre des humains et le système considéré. Le problème est que les interactions avec des humains sont potentiellement très complexes et imprévisibles, et ni les modèles ni les spécifications ne peuvent inclure de manière satisfaisante toutes ces possibilités.

Théoriquement, il serait possible de définir l'ascenseur uniquement en fonction de ses interactions avec son environnement : tout objet peut se définir seulement par ses interactions avec le monde externe, y compris ses "caractéristiques". Par exemple, la masse d'un objet peut se définir comme sa résistance à l'accélération, sa couleur peut être définie en fonction de la longueur d'onde de la lumière réfléchie, etc. Mais définir l'ascenseur seulement par ses interactions avec son environnement n'est ni pratique, ni même généralement faisable dans une spécification.

Une solution possible est que "cabine" ou "l'utilisation par inadvertance comme une marche ou comme une prise" soient définis, pas de manière formelle, mais en tout de même plus précisément qu'en utilisant seulement ces mots.

Financement

Cette thèse a été financée par la chaire Blériot-Fabre entre Safran et CentralSupélec.

Bibliographie

Abrial, J.-R. 2010, *Modeling in Event-B : system and software engineering*, Cambridge University Press. 4, 16, 73

de Almeida Ferreira, D. et A. R. da Silva. 2009, «A controlled natural language approach for integrating requirements and model-driven engineering», dans *Proceedings of the fourth International Conference on Software Engineering Advances (ICSEA'09)*, IEEE, p. 518–523. 16, 131

Badreau, S. et J.-L. Boulanger. 2014, *Ingénierie des exigences : Méthodes et bonnes pratiques pour construire et maintenir un référentiel*, Dunod. 1, 5, 7, 8, 39, 40

Bano, M., D. Zowghi et N. Ikram. 2014, «Systematic reviews in requirements engineering : A tertiary study», dans *Proceedings of the fourth international workshop on Empirical Requirements Engineering (EmpiRE), at RE'14*, IEEE, p. 9–16. 2

Batteux, M., T. Prosvirnova et A. Rauzy. 2015, «System Structure Modeling Language (S2ML)», URL https://hal.archives-ouvertes.fr/hal-01234903, consulté le 23 octobre 2017. 52

Bechhofer, S., F. van Harmelen, J. Hendler, I. Horrocks, D. L. McGuinness, P. F. Patel-Schneider et L. A. Stein. 2009, «OWL : Web ontology language reference», URL https://www.w3.org/TR/owl-ref/, consulté le 23 octobre 2017. 15

Beck, K., M. Beedle, A. Van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, J. Grenning, J. Highsmith, A. Hunt, R. Jeffries et collab.. 2001, «Manifesto for agile software development», URL http://agilemanifesto.org/, consulté le 23 octobre 2017. 38

Behm, P., P. Benoit, A. Faivre et J.-M. Meynadier. 1999, «METEOR : A successful application of B in a large project», dans *Proceedings of the world conference on Formal Methods in the Development of Computing Systems (FM'99)*, Springer LNCS 1708, p. 369–387. 28

Bernard, Y. 2012, «Requirements management within a full model-based engineering approach», *Systems Engineering*, vol. 15, n° 2, p. 119–139. 17

Betts, R. G. 2003, «Wycliffe associates EasyEnglish : Challenges in cross-cultural communication», *Proceeding of EAMT-CLAW 2003*. 9

Bijan, Y., J. Yu, J. Stracener et T. Woods. 2013, «Systems requirements engineering–state of the methodology», *Systems Engineering*, vol. 16, n° 3, p. 267–276. 3, 6

Bone, M. et R. Cloutier. 2010, «The current state of model based systems engineering : Results from the OMG™ SysML request for information 2009», dans *Proceedings of the 8th Conference on Systems Engineering Research (CSER)*. 12, 13, 14

Cabral, G. et A. Sampaio. 2008, «Formal specification generation from requirement documents», *Electronic Notes in Theoretical Computer Science*, vol. 195, p. 171–188. 16

Cameron, B., E. Crawley et D. Selva. 2016, *Systems Architecture, Global Edition*, Pearson Education Limited, ISBN 9781292110851. 10

Chantree, F. J. 2006, *Identifying nocuous ambiguity in natural language requirements*, thèse de doctorat, THE OPEN UNIVERSITY. 6

Cheng, B. H. et J. M. Atlee. 2007, «Research directions in requirements engineering», dans *FOSE'07 : Future of Software Engineering*, IEEE Computer Society, p. 285–303. 2

Clocksin, W. et C. S. Mellish. 2003, *Programming in PROLOG*, Springer. 15

Cockburn, A. 2001, *Writing Effective use Cases*, Addison-Wesley. 3

Colomb, R., K. Raymond, L. Hart, P. Emery, C. Welty, G. T. Xie et E. Kendall. 2006, «The object management group ontology definition metamodel», dans *Ontologies for software engineering and software technology*, Springer, p. 217–247. 18

Cowan, J. W. 1997, *The complete Lojban language*, Logical Language Group. 15

Cycorp. 2006, «ResearchCyc», URL http://www.cyc.com/platform/researchcyc/, consulté le 23 octobre 2017. 18

Dodd, T. et A. Dodd. 1990, *PROLOG ; A Logical Approach*, Oxford University Press. VII

Dori, D. 2011, *Object-process methodology : A holistic systems paradigm*, Springer. 11, 12

Estefan, J. A. et collab.. 2007, «Survey of model-based systems engineering (MBSE) methodologies», *Incose MBSE Focus Group*, vol. 25, n° 8. 11

Fraga, A., J. Llorens, L. Alonso et J. M. Fuentes. 2015, «Ontology-assisted systems engineering process with focus in the requirements engineering process», dans *Proceedings of Complex Systems Design & Management (CSDM'14)*, Springer, p. 149–161. 8, 17

Friedenthal, S., R. Griego et M. Sampson. 2007, «INCOSE model based systems engineering (MBSE) initiative», dans *INCOSE 2007 Symposium*. 11

Friedenthal, S., A. Moore et R. Steiner. 2014, *A practical guide to SysML : the systems modeling language*, Morgan Kaufmann. 11, 18, 52

Fuchs, N. E. et R. Schwitter. 1996, «Attempto Controlled English (ACE)», dans *Proceedings of the First International Workshop on Controlled Language Applications*, Katholieke Universiteit Leuven, p. 124–136. 8, 16

Gal, A., G. Lapalme et P. Saint-Dizier. 1989, *Prolog pour l'analyse automatique du langage naturel*, Éditions Eyrolles. 15

Gervasi, V. et B. Nuseibeh. 2002, «Lightweight validation of natural language requirements», *Software : Practice and Experience*, vol. 32, n° 2, p. 113–133. 16

Glinz, M. 2002, «Statecharts for requirements specification – as simple as possible, as rich as needed», dans *Proceedings of the ICSE'02 workshop on scenarios and state machines : models, algorithms, and tools*. 3

Gubisch, M. 2016, «Farnborough : Honeywell and safran halt electric taxi project», URL https://www.flightglobal.com/news/articles/farnborough-honeywell-and-safran-halt-electric-tax-427400/, consulté le 23 octobre 2017. 82

Guillerm, R., H. Demmou et N. Sadou. 2010, «Information model for model driven safety requirements management of complex systems.», dans *Proceedings of Complex Systems Design & Management (CSDM'10)*, Springer, p. 99–111. 4

Hall, A. D. et R. E. Fagen. 1956, «Definition of system», *General Systems*, vol. 1, n° 1, p. 18–28. 9

Harel, D. 1987, «Statecharts : A visual formalism for complex systems», *Science of Computer Programming*, vol. 8, n° 3, p. 231–274. 23

Hutchinson, J., J. Whittle et M. Rouncefield. 2014, «Model-driven engineering practices in industry : Social, organizational and managerial factors that lead to success or failure», *Science of Computer Programming*, vol. 89, p. 144–161. 12

INCOSE. «What is systems engineering ?», URL http://www.incose.org/AboutSE/WhatIsSE, consulté le 23 octobre 2017. 10

Ingham, M. D., R. D. Rasmussen, M. B. Bennett et A. C. Moncada. 2004, «Engineering complex embedded systems with state analysis and the mission data system», *Journal of Aerospace Computing, Information, and Communication*. 11

ISO/IEC/IEEE. 2011, «ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering», . 7, 19, 39

Issad, M., L. Kloul, A. Rauzy et K. Berkani. 2015, «ScOLA, a scenario oriented modeling language for railway systems», *INSIGHT*, vol. 18, n° 4, p. 34–37. 13

Jacobson, I. 1993, *Object-oriented software engineering : A use case driven approach*, Addison-Wesley. 13

Jeannet, B. et F. Gaucher. 2015, «Debugging real-time systems requirements : Simulate the "what" before the "how"», dans *Embedded World Conference*. 4

KAINDL, H. 2005, «A scenario-based approach for requirements engineering : Experience in a telecommunication software development project», *Systems Engineering*, vol. 8, n° 3, p. 197–210. 17, 18

KAIYA, H. et M. SAEKI. 2005, «Ontology based requirements analysis : Lightweight semantic processing approach», dans *Proceedings of the Fifth Quality Software International Conference (QSIC'05)*, IEEE, p. 223–230. 7, 17

KAMSTIES, E. et B. PEACH. 2000, «Taming ambiguity in natural language requirements», dans *Proceedings of the thirteenth International Conference on Software and Systems Engineering and Applications (ICSSEA'00)*. 3, 7

KINCAID, J. P., R. P. FISHBURNE JR, R. L. ROGERS et B. S. CHISSOM. 1975, «Derivation of new readability formulas for navy enlisted personnel», cahier de recherche, Naval Technical Training Command Millington TN Research Branch. 7

KNORRECK, D., L. APVRILLE et P. DE SAQUI-SANNES. 2011, «TEPE : a SysML language for time-constrained property modeling and formal verification», *ACM SIGSOFT Software Engineering Notes*, vol. 36, n° 1, p. 1–8. 15

KÖRNER, S. J. et T. BRUMM. 2009, «RESI - A natural language specification improver», dans *International Conference on Semantic Computing (ICSC'09)*, IEEE, p. 1–8. 7, 18

KROB, D. 2017, «CESAM : CESAMES systems architecting method», . 52

KUHN, T. 2014, «A survey and classification of controlled natural languages», *Computational Linguistics*, vol. 40, n° 1, p. 121–170. 8

LAMPORT, L. 1989, «A simple approach to specifying concurrent systems», *Communications of the ACM*, vol. 32, n° 1, p. 32–45. 4

VAN LAMSWEERDE, A. 2001, «Goal-oriented requirements engineering : A guided tour», dans *Proceedings of the fifth international symposium on Requirements Engineering*, IEEE, p. 249–262. 2, 3

LEBEAUPIN, B. 2015, «A language for writing system specifications in an aeronautical context», dans *Requirements Engineering Conference (RE)*.

LEBEAUPIN, B., A. RAUZY et J.-M. ROUSSEL. «Towards a better integration of requirements and model-based specifications», *en révision pour publication dans Systems Engineering*.

LEBEAUPIN, B., A. RAUZY et J.-M. ROUSSEL. 2017, «A language proposition for system requirements», dans *Proceedings of the 11th Annual IEEE International Systems Conference (SysCon)*, IEEE.

LEGENDRE, A., A. LANUSSE et A. RAUZY. 2017, «Toward model synchronization between safety analysis and system architecture design in industrial contexts», dans *Proceedings of the International Symposium on Model-Based Safety and Assessment*, Springer, p. 35–49. 12

Li, J., H. Zhang, L. Zhu, R. Jeffery, Q. Wang et M. Li. 2012, «Preliminary results of a systematic review on requirements evolution», dans *Proceedings of the 16th international conference on Evaluation & Assessment in Software Engineering (EASE'12)*, IET, p. 12–21. 5

Mavin, A., P. Wilkinson, A. Harwood et M. Novak. 2009, «Easy approach to requirements syntax (EARS)», dans *Proceedings of the international symposium on Requirements Engineering*, IEEE, p. 317–322. 8

Mhenni, F., N. Nguyen et J.-Y. Choley. 2014, «Automatic fault tree generation from SysML system models», dans *Proceedings of the IEEE/ASME international conference on Advanced Intelligent Mechatronics (AIM)*, IEEE, p. 715–720. 15

Micouin, P. 2014, *Model Based Systems Engineering : Fundamentals and Methods*, John Wiley & Sons. 10

Miller, G. A. 1995, «WordNet : A lexical database for English», *Communications of the ACM*, vol. 38, n° 11, p. 39–41. 18

Nicolás, J. et A. Toval. 2009, «On the generation of requirements specifications from software engineering models : A systematic literature review», *Information and Software Technology*, vol. 51, n° 9, p. 1291–1307. 17

Nuseibeh, B. et S. Easterbrook. 2000, «Requirements engineering : A roadmap», dans *Proceedings of the Conference on the Future of Software Engineering*, ACM, p. 35–46. 5

Ogden, C. 1930, *Basic english : A general introduction with rules and grammar*, Kegan Paul, Trench, Trubner & Co. 9

OMG. 2016, «Requirements Interchange Format», URL http://www.omg.org/spec/ReqIF/, consulté le 23 octobre 2017. 18

Ormandjieva, O., I. Hussain et L. Kosseim. 2007, «Toward a text classification system for the quality assessment of software requirements written in natural language», dans *Proceedings of the fourth international workshop on Software Quality Assurance (SOQUA'07)*, ACM, p. 39–45. 7, 127

Parra, E., C. Dimou, J. Llorens, V. Moreno et A. Fraga. 2015, «A methodology for the classification of quality of requirements using machine learning techniques», *Information and Software Technology*, vol. 67, p. 180–195. 7

Perrouin, G., E. Brottier, B. Baudry et Y. Le Traon. 2009, «Composing models for detecting inconsistencies : A requirements engineering perspective», dans *Proceedings of the international working conference on Requirements Engineering : Foundation for Software Quality (REFSQ'09)*, Springer LNCS, p. 89–103. 2

Pohl, K. 2010, *Requirements engineering : Fundamentals, principles, and techniques*, Springer. 1, 5, 7, 8

Prosvirnova, T., M. Batteux, P.-A. Brameret, A. Cherfi, T. Friedlhuber, J.-M. Roussel et A. Rauzy. 2013, «The AltaRica 3.0 project for model-based safety assessment», *Proceedings of the fourth IFAC workshop on Dependable Control of Discrete Systems (DCDS)*, vol. 46, n° 22, p. 127–132. 15

Rajan, A. et T. Wahl. 2013, *CESAR : Cost-efficient methods and processes for safety-relevant embedded systems*, Springer. 6, 7, 39, 41

Regnell, B., K. Kimbler et A. Wesslén. 1995, «Improving the use case driven approach to requirements engineering», dans *Proceedings of the second international symposium on Requirements Engineering (RE'95)*, IEEE, p. 40–47. 13

Reif, K., éd.. 2015, *Automotive Mechatronics : Automotive Networking, Driving Stability Systems, Electronics*, Springer Vieweg. XV, 49

Robertson, S. et J. Robertson. 2012, *Mastering the requirements process : Getting requirements right*, Addison-wesley. 2

Saavedra, R., L. C. Ballejos et M. Ale. 2013, «Quality properties evaluation for software requirements specifications : An exploratory analysis.», dans *WER'13*, p. 6–19. 7, 39, 40, 130

Sabetzadeh, M. et S. Easterbrook. 2005, «Traceability in viewpoint merging : A model management perspective», dans *Proceedings of the third international workshop on Traceability in Emerging Forms of Software Engineering (TEFSE'05)*, ACM, p. 44–49. 2

Safwat, H. et B. Davis. 2014, «A brief state of the art of CNLs for ontology authoring», dans *Proceedings of the fourth workshop on Controlled Natural Language (CNL'14)*, vol. 8625, Springer LNAI, p. 190–200. 16

Scanniello, G., M. Staron, H. Burden et R. Heldal. 2014, «On the effect of using SysML requirement diagrams to comprehend requirements : Results from two controlled experiments», dans *Proceedings of the 18th international conference on Evaluation and Assessment in Software Engineering (EASE'14)*, ACM, p. 433–442. 14

Schmidt, R., K. Lyytinen, M. Keil et P. Cule. 2001, «Identifying software project risks : An international Delphi study», *Journal of Management Information Systems*, vol. 17, n° 4, p. 5–36. IV

Shah, U. S. et D. C. Jinwala. 2015, «Resolving ambiguities in natural language software requirements : A comprehensive survey», *ACM SIGSOFT Software Engineering Notes*, vol. 40, n° 5, p. 1–7. 3

Somé, S. S. 2006, «Supporting use case based requirements engineering», *Information and Software Technology*, vol. 48, n° 1, p. 43–58. 13

Stecklein, J. M., J. Dabney, B. Dick, B. Haskins, R. Lovell et G. Moroney. 2004, «Error cost escalation through the project life cycle», dans *INCOSE 14th Annual International Symposium*, p. 19–24. 24

STEMG. 2017, «ASD Simplified Technical English», URL http://www.asd-ste100.org/, consulté le 23 octobre 2017. 8, 9

Torkar, R., T. Gorschek, R. Feldt, M. Svahnberg, U. A. Raja et K. Kamran. 2012, «Requirements traceability : A systematic review and industry case study», *International Journal of Software Engineering and Knowledge Engineering*, vol. 22, n° 3, p. 385–433. 2

Vilela, J., J. Castro, L. E. G. Martins et T. Gorschek. 2017, «Integration between requirements engineering and safety analysis : A systematic literature review», *Journal of Systems and Software*, vol. 125, p. 68–92. 4

Vogelsang, A., T. Amorim, F. Pudlitz, P. Gersing et J. Philipps. 2017, «Should I stay or should I go ? On forces that drive and prevent MBSE adoption in the embedded systems industry», dans *18th international conference on Product-Focused Software Process Improvement (PROFES)*. 11, 12, 13

Whalen, M. W. 2000, *A Formal Semantics for the Requirements State Machine Language without Events*, mémoire de maîtrise, University of Minnesota. 4, 16

Wymore, A. W. 1993, *Model-based systems engineering*, CRC press. 29

Zave, P. et M. Jackson. 1997, «Four dark corners of requirements engineering», *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 6, n° 1, p. 1–30. 52

Zowghi, D. et V. Gervasi. 2002, «The three Cs of requirements : Consistency, completeness, and correctness», dans *Proceedings of the international workshop on Requirements Engineering : Foundations for Software Quality (REFSQ'02)*, Springer, p. 155–164. 41

université PARIS-SACLAY

ÉCOLE DOCTORALE
INTERFACES
Approches Interdisciplinaires : Fondements, Applications et Innovation

Benoît Lebeaupin

Vers un langage de haut niveau pour une ingénierie des exigences agile dans le domaine des systèmes embarqués avioniques

Towards a high level language for agile requirements engineering in an aeronautical context

CentraleSupélec

Résumé : La complexité des systèmes conçus actuellement devient de plus en plus importante. En effet, afin de rester compétitives, les entreprises concevant des systèmes cherchent à leur rajouter de plus en plus de fonctionnalités. Cette compétitivité introduit aussi une demande de réactivité lors de la conception de systèmes, pour que le système puisse évoluer lors de sa conception et suivre les demandes du marché.

Un des éléments identifiés comme empêchant ou diminuant cette capacité à concevoir de manière flexible des systèmes complexes concerne les spécifications des systèmes, et en particulier l'utilisation de la langue naturelle pour spécifier les systèmes. Tout d'abord, la langue naturelle est intrinsèquement ambiguë et cela risque donc de créer des non-conformités si client et fournisseur d'un système ne sont pas d'accord sur le sens de sa spécification. De plus, la langue naturelle est difficile à traiter automatiquement, par exemple, on peut difficilement déterminer avec un programme informatique que deux exigences en langue naturelle se contredisent. Cependant, la langue naturelle reste indispensable dans les spécifications que nous étudions, car elle reste un moyen de communication pratique et très répandu.

Nous cherchons à compléter ces exigences en langue naturelle avec des éléments permettant à la fois de les rendre moins ambiguës et de faciliter les traitements automatiques. Ces éléments peuvent faire partie de modèles (d'architecture par exemple) et permettent de définir le lexique et la syntaxe utilisés dans les exigences. Nous avons testé les principes proposés sur des spécifications industrielles réelles et développé un prototype logiciel permettant de réaliser des tests sur une spécification dotée de ces éléments de syntaxe et de lexique.

Mot clés : ingénierie des exigences, langue naturelle, ingénierie basée sur les modèles, langage formel, ingénierie des systèmes complexes, ambiguïté.

Abstract : Systems are becoming more and more complex, because to stay competitive, companies which design systems search to add more and more functionalities to them. Additionally, this competition implies that the design of systems needs to be reactive, so that the system is able to evolve during its conception and follow the needs of the market.

This capacity to design flexibly complex systems is hindered or even prevented by various various elements, with one of them being the system specifications. In particular, the use of natural language to specify systems have several drawbacks. First, natural language is inherently ambiguous and this can leads to non-conformity if customer and supplier of a system disagree on the meaning of its specification. Additionally, natural language is hard to process automatically : for example, it is hard to determine, using only a computer program, that two natural language requirements contradict each other. However, natural language is currently unavoidable in the specifications we studied, because it remains very practical, and it is the most common way to communicate.

We aim to complete these natural language requirements with elements which allow to make them less ambiguous and facilitate automatic processing. These elements can be parts of models (architectural models for example) and allow to define the vocabulary and the syntax of the requirements. We experimented the proposed principles on real industrial specifications and we developped a software prototype allowing to test a specification enhanced with these vocabulary and syntax elements.

Keywords : Requirements Engineering, Natural Language, Model-Based Systems Engineering, Formal Language, Complex Systems Engineering, Ambiguity.