En vue de l'obtention de l'Habilitation à Diriger des Recherches

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

Présentée et soutenue le *Lundi 9 septembre 2019* **par :**

Thomas Polacsek

Vérification, validation, certification : approches formelles et informelles pour établir la correction des artefacts et des logiciels

---

JURY

Florence Sèdes	Professeure des universités	référente
Philippe Besnard	Directeur de Recherche CNRS	examinateur
Régine Laleau	Professeure des universités	rapporteure
Daniel Le Berre	Professeur des universités	rapporteur
Lionel Seinturier	Professeur des universités	rapporteur
Mireille Blay-Fornarino	Professeure des universités	examinatrice
Óscar Pastor	Professeur des universités	examinateur
Claude Cuiller	Ingénieur	invité

Sommaire

Note d'édition

La présente édition est une version allégée de mon manuscrit d'Habilitation à Diriger des Recherches. Le but du présent document est de donner une vue générale et des perspectives épistémologiques des problèmes liés à la Vérification, la Validation et la Certification, ainsi qu'une vision des travaux que j'ai menés sur ces sujets.

Ont été enlevés, les annexes contenant des articles publiés ainsi que les descriptifs des projets et des activités d'animation de recherche. Le lecteur désirant ces informations pourra se référer au manuscrit disponible sur l'archive ouverte pluridisciplinaire HAL.

Préambule chronologique

Avril 2019

Après une thèse en Intelligence Artificielle (IA), j'ai décidé de me diriger vers des activités plus appliquées de Recherche & Développement. J'ai ainsi participé à l'élaboration et à la création d'un système expert pour des audits de conseils bancaires réalisée par la Banque de France. Issu de ce que l'on nomme IA, un système expert est un programme informatique qui « *raisonne* », suivant des règles préétablies par des humains qui ont un grand niveau d'expertise dans un domaine donné. J'ai ensuite conçu et réalisé, avec des experts du transport urbain, un logiciel visant à réorganiser en temps réel les parcours de bus urbains selon les aléas du trafic routier. Cette activité consistait notamment à modéliser de façon formelle, c'est-à-dire mathématique, le problème, mais aussi à concevoir des algorithmes d'optimisation afin de proposer la meilleure configuration de l'offre de transport public pour faire face aux événements. Toujours, sur les problématiques de transport, j'ai collaboré avec Vincent Vidal sur une application des technologies de planification et j'ai initié un partenariat entre la RATP [1] et le CRIL [2].

Fort de ces expériences, j'ai décidé de revenir dans le monde de la recherche et j'ai intégré l'ONERA [3]. Avec Laurence Cholvy, dans le cadre du projet européen CRESCENDO [4], j'ai commencé à m'intéresser au thème de l'*Argumentation* et à appliquer les résultats issus de cette discipline pour une aide à la prise de

1. Régie Autonome des Transports Parisiens.
2. Centre de Recherche en Informatique de Lens, le CRIL est une Unité Mixte de Recherche de l'Université d'Artois et du CNRS.
3. Office National Etudes et de Recherches Aérospatiales.
4. Collaborative and Robust Engineering using Simulation Capability Enabling Next Design Optimisation (projet européen FP7, cinquante-huit partenaires).

décisions collaboratives rationnelles [1].

Dans la continuité de ces travaux, j'ai appliqué des techniques issues de l'IA, comme l'usage de solveurs (model checking) pour les systèmes avioniques embarqués. Avec Rémi Delmas, nous avons montré qu'il était possible d'établir, de façon automatique à l'aide de logiciels, qu'un modèle est conforme à certaines propriétés souhaitées par les concepteurs. L'ensemble de ces travaux se sont inscrits dans des projets collaboratifs ayant notamment pour cadre la coopération AIRSYS [2] et l'ITEA3 [3]. Par la suite, dans le cadre du co-encadrement de la thèse d'Anthony Fernandes-Pires avec Virginie Wiels, nous avons étendu ces travaux au code informatique et, plus précisément, à la conformité d'un code par rapport à sa spécification.

Par la suite, toujours avec Rémi Delmas, nous avons appliqué nos travaux portant sur l'utilisation de techniques issues de l'IA à un contexte plus vaste que celui du logiciel embarqué. Ainsi, nous nous sommes intéressés aux Systèmes d'Information (SI) et à la possibilité d'offrir une aide automatique pour leur spécification. Plus précisément, nous nous sommes focalisés sur les exigences qui permettent de caractériser un système d'alarmes et sur les vérifications formelles pouvant être réalisées sur de telles exigences. C'est sur ce thème qu'avec Florence Sèdes, nous avons organisé tous les trois une série de trois ateliers sur ce thème, un dans le cadre de la conférence internationale ABDIS [4] et deux dans le cadre de la conférence Inforsid.

Parallèlement à ces activités, confronté à des modèles excessivement complexes, de par leur taille et l'hétérogénéité des concepts qu'ils manipulent, je me suis intéressé à leur visualisation. Bénéficiant d'un soutien du RNSC [5], avec David Bihanic, nous avons entamé un dialogue interdisciplinaire, dialogue qui s'est poursuivi par la création d'un groupe de travail avec des chercheurs en design, des géographes, des psychologues et des informaticiens spécialisés en interaction homme-machine dans le cadre d'une action financée durant deux années par l'association INFORSID et le GDR I3 [6].

Ces interactions avec le milieu industriel et la position privilégiée de l'ONERA, m'ont fait me confronter aux limitations de l'aide automatique, autrement dit du formel. En effet, dans la pratique, un ensemble d'information n'est pas toujours modélisé formellement, soit par manque de temps ou de compétence, soit parce qu'il n'existe pas encore de cadre mathématique permettant de le faire. Dès lors,

1. Ce travail a donné lieu à un *best paper award* lors de la conférence internationale CSCWD 2011.
2. Architecture et IngénieRie des SYStèmes (AIRSYS) est une coopération entre la société Airbus, l'Institut de Recherche en Informatique de Toulouse (IRIT), le Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS) et l'ONERA.
3. Information Technology for European Advancement.
4. Advances in Databases and Information Systems.
5. Réseau National des Systèmes Complexes.
6. Groupement De Recherche Information Interaction Intelligence.

je me suis intéressé à comment organiser et utiliser une masse d'informations documentaires pour justifier qu'une chose est correct. Pour cela, j'ai défini les *Diagrammes de Justification*, qui se situent à la frontière du formel et de l'informel. Avec des ingénieurs spécialisés en simulations numériques, nous avons utilisé ces diagrammes dans le cadre du projet européen TOICA [1]. J'ai également pu appliquer cette approche à base de Diagrammes de Justification, dans le cadre de projets avec les autorités de certification avionique, pour clarifier certains points concernant la certification. Par ailleurs, dans le domaine médical, l'application de ces diagrammes, ainsi que leur modélisation formelle, a fait l'objet de la thèse de Clément Duffau que j'ai co-encadrée avec Mireille Blay-Fornarino.

Depuis quelques années, j'applique la méthodologie issue des Diagrammes de Justification dans le cadre de la certification avionique. Avec Kevin Delmas et Claire Pagetti, nous avons ainsi défini des patrons, des diagrammes génériques, pour la certification de processeurs multi-cœurs dans le cade d'une convention avec la Direction Générale de l'Aviation Civile (DGAC). Toujours dans le domaine avionique, j'ai participé au groupe de travail RESSAC [2] dont l'objectif était de proposer de nouvelles solutions pour rationaliser les données utilisées dans le cadre de la certification aéronautique.

Il y a deux ans, reprenant nos travaux sur les exigences, avec Rémi Delmas, Juyeon Kang et Florence Sèdes, nous avons cherché à ajouter une dimension linguistique à nos travaux précédents dans le cadre du projet ELENAA [3].

Pour finir, depuis quelques années, avec Stéphanie Roussel, Cédric Pralet et les architectes avion de la société Airbus, nous cherchons à appliquer la modélisation du SI, les techniques d'IA tel que l'optimisation et les problèmes de justification, à un cadre très concret : penser et concevoir conjointement l'avion et l'usine qui va le construire. Ces travaux viennent de donner lieu au démarrage du projet INSTAT [4] de la DGAC.

Après ce préambule qui cherche à donner une perspective chronologique, je vais m'efforcer de présenter, dans ce manuscrit, une vue d'ensemble de mes différents travaux non pas sous un prisme chronologique, mais suivant une cohérence de thèmes, de techniques et de préoccupations.

1. Thermal Overall Integrated Conception of Aircraft (projet européen FP7, trente-deux partenaires).
2. Projet de l'IRT Saint Exupéry.
3. Exigences en LangagE Naturel à leurs Analyses Automatiques (projet région Occitanie).
4. INSTAllation système et Technologies (projet DGAC).

Chapitre 1

De la correction des artefacts

> Alice observait la machine qui produisait des théières en série. Émerveillée elle s'exclama « *c'est extraordinaire elles sont toutes pareilles* ». Le Lièvre et le Chapelier la regardèrent avec étonnement : « *Pareil ? Mais pareil à quoi ?* »
>
> *Alice au pays des machines*, Apocryphe

Les objets conçus par l'homme ne cessent de se complexifier au point que la majorité des objets modernes, comme les voitures, les avions, les constellations de satellites ou même, plus simplement, les téléphones, sont d'une complexité inégalée pour des créations humaines. Si l'on peut rapprocher la conception des premières locomotives à vapeur de l'artisanat, la conception des trains à grande vitesse actuels englobe de nombreuses facettes, telles que l'aérodynamique, la sûreté de fonctionnement ou les contraintes environnementales ainsi que des connaissances techniques de pointe dans des domaines aussi variés que les matériaux composites ou l'informatique embarquée. Cette complexité des techniques nécessite une spécialisation du travail par domaine, spécialisation qui entraîne nécessairement une division du travail. A cette séparation des activités due aux spécialisations techniques vient s'ajouter une division induite par les structures des organisations. Ainsi, les entreprises ne maîtrisent plus à elles seules l'ensemble du processus de conception, mais s'inscrivent dans des réseaux intégrants des partenariats et de la sous-traitance. Dès lors, il devient crucial pour chaque spécialité et pour chaque sous-organisation de s'assurer que ce qu'elle produit est correct, que les « *bonnes choses* » sont faites. Encore faut-il s'entendre sur la signification d'une *bonne chose*. Tout d'abord, nous nous situons dans le cadre de l'ingénierie où les *choses* sont des *artefacts techniques*, c'est-à-dire des objets physiques, mais

aussi des logiciels, des modèles numériques, des études ou des simulations. Par *bonne*, nous cherchons, ici, à établir qu'un artefact est conforme à l'idée que s'en font les concepteurs, en évacuant tout ce qui relève du jugement de valeur, de l'esthétisme ou des opinions. Le mot correct renvoie donc bien ici à sa définition de conforme, qui ne comporte pas d'écart [1]. Comme nous le verrons par la suite, pour nous, la conformité d'un artefact s'exprime au travers de propriétés, mais aussi par rapport à une documentation ou une spécification, voire au moyen de règles régissant le processus de fabrication.

Dans ce mémoire, nous ne nous focalisons pas sur la fabrication, la conception, à proprement parler, mais sur l'élaboration de méthodes qui nous permettent d'établir que le produit final est conforme à ce que l'on en attend. Nous verrons donc comment l'héritage, d'une part d'une tradition argumentative, d'autre part des mathématiques et plus précisément de la logique, peut aider les personnes qui inventent, conçoivent, analysent, construisent ou certifient des artefacts techniques.

1. Division du travail et complexité

Les questions relatives à la conformité d'un produit, à la vérification du travail et plus généralement à tout ce qui relève de la qualité, émergent avec la révolution industrielle. L'arrivée de l'industrie manufacturière, puis de la machine et de la machine à vapeur, a radicalement changé les activités humaines. Avant cela, la fabrication d'objets faisait appel à quelques êtres humains ayant une vue claire et précise de ce qu'ils fabriquaient. Ainsi, un forgeron avait une connaissance des métaux et s'occupait intégralement de la fabrication de tout ce qui relevait de la forge, de la conception jusqu'à la réalisation finale. Il en va de même pour la plupart des corps de métiers. Les métiers les plus techniques s'organisent autour de guildes donnant naissance à l'artisanat où, là encore, l'artisan a une connaissance complète de ce qui est fabriqué. Tout change avec l'arrivée de l'industrie manufacturière. Pour pouvoir produire en masse, l'industrie manufacturière standardise les produits, réduit leur fabrication à des tâches répétitives et instaure une parcellisation du travail. A partir de là se développe une main d'œuvre qui, occupée à réaliser des tâches très spécifiques, finit par perdre de vue le processus global de fabrication. Notons que, même si cette organisation du travail, avec une main-d'œuvre cantonnée à des tâches simples, se retrouve assez loin dans l'histoire de l'humanité [2], elle reste assez anecdotique, cantonnée

1. Définition du Trésor de la Langue Française : « *Qui est conforme à la norme, qui ne comporte pas d'écart par rapport à la norme* »

2. Citons, au 3e siècle avant J.-C., l'armée de terre cuite du mausolée funéraire de Qin Shihuangdila en Chine. En à peu près 40 ans, 7000 statues de guerriers, à taille réelle, peintes et avec un équipement standardisé, ont été conçues [127]. Cet équipement inclus par exemple, des arcs et des flèches. Ainsi, 40000 pointes de flèche en bronze ont été produites, soit un taux de production autour de 1000 pointes par an.

à des domaines très spécifiques comme la fabrication navale avec les Galions espagnols ou l'organisation du travail à la chaîne à l'arsenal de Venise [1] [46]. Les choses s'accélèrent quelque peu au 17e en Europe de l'Ouest avec, par exemple en France sous Colbert, la création de nombreuses manufactures (Manufacture des Gobelins, Manufacture des Dentelles, Manufacture des Glaces qui deviendra Saint-Gobain, etc.).

C'est avec la révolution industrielle que finalement de nombreux objets vont se retrouver standardisés et produits à grande échelle [131]. Cette production massive entraîne un fractionnement du travail, dans des tâches précises et, le plus souvent, répétitives. Cette division du travail, créant une perte de sens dans le travail effectué, sera poussée à son maximum au vingtième siècle, notamment par Frederick Winslow Taylor, au travers de ce qui se nomme en anglais le « *scientific management* » (organisation scientifique du travail) et que nous appelons généralement en français Taylorisme. La production au niveau de l'ouvrier se ramène à des tâches excessivement simples. Cette logique, poussée à l'extrême, amènera à des organisations délétères et Taylor écrira que le travailleur « *qui est apte à manipuler la fonte en tant qu'occupation régulière et qu'il doit être si stupide et si flegmatique qu'il ressemble plus, dans sa constitution mentale, à un bœuf qu'à autre chose.* » [2] [179]. Tout ce mouvement, déconnectant l'homme du produit final, tend à poser des questions relatives à la qualité des tâches produites ainsi que celle du produit. Contrairement à l'artisan qui a une vue globale de ce qui est fait et qui peut corriger une erreur, l'industrie manufacturière demande que chaque poste produise ce dont le poste suivant a besoin, sans déviation possible. Dès lors, à chaque instant de la production, chaque fois qu'un travail est effectué, il doit être possible de l'évaluer, cette évaluation se faisant au moyen de la standardisation. C'est le début du management de la qualité [3], que l'un des fondateurs de la discipline, Walter Andrew Shewhart, définit comme la spécification de la norme de qualité visée, la production de pièces étalons conformes à la qualité visée et l'évaluation de la conformité des pièces produites par rapport à la pièce étalon [4] [170]. Nous voyons, ici, s'établir les bases des questions qui nous préoccupent, c'est-à-dire les questions relatives à la correction d'un produit intermédiaire sans avoir à attendre le résultat final.

Pour répondre à ces questions, des départements sont créés au sein des entre-

1. Fondé autour de 1320, l'arsenal de Venise, ou arsenal de la République Sérénissime, finira dès le 17e siècle par disposer de gréements et de mobiliers de ponts standardisés, d'entrepôts séparés pour chaque article et d'étapes de production à la chaîne sur ce qui ressemblait à une ligne de production.

2. Traduit de : *Is fit to handle pig iron as a regular occupation is that he shall be so stupid and so phlegmatic that he more nearly resembles in his mental make-up the ox than any other type*

3. Même si le gestion de la qualité émerge au début du 20e siècle, comme l'explique Pierre Claude Reynard [158], cette problématique de la qualité est aussi ancienne que l'histoire de l'industrie manufacturière.

4. « *the specification of the aimed-at standard of quality, the production of pieces of product that will be of standard quality and the determination of wheter or not product thus made is of standard quality* ».

prises. Petit à petit, la gestion de la qualité devient une branche de l'industrie à part entière, comme le management, et se retrouve séparée de la production.

Effectuons un saut dans le temps et revenons à notre centre d'intérêt : la conception d'artefacts techniques. Alors que le monde de l'ingénierie semble éloigné de celui de la manufacture du vingtième siècle, force est de constater certains héritages de cette époque dans l'organisation du travail. Ainsi, comme le préconisait Taylor, une division verticale au sein des organisations s'opère, avec des managers qui pensent le travail et des ingénieurs qui l'exécutent. Dans certaines entreprises, bien qu'ayant plus de libertés et de prérogatives, les ingénieurs en développement sont devenus de nouveaux cols bleus. Par ailleurs, nous pouvons retrouver une division horizontale du travail avec des individus qui sont organisés avec des limites précises de ce qui relève de leurs compétences. Par exemple, dans l'industrie du logiciel, nous retrouvons parfois cette séparation avec des concepteurs, des développeurs et des testeurs alors que tous ont les compétences en informatique permettant une polyvalence des tâches. Nous sommes clairement ici face à un héritage de la parcellisation du travail. Dès lors, de par ces contraintes organisationnelles, comme dans l'industrie manufacturière, le besoin d'évaluer à chaque étape si ce qui est produit est correct demeure un problème crucial. Avant d'aller plus avant sur le sujet, nous allons nous pencher sur un autre facteur de division du travail, la spécialisation permettant de concevoir des objets de plus en plus complexes.

Si l'industrialisation a mené à une parcellisation du travail nécessitant des opérations de vérification, il existe un autre mouvement plus profond (et nous oserons dire plus « *positif* ») de division du travail. Pour faire face à la complexité des tâches, l'homme s'est peu à peu spécialisé, divisant ainsi le travail suivant des compétences particulières. Bien qu'il soit difficile d'en établir la preuve, il est plus que probable que, dès les premiers temps de l'humanité, les individus les plus doués pour une tâche en assuraient la réalisation. Pour certains historiens cette division des tâches serait même liée à l'apparition des outils [8]. Cette division, qui a donné lieu aux métiers [1], permettait un travail collaboratif maximisant les compétences de chacun. Nous sommes, ici, dans une division du travail permettant la production d'objets que n'aurait pu fait une personne seule, le tout étant supérieur à la somme de ses parties. De par la spécialisation, et donc la maximisation, du savoir-faire de chacun, il devient possible de concevoir et de fabriquer des objets plus complexes en s'appuyant sur un assemblage de parties plus petites.

Ainsi, la complexification des objets est induite par leur structuration sous forme hiérarchique. Afin d'expliciter cela, nous nous proposons de reprendre la métaphore de Herbert Simon [171]. Il était une fois deux horlogers nommés

1. La division du travail en métiers spécialisés et réglementés est présente dès l'antiquité. Nous pouvons en trouvez une trace, par exemple, dans le code juridique babylonien (1750 avant J.-C.) visible au Musée du Louvre qui fixe les prix de différentes professions.

Hora et Tempus. Ils fabriquaient des montres composées d'un millier de pièces chacune. Les montres de Tempus étaient construites de telle sorte que s'il était interrompu pendant qu'il les assemblait, pour répondre au téléphone par exemple, il devait reprendre son montage depuis le début. Plus les clients aimaient les montres de Tempus, plus ils lui téléphonaient et plus il devenait difficile pour lui de trouver suffisamment de temps ininterrompu pour terminer une montre. De son côté, Hora avait conçu ses montres de manière à pouvoir assembler des sous-ensembles d'une dizaine d'éléments chacun. Dix de ces sous-ensembles pouvaient être assemblés en un sous-ensemble plus grand et un système de dix de ces derniers sous-ensembles constituait l'ensemble de la montre. Ainsi, lorsque Hora devait déposer une montre partiellement assemblée pour répondre au téléphone, il ne perdait qu'une petite partie de son travail. Tempus et Hora étaient très appréciés et de nouveaux clients les appelaient constamment. Cependant, Hora prospérait, tandis que Tempus s'appauvrissait et finit par fermer boutique. Ce que nous voyons à l'œuvre dans cette histoire, c'est que la complexité résulte de l'agrégation ordonnée d'éléments et qu'il est plus facile de maîtriser cette agrégation en définissant des niveaux.

Nous retrouvons cette organisation en éléments agrégés par niveau dans presque tous les produits complexes qui nous entourent. Un ordinateur est composé d'éléments, comme des processeurs, des mémoires, des périphériques, des interconnexions entre ces éléments, sachant que chaque élément est lui-même un objet complexe. Par exemple, le processeur est composé de mémoires, d'unités de calcul, de registres, etc. qui sont composés de portes logiques, qui sont elles-mêmes des ensembles de transistors gravés dans le silicium. En raison de la technicité de chaque élément, concevoir un tel assemblage demande des experts en architecture matérielle différents pour chaque composant et des physiciens pour la partie silicium.

C'est donc grâce à cette division que l'homme a augmenté sa capacité à concevoir et à produire des objets de plus en plus complexes. Mais, cette structuration en ensemble de sous éléments sous-tend une structuration fragmentée du travail. En 1903, les frères Orville et Wilbur Wright réalisent leur premier vol sur un planeur motorisé. Ils sont à la fois les concepteurs, constructeurs et pilotes de l'engin. Tout à la fois chercheurs, ils contribueront à la mécanique du vol, concepteurs et constructeurs, ils tenaient un atelier de fabrication et de vente de bicyclettes, et pilotes, Wilbur réalise les essais en vol, les deux frères embrassent la totalité des activités nécessaires à la conception d'un avion de l'époque. Si nous retrouvons dans les pionniers de l'aviation quelques personnes capables d'embrasser toutes ces compétences, comme les frères Wright ou Louis Blériot, il devient vite nécessaire de diviser le travail laissant chaque domaine aux mains de spécialistes. Ainsi le premier avion à réaction fut le fruit de la collaboration d'un chercheur en physique et d'ingénieurs en aéronautique (et aucun d'eux ne fut pilote pour les premiers vols qui furent effectués par un pilote d'essai

professionnel). Aujourd'hui, comme pour les montres de Hora, la conception d'un avion consiste en un assemblage de sous-ensembles, aussi bien concrets que conceptuels. D'un point de vue concret, nous avons, entre autres, la voilure (les ailes de l'avion) qui nécessite des connaissances particulières sur la portance et qui a ses propres méthodes de fabrication. Nous avons aussi la structure de l'avion, qui fait notamment appel à la physique des matériaux, ou encore tout le câblage électrique. Par ailleurs, du point de vue conceptuel, la conception d'un avion nécessite, par exemple, des analyses de marché, des analyses de sûreté de fonctionnement ou des études de faisabilité de concepts.

Nous sommes, ici, face à une deuxième division du travail. Celle-ci est guidée par une spécialisation excessivement accrue des compétences. Chaque domaine, chaque sous-ensemble d'un objet complexe, fait appel à des experts et peu de personnes, pour ne pas dire personne, ont une vue complète et exhaustive de l'objet fabriqué dans ses moindres détails. L'augmentation de la complexité induit une séparation du travail qui vient s'ajouter à celle organisationnelle que nous avons évoquée précédemment. Et c'est parce que l'organisation est stratifiée et séparée qu'il est nécessaire de pouvoir vérifier la correction de ce qui est produit non seulement pour chaque strate, mais aussi pour chaque sous-ensemble, pour chaque partie. En raison de cette spécialisation, chaque domaine doit mettre en place des méthodes visant à vérifier si ce qui est produit à son niveau est correct. A cela, s'ajoute, bien évidement des vérifications au moment de l'intégration, c'est-à-dire l'agrégation des sous-parties. Cette problématique de la correction de ce qui est accompli par chaque sous-partie est d'autant plus cruciale que nous sommes, le plus souvent, dans des processus de conception longs. Attendre qu'un bateau soit mis à l'eau pour vérifier si le moteur fonctionne ou le premier essai d'un avion pour vérifier son design serait insensé.

2. Un temps long

La fabrication d'une voiture, d'un bateau, d'un applicatif logiciel ou de la plupart des artefacts techniques s'inscrit dans des temps longs, longs dans leur conception. Par exemple, le constructeur automobile SEAT donne la durée de 1400 jours, soit à peu près quatre ans, pour, à partir des premiers croquis, arriver à la sortie d'usine d'un véhicule. De son côté, Mercedes donne les chiffres de douze mois pour la conception d'une nouvelle version d'un véhicule existant et cinq ans pour un modèle totalement nouveau. Si l'on prend le secteur de l'aéronautique, dans le cas de l'A380, Airbus a commencé les études préliminaires dès la fin de l'année 1995 sur ce qui se nommait à l'époque l'A3XX. Le programme A380 sera officiellement lancé en 2000, pour une présentation du premier avion en 2005. Même dans un secteur dématérialisé, comme celui du logiciel, où il n'y a pas de contrainte de fabrication physique ou de transport, les temps sont relativement longs. Entre l'analyse du besoin, la spécification, l'écriture des premières lignes de

code et les tests finaux plusieurs mois, voire plusieurs années, peuvent s'écouler. S'il est très difficile de trouver des informations quantifiées à ce sujet, nous pouvons citer l'exemple des systèmes d'exploitation Windows 95 et XP qui, tout en n'étant que des évolutions des versions précédentes, ont respectivement nécessité un peu plus de trois ans pour l'un et vingt-un mois pour l'autre [1].

C'est en partie la complexité des produits et la complexité des organisations qui justifient ces temps longs. En effet, comme nous l'avons vu avec l'exemple des montres de Tempus et d'Hora, la solution adoptée pour faire face à la complexité consiste à diviser le travail en parties, sur plusieurs niveaux d'abstraction, puis à assembler ces parties niveau par niveau. Cette approche entraîne de facto des cycles de développement séquentiel pouvant être excessivement longs.

Prenons le cas de la conception d'un avion comme ceux de la famille des A320. Le développement commence par expliciter des exigences de très haut niveau qui sont raffinées en exigences de plus bas niveau, puis traduites en spécifications. De façon schématique, les exigences de haut niveau peuvent être vues comme les caractéristiques globales souhaitées de l'avion et le raffinement consiste à détailler l'exigence en un ensemble de sous-exigences. Dans notre exemple, une exigence de haut niveau pourrait être un grand confort pour les passagers et le raffinement de cette exigence pourrait produire l'exigence de disposer de grands écrans se raffinant elle-même en exigences de puissances électriques pour faire fonctionner ces écrans, d'exigences liées au poids des câbles nécessaires, etc. In fine, les exigences de bas niveau permettront de définir les spécifications des éléments qui composeront l'avion, la description de ce que doit faire chaque composant de l'avion (nous reviendrons plus tard sur les différences entre exigences et spécifications).

Notons que cette spécification étape par étape des caractéristiques de l'avion, ne s'effectue pas d'un seul bloc. Premièrement, étant donné la complexité, le raffinement des exigences se fait par spécialités techniques au sein d'équipes spécialisées. Deuxièmement, chaque étape de raffinement donne lieu à différents designs, qui sont évalués et comparés entre eux. D'un point de vue processus industriel, avant de fixer les premières exigences, des phases d'avant-projet ont lieu afin de définir une spécification générale et un dimensionnement global de l'avion. Ces études consistent en une concertation entre les architectes avion, les bureaux d'études et les entreprises partenaires. Suite à cette étape, vient une phase d'études générales. A partir de l'architecture du nouvel avion, les principaux lots à développer sont définis. Débute ensuite la conception à proprement parler, avec la définition détaillée des constituants majeurs : fuselage, poste de pilotage, ailes, empennages, systèmes, trains d'atterrissage, interface moteur. C'est sur la base de cette conception détaillée de l'avion que commencent les phases de

1. Concernant un logiciel qui ne soit pas une évolution, mais une création de bout en bout, seule l'industrie du jeu vidéo communique et elle affiche, elle aussi, des temps relativement longs. Citons pour exemple la société Ubisoft dont le développement de son jeu Assassin's Creed Origins a pris trois ans et demi.

définition des moyens de production industrielle. Pour finir, concernant la division organisationnelle du travail, la structuration du secteur aéronautique, avec ses différents acteurs, impose une répartition des activités prédéfinie pour chaque famille d'avion, peu modifiable, car contractuelle. Nous voyons bien au travers de cet exemple, que concevoir un avion aujourd'hui ne peut se faire que dans un temps long.

Remarquons que, comme nous venons de le voir, de par la nécessité d'enchaîner des étapes et de par la division hiérarchique due à la complexité, peu importe l'artefact technologique, le fait de rajouter de la ressource n'accélérera pas le processus de conception.

3. Des artefacts corrects

Jusqu'ici, nous avons parlé de *choses*, d'*artefacts techniques* ou d'*objets* : essayons de définir de façon un peu plus précise notre objet d'étude. Comme nous l'avons dit, nous nous intéressons aux objets complexes comme un avion ou un programme informatique, nous sommes donc face à des artefacts produits par l'homme. Distinguer ce qui est produit spécifiquement par l'homme du reste des choses occupait déjà Aristote et ses contemporains. Dans Physique [9], Aristote sépare les choses qui « *existent par le seul fait de la nature* » de celles qui « *sont produites par des causes différentes* ». Ainsi, nous avons d'un côté ce que l'on trouve dans la nature comme les plantes, les animaux, mais aussi les éléments tels que le feu ou la terre, et de l'autre les artefacts produits par fabrication humaine.

C'est dans cette veine que la philosophie contemporaine définit un artefact comme le produit, et non le résultat, d'un ensemble d'actions. L'idée de production renvoie à une intention. Comme le souligne Risto Hilpinen, un objet est un artefact si et seulement s'il a un auteur [1] [103]. En d'autres termes, c'est l'auteur qui, parce qu'il a une intention, va produire l'artefact. Cette intentionnalité est soulignée dans la définition donnée par Hilpinen. Reprenant la suite des réflexions de Donald Davidson [45] sur les actions, Hilpinen définit un artefact comme un objet produit intentionnellement suivant une certaine description [102].

Cette définition exclut tout ce qui n'est pas intentionnel. Par exemple, la construction d'une maison va produire un grand nombre de déchets. Ces déchets, bien que produits par l'action humaine, ne sont pas des artefacts. Notons tout de même que, dans le cas de l'industrie, les objets produits automatiquement par des machines sont bien des artefacts. En effet, les personnes qui contrôlent le fonctionnement des machines ont bien l'intention de produire l'objet [103].

Dans cette définition d'artefact, l'idée qu'un objet est produit suivant une certaine description sous-tend simplement que l'artefact a un *but* voulu par son

1. Notons que cette définition est compatible avec la possibilité pour un objet d'avoir plusieurs auteurs, qu'un artefact soit le produit d'actions collectives.

auteur. Par exemple, une bouilloire sert à faire bouillir de l'eau. Par commodité, nous utilisons ici le terme but au singulier, mais, bien évidemment un artefact peut avoir plusieurs usages et plusieurs buts, comme un téléphone mobile qui est conçu pour permettre, entre autres, de téléphoner, de prendre des photos ou d'écouter de la musique. Cette notion de but voulu entraîne que tout ce qui ne relève pas d'un but a priori ne peut être considéré comme un artefact. Le code informatique résultant de l'appui intentionnel de touches au hasard n'est pas un artefact, même si le résultat venait à donner un programme informatique faisant quelque chose d'utile. En effet, l'intention étant simplement d'appuyer sur des touches, mais aucunement dans le résultat obtenu, il n'y a aucun but a priori dans ce programme informatique. Pour finir, le fait qu'un artefact soit une production, le résultat d'un ensemble d'actions, exclut les objets naturels. Remarquons que ce dernier point soulève un problème en archéologie et en anthropologie où l'usage d'un objet naturel peut être considéré comme un artefact s'il est utilisé comme outil. C'est le cas des coquilles de coquillage qui étaient utilisées sans modification comme outils et comme récipients au Néolitique [95].

Se focalisant plus particulièrement sur l'aspect ingénierie, des travaux ont cherché à caractériser ce qu'ils nomment *artefact technique* [120, 189, 47]. Le terme artefact technique est préféré à simplement artefact pour souligner le but utilitaire, pratique, de ces objets. Les auteurs cherchent ainsi à se restreindre aux domaines techniques et évacuent tout ce qui relève, par exemple, de la création artistique. Partant de la définition de l'artefact, et donc de la nécessité ontologique pour un artefact d'avoir une finalité intentionnelle, ils en déduisent qu'un artefact technique a une nature duale composée d'une partie structurelle et d'une partie fonctionnelle [121, 83]. Ainsi, les propriétés structurelles d'un artefact correspondent à sa description physique, par exemple la taille ou le poids d'un avion, et les propriétés fonctionnelles correspondent à son but, pour un avion : se déplacer dans les airs.

Cependant, nous ne nous intéressons pas seulement ici à des objets physiques, comme un avion ou un ordinateur, mais aussi aux programmes informatiques et aux systèmes, comme le système de politique d'échange d'information dans le cadre de la surveillance de la terre. Dès lors, que devient la nature structurelle, la nature physique, d'un tel artefact que nous qualifierons d'abstrait ? Elle est toujours bien présente, mais elle s'appuie sur une structure abstraite. Par exemple, un programme informatique utilisera des structures de données abstraites fournies par le langage de programmation comme des tableaux,un système d'échange d'information pourra être vu comme un graphe. De tels artefacts techniques ont donc bien une nature duale, avec la partie structurelle qui est une abstraction. Cette nature duale des programmes informatiques n'est pas nouvelle, elle était déjà indiquée en 1978 par James Moor qui, de plus, considérait lui aussi qu'un programme informatique avait nécessairement un but [132] [1].

1. A ce sujet, il est intéressant de noter quand 1958, dans son livre posthume *The computer and the Brain*, John Von Neuman utilise trois fois l'expression « *le problème qui doit être résolu, c-a-d*

Si nous revenons à notre problème principal : la correction d'un artefact technique, la nature fonctionnelle des artefacts techniques devient centrale. En effet, comme le contrôle qualité dans l'industrie manufacturière qui consiste à vérifier si une pièce usinée est conforme à une pièce étalon, la correction d'un artefact technique consiste à évaluer s'il est conforme à ses descriptions structurelle et fonctionnelle. Les propriétés structurelles et fonctionnelles deviennent donc ipso facto l'étalon permettant d'établir la justesse de l'artefact [183]. C'est parce que l'avion a pour fonction de voler, que l'on peut vérifier sa correction par apport à son but, c'est-à-dire qu'il vole. Nous voyons apparaître ici le lien étroit entre la fonction et l'intention humaine, un artefact technique est toujours produit pour un but, son créateur a l'intention de s'en servir pour une certaine raison.

Cette nature fonctionnelle est encore plus prégnante dans le cadre d'un artefact abstrait puisque c'est elle seule qui nous permet de déterminer s'il est correct. Elle fournit les critères d'exactitude et de dysfonctionnement. Pour être plus précis, évaluer la correction d'un artefact abstrait revient à l'évaluer par rapport à sa fonction. C'est seulement en connaissant sa fonction, qu'il est possible de justifier les raisons qui font penser qu'un artefact abstrait est correct.

4. La correction par rapport à la spécification

Dans le monde de l'ingénierie, ce que l'on nomme spécification correspond, en partie, à une expression de la fonction d'un artefact. S'il fallait chercher à définir la spécification, nous pourrions dire que c'est un document qui décrit de façon précise un artefact de telle sorte qu'il serve de guide, de plan, aux concepteurs. Comme le dit Raymond Turner « *les spécifications sont considérées comme étant prescrites avant la construction de l'artefact et guident le constructeur* »[1] [182]. Cette expression peut être faite au moyen de texte, de formalismes mathématiques, de plans ou de diagrammes. Comme elle définit la fonction de l'artefact, la spécification permet bien évidemment de vérifier sa correction, mais elle permet aussi de communiquer. De par la division du travail, les documents de spécification permettent des échanges entre les différents acteurs, comme la sous-traitance, les équipes dédiées à la conception de sous-parties, etc.

De ce point de vue la spécification joue un rôle normatif puisque, non seulement elle exprime la correction d'un artefact, mais elle sert de fondement aux contrats pour définir ce qui est attendu. Ces contrats peuvent prendre une forme légale, comme un avionneur qui fournit à un motoriste la spécification du moteur qu'il désire pour son avion, mais peuvent être aussi simplement moraux, comme

l'intention de l'utilisateur » (*the problem to be solved, i.e. the intention of the user*). Von Neuman insiste ainsi sur le fait que le problème que nous cherchons à faire résoudre à une machine au travers d'un programme informatique, n'est que l'expression de l'intention de l'utilisateur [138].

1. Traduit de : *specifications are taken to be prescribed in advance of the artefact construction and guide the constructor*

une description de ce qui est attendu par un autre département au sein d'une même organisation.

Cet aspect de la spécification n'est pas à négliger. De moins en moins d'entreprises fabriquent seules un artefact technique, elles font appel à de la sous-traitance et des collaborations. De par les liens de plus en plus forts entre les différents acteurs qui participent à la conception, la fabrication et la conception d'un artefact, nous avons vu émerger ce que l'on nomme l'*entreprise étendue*, c'est-à-dire un groupe d'entreprises partageant un objectif commun, des ressources et de la connaissance au travers d'alliances telles que des consortiums [32] [67] [1]. Dans un tel contexte, la spécification devient un artefact crucial puisqu'elle permet d'exprimer ce qui doit être fait, comment le faire et comment vérifier sa correction. Notons que cet usage de la spécification est aussi répandu au sein d'une même entreprise, pour communiquer entre les différentes équipes et les différents départements.

Comme nous l'avons vu, les artefacts techniques auxquels nous nous intéressons sont des objets complexes dont la conception nécessite des temps relativement longs. Par conséquent, le développement de tels artefacts s'inscrit dans un cycle itératif où la spécification de l'artefact va se raffiner dans une série de couches d'abstraction jusqu'à arriver à la fabrication d'éléments physiques [2].

Prenons l'exemple d'un avion. Dans son expression la plus haute, la spécification exprime les grandes fonctionnalités de l'avion, comme sa capacité de fret, de transport de passagers, s'il doit réaliser des trajets courts ou intercontinentaux. Nous sommes typiquement dans ce que l'on nomme l'expression du besoin : l'ingénierie des exigences. La spécification exprime, ici, les fonctionnalités de l'artefact à haut niveau. Dans le cadre d'un logiciel, la spécification à ce niveau exprime ce que *désirent* les futurs utilisateurs. Cette spécification va donner lieu à de premières opérations de conception qui vont permettre un découpage en sous-parties. L'avion va être pensé en grands ensembles, comme les ailes ou la pointe avant, et le logiciel en module, un module pouvant être l'interface graphique ou une base de données. Chaque élément, chaque module est ainsi décrit fonctionnellement. Il dispose donc de sa propre spécification pour permettre d'établir sa correction. En effet, la somme étant supérieure à ses parties, il sera nécessaire de vérifier la correction du module au moment de l'intégration de ses sous-parties. Par ailleurs, la spécification va permettre de communiquer avec ceux qui vont réaliser le module. En effet, de par la spécialisation technique et la division du travail, chaque élément peut être réalisé par un département différent voire par de la sous-traitance.

Nous devons souligner le fait que chacun de ces modules est lui-même un artefact technique qui dispose donc d'une nature fonctionnelle donnée par la

1. L'entreprise étendue répond aussi au besoin de partage des risques financiers et une stratégie d'implantation dans un marché mondialisé.

2. Rappelons qu'ici, par physique, nous entendons bien évidemment des éléments physiquement tangibles, mais aussi du code informatique.

spécification. Le train d'atterrissage d'un avion ou le traitement de texte d'une suite bureautique, sont eux-mêmes des artefacts techniques appartenant à un artefact plus complexe. Ne nous y trompons pas, même si nous donnons ici des exemples assez concrets, ces artefacts peuvent être excessivement abstraits. Ainsi, un processus de conception se compose en une série d'étapes qui introduisent de nouveaux artefacts intermédiaires, avec leurs spécifications, jusqu'à arriver aux éléments les plus simples. Aucun processus de conception d'un artefact complexe n'échappe à cela. Même dans le cadre d'un développement informatique où le logiciel consiste, dans un premier temps, en quelque chose de simple, auquel vient s'ajouter dans le temps de nouvelles fonctionnalités, nous avons la production d'artefacts intermédiaires. Chaque itération, chaque nouvelle version avec ses fonctionnalités est bien un artefact en soi.

Reprenons l'exemple simple de la bouilloire. Elle pourrait être décomposée en trois artefacts intermédiaires dont les fonctions seraient chauffer l'eau à la température choisie, fournir de la puissance thermique et contenir le liquide. L'artefact qui a pour fonction de chauffer l'eau est assez abstrait, puisque l'on ne précise ni comment faire chauffer l'eau, ni comment maintenir la température. Au niveau d'abstraction suivant, il pourra être décomposé en artefacts devant mesurer la température, contrôler la puissance et chauffer l'eau.

C'est parce que nous nous retrouvons avec un ensemble d'artefacts intermédiaires que, de la même manière que pour l'industrie manufacturière, il était nécessaire d'établir la correction de chaque élément produit, il est crucial pour l'ingénierie de vérifier la correction de chaque artefact produit. L'ensemble des opérations visant à établir cette correction sont appelées sous le terme générique d'opérations de Vérification et de Validation (V&V).

5. Vérification et Validation

Plus une erreur est détectée tard dans le processus de conception, plus le coût engendré pour sa correction est élevé. Un exemple fréquemment donné pour illustrer cela est celui de la sonde spatiale Mariner 1 de la NASA. Cette sonde, lancée en 1962, avait pour mission le survol de la planète Vénus. La sonde a raté son lancement et a changé de trajectoire suite à une série de pannes et, surtout, suite à une erreur dans le code du logiciel de guidage. Pour des raisons de sécurité, la NASA a décidé de détruire la sonde en vol, dès la détection de la défaillance des commandes de guidage. Évidemment, le coût de la correction, ici, est extrême, puisqu'il s'agit de la destruction pure et simple de la sonde. Plus récemment, en 2016, la société coréenne Samsung a lancé un nouveau téléphone le Galaxy note 7. Après la mise en vente de celui-ci, suite à de nombreux incidents, un défaut de conception a été détecté (il s'agissait de problèmes au niveau de l'interface entre le téléphone et ses batteries). Ce défaut de conception, compris tardivement, a entraîné l'arrêt des ventes et la nécessité de refaire la conception des points

défectueux ainsi qu'une interdiction de ce téléphone dans les vols commerciaux. Aux coûts engendrés par l'étude et la conception d'un nouveau design pour le logement des batteries sont venus s'ajouter ceux liés au reconditionnement des anciens modèles et à la communication de crise. Pour finir, citons l'exemple des jets régionaux MRJ90 du constructeur japonais Mitsubishi, dont le programme a été lancé en 2007. Suite aux premiers essais en vol réalisés en 2015, ainsi qu'à de nombreux autres depuis, des erreurs de conception sont apparues obligeant l'avionneur à refaire le design du train d'atterrissage, de certains systèmes, de la baie avionique et de la configuration électrique. Si la société Mitsubishi ne communique pas sur les coûts engendrés par ces problèmes de conception, il est tout de même clair que s'ils avaient été détectés dans les phases de conception, et non dans les phases de tests, les coûts liés aux essais en vol, aux études pour trouver et comprendre les problèmes et à la fabrication de nouveaux prototypes auraient été largement moindres. Aujourd'hui, un retard de deux ans sur la date de mise en service est annoncé, depuis le lancement du programme, le calendrier a déjà été repoussé cinq fois et l'avion n'est toujours pas certifié.

S'il est difficile d'estimer le gain de la détection d'une erreur au plus tôt, ces exemples nous montrent qu'il est loin d'être négligeable. Dans le monde du logiciel, dans une méta-analyse de 2004 portant sur de nombreuses études menées sur le sujet, les auteurs indiquent des facteurs d'augmentation des coûts deux fois plus importants entre des erreurs détectées à la conception par rapport à celles détectées au moment du codage et dix fois plus importants entre une détection à la conception et une détection dans les phases de tests [175]. Dans le même ordre d'idée, IBM donne à peu près les mêmes chiffres dans un livre blanc de 2008 [1].

Comme nous le voyons, tout le problème consiste donc à détecter les problèmes au plus tôt dans le processus de développement, en d'autres termes, à établir la correction des artefacts intermédiaires. Dans l'exemple du MRJ90, le jet de Mitsubishi, les artefacts intermédiaires correspondent notamment aux documents et aux différents modèles utilisés lors de la spécification et de la conception de l'avion. C'est dans ce but que l'ingénierie a développé un ensemble d'opérations que l'on retrouve regroupées sous le vocable d'opérations de Vérification et Validation (V&V).

L'origine de l'acronyme V&V est fortement attaché à l'histoire de l'informatique et de la simulation numérique. Dans leur article retraçant l'histoire de la V&V des modèles de simulation [166], Robert Sargent et Osman Balci dressent un panorama de la discipline qui débute dans les années cinquante. Ceci n'est pas très surprenant si l'on considère que la simulation numérique est née en 1944 à Los Alamos dans le cadre de programmes militaires. Dans cet article, les auteurs montrent comment la V&V est passée, dans le courant des années 1970 à 80,

1. Notons tout de même que ce livre blanc a pour vocation de promouvoir les outils IBM de détection de bugs logiciels : Minimizing code defects to improve software quality and lower development costs (white paper) [31].

d'un sujet à la périphérie de la simulation à un domaine d'intense réflexion. Le premier article[1] qui donne une définition assez complète, et conforme à l'idée actuelle de ce qu'est la V&V, est celui de Fishman et Philip de 1968 [80]. Pour les auteurs, la vérification détermine si un modèle se comporte comme le suppose un expérimentateur et la validation teste si un modèle se rapproche raisonnablement du système réel qu'il doit simuler. Dans son étude approfondie sur la V&V, Patrick Roache montre comment la V&V est reliée aux origines de la simulation numérique [161]. Il y souligne le fait que les mots vérifier, valider et confirmer sont souvent utilisés sans réelle distinction dans le cadre de ce qui se nomme l'assurance qualité. De plus, même dans le cadre de l'ingénierie, ces termes sont polysémiques. L'auteur propose tout de même de reprendre les définitions données par Barry Boehm et Frederick Blottner : la vérification consiste à « *résoudre correctement les équations* » alors que la validation est vue comme « *s'assurer que l'on a posé les bonnes équations* ». Nous pouvons paraphraser cela par : la vérification consiste à construire de la bonne façon et la validation consiste à construire la bonne chose.

En simulation, la V&V est devenue un élément crucial. C'est le moyen qui permet d'établir une confiance, une crédibilité, dans les résultats de simulation, résultats de plus en plus utilisés dans des prises de décisions stratégiques. En effet, la modélisation et la simulation se sont imposées dans des domaines critiques où des tests sont impossibles comme, par exemple, la sécurité des centrales nucléaires ou le test de scénario de réchauffement climatique de la Terre. Il n'est pas étonnant que le premier guide de V&V, paru en 1987, émane de l'industrie de l'atome[2]. Par ailleurs, dans une logique de réduction des coûts et des risques, de nombreux industriels se sont mis à utiliser la simulation pour réaliser du prototypage et des tests virtuels. Le prototypage virtuel consiste à tester des ébauches d'un produit, seulement au moyen de simulations numériques, sans maquette physique. Dans le cadre aéronautique, cela permet de comparer différentes architectures d'avions suivant des critères tels que le poids, le bruit ou la consommation, seulement à l'aide d'un ordinateur. Les tests virtuels, eux, ont vocation à réduire au maximum les tests sur l'objet réel. En effet, ces tests sont excessivement coûteux et ils n'interviennent que tardivement dans le processus de conception ce qui, comme nous l'avons vu, entraîne des coûts plus importants en cas de détection de problèmes. Les tests virtuels ont donc l'énorme avantage de porter sur des artefacts strictement abstraits et d'intervenir plus en amont dans le processus de conception que les tests classiques.

Si l'acronyme V&V regroupe bien deux choses, la vérification et la validation, il n'existe cependant pas de définition consensuelle de la V&V [140]. Personne ne

1. Il est toujours difficile et périlleux de fixer l'origine de quelque chose. La pérennité de l'article cité ici s'appuie sur le fait qu'il est cité comme tel dans l'ensemble de la littérature que nous avons consultée.

2. Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry, American Nuclear Society, 1987.

semble précisément s'accorder sur la différence entre vérification et validation et chaque domaine dispose de sa propre définition.

Dans le domaine du logiciel, la différence entre vérification et validation se concentre sur les artefacts intermédiaires et le logiciel final. Ainsi, dans son article de 1984, Barry Boehm définit la vérification comme « *établir la véracité de la correspondance entre un produit logiciel et ses spécifications* »[1] et la validation comme « *établir l'adéquation d'un produit logiciel par rapport à sa mission opérationnelle* »[2] [26]. Les opérations de vérification portent donc, ici, sur les artefacts intermédiaires, que cela soit le programme, mais aussi les documentations. La vérification cherche à établir la conformité d'un artefact avec sa spécification définie plus en amont dans le cycle de développement. Les opérations de validation portent, elles, sur les exigences du logiciel (ses buts définis par sa fonction en tant qu'artefact) et cherchent à établir si le logiciel, ou un composant du logiciel, est conforme avec ces exigences, qualifiées parfois d'exigences de haut niveau.

De façon schématique, en reprenant la métaphore des montres de Hora, si l'on considère l'artefact correspondant à l'ensemble des engrenages permettant de faire bouger l'aiguille des minutes, la vérification consistera à établir que cet ensemble est bien conforme à sa spécification, c'est-à-dire qu'il y a bien le bon nombre d'engrenages, qu'ils sont faits avec le bon métal, que les dents des engrenages s'emboîtent bien, etc. La validation, pour sa part, consistera à établir que cet ensemble contribue bien à l'exigence de haut niveau de donner l'heure, notamment en validant que ce sous-ensemble s'interface correctement avec les autres sous-ensembles (par exemple, il permet bien de faire bouger l'aiguille des minutes).

Parfois, en informatique, de manière pragmatique, les opérations de validation se focalisent sur la spécification, cherchant à établir qu'elle est conforme aux attentes des utilisateurs (construire la bonne chose). Cette séparation avec, d'un côté, la vérification qui consiste à établir la correction par rapport à une spécification et, d'un autre côté, la validation qui consiste à établir la correction de la spécification par apport aux fonctions de l'artefact est assez communément admise dans le monde spécifique du *logiciel critique* et se retrouve de façon plus ou moins explicite dans les normes (comme la DO178[3]).

Hors du logiciel critique, la dichotomie entre vérification et validation n'est pas toujours claire, elle est même source de débat dans l'univers de la simulation. Comme l'explique Patrick Roache, il est communément admis que la vérification

1. Traduit de : *to establish the truth of the correspondence between a software product and its specification.*

2. Traduit de : *to establish the fitness or worth of a software product for its operational mission.*

3. La norme DO-178 *Software considerations in airborne systems and equipment certification* spécifique aux logiciels critiques dans le monde de l'avionique définit la vérification comme « *l'évaluation des produits d'un processus pour assurer leur correction et leur consistance par rapport aux entrées fournies et aux standards s'appliquant à ce processus.* », et la validation comme « *déterminer que les exigences sont correctes et complètes* ».

concerne une activité mathématique alors que la validation concerne des aspects science de l'ingénieur. Les opérations de vérification vont donc se concentrer sur la façon dont les équations qui interviennent dans la simulation sont résolues, est-ce que l'aspect mathématique est juste peu importe le phénomène simulé. La validation, de son côté, va se focaliser sur le choix des équations utilisées, représentent-elles bien ce que l'on cherche à simuler. Cette séparation présuppose qu'il est possible d'effectuer une vérification sans utiliser les résultats d'une simulation, puisque l'on se concentre uniquement sur la correction des méthodes de calculs, et qu'il est possible d'évaluer de la pertinence d'un modèle sans effectuer de vérification. Eric Winsberg soutient que cette image est beaucoup trop simpliste [198, 197]. Dans la pratique, il est souvent impossible d'effectuer la validation ou la vérification indépendamment l'une de l'autre. Il peut y avoir plusieurs explications à cela. Par exemple, les modèles qui sous-tendent la simulation peuvent être si complexes qu'il est tout simplement impossible d'offrir des arguments purement mathématiques pour déterminer les résultats attendus. Allant plus loin dans le raisonnement, certains auteurs réfutent le terme même de validation, considérant les opérations de validation comme impossibles [142]. Pour eux, il est juste possible de confirmer un modèle au vu de données expérimentales, mais en aucun cas de le valider.

En ce sens, que cela soit en simulation numérique comme en informatique, l'acronyme V&V, qui est massivement utilisé dans l'ensemble des normes et des standards, est assez révélateur de la difficulté à différencier ces deux activités puisqu'il englobe dans un tout, non discernable, les mots vérification et validation. Cette difficulté apparaît aussi dans les définitions données dans les différentes normes. Ainsi, dans les années 1990, pour l'ensemble de leurs standards, l'Institute of Electrical and Electronics Engineers (IEEE) [107] définissait la vérification comme établir que « *les exigences spécifiées ont été satisfaites* »[1] et la validation comme établir que « *des exigences particulières pour un usage spécifique prévu sont satisfaites* »[2]. Ces définitions ont dû laisser quelques lecteurs dubitatifs. Depuis, les définitions se sont clarifiées, mais aussi complexifiées, le glossaire de 2017 de l'IEEE réalisé en commun avec l'International Organization for Standardization (ISO) fournissant quatre définitions pour la vérification et cinq définitions pour la validation [112].

En définitive, ce que nous devons retenir ici, est : premièrement, qu'il est nécessaire de disposer de la capacité d'établir si un artefact est conforme à sa fonction, peu importe qu'il soit intermédiaire ou abstrait et deuxièmement, que cette capacité doit permettre d'identifier les problèmes de conception au plus tôt.

1. Traduit de : *specified requirements have been fulfilled*
2. Traduit de : *the particular requirements for a specific intended use are fulfilled*

6. Accréditation et certification

Historiquement, dans le contexte de la simulation, une nouvelle notion est venue compléter la V&V : l'accréditation. En effet, les activités de V&V ne sont pas toujours cantonnées au processus de conception, elles peuvent aussi jouer un grand rôle dans les activités d'acceptation d'un artefact technique par un client ou de certification par une autorité. L'une des premières définitions de l'accréditation a été donnée en 1994 par le Département de la Défense américain, pour qui l'accréditation consiste en « *la certification officielle qu'un modèle ou une simulation est acceptable pour une utilisation dans un but spécifique* »[1] [62]. Il devient donc nécessaire de recueillir et d'évaluer tous les éléments de preuve qui permettent de considérer qu'une simulation, ou un modèle, est utilisable pour un emploi donné. Les activités d'accréditation complètent les activités de V&V et impliquent une autorité qui certifie. Notons que cette autorité doit être indépendante et impartiale.

L'ajout des activités d'accréditation a donné naissance à la Verification, Validation & Accreditation (VV&A) [14] qui, si elle trouve son origine dans le domaine de la défense, s'est peu à peu répandue dans les entreprises pour accréditer en interne des simulations particulièrement complexes[2]. La nécessité de produire un ensemble de justifications pour convaincre de la validité d'une activité s'étend maintenant à des domaines tels que la gestion des risques et les décisions stratégiques. Un exemple possible d'utilisation de la VV&A en entreprise peut être le passage de jalon d'un projet ou d'un niveau de Technology Readiness Level[3] (TRL).

Les activités d'accréditation consistent donc, pour celui qui conçoit l'artefact, à collecter une documentation exhaustive, expliquant non seulement les résultats, mais aussi les données d'entrée, les hypothèses faites, les techniques appliquées, etc. Pour l'autorité qui accrédite l'artefact, l'activité d'accréditation consiste à évaluer l'ensemble de cette documentation. Nous pouvons, ici, opérer un rapprochement entre l'accréditation et la certification des systèmes critiques, une définition simple des systèmes critiques étant que le système a le potentiel de mettre en danger la vie d'une personne. La tâche nommée certification pour un système critique consiste à l'évaluation d'un système par une autorité de certification en vue de donner l'autorisation de son exploitation. Des exemples de telles autorités sont : pour l'évaluation du médicament, l'*Agence européenne des médicaments* (EMA) et la *Food and Drug Administration* (FDA) ou pour la sécurité de l'aviation civile, l'*Agence Européenne de la Sécurité Aérienne* (EASA) et la *Federal Aviation Administration* (FAA).

1. Traduit de : *The official certification that a model or simulation is acceptable for use for a specific purpose.*

2. Nous pouvons citer le projet européen TOICA auxquels ont participé de nombreux acteurs du secteur aéronautique et que nous verrons plus en détail dans la Section 3.2.

3. Le TRL est une mesure de maturité technologique. Historiquement employée par les agences gouvernementales américaines, elle est aujourd'hui largement utilisée dans le milieu industriel.

Dans le cadre de la certification des systèmes critiques, pour convaincre l'autorité de certification, il est nécessaire de fournir tous les éléments concernant la conception du système et les opérations de V&V qui ont été effectuées. L'ensemble de ces éléments est généralement appelé un *assurance case* [1]. Un assurance case peut être défini comme « *une argumentation organisée selon laquelle un système est acceptable pour l'usage auquel il est destiné en ce qui concerne des préoccupations spécifiques (telles que la sécurité, la sûreté, la sécurité, l'exactitude)* » [2] [160]. Dans son étude sur les systèmes logiciels fiables et certifiables, le National Research Council souligne que « *des éléments de preuve concrets doivent être présentés pour étayer l'allégation de fiabilité. Ces éléments de preuve prendront la forme d'un* « dossier de fiabilité », *faisant valoir que les propriétés requises découlent de la combinaison des propriétés du système lui-même (c'est-à-dire la mise en œuvre) et des hypothèses environnementales* » [3] [136]. Contrairement à la V&V, la conformité ne concerne donc pas l'artefact technologique à proprement parler, mais les actions réalisées pour le concevoir et le développer.

Finalement, que l'on parle de certification ou d'accréditation, tout le problème consiste, pour les concepteurs, à bien argumenter et, pour l'autorité de certification, à évaluer une argumentation. Nous sommes bien, ici, dans un cadre argumentatif, il faut convaincre l'autorité. Le système doit fournir un service auquel on peut légitimement faire confiance, la confiance s'établissant au travers de liens plausibles entre les éléments de preuves fournis et le fait que le système rend le service attendu [13].

Afin d'organiser l'ensemble des éléments de preuve que sont, par exemple, les techniques utilisées dans le processus de conception, les tests, les résultats de simulation, les données d'entrée ou les hypothèses, de nombreux travaux cherchent à structurer et à donner une cohérence aux assurance cases. Du coté académique, nous pouvons citer *Safety Argument Manager* [129], *Goal Structuring Notation* [117], *ASCAD notation* [70], *Justification Diagram* [147] et *Structured Assurance Case Meta-model* [141], la norme *ISO 15026* [111], *Generic Methodology for Verification and Validation* [163] du côté des organismes de standardisation.

Remarquons tout de même que du côté de l'autorité d'accréditation ou de certification, il n'est pas forcément nécessaire d'être un expert dans tous les domaines couverts par les opérations de V&V pour pouvoir certifier un artefact. Prenons le cas d'un ingénieur qui veut savoir si la conjecture de Fermat est un théorème ou

1. Il n'est pas aisé de traduire le terme assurance case. Nous pourrions tenter : « *dossier de certification* », mais cette traduction n'est pas convaincante. Notons que suivant les domaines le mot assurance case peut être remplacer par un autre idiome, comme, par exemple, « *safety case* » généralement employé dans le cadre de la sûreté de fonctionnement.

2. Traduit de : *an organized argument that a system is acceptable for its intended use with respect to specified concerns (such as safety, security, correctness).*

3. Traduit de : *concrete evidence must be present that substantiates the dependability claim. This evidence will take the form of a* « dependability case, » *arguing that the required properties follow from the combination of the properties of the system itself (that is, the implementation) and the environmental assumptions*

non (ce qui revient à savoir s'il existe une démonstration mathématique de cette conjecture). Il se trouve qu'Andrew Wiles a donné une démonstration de cette conjecture. Comme notre ingénieur n'est pas un expert en théorie des nombres, il n'en comprend pas la preuve. Est-ce à dire que notre ingénieur n'a pas confiance en la preuve de Wiles ? Bien sûr que non. Il peut avoir confiance, car suffisamment de chercheurs ont validé la preuve et qu'il existe en science un système de relecture par les pairs. Par conséquent, il semble raisonnable de considérer que la preuve d'Andrew Wiles est correcte. De la même manière, pour avoir confiance dans un artefact technique, il est nécessaire d'avoir confiance dans les personnes impliquées dans le processus de conception et de disposer de tous les éléments clés qui étayent la forte plausibilité que l'artefact soit correct.

Pour finir, nous pouvons dresser un parallèle entre le droit et la certification. Un des principes de droit, hérité du droit romain [1], est que la charge de la preuve appartient au demandeur. En d'autres termes, c'est à celui qui avance un fait de le prouver. Ce principe est résumé par les deux adages juridiques latins *actori incumbit probatio*, qui signifie que la preuve incombe au demandeur, et *necessitas probandi incumbit ei qui agit* : la preuve incombe à celui qui agit [101]. Dans le cadre de la certification, la même logique s'applique. C'est à celui qui demande une certification, aux concepteurs de l'artefact, qu'il incombe de fournir une argumentation étayée. Cette position est excessivement rationnelle. Effectivement considérer que cela soit à l'autorité de prouver qu'un système n'est pas certifiable consisterait à opérer un renversement de la charge de la preuve, ce qui correspondrait à un sophisme, à un raisonnement fallacieux, et plus particulièrement à l'appel à l'ignorance. Le sophisme d'appel à l'ignorance, nommé *argumentum ad ignorantiam*, consiste à considérer que quelque chose est vrai sur la base qu'il n'a pas été démontré quelle est fausse [192].

1. Que l'on retrouve notamment dans le « *Copus de droit civil* » (Corpus iuris) qui est une vaste compilation de droit romain élaborée sur l'ordre de l'empereur romain d'Orient Justinien (VI^{e} siècle).

Chapitre 2

Des méthodes formelles pour prouver la correction

« *On the other side of the screen, it all looks so easy.* »

Kevin Flynn, Tron

1. Des démonstrations faites par des machines

Que l'on nomme cela vérification, validation ou opérations de V&V, établir la correction d'un artefact technique est une activité cruciale. Dans la pratique, cette correction est généralement établie à l'aide d'inspections et de tests. L'inspection, ou la revue, consiste à s'appuyer sur un humain qui, dans le cadre de documents ou de code informatique, va relire très attentivement ce qui a été écrit [72], dans le cadre d'une simulation, va observer finement les résultats donnés et, dans le cadre d'un objet physique, va observer à l'aide de son expertise l'absence de défaut d'un objet. Les tests, pour leur part, consistent à appliquer une opération sur un artefact et à vérifier que le résultat de cette opération est conforme à un résultat attendu et connu à l'avance. On ne teste pas vraiment la fonction de l'artefact, mais plutôt la fonction dans des cas particuliers.

Ces deux méthodes ont le défaut de ne pas être complètement fiables. Un humain, malgré toute l'attention qu'il peut porter à sa tâche, laissera toujours passer des défauts, quant aux tests, ils sont rarement suffisants. En effet, concernant les tests, chacun d'eux vérifie seulement si un artefact est correct pour un cas précis unique. Afin de réellement établir la correction d'un artefact, il faudrait être

capable d'être complètement exhaustif, c'est-à-dire tester tous les cas possibles, et donc connaître à l'avance toutes les réponses attendues dans toutes les configurations. Sur un artefact technique complexe, cette exhaustivité est totalement irréaliste, et ce, pour trois raisons. Premièrement, dans certains cas l'ensemble des tests est potentiellement indénombrable. Prenons l'exemple d'une machine qui multiplie par deux un nombre, l'ensemble des nombres étant infini, il serait impossible de tous les tester. Deuxièmement, il est très souvent matériellement impossible de réaliser tous les tests pour des raisons de coûts et de temps. Prenons le cas d'un avion, les essais en vols sont excessivement coûteux et il faudrait pouvoir tester un avion à toutes les températures, toutes les vitesses, toutes les altitudes et toutes les positions possibles ! Dans le cas d'un logiciel, tester toutes les réponses du programme pour toutes les entrées possibles prendrait un temps beaucoup trop long. Troisièmement, pour réaliser un test, il faut absolument connaître le résultat attendu à l'avance. Or, il n'est pas forcément possible d'imaginer tous les cas de figure qui se présenteront. Dans le cas d'un avion, cela signifierait imaginer tous les événements dus à l'environnement extérieur, sans en oublier aucun. Dans certains cas, les résultats attendus sont tout simplement inconnus et le but de l'artefact est précisément de les trouver. Prenons le cas de simulations comme celles sur l'évolution du climat de la Terre ou sur la prévision des éruptions solaires, le but de ces simulations est justement de connaître ces résultats, il est donc impossible de les déterminer à l'avance pour réaliser des tests.

La limitation des tests est très vite apparue dans le domaine de l'informatique. En 1972, Edsger Dijkstra remarque que si les tests sont utiles pour trouver des bugs, ils sont insuffisant pour démontrer leur absence [60]. Il reviendra sur ce point dans un article de 1974, précisant que « *tester peut être utilisé de manière très convaincante pour montrer la présence de bugs, mais jamais pour démontrer leur absence, car le nombre de cas que l'on peut réellement essayer est absolument négligeable par rapport au nombre de cas possibles* » [1] [61]. Dès lors, Dijkstra considère que le moyen d'établir la correction d'un programme passe nécessairement par une preuve, par une démonstration mathématique. Il est intéressant de noter que Dijkstra s'interrogeait sur la correction des programmes informatiques à l'époque de ce qui se nommait la crise du logiciel. En 1968, lors d'un congrès sur le logiciel [137], de nombreux experts ont considéré que le monde de l'informatique traversait une véritable crise avec des logiciels inefficaces, de trop pauvre qualité, ne satisfaisant pas les demandes des utilisateurs, avec du code difficilement maintenable, etc. Quelques décennies plus tard, force est de constater que la crise n'est pas vraiment terminée ...

Cette idée d'établir des preuves mathématiques est présente dès les origines de l'informatique [25]. Ainsi certains des fondateurs de la discipline, comme

1. Traduit de : *testing can be used very convincingly to show the presence of bugs, but never to demonstrate their absence, because the number of cases one can actually try is absolutely negligible compared with the possible number of cases.*

Alan Turing [1], James Wilkinson ou John von Neumann, travaillaient sur des démonstrations concernant les erreurs d'arrondis faites par les machines (la différence entre la valeur calculée par un ordinateur et la valeur mathématique exacte). Un peu plus tard, en 1967, Robert Floyd, dans son article séminal, présente un cadre formel permettant de prouver mathématiquement des propriétés sur des programmes informatiques (à l'aide d'une représentation sous forme de flowchart) [82]. Dans la lignée de ces travaux, Tony Hoare propose en 1969 un système axiomatique permettant de démontrer des propriétés sur du code [104]. Ces travaux posent un jalon dans ce qui va devenir les *méthodes formelles*, puisqu'à partir de là, il devient envisageable de donner une preuve mathématique pour établir qu'un programme est correct par rapport à sa spécification.

En informatique, les méthodes formelles sont une discipline consistant à utiliser des cadres mathématiques pour montrer la correction de programmes, de spécification de logiciel et parfois même du matériel (hardware). Nous allons ici étendre cette définition et considérer les méthodes formelles dans un sens beaucoup plus large. Pour nous, les méthodes formelles sont des techniques permettant d'établir la preuve, au sens démonstration mathématique, qu'un artefact est conforme à sa fonction ou, pour être plus précis à l'expression de sa fonction [2]. Peu importe que l'artefact soit un logiciel, un avion ou un artefact intermédiaire. Notons ici deux difficultés majeures auxquelles n'échappe pas non plus le domaine de la preuve sur des programmes informatiques. Premièrement, la complexité des artefacts rend difficile le passage à l'échelle des méthodes formelles, leur application sur des cas réels. Deuxièmement, il n'est pas simple d'exprimer la fonction d'un artefact, autrement dit sa spécification, dans un cadre mathématique.

A cette définition générique des méthodes formelles, nous rajoutons une restriction : les méthodes mathématiques employées doivent être automatisables, nous voulons que la correction puisse être réalisée de façon automatique par un ordinateur. Nous sommes clairement ici dans une des différences entre mathématique et informatique où, comme le disait Dijkstra « *Un programmeur conçoit des algorithmes, destinés à une exécution mécanique, destinés à contrôler des équipements informatiques existants ou concevables* » [3] [61].

L'histoire des méthodes formelles est intimement liée à celle de l'*Intelligence Artificielle* (IA). Le terme IA est né en 1955 [4] à l'initiative de John McCarthy [5],

1. Dès 1949, Alan Turing envisageait à la conférence sur *High Speed Automatic Calculating Machines* une méthode de preuve de programme à l'aide d'assertions [133].

2. La fonction d'un artefact reste quelque chose d'assez abstrait, présent dans l'esprit des concepteurs et/ou des utilisateurs. Ici, ce qui est réellement vérifié c'est la correction par rapport à l'expression de cette fonction, sa traduction, en langage mathématique.

3. Traduit de : *A programmer designs algorithms, intended for mechanical execution, intended to control existing or conceivable computer equipment*

4. voir : A proposal for the Dartmouth summer research projet on Artificial Intelligence, 31 août 1955.

5. John McCarthy dont les travaux sur une théorie mathématique pour l'informatique serviront

Marvin Minsky, Nathaniel Rochester et Claude Shannon lors de l'organisation d'un atelier de travail. Cet atelier, qui a eu lieu durant tout l'été 56, a notamment porté sur la question de comment une machine pouvait résoudre des problèmes réservés, pour le moment, aux humains. Lors de cet atelier, Allen Newell et Herbert A. Simon ont présenté le *Logic Theorist*, un logiciel qu'ils ont programmé avec Cliff Shaw [96]. Newell et Simon avaient conçu un logiciel capable de manipuler des expressions symboliques et, plus précisément, des expressions logiques. Ainsi, en exprimant des théorèmes mathématiques en langage logique, le *Logic Theorist* pouvait trouver tout seul des preuves à ces théorèmes [1] [139, 176].

La capacité de raisonner logiquement a été au cœur de la recherche en IA. Cette capacité pour une machine à raisonner prend généralement la forme de programmes informatiques qui sont capables de dire si une formule mathématique, plus précisément une formule de logique, est toujours vraie ou pas. C'est précisément ici qu'a lieu le point de jonction avec les méthodes formelles qui cherchent à établir la véracité d'une propriété représentant la correction d'un artefact. En modélisant l'artefact et sa fonction sous la forme d'une formule logique, il est possible, pour un ordinateur, de vérifier si cette formule est toujours vraie : de prouver la correction de l'artefact par rapport à sa fonction. De plus, si la formule n'est pas vraie, la machine est généralement capable de donner un contre-exemple, c'est-à-dire un cas de dysfonctionnement de l'artefact au sens large. En d'autres termes, la machine renvoie un exemple dans lequel la formule est fausse, et cet exemple peut être l'expression d'un défaut de l'artefact, d'un cas d'utilisation possible non considéré, d'une erreur dans l'expression de la propriété, etc.

2. Etablir formellement la correction d'un artefact

Il est tout à fait possible pour un programme informatique de trouver une démonstration mathématique. Ce champ de recherche se situe entre la logique, le raisonnement automatique (qui est une branche de l'IA) et la vérification de programmes. Il existe de nombreuses techniques pour établir si une formule, une propriété, est vraie et chacune de ces techniques a ses limitations [2]. Certaines ne garantissent pas de trouver une preuve, même si elle existe, d'autres peuvent ne pas finir [3] et certaines utilisent des langages excessivement simples et peu expressifs. Chacune ayant ses avantages et ses inconvénients, et aucune n'étant

de base à Robert Floyd.

1. Le *Logic Theorist* sera capable de trouver la démonstration de 38 théorèmes, exprimés en calcul propositionnel, tirés du Principia Mathematica (1910-13), ouvrage en trois volumes des philosophes mathématiciens britanniques Bertrand Russell et Alfred North Whitehead.

2. Même si ce n'est pas notre sujet, nous pouvons citer la méthode des tableaux, le superposition calculus, toutes les techniques pour problème de satisfaisabilité booléenne ou les techniques employées pour le satisfiability modulo theories.

3. Ne pas finir signifie que le programme ne s'arrête jamais et qu'il n'est pas possible de savoir si la propriété est vraie ou fausse.

intrinsèquement supérieure aux autres, c'est la forme du problème qui va guider le choix d'une technique plutôt qu'une autre. Par la suite, nous désignerons sous la dénomination générique *solveur*, l'ensemble des programmes cherchant à établir si une propriété est vraie ou fausse, peu importe la technique sous-jacente.

L'utilisation de méthodes formelles, et donc de solveurs, pour établir la correction d'artefacts informatiques, a pris son essor durant les années 1970. Elle a donné lieu en 2007 à l'obtention d'un prix Turing [1] par Edmund M. Clarke, E. Allen Emerson et Joseph Sifakis « *pour leur rôle dans le développement du Model-Checking en une technologie de vérification hautement efficace qui est largement adoptée dans les industries du matériel et du logiciel* » [2].

Concernant la correction d'un artefact, l'usage des méthodes formelles permet d'établir la correction de la définition de l'artefact, mais rarement la correction de l'artefact en tant que tel. Prenons le cas d'un artefact abstrait comme du code informatique, la plupart des preuves de correction vont certes s'appuyer sur le code, mais aussi sur une représentation au moyen de formules mathématiques de la machine et de son modèle d'exécution. C'est donc bien la correction de la représentation du code qui est prouvée et pas le code dans l'absolu. Du point de vue de l'artefact logiciel produit à partir du code, autrement dit le programme exécutable, quand bien même cette représentation de la machine serait l'exacte réalité, cela ne prouve aucunement que le programme s'exécutera correctement. Nous avons tendance à l'oublier, mais, à son exécution, un programme informatique n'est qu'un échange de charges électriques et, au vu de la complexité de ces échanges, il est inenvisageable de faire des preuves de correction de programme à ce niveau là.

Quoi qu'il en soit, les méthodes formelles ont montré leur intérêt pour la preuve de correction d'artefacts tels que des programmes informatiques ou des spécifications électroniques et les techniques sous-jacentes aux outils que sont les solveurs ne cessent de progresser. Dans ce contexte, il semble pertinent d'une part de chercher à pousser l'usage des méthodes formelles partout où cela peut être utile dans l'industrie et d'autre part de trouver de nouveaux domaines où elles peuvent être utilisées. C'est précisément sur ces deux axes que se situent mes travaux de recherche concernant la preuve de correction d'artefacts. Plus précisément, comme nous allons le voir par la suite, je me suis intéressé à l'utilisation de solveurs :

— premièrement, pour une vérification des exigences et une aide à la conception de systèmes d'information et plus précisément de système d'information avec échanges d'informations critiques (projet interne ONERA pour un système international de détection des débris spatiaux) ;

— deuxièmement, pour une aide à la spécification de systèmes exprimés à

1. Prix décerné tous les ans par l'Association for Computing Machinery (ACM).

2. Traduit de : *for their role in developing Model-Checking into a highly effective verification technology that is widely adopted in the hardware and software industries.*

base de modèles pour des systèmes avioniques embarqués (projets dans le cadre de la coopération AIRSYS [1] *Architecture et IngénieRie des SYStèmes* et de l'ITEA3 [2]) ;

— troisièmement, pour une vérification de la conformité d'un code par rapport à sa spécification dans le contexte de la certification avionique, (encadrement de la thèse d'Anthony Fernandes Pires [3]).

3. Méthodes formelles et exigences

Avant de concevoir un artefact technique, il faut tout d'abord en définir la fonction, le but. L'expression de cette fonction se nomme exigence. Dans la pratique, un artefact technique n'a pas une, mais des exigences. Il doit remplir plusieurs objectifs et répondre à diverses contraintes. Généralement, les exigences sont définies comme les fonctionnalités demandées par les futurs utilisateurs ou comme les fonctionnalités que doit posséder un artefact pour satisfaire un ensemble de demandes (telles les demandes des utilisateurs, mais aussi des contraintes contractuelles ou liées à l'application de normes). Si l'on se réfère au glossaire de l'IEEE, une exigence est une « *fonctionnalité, ou une condition, que doit posséder, ou remplir, un système pour satisfaire des documents imposés* » [4] [108]. De son côté, l'organisme de normalisation ISO, dans le cadre de l'ISO 9001, donne une définition très générique des exigences : « *besoin ou attente formulée, généralement implicite ou obligatoire* » [110]. Finalement, la définition la plus claire est peut-être celle donnée par Gerald Kotonya et Ian Sommerville dans leur ouvrage de référence : « *les exigences définissent les services que le système doit fournir et définissent les contraintes qui régissent le fonctionnement du système* » [5] [119].

Comme nous le voyons au travers de ces définitions, les exigences recouvrent les attentes sur la fonction de l'artefact, comme *la bouilloire doit chauffer l'eau*, mais aussi possiblement sur sa structure, *elle doit être en verre*, sur son processus de fabrication, *aucune substance dangereuse ne devra être utilisée* et sur toutes autres attentes comme *la bouilloire devra être conforme à l'ensemble des normes électriques françaises*. Etant donné leur diversité et afin d'y voir plus clair, on structure généralement les exigences en différentes catégories. S'il existe plusieurs découpages possibles, dans le monde logiciel, le plus simple consiste à les classer en deux groupes : les

1. AIRSYS est une coopération entre la société Airbus, l'IRIT (Institut de Recherche en Informatique de Toulouse), le LAAS (Laboratoire d'Analyse et d'Architecture des Systèmes) et l'ONERA.

2. Information Technology for European Advancement.

3. Thèse CIFRE, en partenariat avec la société Atos, soutenue en 2014.

4. Traduit de : *condition or capability that must be met or possessed by a system ... to satisfy an agreement, standard, specification, or other formally imposed documents*

5. Traduit de : *the requirements define the services that the system should provide and they set out constraints on the system's operation.*

exigences fonctionnelles, qui portent sur la fonction de l'artefact, et celles non fonctionnelles, qui portent sur tout le reste. A ce découpage simple, viennent s'ajouter des propositions de classification plus fines, propositions donnant lieu à des normes et des standards [1].

L'histoire de l'ingénierie des exigences est fortement liée à celle de l'informatique. Comme le souligne Barry Boehm, diriger le développement logiciel par les exigences était déjà bien établi dès les années cinquante [27]. Ce lien avec le logiciel entraîne un piège sémantique sur le terme *exigences fonctionnelles*. En fait, par fonctionnel, nous parlons des services que doit rendre un logiciel. Dans le cas d'un artefact avec une réalité physique, les exigences fonctionnelles renvoient bien évidemment à la partie fonctionnelle, mais elles peuvent aussi inclure la partie structurelle eu égard au fait que la structure participe à la réalisation de la fonction de l'artefact.

Contrairement à ce que l'on pourrait penser, les exigences sont rarement fixées de façon ferme et définitive. Elles peuvent évoluer durant un projet. L'une des premières raisons de ces évolutions est tout simplement le changement d'avis des futurs utilisateurs, le fait qu'ils ne désirent plus exactement la même fonctionnalité qu'au début du projet. Bien que très fréquent, ce type de changement, dû à une modification des buts, est redouté par les concepteurs et est à l'origine de nombreux échecs de projets. Une autre raison d'évolution peut être que les fonctionnalités demandées sont irréalisables pour des questions de coûts ou, tout simplement, parce qu'elles sont fonctionnellement impossibles (cette impossibilité n'apparaissant qu'après une étude approfondie).

En dépit de ce côté évolutif, il faut bien comprendre le rôle normatif des exigences : elles sont l'expression d'un contrat portant sur l'artefact et sur sa conception. De par cet aspect normatif, leur expression revêt un enjeu tout particulier et fait l'objet de nombreuses normes et recommandations. Concernant leur forme proprement dite, les exigences sont dans une grande majorité des cas écrites en langage naturel [2] et de nombreux organismes proposent des contraintes de rédaction sur la syntaxe, le style et le lexique à utiliser [3]. Des exemples de contraintes peuvent être d'éviter les termes flous, l'usage des conjonctions de coordination ou l'utilisation de l'anglais « *shall* » (qui traduit un devoir) pour une

1. Par exemple, dans l'aviation civile, l'ARP 4754 propose un classement avec notamment des exigences de sûreté de fonctionnement, des exigences de certification venant directement de la régulation aérienne et des exigences fonctionnelles. Les exigences fonctionnelles se subdivisent à leur tour en exigences opérationnelles, de performance, physique et installation, de maintenance, utilisateur et d'interface [165].

2. Par langage naturel, nous entendons les langues qui se sont forgées à l'usage, dans le temps, sans plan d'ensemble précis. Le français, l'anglais ou le chinois sont des langages naturels, alors que la logique mathématique ou les langages de programmation ne le sont pas.

3. Parmi les exemples possibles, nous avons les guides de l'International Requirements Engineering Board (IREB), l'International Council on Systems Engineering (INCOSE) avec son guide « *Guide for Writing Requirements* », l'ISO avec les normes ISO 9001 ou l'IEEE avec IEEE 830-1998 et ISO/IEC/IEEE 29148 :2011.

exigence qui doit absolument être respectée.

A ces contraintes linguistiques, viennent parfois se rajouter des contraintes de présentation et donc de forme. Par conséquent, les exigences se présentent rarement sous la forme d'un texte libre, mais ressemble plutôt à des listes soumises à des règles d'écritures strictes. Malgré cela, de par l'usage du langage naturel, les exigences sont souvent floues, ambiguës, sujettes à interprétation, voire peu claires. Pour pallier cela, de nombreux travaux ont défini des langages pour l'expression des exigences (comme i^* [201, 44], *KAOS* [187], *Z* [174] ou *B* [2]). Certains de ces langages disposent d'une sémantique formelle, ce qui permet non seulement de lever les ambiguïtés, mais également d'utiliser des méthodes formelles. Mais ne nous leurrons pas, de par la difficulté d'écrire des formules mathématiques, les exigences en langage naturel ont encore de beaux jours devant elles. Cependant, que ce constat ne nous décourage pas à promouvoir et à étudier des cadres formels pour exprimer les exigences. En effet, dans certains cas, de par la criticité des applications, l'usage des méthodes formelles peut s'imposer pour écrire des exigences. Pour notre part, nous nous sommes focalisés sur l'expression des exigences à l'aide de la logique.

3.1. Vérifier des exigences formelles

En partant d'exigences écrites de façon formelle, il peut être intéressant de fournir un ensemble d'analyses automatiques pour une aide à la spécification d'exigences. Ces analyses ont pour but de vérifier que les exigences capturent correctement tous les aspects du problème [1], sans contradiction ou sans redondance afin d'établir la confiance des concepteurs dans leur spécification surtout lorsque l'environnement du système évolue, ou lorsque des exigences sont ajoutées, supprimées ou modifiées. L'aspect automatique de ces analyses est un point clé, il permet d'analyser facilement une spécification, le travail étant dévolu à une machine. Notre but est donc de détecter des erreurs dès l'expression des exigences, car, comme nous l'avons vu, les erreurs détectées au plus tôt réduisent les coûts de développement.

Nos analyses automatiques portent sur deux familles de propriétés : des propriétés propres à chaque système et des propriétés que nous qualifions de *génériques*. Par générique, nous entendons que l'on peut raisonnablement s'attendre à ce que ces propriétés s'appliquent à n'importe quel système et capturent de bonnes pratiques pour la rédaction d'exigences. Nous avons ainsi défini quatre propriétés génériques : la consistance, la complétude, l'applicabilité et la minimalité [54, 53].

La propriété de consistance signifie qu'une chose n'est pas à la fois requise et interdite, en d'autres termes, que les exigences ne se contredisent pas entre elles. Cette définition fait écho à la définition de consistance en logique. En logique, une théorie est consistante si elle ne contient pas de contradiction, s'il n'est pas possible

1. Dans le mesure où il est possible d'exprimer formellement l'environnement de l'artefact.

de démontrer à la fois un énoncé et sa négation. Nous retrouvons cette notion de consistance qui établit qu'une chose ne peut être à la fois obligatoire et interdite dans le contexte des politiques de confidentialité. Ainsi, deux politiques d'accès sont dites consistantes entre elles si, lorsqu'elles sont fusionnées, aucun utilisateur ne peut avoir la permission, selon la première politique, et l'interdiction, selon la seconde, de savoir quelque chose [20]. L'inconsistance est une notion fondamentale, autoriser et interdire une même chose revient à demander l'impossible et renvoie donc à l'impossibilité de concevoir un artefact répondant aux exigences.

Nous définissons la propriété de complétude comme le fait que les exigences couvrent toutes les situations possibles. En d'autres termes, il n'y a pas d'absence d'exigences applicables pour une situation donnée. Cette définition de la complétude est assez standard et ressemble à celle donnée dans le contexte des politiques de contrôle d'accès [3, 59]. L'incomplétude revient à laisser aux concepteurs le soin de définir une partie des fonctionnalités de l'artefact. Cette liberté peut entraîner in fine la réalisation d'un artefact ne correspondant pas complètement aux besoins des utilisateurs puisque justement les exigences étaient incomplètes. Néanmoins, dans les phases préliminaires de conception d'un système, il peut être utile d'avoir un ensemble d'exigences incomplet. En effet, les exigences peuvent être conçues dans le contexte d'une collaboration entre des parties bien distinctes, chacune portant seulement attention au sous-ensemble des sujets qui la concernent, la propriété de complétude n'étant alors souhaitable qu'au moment de la mise en commun de toutes les exigences.

Concernant l'incomplétude, remarquons que dans le cas particulier d'exigences exprimant des autorisations et des interdictions, il est possible de choisir une approche permissive ou restrictive. Une approche permissive, correspond à un des principes fondamentaux du droit : « *nulla poena sine lege* », il n'y a pas de peine sans une loi, tout ce qui n'est pas couvert par les exigences est autorisé. L'approche restrictive est exactement l'inverse, où tout ce qui n'est pas explicitement autorisé est interdit. Dès lors, le caractère incomplet des exigences peut devenir un problème majeur s'il se rapporte à des systèmes sensibles où l'absence d'exigence peut donner lieu à des autorisations non désirées dans un cadre permissif ou un blocage complet du système, avec trop d'interdiction, dans un cadre restrictif.

Pour finir, nous définissons deux autres propriétés génériques : l'applicabilité et la minimalité. Un ensemble d'exigences est dit applicable si aucune des exigences n'est inutile, autrement dit, il n'existe pas d'exigence portant sur une situation impossible, et il est dit minimal si aucune de ces exigences n'est déductible à partir des autres, i.e. il n'y a rien de redondant.

3.2. Applications pratiques

Nous avons appliqué notre approche dans le contexte des systèmes d'information avec alarmes. Plus précisément, nous avons défini le concept de *Système avec*

Criticité dans l'Echange d'Information (CIDS [1]) [55].

Un CIDS est un système où certains échanges d'information sont de nature critique pour des agents (critique d'un point de vue sûreté, sécurité, stratégique, économique, etc.). Des exemples de tels systèmes sont les réseaux de surveillance pour la détection et la prévention d'épidémie, de catastrophes naturelles ou la surveillance de débris spatiaux. En effet, la prévention et la gestion des risques sont aujourd'hui gérées au niveau international et il serait illusoire de penser qu'un acteur seul soit en mesure de gérer convenablement de tels problèmes. La gestion de ces risques implique donc la mise en place de structures d'échanges d'information, de systèmes d'alertes coopératifs. Dans de tels systèmes, l'envoi d'un message d'alerte peut avoir des conséquences graves s'il n'est pas envoyé à la bonne personne, ou au bon groupe de personnes, ou encore s'il ne contient pas les informations nécessaires. Toutefois, par souci de confidentialité et par crainte de perdre la maîtrise de l'information qu'ils possèdent, les entreprises, les agences, les organisations et les états impliqués dans ces échanges ne communiquent pas de façon transparente et sans contraintes. La particularité des CIDS est donc la nécessité de garantir la diffusion d'informations cruciales à certains agents. Au niveau des exigences, cette nécessité vient généralement se heurter à des exigences de non-diffusion d'informations sensibles.

Cette thématique des systèmes d'alarmes a donné lieu à l'organisation d'un atelier lors de la conférence internationale Advances in Databases and Information Systems (ADBIS) en 2015, de deux ateliers lors des conférences Inforsid 2016 et Inforsid 2017, ainsi qu'un numéro spécial de la revue Ingénierie des Systèmes d'Information [168], ateliers et numéro spécial que nous avons coordonnés avec Rémi Delmas et Florence Sèdes.

Dans le contexte des CIDS, avec Rémi Delmas, nous avons proposé une approche formelle pour la spécification des exigences de politiques d'échange d'information et nous avons fourni un outil pour aider l'utilisateur à définir une politique et pour vérifier automatiquement si cette politique est conforme à nos propriétés génériques. Par ailleurs, nous avons formalisé deux types d'exigences antagonistes qui sont la nécessité de partager de l'information et l'obligation de ne pas diffuser certaines informations. De plus, nous avons montré comment il était possible, formellement, de les concilier dans une même politique à l'aide d'une opération de filtrage d'information dont la spécification apparaît naturellement [57, 56, 58].

D'un point de vue pratique nous avons proposé un langage, nommé PEPS pour « *Peps for Exchange Policy Specification* », qui permet d'exprimer les exigences liées à la diffusion d'information dans un CIDS. Nous avons choisi de baser le langage PEPS sur la logique des prédicats avec sortes [2]. Ce choix est guidé par une volonté de simplicité et surtout un besoin d'outils efficaces pour analyser les exigences de

1. CIDS pour Critical Information Diffusion System

2. La logique des prédicats avec sortes est un cadre de la logique où les objets du domaine, l'univers du discours représenté par des variables, possèdent une sorte i.e. un type [85].

diffusion d'information. D'autres choix auraient été possibles. Par exemple, de par leur aspect normatif, il semble naturel d'utiliser la logique déontique pour modéliser les exigences. La logique déontique est une logique modale conçue pour formaliser les normes, les lois et les règlements [190, 37]. Pour cela, elle définit la notion d'obligation à l'aide d'un opérateur modal qui définit ce qui est obligatoire et ce qui est interdit. Le point clé de cette approche est de pouvoir parler d'une interdiction sans rien dire sur la réalité tangible. Ainsi, il est possible d'exprimer qu'il est interdit de fumer dans un lieu, tout en rendant compte de la réalité qui peut être que quelqu'un fume. Cependant, comme toute logique modale, la logique déontique demande un certain niveau d'expertise pour être utilisée. Ainsi pour John McCarthy, si la logique modale est un outil permettant de manipuler aisément des concepts complexes, elle est aussi particulièrement difficile à manipuler pour les non-experts et reste limitée au seul usage des logiciens [128]. Cette critique peut bien évidement être étendue à tout formalisme mathématique, et c'est ce que fait Bruce Edmonds pour qui toutes les approches à base de logiques formelles sont inutilisables par la plupart des gens étant donnée la complexité même de ces langages logiques [68]. On peut toutefois objecter à cet argument que seul le langage naturel peut être compris par tout le monde, nous ramenant ainsi à l'ensemble des questions se rapportant aux ambiguïtés du langage naturel et aux limitations des possibilités d'analyses automatiques de celui-ci, questions qui nous ont justement motivés dans l'utilisation de langages formels pour l'expression des exigences.

Notre choix de ne pas considérer les formalismes basés sur une logique modale est surtout motivé par notre envie de fournir des outils efficaces d'analyses automatiques. Pour l'heure, force est de constater que les solveurs dédiés aux logiques modales sont moins efficaces que les solveurs pour les logiques standards (logique propositionnelle et logique du premier ordre) [167].

En plus du langage PEPS, nous avons défini les algorithmes permettant de vérifier nos propriétés génériques que sont la consistance, la complétude, l'applicabilité et la minimalité. A partir de ces différents algorithmes, nous avons créé le logiciel PEPS-analyzer pour une assistance à l'écriture d'exigences de diffusion d'information [1]. En interagissant avec PEPS-analyzer, il est possible pour un concepteur de vérifier si des exigences sont conformes à nos propriétés génériques et de les modifier en conséquence en se servant de contre-exemples renvoyés par l'outil. Sans outil, cette tâche de mise au point peut s'avérer impossible, la combinatoire des interactions faisant qu'il est humainement impossible d'identifier toutes les incohérences entre exigences, de garantir la complétude de la politique ou de garantir l'absence de redondance entre règles.

1. Il me faut préciser que la création du langage PEPS, des propriétés génériques et des algorithmes de vérifications qui leur sont associés ainsi que le concept de CIDS relèvent d'un travail conjoint avec Rémi Delmas, mais que la paternité de l'outil PEPS-analyser lui revient entièrement.

4. Méthodes formelles et spécification

La spécification d'un artefact correspond à une description précise de ce que l'on cherche à concevoir. Théoriquement, les exigences décrivent ce que doit faire l'artefact et les spécifications décrivent ce qu'est l'artefact, ce qui doit être construit. Suivant cette définition, les exigences d'une maison correspondraient à savoir si elle doit pouvoir se fermer à clé, s'il doit y avoir une vue sur l'extérieur ou s'il doit y avoir une salle de bain. Les exigences de bas niveau donneraient le nombre exact de pièces et leurs fonctions, la superficie, le nombre de fenêtres, etc. La spécification, quant à elle, correspondrait tout simplement aux plans de construction. Néanmoins, cette frontière entre exigences et spécifications est assez théorique. Les exigences sont parfois si précises qu'elles peuvent être utilisées comme spécification [1] et il arrive que des spécifications de haut niveau soient si peu précises qu'elles tendent plutôt à ressembler à des exigences.

La conception d'un artefact technique fait intervenir de nombreux artefacts intermédiaires, chaque artefact intermédiaire ayant lui-même sa spécification. Ainsi, il existe des spécifications de haut niveau, qui vont décrire des assemblages complexes, et des spécifications de bas niveau, qui vont finement décrire l'artefact dans ses moindres détails. De nombreux langages existent pour exprimer les spécifications et, suivant les domaines, les disciplines, le niveau de détail, elles seront exprimées en langage naturel, au moyen de modèles numériques, dans des langages mathématiques ou à l'aide de *modèles conceptuels*.

Un modèle conceptuel est une représentation schématique avec des concepts, des catégories de choses, connectés entre eux par des relations : on utilise aussi le vocable de *modèle entité-association* [2] [41]. Pour exprimer la spécification d'un artefact à l'aide d'un modèle entité-association, il est possible de définir son propre langage de modélisation spécialisé, on parle généralement de langage dédié ou métier, ou d'utiliser un langage existant. Parmi les langages existants, citons l'*Universal Modeling Language* (UML) et son descendant direct *Systems Modeling Language* (SysML). UML est un langage de modélisation très orienté informatique, disposant de différents types de diagrammes et permettant de spécifier facilement l'architecture d'un logiciel. SysML, quant à lui, est l'adaptation d'UML à ce qui se nomme l'*ingénierie système*. Il permet de représenter, au moyen de diagrammes, n'importe quel système, n'importe quel artefact technique. Notons qu'exprimer une spécification à l'aide de modèles conceptuels relève d'une approche très descriptive, les diagrammes représentant la structure de l'artefact.

Comme nous l'avons vu précédemment, la conception d'un artefact complexe

1. On utilise même le terme d'exigences de spécification.

2. Nous choisissons de faire l'amalgame entre modèle entité-association et modèle conceptuel. S'il est vrai que le paradigme entité-association est bien plus vaste puisqu'il englobe, par exemple, les schémas de bases de données, l'une des conférences de référence du domaine, ER (pour Entity-Relationship), se définit elle-même comme la conférence internationale de la modélisation conceptuelle (International Conference on Conceptual Modeling).

induit une structuration hiérarchique. Dès lors, un artefact est divisé en sous-ensembles plus simples, eux-même composés de sous-ensembles et ainsi de suite jusqu'à avoir des éléments qui peuvent être construits. Prenons deux exemples : un avion et un logiciel. De façon très schématique, dans le cas de l'avion, une étude préliminaire va décider, en fonction d'objectifs, des grands ensembles et la façon dont ces grands ensembles s'agencent entre eux. Cette structuration est un modèle de l'avion, très abstrait, et correspond donc à un artefact intermédiaire. A partir de ce modèle, chaque sous-ensemble va donner lieu à une modélisation 3D permettant de définir les éléments électriques, de conditionnement d'air, etc. Là encore, nous avons la production d'un artefact intermédiaire, le modèle 3D, qui va servir de spécification à l'étape suivante où chaque élément du modèle va être lui-même conçu dans les moindres détails, donnant la production de plan de conception. Ces plans vont à leur tour servir de spécification pour la fabrication des éléments. Ainsi, nous avons un processus où chaque étape produit des artefacts intermédiaires, ces artefacts intermédiaires devenant la spécification utilisée à l'étape suivante. Nous pouvons retrouver la même chose dans le cas d'un logiciel, avec une structuration par grands éléments qui va servir de spécification à la conception de chaque élément, éléments servant eux-mêmes à définir les entrées/sorties de chaque fonction du programme.

4.1. Vérifier et assister la spécification basée modèle

Partant du constat que, pour un artefact technique, la structuration hiérarchique entraîne un processus de conception dans lequel nous avons à chaque étape la création d'artefacts intermédiaires qui deviennent éléments de spécification à l'étape suivante, il peut être intéressant d'utiliser des méthodes formelles pour d'une part vérifier la correction de ces artefacts intermédiaires et, d'autre part, fournir une aide à la conception. Ainsi, en nous focalisant plus particulièrement sur les modèles conceptuels, nous avons proposé un processus de conception qui tire parti des avantages que nous offrent les méthodes formelles [51, 49, 50, 52].

Si les modèles conceptuels, comme UML ou SysML, permettent de modéliser un système de façon visuellement intelligible et compréhensible par des personnes d'horizons différents, ils manquent cependant d'expressivité pour capturer complètement une spécification. En effet, suivant la complexité de l'artefact à construire, il est souvent nécessaire d'exprimer des contraintes sur les modèles. Prenons l'exemple simple de tâches devant être attribuées à des personnes suivant leurs compétences, sachant que toutes les tâches doivent être attribuées. Cet exemple est un problème d'allocation que l'on peut retrouver aussi bien dans la gestion des ressources humaines que dans la conception d'architectures embarquées critiques. Il nous faut pouvoir exprimer ici la contrainte que chaque tâche doit être assignée à une personne possédant les compétences requises. A cela vient s'ajouter la contrainte qu'aucune personne ne doit être en surcapacité, autrement dit que la somme des tâches assignées à une personne ne doit pas dépasser sa

charge de travail maximale.

Pour exprimer de telles contraintes sur le modèle, il est possible d'utiliser le langage naturel, mais, comme pour les exigences, ce choix peut être source d'ambiguïté. De plus, étant dans un cadre de modélisation technique, il est préférable d'utiliser un langage compréhensible par une machine afin de pouvoir réaliser des opérations automatiques sur les modèles. C'est pour ces raisons que nous avons choisi d'utiliser un langage formel pour l'expression de nos contraintes.

La première étape de notre processus consiste à établir un modèle de spécification qui décrive les types de composants du système, leurs relations et leurs contraintes sans donner la liste précise de tous les éléments (définir un modèle, mais pas les instances). Dans notre exemple, cette étape consiste à définir de façon générique les relations et contraintes entre tâches, compétences, personnes et charges de travail maximales.

Dans la deuxième étape, nous cherchons à aider le concepteur à avoir confiance en son modèle. Pour cela, nous utilisons les solveurs pour vérifier que le modèle ne contienne pas d'incohérence, autrement dit qu'il existe bien une réalisation possible de la spécification, des instances du modèle [1]. Pour notre allocation de ressources, cela correspond à générer un exemple où il y a des tâches, des personnes et des allocations de tâches, le tout en respectant les contraintes. Si un solveur n'arrive pas à générer un tel exemple, cela signifie que la spécification n'est pas bonne : il est impossible de construire un artefact qui respecte le modèle de spécification. En renvoyant plusieurs exemples simples au concepteur, nous lui permettons de tester sa spécification et, dans une certaine mesure, de vérifier si elle est bien conforme à ce qu'il souhaite. Notons que dans cette étape, même si au travers des solveurs nous faisons appel à des méthodes de vérification, nous réalisons une opération de validation. En effet, l'artefact ici est un modèle qui représente une spécification et nous ne disposons pas d'étalon (de spécification de la spécification) pour pouvoir établir la correction de l'artefact donc sa vérification. Nous sommes clairement dans une démarche de validation au plus tôt, nous cherchons à nous assurer, le plus possible, de l'exactitude d'une spécification avant d'entamer les phases suivantes du processus de conception.

Une fois le modèle construit, il devient à son tour une spécification pour la conception de nouveaux artefacts intermédiaires qui seront des instances de ce modèle. De par la taille et la complexité des modèles employés dans l'industrie, il est de plus en plus difficile pour un humain de concevoir un artefact conforme à une spécification. Si l'on rajoute à cela, non pas la recherche d'une solution quelconque, mais d'une solution optimisée par rapport à des critères quantitatifs, la tâche devient humainement impossible. Nous avons donc proposé d'apporter une aide aux concepteurs, sous la forme d'outils qui prennent une spécification basée modèles, avec des contraintes à respecter et des critères à optimiser, ainsi

1. Notons que ce n'est pas l'unicité qui est ici visée, mais l'existence d'au moins une solution.

que l'ébauche d'une solution qui prend la forme d'une instance du modèle de spécification. Notre méthode, dans la pratique, consiste à fournir à un outil logiciel : un modèle qui sert de spécification, ainsi qu'une instance partielle de ce modèle. Cette instance est dite partielle car, si elle doit contenir tous les objets, il n'est cependant pas nécessaire d'instancier toutes les relations entre les objets. Dans notre exemple, l'instance partielle décrira toutes les tâches, les personnes et les charges de travail maximales, mais pas les allocations de tâches (il est tout de même possible de fournir un certain nombre d'allocations qu'un concepteur veut absolument voir réalisées). Faisant appel à un solveur, l'instance est automatiquement complétée de manière à ce qu'elle réalise les allocations de tâches en étant conforme aux contraintes définies dans la spécification.

Pour cette dernière étape, l'utilisation de solveurs nous permet non seulement de concevoir un artefact conforme à sa spécification, mais aussi d'optimiser la solution trouvée par rapport à des critères quantitatifs. Généralement, à partir d'une seule spécification plusieurs solutions peuvent être construites, certaines pouvant être considérées meilleures que d'autres. Pour notre exemple, un critère pourrait être que nous préférons que les personnes soient uniformément occupées, personne ne doit travailler plus que les autres. De nombreux solveurs disposent de la capacité d'intégrer des fonctions de coût pour pouvoir classer les différentes solutions et être guidé pour trouver la meilleure. En exploitant ces fonctionnalités, il est possible de compléter une instance partielle pour obtenir une solution qui, non seulement respecte sa spécification avec ses contraintes, mais en plus optimise un critère donné.

4.2. Applications pratiques

Nous avons appliqué nos méthodes de vérification et d'aide à la conception pour de l'informatique embarquée. En effet, les logiciels intervenant dans le pilotage d'un avion sont soumis à un ensemble de règles permettant de garantir la sûreté de fonctionnement. Par exemple, il peut être nécessaire qu'un logiciel soit dupliqué et que chacune de ses duplications s'exécute sur des ordinateurs qui ne sont pas du même côté de l'avion (une sur la partie droite et l'autre sur la partie gauche). Cet ensemble de règles peut être modélisé sous la forme de contraintes et de critères de préférence sur des modèles de spécification d'architectures avioniques.

Dans le cadre de la coopération AIRSYS, nous avons travaillé sur le problème de l'allocation de fonctions logiciel sur une architecture afin de garantir au mieux des contraintes de sûretés de fonctionnement. Parallèlement, dans le projet *Open Platform for the Engineering of Embedded Systems* (OPEES), nous avons montré comment outiller notre processus de conception, le but du projet OPEES étant de disposer de technologies d'ingénierie innovantes dans le domaine des systèmes embarqués disponibles sur le long terme. Même si notre processus n'a été utilisé que dans des contextes de conception d'architectures logicielles embarquées, il

nous semble tout à fait possible, et utile, de l'appliquer à d'autres domaines du moment que nous sommes face à des modèles difficilement appréhendables par un être humain en raison de contraintes trop complexes ou d'instances de trop grandes tailles.

Du point de vue pratique, nous avons intégré à notre processus des outils existants tels que USE [88], UML2Alloy [7] et UMLtoCSP [33]. De plus, nous avons développé nos propres outils [1] de traduction de modèles de spécification munis de contraintes et de critères en problèmes pseudo-Booléens qui sont analysés avec le solveur Sat4J-PB [124] ou le solveur WBO [126]. Notons qu'à l'époque de ces travaux, l'idée d'analyser des modèles et des contraintes à l'aide de méthodes formelles était assez en vogue [7, 172, 122, 34].

5. Méthodes formelles, du modèle au code

Comme nous l'avons vu, la production d'un artefact technique passe généralement par un ensemble d'étapes, ces étapes produisant des artefacts intermédiaires. Dans le domaine du logiciel, cette production d'artefacts intermédiaires est présente depuis les origines. Le verbe *programmer*, qui consiste à donner des instructions à un ordinateur, a été inventé par les premiers programmeurs de l'histoire de l'informatique qui avaient pour particularité d'être des programmeuses. C'est en 1945 qu'est mis en service à l'Université de Pennsylvanie le premier ordinateur entièrement électronique nommé ENIAC pour *Electronic Numerical Integrator And Computer*. Le but de l'ENIAC était de résoudre des équations beaucoup plus rapidement qu'un être humain. Afin de lui transmettre les informations pour réaliser un calcul, il est nécessaire de brancher des câbles, d'utiliser des tableaux de connexion et d'enclencher des commutateurs. Herman Goldstine, qui était en charge du projet, décide de recruter pour réaliser ces tâches celles que l'on appelle aujourd'hui les six de l'ENIAC : Jean Jennings, Frances « *Betty* » Snyder Holberton, Frances Bilas, Marlyn Wescoff, Kathleen « *Kay* » McNulty, et Ruth Lichterman [2] [84, 125]. A leur arrivée, ne disposant pas encore des accréditations pour être autorisées à entrer dans la salle où se trouve l'ordinateur, elles doivent concevoir des diagrammes uniquement sur papier pour donner la séquence d'opérations permettant de réaliser le calcul. La « machine » n'étant munie d'aucun manuel, d'aucune explication, il faut tout inventer et tout comprendre par soi-même. C'est donc aidées par Adele Katz Goldstine, professeur de mathématique pour les femmes calculateurs [3], que les six programmeuses vont concevoir comment mani-

1. La partie implémentation de ces outils est entièrement le fait de Rémi Delmas et David Doose.

2. A ces six noms que l'on retrouve dans la littérature, Barkley Fritz rajoute Ruth Rauschenberger, Lila Todd, Homé McAllister, Marie Bierstein et Willa Wyatt Sigmund [84].

3. Pendant la deuxième guerre mondiale, l'armée américaine employait de nombreux mathématiciens pour effectuer les calculs tels que les calculs balistiques. Du fait que les hommes étaient

puler les interrupteurs et les câbles, à partir des plans et de la structure machine. C'est d'ailleurs Adele Goldstine qui écrira en 1946 le premier manuel d'utilisation de la machine [89].

Si les inventeurs de l'ENIAC pensaient que pour réaliser des calculs il suffisait de simplement brancher des câbles, ils ont découvert que les choses étaient beaucoup plus complexes dans la réalité. Faire faire un calcul à la machine nécessitait toute une séquence d'opérations loin d'être triviales [177]. Il fallait, premièrement, transformer le calcul en un algorithme, pour ensuite transformer cet algorithme en un algorithme prenant en compte les limitations de calcul de l'ordinateur ainsi que ses avantages, comme le parallélisme et la capacité de faire des boucles. Venait ensuite l'implantation de l'algorithme dans la machine, avec la gestion du flot de données avec le raccordement, dans le bon ordre, d'unités de calcul à l'aide de câbles, et l'initialisation de la machine, avec la mise au point de techniques pour contourner les limitations telles que la taille maximum d'un nombre qui peut être manipulé. Pour la réalisation de tout ce processus, les six de l'ENIAC se sont mises à utiliser le terme « *programmer* » [98].

Nous voyons bien, ici que, dès les débuts de la programmation, construire un logiciel passe par une chaîne de production d'artefacts intermédiaires abstraits comme l'équation représentant le problème, l'algorithme, l'algorithme adapté à la machine et les plans de câblages. Aujourd'hui, le simple fait d'écrire un programme s'inscrit toujours dans une telle démarche avec un code source, écrit dans un langage plus compréhensible par l'être humain, qui est traduit par un compilateur dans un langage interprétable par un ordinateur. Le premier pas vers cette abstraction a été réalisé par Frances « *Betty* » Holberton. En effet, en 1951, elle écrit le premier programme qui génère un programme dans l'objectif de généraliser des procédures de fusion et de tri de données. Inspirée par ce travail, Grace Murray Hopper, qui connaissait Holberton, crée le premier programme générant des programmes à partir d'instructions écrites en anglais. En d'autres termes, elle a conçu le premier compilateur [97]. Les travaux de Grace Hopper donneront le langage COBOL et seront poursuivis notamment par Frances « *Fran* » Allen qui recevra un prix Turing pour avoir « *posée les bases conceptuelles d'une analyse et d'une transformation systématique des programmes informatiques.* » [1].

Depuis, les langages informatiques ont bien évolué et le développement logiciel, dans le but d'être toujours plus intuitif, n'a cessé de rajouter des étapes de productions d'artefacts intermédiaires. Ces modèles, plus faciles à appréhender, sont autant d'artefacts intermédiaires intervenant dans la production d'un logiciel. Au travers d'un langage graphique simple, ils permettent de spécifier un système logiciel. Plusieurs raisons permettent d'expliquer le succès des approches basées modèles. Une raison non négligeable est que les modèles, de par leur aspect

envoyés au combat, beaucoup de calculateurs étaient des femmes.

1. Traduit de : *laid the conceptual basis for systematic analysis and transformation of computer programs*.

graphique, sont intuitifs et facilitent la communication entre les différents acteurs qui interviennent dans un cycle de développement. En effet, la conception d'un logiciel fait intervenir, non seulement plusieurs équipes de programmeurs avec des compétences et des spécialités différentes [1], mais aussi des non-informaticiens comme des spécialistes des métiers concernés, des ergonomes, des graphistes, de futurs utilisateurs, etc. A cette hétérogénéité des métiers vient, de plus en plus, s'ajouter la complexité des organisations, avec des projets faisant intervenir des acteurs de différents pays, de différentes cultures et de langues différentes. Là encore, l'utilisation de modèle permet d'avoir un langage compréhensible par tous et offre un support graphique simple pour pouvoir partager, discuter et travailler ensemble. Pour finir, notons que, même dans une équipe homogène, l'utilisation de modèles peut être motivée par la volonté de limiter les erreurs d'ambiguïté dans les spécifications comme cela peut être le cas quand elles sont écrites en langage naturel.

Le succès de l'approche par modèles est tel qu'il a donné lieu à une discipline, l'*Ingénierie dirigée par les modèles*, et à de nombreux outils informatiques, ces outils permettant, par exemple, de générer automatiquement à partir de modèles de spécifications le squelette d'un code source et la documentation associée. Dans cette logique, les modèles étant de plus en plus utilisés dans le monde de l'entreprise par des non-informaticiens, des travaux cherchent à aller plus loin en définissant des transformations permettant de produire du code source à partir des exigences et des processus exprimés à l'aide de modèles [143, 69].

5.1. Vérifier que le code est conforme au modèle

Comme nous venons de le voir, la conception d'un logiciel entraîne la production d'un grand nombre d'artefacts techniques parmi lesquels se trouvent des modèles et du code source. Nous avons déjà évoqué l'usage de méthodes formelles pour la correction d'exigences et de modèles, nous allons donc, ici, nous focaliser sur la correction du code. Comme toujours, la correction s'établit en fonction de la définition de ce que doit faire l'artefact. Concernant le code, dans une approche basée modèle, la définition de l'artefact nous est donnée par son modèle de spécification. La correction revient donc à établir la conformité du code source par rapport à ce qui est défini par le modèle.

Comme toujours, il est possible de chercher à établir cette conformité par des tests ou par des relectures attentives. Pour notre part, nous nous sommes focalisés sur une approche formelle afin d'établir la preuve mathématique de cette conformité. Pour ce faire, dans le cadre de la thèse d'Anthony Fernandes Pires

1. L'informatique s'étant grandement complexifiée, le développement logiciel a, lui aussi, subi une division du travail avec des équipes spécialisées dans des domaines comme le réseau, les bases de données, le graphisme ou l'architecture logiciel. A cette diversité de métiers, s'ajoute une diversité des langages de programmation, entraînant un besoin de disposer d'un socle commun, besoin auquel tente de répondre l'utilisation de modèles.

que j'ai encadrée avec Virginie Wiels, nous avons choisi d'utiliser les méthodes de preuve de programme basées sur les travaux de Robert Floyd et de Tony Hoare [82, 104]. Schématiquement, cette méthode consiste : premièrement, à exprimer de façon mathématique une précondition, c'est-à-dire ce qui est vrai avant l'exécution d'un programme, et la postcondition, ce qui est vrai après ; deuxièmement, à établir que la postcondition est toujours vraie après l'exécution du programme si la précondition est vraie. Pour ce faire, on s'appuie sur un ensemble de règles logiques associées aux instructions du langage de programmation dans lequel est écrit le code du programme. Prenons l'exemple d'un programme de tri de nombres, la précondition sera une formule explicitant que nous avons un tableau de nombres et la postcondition consistera en une formule explicitant le fait que nous avons les mêmes nombres qu'en entrée et qu'ils sont maintenant triés. La preuve consistera à montrer, au moyen des règles logiques associées au code source de ce programme, que la précondition implique bien la postcondition après exécution du code.

Cette méthode de preuve, nommée *analyse statique de programme par méthodes déductives* ou *logique de Hoare*, est implémentée dans des outils informatiques. Ces outils réalisent automatiquement la partie consistant à prouver que le programme et la précondition impliquent bien la post-condition, à charge à l'utilisateur d'écrire correctement les pré et post conditions [1]. Pour écrire les pré et post conditions, chacun de ces outils dispose d'un langage dédié, appelé *langage d'annotation de preuve*. Citons, pour exemple, le *Java Modeling Language* pour langage de programmation JAVA, ACSL [2] [16] pour le langage C et SPARK [43] qui est à la fois un langage de programmation et d'annotation.

Pour pouvoir utiliser une telle technique, et donc prouver la correction d'un code par rapport à son modèle, il faut être capable de transcrire la spécification du programme, exprimée par le modèle, en pré et post conditions. Une telle tâche demande de bonnes connaissances en mathématique et en méthodes formelles, ce qui peut être un problème. Ainsi, nous avons mené une enquête auprès de soixante-quinze personnes dans le domaine des systèmes embarqués et du logiciel. Dans cette enquête, si 95% des personnes trouvaient un avantage à la preuve formelle, 70% considéraient la formalisation de propriétés comme la principale difficulté [3]. Pour pallier cette difficulté, avec Stéphane Duprat, Anthony Fernandes Pires et Virginie Wiels, nous avons choisi de développer une méthode pour générer automatiquement, à partir d'un modèle de conception, des annotations de preuve [76, 77].

1. Nous simplifions, ici, quelque peu les choses, ces outils utilisent de nombreuses techniques et des évolutions de la logique de Hoare. De plus, il ne faut pas seulement écrire une pré et une post condition, mais aussi d'autres formules relatives à des hypothèses et à des propriétés intrinsèques du système.

2. ACSL pour ANSI / ISO C Specification Language

3. Plus de détails sur cette enquête sont disponible dans la thèse d'Anthony Fernandes Pires[73].

5.2. Applications pratiques

Nous avons appliqué notre méthode aux systèmes réactifs embarqués avioniques. Dans le contexte du logiciel embarqué, et pas seulement avionique, l'automatisation de méthodes permettant de prouver la correction de code revêt un enjeu tout particulier. Comme le souligne un rapport de 2010 demandé par le ministère de l'Industrie à des experts du domaine, les opérations de V&V représentent, à elles seules, entre 40% et 50% des coûts de conceptions [153]. Ceci est d'autant plus vrai dans le milieu avionique où, sur certains projets, ces coûts avoisinent les 60% [74].

Il faut bien comprendre que ces coûts de V&V sont induits par des exigences liées à la certification de tels systèmes. Dans notre cas, nous nous sommes intéressés à la correction de programme dans le cadre de la norme avionique DO178 [1] et de son supplément dédié aux méthodes formelles la DO-333. Ces normes définissent des objectifs que doit remplir tout développement logiciel qui veut être certifié. Notre approche, en plus de permettre à des non-experts en méthodes formelles de réaliser de la preuve de correction de façon complètement transparente, répond à certains de ces objectifs de certification [78].

Concernant les aspects pratiques, pour pouvoir générer les annotations de preuve issues d'un modèle, il faut être sûr de ce que signifie le modèle, qu'il soit sans aucune ambiguïté. Dans cette optique, nous avons identifié un sous-ensemble des machines à états UML suffisant pour modéliser les systèmes réactifs embarqués avioniques et nous avons défini une sémantique formelle pour ce sous-ensemble. A partir de cette sémantique, il nous a été possible de concevoir les règles de génération automatique d'annotations permettant, à un outil de preuve, de démontrer que le code correspond bien au comportement spécifié par son modèle [76, 74]. L'ensemble de ce processus a été implémenté dans un logiciel dans le cadre de cette thèse [75].

Pour finir, notons l'aspect crucial du contexte de certification dans ce travail. En effet, nous pouvons nous interroger sur les apports d'une approche comme la nôtre par rapport à une approche plus classique de génération de code à partir d'un modèle. Le principal atout de la génération d'annotations réside dans le fait qu'elle ne crée, ni ne modifie, du code exécutable. S'il existe des erreurs dans l'outil de génération d'annotations et de vérification, cela n'introduira aucune erreur dans le code. Par conséquent, la qualification d'un tel outil dans le cadre d'une certification est beaucoup plus simple que celle d'un générateur de code qui peut, lui, introduire des erreurs dans le programme.

1. Notons que nous employons le mot « *norme* », mais la DO-178 (*Software considerations in airborne systems and equipment certification*) est un standard reconnu comme moyen de conformité par les autorités de certification.

Chapitre 3

Des justifications pour argumenter la correction

> « *. . . you are listening to a machine. Do the world a favor and don't act like one.* »
>
> *Stephen Falken*, Wargame

1. Une connaissance basée sur des justifications

En dépit de leur aspect prometteur, c'est un fait, les méthodes formelles sont peu utilisées dans la pratique industrielle. Comme le souligne Tony Hoare dans son article de 1996 « *Comment les logiciels sont-ils devenus si fiables sans preuve ?* »[1] [105], ce n'est pas grâce à l'utilisation de méthodes formelles que les logiciels sont devenus plus fiables, mais par l'usage de techniques déjà employées dans les autres branches de l'ingénierie comme : des procédures rigoureuses de relecture des spécifications de conceptions, de l'assurance qualité fondée sur de larges éventails de tests ou de l'amélioration continue. Dès lors, comment, à partir de ces éléments informels, être sûr que l'artefact final est correct ?

Par ailleurs, l'utilisation de méthodes formelles ne nous ôte pas de certains doutes. En effet, considérons que nous disposons de la preuve mathématique de la correction d'un artefact, sommes-nous sûrs que cette preuve ne contienne pas elle-même des erreurs ? Si elle a été établie par une machine, sommes-nous sûrs

1. Traduit de : *How Did Software Get So Reliable Without Proof ?*.

que le programme utilisé est lui aussi prouvé ? Nous pouvons ainsi remettre en question tous les éléments, chercher des preuves aux preuves, sans jamais trouver de fin à nos questionnements. Nous sommes typiquement face au problème épistémologique de la régression infinie, problème qui fut posé par le sceptique Sextus Empiricus [1] dans les termes suivants : « *il faudra qu'il apporte une démonstration pour prouver cela : et puis il faudra qu'il donne une démonstration de sa démonstration pour prouver qu'elle est vraie, et ainsi à l'infini* » [71].

Loin d'être un problème purement philosophique, le problème de la confiance dans les moyens utilisés pour établir la preuve de correction, se pose cruellement dans le monde de l'ingénierie. En effet, toute la démarche visant à établir la correction d'un artefact n'a qu'un seul but : prévenir les erreurs. Il est donc crucial que les moyens utilisés ne soient pas eux-mêmes entachés d'erreurs ou, tout au moins, que nous ayons confiance en eux. A ce sujet, Richard Bloch raconte l'anecdote suivante : à l'époque où il travaillait à Harward avec Howard Aiken sur l'un des tout premiers calculateurs universels et entièrement électromécaniques [2], le MARK 1, les résultats étaient vérifiés en utilisant deux méthodes différentes : « *nous avions passé beaucoup de temps à nous assurer que si cela était fait de deux manières différentes qui étaient vraiment différentes, un bogue particulier ne pouvait pas se trouver dans la machine et apparaître de manière à créer la même mauvaise réponse dans les deux cas. Ce que nous avons fait, c'est comparer les réponses, l'idée étant que si elles étaient correctes, tout était probablement correct* » [11]. Ce que nous voyons apparaître ici, c'est le besoin d'être convaincus. Pour Aiken et son équipe, les résultats du calcul n'étaient exempts d'erreur que quand ils étaient obtenus de deux manières différentes. Remarquons qu'ils ne remettaient aucunement en cause les fondements mathématiques de leurs programmes informatiques, ils cherchaient seulement à s'assurer qu'aucune erreur n'était venue se glisser ni dans le code, ni dans l'exécution du code par la machine. Finalement, ils cherchaient à *savoir* que le résultat était correct, ce savoir passant, en plus des démonstrations mathématiques, par une redondance des méthodes de calculs.

Nous cherchons à « *savoir si un artefact est correct* », la preuve formelle n'étant qu'un élément parmi d'autres permettant d'établir cette connaissance. Et c'est justement cette connaissance, dénuée de doute, qui est difficile à atteindre. Mais, penchons-nous quelque peu sur le terme *savoir*. Dans le *Théétète*, Platon propose diverses analyses de ce que pourrait être la connaissance. La définition de Platon retenue par l'histoire de la philosophie est qu'une connaissance est une *croyance*,

1. Sextus Empiricus était un médecin et un philosophe grec qui a vraisemblablement vécu entre le II^e^et le III^e^ siècle.

2. L'Automatic Sequence Controlled Calculator (ASCC) est un ordinateur conçu par Howard Aiken en 1937 et fabriqué en 1944 par IBM pour l'Université de Harward. L'équipe de programmeur qui travaillait sous les ordres d'Aiken sur le Mark 1 était composée de Richard Bloch, Robert Campbell, et Grace Hopper. L'ENIAC construit en 1945 est généralement considéré comme le premier ordinateur, car il était entièrement électronique contrairement au MARK 1 qui était électromécanique.

vraie et justifiée où, comme Platon le fait dire à Socrate, « *l'opinion vraie accompagnée de raison est science, mais que, dépourvue de raison, elle est en dehors de la science, et que les choses dont on ne peut rendre raison sont inconnaissables* » [145]. Cette définition est dite définition tripartite. Elle consiste à dire qu'une chose est sue : (1) si elle est vraie, (2) si on la croit vraie et (3) si l'on possède des *justifications* sur cette croyance [1]. Elle est généralement représentée dans la littérature par l'acronyme anglais *JTB* pour *justified true belief* [40].

Si nous acceptons que la connaissance est une croyance, vraie et justifiée alors notre point focal devient la notion de justification. En effet, si nous cherchons à établir la correction d'un artefact, c'est que nous croyons qu'il est correct. Nous évacuons tout ce qui relèverait de la malhonnêteté, qui n'a pas lieu d'être ici. Dès lors, suivant la définition tripartite JTB, il ne reste plus qu'à établir les justifications garantissant la correction [2]. Pour cela, nous pouvons nous pencher sur les travaux réalisés par la communauté de l'argumentation et par certains logiciens.

Quand on s'intéresse à la justification d'une propriété, il n'est plus question de validité, au sens où la propriété serait vraie ou fausse, mais d'étudier pourquoi une propriété est jugée comme acceptable. Dans son ouvrage *Fallacies*, Charles Hamblin remet en question l'utilisation de la logique formelle face à l'étude de l'argumentation [99]. Plus précisément, il s'intéresse aux raisonnements fallacieux et reprend les travaux d'Aristote sur les sophismes. Dès la première ligne des *Réfutations Sophistes*, Aristote définit un raisonnement fallacieux comme un raisonnement qui semble être valide, mais qui ne l'est pas [10]. Un raisonnement fallacieux est donc un raisonnement incorrect, faux, et qui pourtant peut paraître logiquement correct. Dans le cadre particulier du sophisme, l'erreur de raisonnement est connue, mais elle est réalisée dans le but de convaincre, de persuader. Le mot sophisme trouve son origine dans la Grèce antique où les sophistes étaient des orateurs. Chez Platon, le mot prendra une connotation péjorative et deviendra synonyme de malhonnêteté intellectuelle, de manipulation au mépris du raisonnement logique.

Cependant, tous les raisonnements fallacieux ne sont pas des sophismes. Bien qu'incorrect, un raisonnement fallacieux peut être énoncé de bonne foi. Prenons l'exemple suivant donné par Charles Dodgson [3] : si « *tous les lions sont féroces* » et que « *certains lions ne boivent pas de café* » il peut être tentant de conclure que « *certaines créatures qui boivent du café ne sont pas féroces* » [35]. Cette conclusion

1. Notons que, dans le *Théétète*, Platon étudie plusieurs définitions possibles de connaissance et que la pièce se conclut par un constat d'échec de Socrate puisque définir la connaissance semble être un problème insoluble. Toutefois, par delà le texte, la tradition philosophique considère que Platon donne la définition tripartite (JTB), de la connaissance comme une croyance, vraie et justifiée dans le *Théétète* [40].

2. Nous omettons volontairement la notion de « *vraie* », laissant aux philosophes le soin de débattre de la réalité des choses, cette question ne nous semblant pas à propos dans une approche d'ingénierie.

3. Charles Dodgson est généralement plus connu sous le pseudonyme de Lewis Carroll comme auteur d'*Alice au pays de merveilles*.

relève d'un raisonnement fallacieux, la conclusion logique que nous pouvons tirer de ces deux prémisses est que « *certaines créatures féroces ne boivent pas de café* ».

Dans *Fallacies*, Charles Hamblin reprend les différents sophismes répertoriés par Aristote, qu'il réécrit de façon plus moderne, et y ajoute un ensemble de raisonnements fallacieux qu'il a lui-même identifiés. Le but de sa démarche est de comprendre ce qui rend une argumentation acceptable. Pour cela, il donne un nouveau modèle pour la validité d'un raisonnement : la validité ne dépendant plus de critères logiques relatifs à la vérité des prémisses, mais de justifications et de la façon d'organiser ces justifications. La relation entre des prémisses et une conclusion n'est plus de l'ordre de l'implication logique mais d'une dialectique qui autorise, ou interdit, des comportements discursifs.

Notons que cette démarche coïncide avec deux autres travaux majeurs dans le domaine de l'argumentation. Premièrement, la publication de *Traité de l'argumentation : La nouvelle rhétorique* [144] où Chaim Perelman et Lucie Olbrechts-Tyteca définissent une nouvelle théorie de l'argumentation basée sur une approche dialectique qui complète la logique. Pour eux, si, comme les logiciens, seule la démonstration purement formelle est admise, il est dès lors impossible d'établir des raisonnements autres, ce qui est « *. . . une limitation indue et parfaitement injustifiée du domaine où intervient notre faculté de raisonner et de prouver* ». Deuxièmement, la création par Stephen Toulmin [180] d'un modèle présentant de façon générique comment passer de données à une conclusion, modèle que nous verrons dans les détails plus avant.

2. Argumenter de la correction d'un artefact

Le problème qui nous préoccupe est un problème d'inférence. Nous cherchons à déterminer s'il est acceptable ou pas de passer d'un ensemble de justifications à une conclusion. Pour reprendre l'anecdote de Richard Bloch, est-il raisonnable de conclure qu'un résultat est correct s'il a été obtenu à l'aide de deux programmes différents et que chacun de ces programmes emploie une méthode mathématique prouvée ? L'étude de tels raisonnements a connu un essor en Amérique du Nord depuis les années 1970 et notamment depuis la publication de *Logical Self-Defense* de Ralph Johnson et d'Anthony Blair [113]. Dans cet ouvrage, les auteurs tentent de définir une approche systématique pour étudier l'argumentation informelle. Ainsi, depuis quelques années, la recherche qui s'intéresse à tout ce qui relève du raisonnement non formalisé mathématiquement se retrouve sous le vocable : *Informal Logic*, *Critical Thinking* et *Argumentation* [1].

De notre côté, nous avons, ici, une ambition bien moindre. En effet, nous sommes dans un contexte d'ingénierie et les raisonnements qui nous intéressent,

1. Nous avons volontairement choisi ici de garder les noms anglais. Les termes Logique Informelle, Pensée Critique et Argumentation semblant des traductions trop approximatives et pas forcément en adéquation avec ces courants de pensée.

bien qu'informels, restent assez factuels. Nous ne visons pas l'étude des débats sur des positions éthiques ou des choix de société, mais l'étude des documentations techniques qui visent à l'accréditation, ou à la certification, d'artefacts. Nous trouvons ce type de documents, par exemple, dans le domaine de l'évaluation des risques et de la fiabilité des systèmes critiques sous le nom de *safety case* [1]. Le safety case est un document structuré qui fournit une justification et des arguments valables sur le fait qu'un système satisfait des propriétés relatives à sa sécurité. Comme le soulignent deux standards britanniques, c'est « *une argumentation raisonnée et auditable, créée pour soutenir l'affirmation selon laquelle un système satisfait des exigences* » [2] et « *une argumentation structurée, soutenue par un ensemble de données* » [3]. Nous voyons apparaître ici la notion de structuration des justifications, structuration qui est présente dans de nombreux standards comme la norme ISO 15026 [4] qui propose, sans expliquer comment, d'assembler un ensemble structuré d'affirmations pour disposer d'une argumentation concernant des objectifs de haut niveau.

S'il n'existe pas d'obligation pour la représentation des safety cases, des propositions de structuration graphique commencent à être utilisées en sûreté de fonctionnement. Citons, par exemple, la *Goal Structured Notation* (GSN) qui consiste en une notation graphique et un ensemble de bonnes pratiques. Basée sur les travaux de Tim Kelly et Rob Weaver [117, 116], elle cherche à définir une approche cohérente pour la construction, la présentation, la maintenance et la réutilisation d'une argumentation de sûreté de fonctionnement [5, 194].

Une des limitations majeures des notations de type GSN [5] est leur approche décompositionnelle. Pensées pour être uniquement utilisées dans le cadre de la sûreté de fonctionnement [6], ces notations s'intéressent en premier lieu à fournir une aide méthodologique pour déduire les prémisses qui garantissent une conclusion et ne se focalise pas vraiment sur le mécanisme d'inférence proprement dit. L'approche est donc clairement orientée de la conclusion vers les données : elle part d'un but de sûreté pour le décomposer, le raffiner, en éléments plus simples

1. Dans la littérature, et suivant le domaine d'application, avionique, nucléaire, ferroviaire ou militaire, nous ne trouverons pas forcément le terme de safety case, mais ceux de : *assurance case, safety assessment, accomplishment summary, certification evidence, security case, assurance case*.

2. Traduit de : *a reasoned, auditable argument created to support the contention that a defined system will satisfy the . . . requirements*, dans [186, section 4.1].

3. Traduit de : *The Safety Case shall consist of a structured argument, supported by a body of evidence*, dans [185, p9, section 9.1].

4. La norme ISO/IEC 15026 [111] est une norme applicable aussi bien à des systèmes qu'à des logiciels. Elle permet de définir des niveaux d'intégrité. Le but de ces niveaux d'intégrité est, par exemple, d'aider à assurer la sûreté ou la sécurité d'un système.

5. Dans la lignée de GSN, nous pouvons citer la notation Claim-Argument-Evidence [70] ou l'approche de forme purement textuelle de John Rushby [164].

6. Le terme *safety*, dans le cadre de la conception des systèmes critiques, est traduit en français par *sécurité*, quant au terme anglais *security*, il est traduit par *sûreté*. De son côté, le vocable *sûreté de fonctionnement* (*systeme dependability* en anglais) englobe de son côté la notion de *sécurité* et parfois même la notion de *sûreté*.

jusqu'à l'obtention d'éléments unitaires auxquels des solutions sont adressées. Comme le soulignent Valentin Cassano et Thomas Maibaum, de telles approches se sont largement éloignées de la question de la validité de l'inférence. Déterminer si une déduction est bien-fondée, ne fait plus vraiment partie de leurs objectifs [36].

Par ailleurs, il nous faut ajouter que, contrairement à ces différents travaux, nous adressons pour notre part un problème beaucoup plus large que celui de la sûreté de fonctionnement. Nous nous intéressons à structurer et comprendre toute argumentation qui relève de l'accréditation ou de la certification d'un artefact.

Remarquons tout de même que l'ensemble de ces travaux donne une vue sous forme de diagramme de l'ensemble du raisonnement. Cette représentation visuelle peut, dans une certaine mesure, aider à la compréhension. Comme le montre Charles Twardy dans le domaine de l'enseignement, l'utilisation d'outils graphiques de représentation de l'argumentation aide les élèves à mieux comprendre les liens entre les arguments et les étapes du raisonnement [184]. Dans le même ordre d'idée, Bart Verheij s'intéresse aux avantages que présentent les outils de visualisation d'argumentation dans le domaine juridique [188]. Pour finir, impossible de ne pas citer les travaux de Bob Horn dans le domaine de la cartographie visuelle de débats [106]. Il a notamment proposé une représentation graphique claire et structurée de 700 arguments donnés par 380 chercheurs en philosophie, en informatique, en psychologie et en mathématique sur le sujet : « *L'ordinateur peut-il penser ?* ».

L'idée de disposer d'une visualisation graphique du raisonnement pour mieux l'appréhender n'est pas nouvelle. En 1836, dans son ouvrage de logique formelle *Elements of Logic*, Richard Wately donnait une figure représentant une démonstration constituée d'une série d'inférences logiques avec la mention suivante : « *Beaucoup d'étudiants trouveront probablement que dessiner sous la forme d'un arbre un raisonnement est un moyen très clair et pratique d'en montrer la logique* » [1] [195]. Dans le domaine légal, John Wigmore propose dès 1917, une représentation visuelle permettant de structurer les hypothèses, les témoignages et les éléments de preuve d'un procès [91]. Si ces premiers diagrammes étaient dessinés à l'époque avec un papier et un crayon, il existe de nos jours de nombreux logiciels pour représenter une argumentation.

Pour conclure, il nous faut souligner que le problème très spécifique de la justification de résultats donnés par une machine n'est pas nouveau [178]. Au début des années quatre-vingt-dix, avec l'essor des systèmes intelligents tels que les systèmes experts, les systèmes d'aide à la décision et les systèmes de prévision, de nombreux travaux ont insisté sur la nécessité, pour l'utilisateur, de comprendre comment a été obtenu le résultat d'un programme. Cette thématique a même

1. Traduit de : *Many students probably will find it a very clear and convenient mode of exhibiting the logical analysis of a course of argument, to draw it out in the form of a Tree*, note de bas de page, page 342.

donné lieu à un symposium de l'Association Américaine d'IA en 1992 [12], ainsi qu'à un atelier dédié lors de conférence IJCAI [1] de 1993 [109]. La motivation principale de ces travaux est que : la confiance d'un humain dans le résultat donné par une machine augmente quand il dispose d'une explication lui permettant de comprendre le raisonnement utilisé par la machine.

En effet, en partant d'une étude empirique, Richard Ye montre que donner des explications aide grandement à l'acceptation des résultats d'un système qui produisait des audits automatiques [199]. De plus, ce sont des explications qui justifient les fondements du système qui ont le plus d'impact sur les utilisateurs [200]. Contrairement à de simples traces d'exécution d'un programme, ces explications, qualifiées de profondes [39, 173], fournissent les théories à partir desquelles le résultat a été généré, les logiques sous-jacentes et les justifications de la base de connaissances sur laquelle s'appuie le programme. Dans leur méta-étude, Shirley Gregor and Izak Benbasat montrent que, en plus du type d'explication, la façon de présenter cette explication est également importante [94]. Ainsi, en faisant une enquête sur de nombreuses études, ils montrent que non seulement fournir une justification et de préférence une explication profonde est nécessaire, mais que, pour être particulièrement efficace, elle doit être conforme au schéma proposé par Stephen Toulmin.

Notons que cette question de la confiance accordée aux traitements réalisés par des logiciels est cruellement d'actualité. Par exemple, dans le contexte de l'internet des objets, la crainte concernant la divulgation de données et la violation de la vie privée ne pourra être dépassée que par l'établissement d'une confiance basée sur des explications claires de ce que fait le logiciel [1]. Nous pouvons aussi citer l'apprentissage automatique qui suscite de nombreuses questions concernant sa validation dans le monde de la certification avionique et qui provoque une demande de « *droit à l'explication* [2] » dans le cadre de son usage pour prendre des décisions impactant des citoyens [90].

Dans la lignée de ces travaux, et plus précisément des travaux cherchant à caractériser une bonne inférence, il est possible de s'intéresser à l'argumentation qui vise à justifier de la correction d'un artefact. Précédemment, au travers de l'utilisation des méthodes formelles, nous avons cherché à établir des preuves mathématiques. En portant notre attention sur les aspects justification, nous opérons un glissement du raisonnement déductif, de la preuve logique, vers une forme de raisonnement plus informel. Cet aspect informel ne doit pas nous arrêter dans notre démarche. En effet, il est possible de dégager des structures permettant de capturer la rationalité de telles argumentations et de définir des approches pour prévenir les raisonnements fallacieux. Plus précisément, comme nous verrons plus en détail par la suite, je me suis intéressé à la théorie de l'argumentation :

— premièrement, pour une aide à l'accréditation de simulations numériques

1. International Joint Conferences on Artificial Intelligence
2. Traduit de : *right to explanation*.

aéronautiques (dans le cadre du projet européen TOICA [1]), pour la certification avionique (projet MIMOSA [2]) ainsi que pour définir un processus industriel (dans le cadre d'une collaboration directe avec un avionneur),

— deuxièmement, pour la création et l'automatisation d'un processus de justification dans le contexte de la certification médicale (encadrement de la thèse de Clément Duffau [3]) et dans le contexte avionique (projet Phylog de la Direction Générale de l'Aviation Civile).

3. Argumentation et certification

De façon très schématique, la certification peut être vue comme une inférence permettant de conclure qu'un artefact est utilisable pour un *emploi donné* sur la base d'un ensemble de justifications. Cependant, cet ensemble de justifications n'est pas un continent plat, sans aucune structure ni logique. Il doit prendre la forme d'une argumentation convaincante, agrégeant des éléments de preuve qui peuvent être, par exemple, les résultats d'opérations de Vérification & Validation (V&V) tels que des tests, des rapports de relecture ou des preuves. Cette argumentation n'étant pas formelle, ou du moins pas dans son intégralité, il paraît vain de chercher à en établir la validité au sens logique du terme, de façon uniquement mathématique. Nous ne sommes donc pas dans un univers fait d'abstractions mathématiques. Dès lors, pour évaluer le bien-fondé d'une telle inférence, nous avons choisi de nous tourner vers les travaux de l'Argumentation et plus précisément vers le modèle du philosophe britannique Stephen Toulmin.

C'est en 1958, dans son livre *The Use of Argument*, que Stephen Toulmin expose son modèle [180]. Son but était de définir une structure aidant à évaluer la validité d'un jugement émis sur la base de faits. Ce modèle est aujourd'hui enseigné dans des universités et parfois même dans des écoles afin d'expliquer les mécanismes de l'argumentation [196, 100, 93, 157].

Dans le modèle de Toulmin, toute argumentation est composée d'une *conclusion* [4], « *conclusion dont nous cherchons à établir le bien-fondé* » [5], et de *raisons*, ou *faits*, « *les faits sur lesquels nous nous appuyons pour fonder la thèse* » [6]. Très basiquement, pour Toulmin qui a une vue légaliste, bien argumenter consiste à énoncer une conclusion en s'appuyant sur des faits.

1. Projet européen FP7, TOICA pour *Thermal Overall Integrated Conception of Aircrafts*
2. MIMOSA pour Means of engIneering for MOdelling and analysis of modular embedded aeronautic Systems and Architectures.
3. Thèse CIFRE, en partenariat avec la société Axonics, soutenue en 2018.
4. Nous avons ici choisi la traduction suivante : conclusion pour *claim*, raison ou fait pour *data*, garanties pour *warrant*, fondements pour *backing*, qualificateurs modaux pour *qualifier* et réfutation pour *rebuttal*.
5. Traduit de : *conclusion whose merits we are seeking to establish*.
6. Traduit de : *the facts we appeal to as a foundation for the claim*.

Les données sont cryptées avec OpenSSL et une clé de 128 bits	$\longrightarrow$	*Donc* Certainement les informations contenues dans le fichier ne sont pas accessibles à quelqu'un de non autorisé *Sauf si* quelqu'un vole la clé
	Vu que une clé de 128 bits suffit pour protéger les données d'après l'Advanced Encryption Standard	
	En vertu de (*B*) l'Advanced Encryption Standard publié en 2001 préconise une taille de clé de 128 bits (FIPS PUB 197)	

Figure 1 – *Exemple du schéma de Toulmin pour : "seules les personnes autorisées peuvent lire les données protégées"*

A ces faits viennent se rajouter des informations relevant du processus de raisonnement. Ce sont des informations additionnelles qui viennent expliciter en quoi l'inférence est acceptable. Typiquement, dans le domaine légal, cela correspond à un article de loi. Toulmin écrit d'ailleurs que cette distinction « *est semblable à la distinction établie dans les cours de justice entre les questions de fait et les questions de droit* »[1]. Ces données supplémentaires, parfois utilisées de manière implicite, sont appelées *garanties*. Les garanties sont donc ce qui permet le passage des faits à la conclusion, elles justifient l'inférence. Distinguer raisons et garanties n'est pas toujours chose aisée, les garanties attestent de la solidité de l'argumentation, elles sont générales, alors que les raisons dépendent plus de données considérées vraies dans le cadre d'une argumentation précise. A la notion de garantie, Toulmin ajoute le concept de *fondement*, « *d'autres garanties, sans lesquelles les garanties elles-mêmes n'auraient ni autorité ni crédit* »[2]. Les fondements sont la justification de pourquoi des garanties sont acceptables de façon indiscutable.

Une conclusion n'ayant pas toujours un caractère absolu, il est possible d'exprimer des réserves avec des *qualificateurs modaux*. Les qualificateurs modaux correspondent à des notions telles que « *possiblement* » ou « *probablement* ». Enfin, à la conclusion se rajoutent des conditions de *réfutation*. Ces conditions expriment les circonstances dans lesquelles la conclusion ne serait pas vraie, en d'autres termes les exceptions possibles.

La Figure 1 représente un exemple d'application du modèle de Toulmin. Ici, la conclusion « *seules les personnes autorisées peuvent lire les données protégées* » est établie à partir de la raison « *les données protégées sont cryptées par un logiciel avec*

1. Traduit de : *is similar to the distinction drawn in the law-courts between questions of fact and questions of law*.

2. Traduit de : *other assurances, without which the warrants themselves would posses neither authority nor currency*.

une clé de 128 bits ». Le passage des raisons à la conclusion s'effectue sur la base de la garantie qu'« *une clé de 128 bits est suffisante pour protéger les données* », garantie que nous avons en vertu de l'« *Advanced Encryption Standard* », qui définit la taille d'une clé suffisante pour une clé de cryptage. Ce standard est ici le fondement de la garantie. Dans notre exemple, comme indiqué par le qualificateur modal « *certainement* », la conclusion n'est pas toujours vraie. Un exemple possible de réfutation de la conclusion est le cas où « *quelqu'un vole la clé* ».

3.1. Organiser les justifications

Notre objectif est de structurer des justifications soit pour aider un « *appliquant* »[1] à la constitution d'une argumentation de certification, soit pour faciliter l'évaluation de celle-ci par une autorité. Notons que cette argumentation ne peut pas être fabriquée ex nihilo, elle doit faire partie d'un processus qui accompagne les différentes étapes de V&V. A chaque étape de V&V, l'argumentation est construite en agrégeant les preuves et les documents produits qui constituent autant d'éléments de justification. Cependant, nous ne voulons pas nous concentrer seulement sur un pas unique de raisonnement, mais établir des conclusions complexes impliquant plusieurs étapes de raisonnement. Toutes ces étapes de raisonnement, prises ensemble, forment ce que nous nommons un *diagramme de justification*. En ce sens, nous sommes conformes à la définition du raisonnement donné dans certains travaux de psychologie où le raisonnement consiste en un effort volontaire pour coordonner des inférences dans le but d'aboutir à une conclusion justifiée [134].

A l'origine de ces travaux, nous appelions ce diagramme : *diagramme d'argumentation*, mais, afin de lever toute ambiguïté, nous avons finalement opté pour une nouvelle dénomination utilisant justification plutôt qu'argumentation. En effet, nous ne nous inscrivons pas vraiment dans la même veine que les travaux classiques en théorie de l'*argumentation* [17]. Ceux-ci, pour la plupart, se concentrent sur la façon de résoudre les conflits entre un ensemble d'arguments, soit en s'appuyant sur le travail séminal de Phan Minh Dung [66], soit au travers d'un processus dialectique [154].

Pour guider la construction d'un tel diagramme, j'ai défini un modèle d'argumentation qui correspond à un pas de raisonnement, à l'inférence non formelle permettant de déduire une conclusion à partir de faits [147, 148]. Fruit de tâtonnements et d'expérimentations menés dans le cadre de divers projets et expérimentations industrielles, mon modèle diffère quelque peu de celui de Toulmin. Contrairement à Stephen Toulmin, ne cherchant pas à définir un schéma pour l'argumentation légale, et encore moins à caractériser ce qu'est l'argumentation dans ses aspects philosophiques, j'ai adapté le modèle de Toulmin au besoin

1. Dans le cadre de la certification, l'appliquant est l'organisme, ou la société, qui présente une demande officielle de certification.

spécifique de l'accréditation d'artefacts techniques.

La première différence entre les deux modèles consiste en la suppression des qualificateurs modaux. Dans le cadre de la certification, une conclusion est acceptable ou ne l'est pas. Nous ne nous intéressons pas à des conclusions qui seraient « *en général* », « *parfois* », etc. acceptables. Le but n'est pas qu'un artefact soit parfois correct ou réponde souvent à une exigence, mais bien d'établir s'il y répond ou pas.

Ensuite, ce nouveau modèle conserve bien évidemment le concept de garantie, mais, pour être en accord avec le vocable utilisé dans le cadre des *safety cases*, elle est renommée *Stratégie*. Pierre angulaire du raisonnement, la stratégie permet de légitimer le passage des raisons à une conclusion. Dans le cadre de l'ingénierie, la stratégie correspond souvent à la méthode utilisée pour établir le passage des faits à la conclusion. Par exemple, si la conclusion est « *les tests sont suffisants* », une stratégie possible pourrait être « *couverture des tests* » et pour la conclusion « *le modèle de turbulence est acceptable* » une stratégie possible serait « *déjà utilisé dans des projets similaires* ». Comme Toulmin, nous accolons à la stratégie des *Fondements*. Les fondements sont l'explication de pourquoi une stratégie est applicable dans un cas précis. Ils détaillent les raisons pour lesquelles une stratégie, et donc une méthode, est acceptable. Imaginons une stratégie qui consiste à suivre un protocole défini dans une norme, alors la stratégie est l'application de ce protocole et le fondement est la norme et les raisons pour lesquelles la norme est applicable.

Nous ajoutons le concept de *domaine d'usage* correspondant au fait qu'une stratégie, et donc une argumentation, n'est valable que dans un contexte précis qu'il peut être nécessaire d'expliciter. Le domaine d'usage donne les conditions précises d'utilisation et les limites d'une méthode. Si nous reprenons l'exemple de l'application d'une norme alors le domaine d'usage décrira dans quels domaines et dans quelles conditions cette norme est applicable.

Nous remplaçons la notion de réfutation, cas particuliers où la conclusion ne s'applique pas, en *limitation* de la conclusion. Contrairement à la réfutation, la limitation n'est plus un cas particulier dans laquelle la conclusion ne serait plus valide, mais le cadre dans lequel la conclusion est vraie. Par exemple, un artefact peut réaliser sa fonction seulement quand il fonctionne dans des températures comprises entre zéro et cent degrés, cet intervalle de température étant ici une limitation de la conclusion que l'artefact réalise sa fonction.

Pour finir, à la place de *fait*, nous employons le terme *éléments de preuve*. Dans le contexte de l'argumentation légale, les éléments de preuve sont des faits et des opinions présentés comme des preuves pour une affirmation [159]. Si l'on met de côté les vérités communément admises, les éléments de preuve ont la particularité de reposer sur l'autorité de celui qui les énonce. L'acceptation d'un fait ne repose donc plus sur le fait lui-même, mais sur la confiance que l'on accorde à celui qui l'énonce. Pour nous, les éléments de preuve se divisent en deux catégories, tout d'abord des faits, que l'on considère comme établis, dont la validité est évaluée

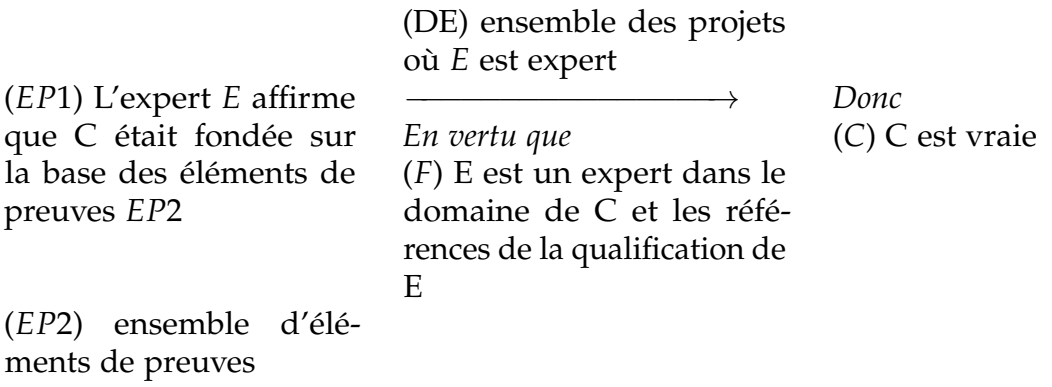

Figure 2 – *Exemple de modèle de Justification : "jugement d'expert"*

hors de l'argumentation, par exemple par une autorité de certification. De tels éléments sont, par exemple, un résultat trouvé dans un article scientifique, une information donnée par un expert, des comptes rendus de tests, une pratique définie dans une norme ou le résultat d'un calcul, ensuite. La deuxième catégorie d'éléments de preuve concerne les éléments qui sont eux-mêmes la conclusion résultant d'une autre étape d'argumentation.

Concernant les éléments de preuve, revenons un instant sur le problème de la régression infinie que nous évoquions en introduction. S'il n'existe pas de solution philosophiquement définitive à ce problème, nous pouvons toutefois apporter une réponse dans le cadre de l'ingénierie. Toute argumentation, ou démonstration, se fonde sur des vérités préétablies. Par exemple, une démonstration dans un système formel postulera toujours qu'un ensemble d'axiomes est vrai. Ces axiomes sont vrais par nature, il n'existe pas de démonstration pour prouver leur validité. Ils sont l'élément de base de tout raisonnement dans ce système. De façon analogue, toute argumentation repose sur un ensemble de postulats acceptés par celui qui énonce la démonstration ainsi que par son auditoire. Dans le cadre de la justification de la correction d'un artefact, la régression des justifications prend fin quand l'argumentation prend pour éléments de preuve, des faits qui sont acceptés par l'autorité et considérés comme vrais.

Finalement, nous marchons sur les traces de Toulmin et, comme lui, nous cherchons au travers de notre modèle à capturer les raisons qui permettent d'établir une conclusion à partir d'éléments de preuve. Par l'usage d'un fondement, notre modèle met en lumière la méthode employée pour justifier l'inférence et les raisons pour lesquelles cette méthode est applicable. Prenons l'exemple d'un projet où un artefact serait considéré correct à partir du jugement d'un expert basé sur un ensemble de preuves. L'utilisation de notre modèle oblige à expliciter les fondements, comme la qualification de l'expert, et le domaine d'emploi, le cadre d'expertise de l'expert, par exemple son domaine ou l'ensemble des projets pour lesquels il est considéré comme expert. La Figure 2 reprend cet exemple. Comme

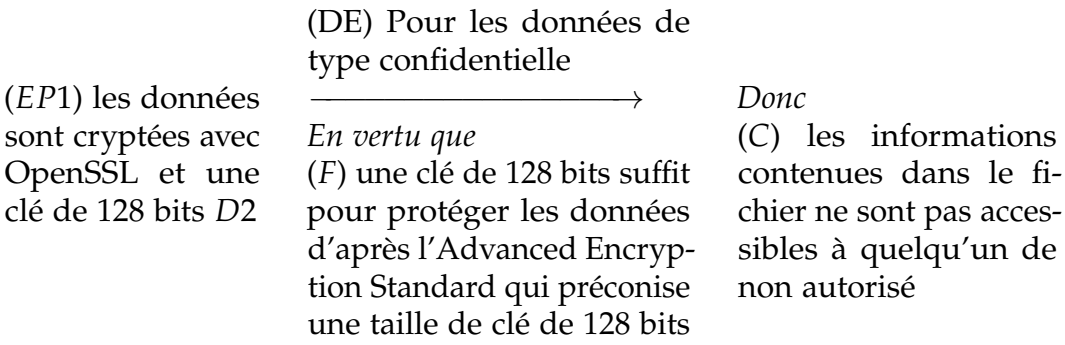

Figure 3 – *Exemple de modèle de Justification : "seules les personnes autorisées peuvent lire les données protégées"*

nous le voyons, il y a deux éléments de preuve, le document qui rapporte le jugement de l'expert et les éléments de preuve sur lesquels se base l'expert. Notons que ces éléments font sûrement, pour chacun d'eux, l'objet d'une argumentation. De plus, les éléments indiqués sur le modèle renvoient nécessairement à des documents. Dans la pratique, nous ne nous satisferions pas d'un simple *références de la qualification*, mais il faudrait indiquer les natures de ces références et les documents qui les attestent.

Sur la Figure 3 nous reprenons l'exemple des données cryptées. Comme nous pouvons le voir, la garantie et les fondements ont été fusionnés, mais nous avons ajouté un domaine d'usage, c'est le cadre dans lequel s'applique cette argumentation. A charge pour le relecteur ou l'autorité, de vérifier que l'*Advanced Encryption Standard* est bien applicable dans ce domaine d'usage.

3.2. Applications pratiques

Un des objectifs de notre modèle est de pouvoir être utilisé par des êtres humains. Il peut servir à organiser et structurer l'ensemble des documents de V&V, être intégré dans des outils informatiques pour faire le lien avec les exigences ou encore indiquer lorsque certaines informations sont manquantes (comme le domaine d'usage ou un support, . . .). De plus, agrégé sous la forme de diagramme de justification, il fournit une vue d'ensemble qui permet de saisir rapidement les tenants et les aboutissants d'une argumentation. Bien évidemment, cette représentation n'est que visuelle, mais, mise en œuvre dans un logiciel, chaque boîte de cet élément est censé renvoyer à des documents.

L'objectif du projet TOICA, projet européen composé de trente-deux partenaires de huit pays différents, était la modélisation et la simulation multidisciplinaires pour la conception d'architectures aéronautiques. Etant donnée la complexité des modèles et des simulations, des revues de conception étaient réalisées régulièrement, le but de ces revues étant d'évaluer la maturité des artefacts produits et de s'assurer qu'ils sont prêts pour le passage à l'étape suivante du cycle

de développement. L'ensemble de ces modèles et simulations numériques s'inscrit dans un processus plus global visant à déterminer des architectures d'avions prometteuses. Ainsi, c'est au cours de réunions d'experts que les solutions de design qui méritent d'être étudiées vont être choisies sur la base des performances évaluées par les simulations. Le processus global créé dans TOICA fait appel à des simulations numériques, des méthodes de choix, des analyses d'experts et l'emploi de logiciels.

Dans ce contexte, il est très difficile pour les experts d'évaluer le bien-fondé des architectures proposées, car : il y a de nombreuses données d'entrées, des critères de sélection, des contraintes spécifiques et des résultats intermédiaires s'appuyant sur des prérequis complexes. Afin de les aider dans cette tâche, tous les éléments importants, comme la définition d'architecture, les caractéristiques de rendement, les exigences et les résultats de simulation 3D sont présentés dans un logiciel unique.

Avec Claude Cuiller, Sanjiv Sharma et Vincent Tuloup [152], nous avons réalisé des diagrammes de justification, basés sur mon modèle, pour faciliter la communication entre les parties prenantes et mettre en évidence des éléments clés permettant de comprendre pourquoi il est possible d'avoir confiance dans les différents résultats de calcul. Pour ce faire, nous avons interrogé des spécialistes dans divers domaines, des architectes avion et nous avons étudié 70 documents différents (des supports de présentations, des géométries numériques, des documents sur les simulations, de la documentation informelle et des critères de préférences stockés dans des feuilles de calcul).

Nous avons utilisé un diagramme de justification lors d'une revue pour un choix d'architectures possibles. En montrant les éléments clés (principalement des éléments V&V), ce diagramme a aidé les décideurs à comprendre pourquoi le résultat est fiable et quels sont les points faibles. De plus, *Dassault Systèmes* a mis en place un prototype de diagramme de justification dans son logiciel 3DExperience. Le logiciel offre de nouvelles possibilités par rapport à la version papier. Par un simple clic, il est possible d'accéder à toutes les données et documents liés au diagramme. Cette fonctionnalité augmente la confiance grâce à un accès direct aux informations liées à l'argumentation. De son côté, à l'issue de cette revue, l'architecte avion a souligné que ce diagramme lui permettait d'avoir confiance dans les résultats et mettait en évidence les conséquences des incertitudes liées à certains paramètres, atout clé pour décider des marges de conception à prendre en compte pour gérer ces incertitudes. De plus, il a apprécié le fait d'avoir à sa disposition l'information pertinente, le diagramme de justification permettant de lier chacun des 70 documents de départ à des éléments de l'argumentation. Dans le cadre de cette revue, concernant une conclusion précise, il a été possible de voir que seulement 16 documents intervenaient dans le raisonnement. Par conséquent, pour vérifier le bien-fondé de cette conclusion, grâce à notre diagramme, l'architecte disposait de la liste des documents pertinents et de leur articulation dans le

raisonnement.

Nous avons eu l'occasion d'appliquer notre modèle dans un autre contexte, celui de la fabrication. Nous nous sommes servis des diagrammes de justification pour établir les documents et les opérations de validation nécessaires pour avoir confiance dans un processus d'évaluation de la charge de travail sur une chaîne d'assemblage. Dans cette étude, nous n'avons pas organisé une documentation existante a posteriori, mais, nous situant *a priori*, nous avons cherché à comprendre quels étaient les éléments nécessaires pour justifier cette évaluation. A l'aide de notre modèle, nous avons pu définir quels étaient les éléments de preuve nécessaires. L'usage de nos diagrammes servait ici, non seulement à trouver les éléments de V&V, mais également à ne pas demander d'éléments inutiles, qui n'interviennent pas dans la justification.

Pour finir, nous avons appliqué notre modèle dans le cadre du projet MIMOSA, projet réalisé avec les autorités de certification qui ciblait le logiciel et les architectures embarquées avioniques [19]. Dans ce projet, un support de certification pour l'*avionique modulaire intégrée* a été défini en étroite collaboration avec les autorités. De notre côté, nous avons défini, avec l'aide d'experts, des argumentations spécifiques aux problématiques temps réel du logiciel embarqué. Ce projet a permis de confirmer que l'utilisation de notre modèle pour présenter les éléments de V&V pouvait aider l'autorité de certification à trouver les manques ou la mauvaise utilisation d'éléments de preuve. De plus, comme dans les autres projets, l'autorité de certification a pu constater que les diagrammes de justification permettent d'avoir une vue claire de l'ensemble de la documentation.

4. Argumentation et patron de justification

Si l'usage de notre modèle permet de structurer une argumentation, il ne nous prémunit nullement du risque de réaliser des raisonnements fallacieux, des raisonnements qui semblent être corrects, mais qui pourtant ne le sont pas. Dans le cadre de raisonnements informels, non mathématiques, l'évaluation du caractère fallacieux ne peut être réalisée que par un être humain, mais il est pourtant possible d'apporter une aide pour se prémunir de ces raisonnements entachés d'erreurs.

Une première voie dans ce sens est celle suivie par Aristote et Richard Hamblin qui cherchaient à identifier et cataloguer les erreurs de raisonnement afin de les reconnaître plus facilement. Une autre voie possible consiste à prendre le problème par l'autre versant, c'est-à-dire chercher à cataloguer les raisonnements non plus fallacieux, mais les raisonnements corrects. Dans son ouvrage *Système de logique déductive et inductive* [130] écrit en 1843, John Stuart Mill s'intéresse non seulement aux raisonnements fallacieux dans le Livre II et V, mais consacre la plus grande partie de l'ouvrage aux raisonnements corrects. Ainsi, dans le livre II nommé *De l'induction*, Mill cherche à caractériser les différentes méthodes d'induction. L'idée

n'est plus d'aider à identifier des erreurs de raisonnement, mais de fournir des cadres pour raisonner correctement.

Cette approche visant à donner des formes génériques de raisonnements corrects trouve écho dans le concept des *patrons de conception*. En ingénierie, l'approche par patrons de conception est un moyen de décrire un problème récurrent et la solution type associée [4]. Un patron de conception associe à la définition d'un problème conceptuel, générique, une solution considérée comme « *bonne* ». Le but premier de l'approche par patron est de capitaliser des solutions basées sur l'expérience. L'utilisation de patrons est particulièrement efficace quand il s'agit de communiquer de bonnes pratiques d'ingénierie. Notons qu'il n'existe pas de langage unique pour exprimer des patrons puisque l'on trouve leur usage dans un large éventail de domaines très hétérogènes comme, par exemple, la conception de bâtiments ou la conception de logiciels [86].

Dans cet esprit, nous pouvons à notre tour chercher à définir des patrons de justification. Ces patrons consisteraient alors en des diagrammes génériques qui listent, pour une solution donnée, les éléments de preuves et la conclusion attendus. Conforme à notre modèle, chaque patron définirait, pour un objectif générique donné, quelles sont les prémisses nécessaires et quelles sont les restrictions et les conditions d'utilisation d'une stratégie.

L'usage de tels patrons est particulièrement intéressant dans le cadre de la certification. En effet, la plupart des standards et des guides utilisés dans la certification ont le désavantage d'être complexes à appréhender. Définir ces standards aux moyens de patrons de justification expliciterait clairement les raisonnements sous-jacents aux objectifs demandés, ce qui n'est pas toujours le cas. De plus, redéfinir les standards actuels à l'aide de patrons de justification permettrait d'identifier les lacunes et les points d'ambiguïté existants. Pour finir, l'usage de patrons pourrait aller dans le sens d'une rationalisation des diagrammes de justification. Avec un patron, un diagramme ne prendrait plus une structure ad hoc définie au cas par cas pour une conclusion donnée, mais serait conforme à une structure générique, définie dans le standard, devenant ipso facto plus simple à évaluer pour une autorité de certification..

Dans cet esprit, plusieurs travaux ont cherché à promouvoir une approche par patrons. Ainsi, Tim Kelly et John McDermid ont défini un format collectant les informations importantes relatives à un patron, comme les motivations pour sa création ou ses limites, dans une optique de réutilisation entre différents projets [115]. De son côté, John Knight, dans une proposition pour une nouvelle version du standard avionique DO178, considère que celui-ci devrait fournir un ensemble de patrons de justification [118]. Pour finir, dans un article sur les standards utilisés dans le cadre des dispositifs médicaux, les auteurs donnent une argumentation étayée pour l'utilisation de patrons de justification dans les normes [193]. Notons qu'aucune de ces approches n'a cherché à caractériser la relation entre un patron et ses instances ou à définir une méthodologie permettant de concevoir de tels

patrons.

Si nous revenons à une perspective plus large, l'argumentation de la correction d'artefact technique, ces patrons de justification peuvent être vus suivant deux dimensions : l'activité et le niveau de maturité. L'activité correspond au moyen mis en œuvre pour démontrer une conclusion, c'est l'objet de la stratégie. La maturité dépend du niveau de détail attendu pour l'argumentation. Ainsi, un patron spécifique à une activité pourra se décliner sous plusieurs patrons, certains très simples, pour des activités peu critiques et/ou de faible niveau de maturité, comme une étude de conception préliminaire, tandis que d'autres patrons pourront être très détaillés s'ils sont utilisés dans un contexte très strict comme la certification d'artefacts critiques.

Bien évidemment, un patron est étroitement lié à la notion de stratégie. Il correspond à une activité particulière comme l'utilisation d'un logiciel de preuve, la réalisation de tests ou la validation par un comité d'expert. Cependant, un patron ne se résume pas à sa stratégie. Une activité définit une stratégie, mais aussi un domaine d'utilisation, une conclusion, un fondement et une liste d'éléments de preuve obligatoires.

4.1. Définir des patrons de justification

Notre objectif est de disposer d'un ensemble de patrons de justification dédiés à un domaine. Cet ensemble de patrons forme une bibliothèque dans laquelle il est possible de venir piocher pour pouvoir guider et structurer une justification de correction, dans le cadre de la réalisation d'un artefact. Les patrons peuvent donc être vus comme des guides, listant les éléments nécessaires à la justification d'un objectif de certification. La réalisation de cette bibliothèque ne peut être faite que sur la base d'experts qui vont définir ces patrons en fonction de leurs connaissances, de bonnes pratiques établies, de normes, d'exigences de qualité, etc.

Tout le problème consiste à ne pas commettre d'erreur dans l'écriture de ces patrons, puisque ce sont eux qui sont censés garantir la validité d'un raisonnement. Il nous faut donc être attentifs à deux points importants : la légitimité des experts et les biais cognitifs.

Concernant la légitimité, les experts qui réalisent les patrons doivent être considérés comme des experts du domaine par les personnes qui vont utiliser les patrons. Cette légitimité ne peut être acquise que par des références, une reconnaissance de la compétence par les pairs. De notre expérience, nous pouvons faire remarquer que, dans le cas particulier de la certification faisant intervenir une autorité, les patrons sont très bien accueillis s'ils sont réalisés conjointement par des experts travaillant du côté de ceux qui fabriquent l'artefact et par des experts appartenant à l'autorité de certification.

Concernant les biais cognitifs, il existe de nombreux biais qui influencent le raisonnement humain. Parmi eux, existe la tendance à considérer sa propre

interprétation subjective comme la vérité sur la réalité. Des travaux en psychologie ont montré que l'une des implications de ce biais cognitif est notre incapacité à juger « *notre compréhension* » et notre méconnaissance de ce que nous connaissons. En d'autres termes, nous pensons comprendre et avoir des explications valides pour des phénomènes que nous ne comprenons pas vraiment. Sur des sujets sensibles, la situation est telle que nous pouvons largement surestimer la qualité de nos justifications et de notre raisonnement [79]. Il est néanmoins possible de compenser ce biais par le dialogue. Comme le confirment de nombreuses études, raisonner en groupe, de façon collaborative, est plus efficace que de raisonner de façon individuelle, surtout pour résoudre des problèmes de raisonnements et de logiques [1] [135, 181, 123].

Nous avons donc défini une méthodologie qui s'inscrit dans un travail collaboratif, où un groupe d'experts, aidé d'un facilitateur, élaborent collaborativement des patrons de justification [48].

Parallèlement, durant la thèse de Clément Duffau que j'ai encadrée avec Mireille Blay-Fornarino, nous avons mené différents travaux. Nous avons tout d'abord clarifié les relations entre les différents concepts de notre modèle, ainsi que les liens entre patron de justification et instance, nous avons défini les premières pistes de ce que pourrait être un logiciel d'aide à la conception, la gestion et l'utilisation de tels patrons et nous avons établi des liens avec d'autres formalismes existants [63].

Ensuite, dans le but d'utiliser clairement et sans ambiguïté les patrons de justification, nous avons défini une sémantique formelle [65]. Cette sémantique s'appuie sur une relation, dite de conformité, qui différencie un patron de l'usage du patron. L'idée sous-jacente à cette notion de conformité est celle d'un *raffinement*, qui pourrait être vu comme « *est un modèle de* » ou « *est une spécialisation de* ».

De plus, pour pouvoir construire des patrons et des diagrammes et les utiliser, nous avons défini un méta-modèle, méta-modèle qui est bien évidemment en conformité avec notre sémantique. A partir de ce méta-modèle, Clément Duffau, a réalisé un logiciel permettant d'éditer des patrons et des diagrammes, gérer leur cohérence et leur évolution dans le cycle de vie d'un produit.

Dans une optique de standardisation des pratiques, l'OMG [2] a établi un méta-modèle pour l'*assurance case*, le *Structured Assurance Case Metamodel* (SACM) [141]. Basée sur le modèle de Toulmin, le SACM diffère de notre modèle sur quelques points comme, la non nécessité de justifier l'inférence, l'absence des garanties

1. David Moshman et Molly Geil ont montré sur un problème de raisonnement, avec une cohorte de 20 groupes et de 32 individus, que 75% des groupes trouvaient la bonne réponse pour 9.4% des individus seuls [135]. Il est à noter d'ailleurs que les groupes construisent des argumentations beaucoup plus sophistiquées et de meilleures qualités qu'un individu seul.

2. L'Object Management Group (OMG) est un consortium international, à but non lucratif, de normalisation de l'industrie informatique. Il est composé de représentants du gouvernement, de l'industrie et académiques.

de Toulmin, et la possibilité qu'une inférence puisse avoir plusieurs conclusions distinctes. En dépit de ces divergences, nous avons montré qu'il était possible de transformer nos diagrammes de justification en diagrammes de type SACM, en d'autres termes, notre proposition est conforme au standard de l'OMG.

4.2. Applications pratiques

La dernière décennie a vu l'émergence de nouvelles architectures de processeurs, les *multi-cœurs*. Un processeur multi-cœur est une puce intégrant plusieurs cœurs interconnectés soit par un bus partagé soit par un réseau embarqué sur la puce. Bien que ces architectures soient prometteuses en termes de gains de performances, elles sont également un défi pour leur intégration dans un environnement critique pour la sécurité. Dans le cadre du projet DGAC Φ-log, nous avons proposé une approche de certification basée sur des patrons de justification pour de tels processeurs. Nous nous sommes notamment focalisés sur le CAST32 [38] qui est un document de la *Federal Aviation Administration* (FAA) proposant des objectifs de certification pour les multi-cœurs. Avec un groupe d'experts nous avons défini des patrons permettant de répondre à ces objectifs de haut-niveau. L'usage de nos patrons de justification a permis de clarifier le CAST32, de mettre en évidence des ambiguïtés, de fournir une aide générique de haut niveau pour répondre aux objectifs de certification et de proposer des modifications dans ces objectifs[28, 18, 48].

Par ailleurs, j'ai eu l'occasion de participer au projet RESSAC, projet réunissant plusieurs experts internationaux en certification des systèmes embarqués aéronautiques et visant à définir de nouvelles modalités de certification, pour les aspects système, logiciel et matériel, basées sur des méta-objectifs. Cette démarche s'inscrit dans un mouvement général de rationalisation (et de simplification) des normes avioniques fortement poussé par l'initiative *Streamlining Development Assurance* de la FAA [1]. Dans le cadre de ce groupe de travail, nous avons montré que les patrons de justification pouvaient être un élément de réponse puisqu'ils mettent l'accent sur l'explication de pourquoi un artefact est certifiable plutôt que sur la démonstration par un processus de respect de normes.

Dans le cadre de sa thèse, Clément Duffau a pu appliquer l'approche dans le domaine médical. Plus précisément, il a défini avec des experts des patrons pour établir la conformité d'essais cliniques et de développements logiciels avec un ensemble de normes. L'approche s'inscrivant dans un contexte d'agilité, il a montré comment il était possible de définir des patrons et des instances, en gérant l'évolutivité, le temps dans le cadre de l'amélioration continue [64]. De plus, Clément a intégralement outillé l'approche au travers d'une usine de justification.

1. Initiative qui s'inscrit dans le cadre du « *FAA Modernization and Reform Act of 2012* » et qui consiste à réformer et rationaliser le processus de certification (« *reforming and streamlining the FAA's regulatory certification process* »)

Cette usine de justification est une chaîne d'outils logiciels qui permet d'éditer des patrons, générer automatiquement des diagrammes de justification, gérer le cycle de vie, et donc les évolutions, des patrons et des diagrammes. Elle se compose pour partie de logiciels open source et pour partie d'un logiciel, nommé *Justification Factory*, réalisé pendant cette thèse.

Chapitre 4

Perspectives

« *La fin est importante en toutes choses.* »

葉隠

1. Correction des artefacts . . .

Concernant la suite directe des travaux déjà menés sur les Diagrammes de Justification, nous pouvons nous intéresser aux *éléments de preuve*. Les éléments de preuve sont les éléments unitaires sur lesquels repose la justification. Ce sont des faits qui sont acceptés et considérés comme vrais. Des éléments de preuve possibles sont des résultats de calcul, une preuve formelle, la parole d'un expert ou un élément donné dans une norme. Nous pouvons dresser un parallèle, ici, entre élément de preuve et témoignage d'expert dans un procès. En effet, dans les deux cas, ces éléments relèvent d'un niveau d'expertise dont ne dispose pas forcément l'autorité, que cela soit un juge, un jury, ou une autorité en charge de la certification. Notons qu'aujourd'hui dans le droit anglo-américain, le témoignage d'expert est tellement utilisé qu'il est devenu une preuve à part entière [1]. Cette spécialisation de la connaissance scientifique et technique constitue un dilemme pour le système juridique, si les décisions juridiques tendent à se fonder sur des connaissances scientifiques, les tentatives d'intégrer les éléments scientifiques dans la procédure judiciaire soulèvent bien des problèmes [87].

1. A ce sujet, un comité de l'académie américaine des sciences a produit un rapport expliquant qu'il ne faut pas accorder une confiance aveugle à ce que l'on qualifie à tort de « *preuve scientifique* », alors qu'il s'agit d'une « *présomption* » [42].

Depuis de nombreuses années, Douglas Walton cherche à définir un schéma représentant la parole d'un expert et à analyser comment une parole d'expert peut être réfutée ou affaiblie [156, 191]. Pour cela, il a défini ce qu'il appelle des *questions critiques*. Le but de cet ensemble de questions est d'évaluer et d'analyser de manière simple la parole d'un expert. Le point central de son approche est que la parole d'un expert n'est pas *vraie* dans l'absolu, elle est vecteur de plausibilité et les questions critiques cherchent à mettre en lumière les éléments permettant de considérer la parole de l'expert comme plausible.

Walton propose six questions critiques de base. La première question relève de la compétence de l'expert. La deuxième cherche à établir si l'expert possède bien des connaissances dans le domaine. La question trois cherche à vérifier la causalité entre la parole de l'expert et ce qui est énoncé comme élément de preuve. La quatrième question interroge la fiabilité de l'expert, s'il est digne de foi. La question cinq cherche à établir si ce que dit l'expert est conforme avec la parole d'autres experts. Pour finir, la dernière question interroge les fondements de ce qui est soutenu par l'expert, sur les preuves sous-jacentes à ce qu'énonce l'expert. A partir de là, suivant le contexte, chaque question critique peut se raffiner, en sous-questions plus précises.

Sur le modèle de Walton, nous pourrions chercher à établir des questions critiques pour chaque famille d'éléments de preuve. Nous pourrions, ainsi, ajouter à nos patrons de justification un ensemble de questions critiques dédiées qui viendraient ajouter des informations permettant d'évaluer l'argumentation. Par exemple, dans le cadre de l'usage de résultats issus d'un logiciel, nous pourrions nous interroger sur la crédibilité du logiciel ou sur pourquoi le résultat donné par le logiciel permet d'énoncer ce qui est donné comme élément de preuve. Définir de telles questions demande, comme pour les patrons, un véritable travail de fond avec des experts de chaque domaine concerné.

Toujours concernant les diagrammes de justification, avec Mireille Blay-Fornarino et Clément Duffau nous avons commencé à élaborer une sémantique formelle permettant d'exprimer la relation entre patrons et instances (application pratique du patron) [65]. Disposer d'une telle sémantique permet de définir sans ambiguïté les relations entre les différents concepts manipulés. Pour le moment, en nous focalisant sur la relation entre patrons et instances, nous avons pu caractériser formellement la notion de raffinement de patrons. De plus, la formalisation de la relation de raffinement nous permet de spécifier mathématiquement un ensemble d'opérations et de propriétés, premières briques vers des outils logiciels d'assistance à la gestion de diagrammes de justification. Dans cet esprit, de futurs travaux devront chercher à caractériser formellement les relations entre les différents éléments des diagrammes que sont la conclusion, la stratégie, le domaine d'usage, le fondement et les supports. In fine, une sémantique formelle des diagrammes de justification permettra non seulement d'apporter une aide à la création des diagrammes, mais aussi de caractériser les opérations liées à leur

cycle de vie durant tout le processus de conception et d'évolution de l'artefact.

Au sujet du cycle de vie, une autre piste de recherche consisterait à s'intéresser à la réutilisation et l'évolution d'une argumentation dans le cas d'une évolution ou d'une mise à jour de l'artefact. Prenons l'exemple d'un avion, quand il y a une évolution mineure, un changement de seulement quelques composants, il n'est pas nécessaire de reprendre la certification dans son ensemble, il suffit d'argumenter uniquement sur les points de changements et leur impact. Dès lors, il faut être capable de justifier de la maîtrise de l'impact des changements et de pouvoir reprendre et faire évoluer l'argumentation de certification [1]. Là encore un travail pourra être mené sur les mécanismes permettant de faire évoluer un diagramme d'argumentation dans le cadre de la modification d'un artefact. A cela devrait aussi se rajouter les questions liées à la traçabilité de l'évolution de la justification et de l'artefact.

Si les diagrammes de justification ont été utilisés dans divers contextes comme la simulation numérique ou la certification avionique et médicale, j'ai été impliqué dans chacun de ces projets. Dans l'avenir, il faudra s'attacher à définir une méthode reproductible pour que l'approche par patrons de justification puisse être utilisée dans des projets où aucune des personnes qui ont contribué à la création des diagrammes de justification ne soit impliquée. Cette méthode devra se focaliser sur la définition du processus à mettre en œuvre, les bonnes pratiques et la définition des rôles permettant de définir, d'éliciter, des patrons de justification. Un tel travail devrait s'inscrire hors de toute notation graphique imposée, permettant à chacun de choisir son langage afin de pouvoir exprimer ses patrons de justification.

Il existe encore de nombreux champs d'application des diagrammes de justification. A ce sujet, nous pourrions chercher à faire un pont entre la partie formelle et la partie informelle que nous avons évoquée dans ce mémoire. Ainsi, nous pourrions développer un catalogue des patrons correspondant aux méthodes formelles utilisées dans l'industrie pour établir la correction d'un artefact.

Concernant les exigences, dans la continuité de nos travaux portant sur l'analyse automatique formelle d'exigences écrites en logique, nous pourrions maintenant chercher à analyser des exigences écrites en langage naturel. Pour ce faire, il faudrait coupler une approche basée méthodes formelles et du traitement automatique des langues. Dans le domaine de l'ingénierie, de par les contraintes linguistiques imposées à l'expression des exigences par les guides et les normes, nous pouvons aisément considérer que nous sommes dans le registre d'une langue très normée, avec de fortes particularités langagières. Dès lors, il paraît possible de s'appuyer sur l'utilisation de formalismes comme celui de la Grammaire Catégo-

1. Qualifier s'il s'agit d'une mise à jour ou d'un nouvel artefact, décider de ce qui doit être à nouveau justifié et qualifier l'impact de la modification sur l'ancienne argumentation sont des problèmes extensivement complexes et sensibles d'un point de vue industriel. Pour preuve, des questions se sont rapidement posées au sujet de la certification du Boeing 737 MAX, qui a été considéré comme une version améliorée et remotorisée du 737 NG, suite aux accidents de deux 737 MAX 8 qui ont eu lieu en Indonésie et en Éthiopie.

rielle Combinatoire et Applicative [114] pour traduire des exigences en formules logiques. De futurs travaux pourraient donc s'attacher non seulement à réaliser une transformation des exigences en langage naturel vers la logique, mais aussi à analyser automatiquement ces exigences.

Nous finirons ce tour d'horizon par des considérations plus prospectives. Il est quasiment certain que la complexification des objets conçus par l'homme va continuer, la tendance étant à des objets toujours plus interconnectés, toujours plus « *intelligents* » et s'appuyant sur des briques de bases de plus en plus complexes. Pour s'en convaincre, il suffit de regarder l'architecture matérielle d'un téléphone ainsi que l'ensemble de ses composants logiciels. Nous pouvons dresser plusieurs tendances lourdes menant à cette complexification. Premièrement, la prolifération de l'informatique, avec bien évidemment sa présence dans les voitures, les avions et la téléphonie, mais aussi dans de plus en plus d'objets, comme les aspirateurs ou certains feux de signalisation. Deuxièmement, la tendance aux artefacts connectés et aux agents artificiels qui rajoute un nouveau niveau d'agrégation, il faut en plus de l'artefact penser le système ainsi connecté avec ses interactions. Un exemple de cela est l'ajout de capteurs dans les villes afin d'orienter la circulation et la distribution des flux, que cela soit l'électricité ou la circulation. Pour finir, citons une troisième tendance à la complexification, le fait de devoir concevoir un artefact en intégrant de plus en plus de facettes. C'est notamment le cas de l'avion où il devient nécessaire, dès les étapes de conception, de prendre en compte les contraintes liées à sa fabrication, à sa maintenance ou celles liées à son démontage et son recyclage en fin de vie. Par conséquent, de la même façon que la qualité s'est imposée dans l'industrie manufacturière, le besoin d'établir la correction de tous les artefacts intermédiaires va aussi s'imposer dans l'ingénierie. C'est bien évidemment aujourd'hui déjà le cas dans bien des domaines, mais nous pouvons envisager que cela devienne la norme, notamment dans le monde du logiciel.

En effet, l'industrie logicielle, du moins en partie, semble faire preuve d'une certaine immaturité. Alors que dans l'univers du logiciel se multiplient les métiers, les compétences et les strates (nous pouvons penser, ici par exemple aux systèmes d'exploitation, à la virtualisation, à l'orienté service, etc.), de nombreux développements continuent de se faire sans rigueur, sans plan et sans méthode permettant d'établir la correction. Contrairement aux autres domaines de l'ingénierie, où il est classique d'établir un plan et les moyens de vérification avant de commencer à fabriquer, de nombreux codes sont produits directement, sans aucune étude préliminaire. Dans le futur, qu'elles soient formelles ou argumentatives les opérations de conformité des artefacts produits dans le cadre d'un développement logiciel vont devoir être systématisées. Peut-être verrons-nous un retour aux idées de Walter Andrew Shewhart avec le besoin de définir un étalon pour chaque artefact produit et l'opération de comparaison à cet étalon. Ce ne sera cependant pas chose aisée. Définir un étalon et la correction à un étalon pour des artefacts relevant plus du concept que d'une réalité matérielle est une véritable gageure. Quel est l'étalon,

la fonction, que doit vérifier un modèle conceptuel ? Comment établir cette correction ? Dans ce mémoire, nous avons vu quelques pistes permettant de répondre à ces questions. Concernant les artefacts intermédiaires, nous avons vu qu'il était possible d'effectuer des preuves formelles sur des exigences et des modèles conceptuels. Ce ne sont que des résultats préliminaires qui doivent être poussés plus avant dans la perspective plus large du développement logiciel dans son ensemble. Dans certains cas, définir l'étalon, la fonction est un problème ouvert. Il existe de nombreux logiciels qui peuvent être considérés comme corrects, sans bug majeur, mais ils ne font pas ce que l'on veut, car, finalement, nous ne savons pas vraiment ce que nous voulons. Toujours concernant la correction, les méthodes formelles peuvent se retrouver inopérantes face à la complexité des systèmes et de la fonction à vérifier. Dans de tels cas, l'approche informelle, argumentative, pourrait pallier à ce problème. Là encore, nous ne sommes qu'aux prémisses, car appliquer une approche informelle pour établir la correction demande à agir avec prudence et avec un maximum de garantie.

Dans tous les cas, de par les enjeux de maîtrise de la complexité et la nécessité de disposer de logiciels de qualité, le développement logiciel devra intégrer le *besoin de correction*. Cependant, quelques soient les moyens choisis pour établir cette correction, l'industrie logicielle devra veiller à ne pas répéter les erreurs de l'industrie manufacturière en imposant une division du travail tellement forte qu'elle mène à une perte de sens.

Eloignons-nous des aspects strictement logiciels et revenons à l'ingénierie en général. Comme nous l'avons vu, nous n'avons pas uniquement utilisé les Diagrammes de Justification dans le monde du logiciel. Ce passage de techniques employées dans le monde de l'informatique vers celui de l'ingénierie dans son ensemble semble être une tendance de fond qui devrait également s'appliquer aux méthodes développées dans ce mémoire. Pour pouvoir embrasser des problèmes de plus en plus complexes, comme la fabrication d'un avion, mais aussi les composantes liées à son démantèlement et son recyclage ou son intégration dans un espace aérien dense et connecté, l'ingénierie fait appel à des pratiques issues de l'informatique comme l'utilisation de modèles conceptuels, l'ingénierie des exigences ou le développement agile. Là encore, le besoin d'établir la correction des artefacts produits par l'introduction de ces pratiques va se faire cruellement sentir. Dans un tel contexte, il serait intéressant de généraliser l'usage des techniques présentées ici, qu'elles soient formelles ou informelles, à l'ingénierie dans son ensemble.

2. . . . et autres considérations

2.1. Concevoir des artefacts complexes fabricables

Dans le cadre de différentes études, j'ai été amené à travailler avec des concepteurs et des architectes du milieu aéronautique. Si les collaborations portaient à l'origine sur des problèmes de V&V de simulations numériques, différentes rencontres m'ont amené dans le monde des chaînes d'assemblage et à m'intéresser aux liens existant entre conception et fabrication [1].

Pour certains artefacts techniques particulièrement complexes, tels qu'un avion, un bateau ou un satellite, il est nécessaire de disposer d'un système industriel spécifiquement conçu. Par système industriel, il faut entendre tous les moyens utilisés pour fabriquer et assembler l'artefact : main d'œuvre, machines, usines, logistique, outillage, etc. Dans de tels cas, le système industriel est construit dans un seul but : la construction spécifique de cet artefact. Cependant, il n'est pas rare dans le cycle de développement d'objets aussi complexes, que le système industriel ne soit spécifié, pensé, qu'après la conception complète de l'artefact. Une telle séquentialité conduit invariablement à une faible prise en compte des interactions entre l'artefact et son moyen de fabrication, induisant des contraintes fortes sur le système industriel, qui doit être capable d'assembler l'artefact à tout prix.

Une des pistes possibles pour gérer ce problème de fabricabilité [2] consiste à concevoir les artefacts de manière à simplifier leur fabrication. Pour ce faire, il est nécessaire de prendre en compte, dès le début d'un projet, les contraintes inhérentes aux moyens de production, que cela soit le coût possiblement rédhibitoire de certains éléments ou l'impossibilité matérielle de réaliser certaines conceptions en raison de l'absence d'outils spécifiques. On parle parfois de *Design for manufacturability* ou de *conception simultanée*. La philosophie d'une telle approche est de résoudre les problèmes de fabrication dès les phases de conception et donc de réduire drastiquement les coûts. En outre, elle permet de tirer pleinement avantage des nouvelles méthodes de fabrication comme la robotique ou encore la fabrication additive (impression 3D). Si nous prenons le cas de la fabrication additive, cette dernière permet d'ouvrir de nouvelles possibilités de conception tout en imposant des contraintes telles que la taille de ce qui peut être imprimé ou les matériaux pouvant être utilisés. Ainsi, les architectures conçues avec fabrication additive sont parfois très différentes des conceptions classiques. Par conséquent, il est crucial d'intégrer les possibilités de fabricabilité au plus tôt dans le cycle de développement d'un artefact et, ce faisant, de disposer d'une approche de

1. Je tiens tout particulièrement à remercier ici Claude Cuiller et François Bouissiere qui m'ont ouvert les portes de cette thématique et qui ont toujours su répondre avec patience à mes nombreuses questions de béotien.

2. Nous utilisons le néologisme « *fabricabilité* » pour traduire le terme anglophone « *manufacturability* » qui signifie « *facilement fabricable* » (à moindre coût et de façon fiable).

conception prenant en compte l'artefact et son système industriel.

Dans les approches de type conception simultanée, l'artefact et son système industriel sont conçus en même temps et non séquentiellement. Tous les acteurs d'un projet sont impliqués dès le début afin d'avoir une compréhension globale des objectifs et des interactions entre toutes les activités qui devront être réalisées. La conception simultanée est un concept ancien [169], mais elle est principalement appliquée pour l'optimisation de pièces détachées dans l'industrie automobile [92], et non dans un cadre global pour la conception d'objets complexes tels qu'un avion ou un bateau. Il est donc intéressant de chercher à définir des méthodes et des outils qui permettraient d'appréhender l'artefact et son système industriel comme un tout, que l'on peut optimiser conjointement. Dans la pratique cela signifie, par exemple, de pouvoir évaluer à un stade précoce l'impact, pour un artefact, de différentes architectures possibles sur un système industriel donné et de comparer, pour une même architecture, différents systèmes industriels entre eux. Dès lors, il serait possible d'envisager la conception conjointe du couple artefact/production comme un tout. Par ailleurs, dans le cas d'un artefact déjà existant, une telle approche permettrait d'étudier les changements dans la conception de l'artefact et dans l'organisation du système industriel afin d'être plus efficace et de réduire les durées de fabrication.

C'est donc dans ce but que nous avons commencé, avec des architectes avion Airbus, à définir une approche basée modèle. Sur la base des données de production de la pointe avant de l'A320 [1], nous avons identifié les éléments clés d'une chaîne d'assemblage, puis nous avons résumé ces éléments en un modèle générique [29]. Ce modèle générique permet d'unifier la vue bureau d'études, partie conception, et la vue usine, partie fabrication. Puis, avec Stéphanie Roussel, nous avons proposé, sur la base de ce modèle générique, des modèles de l'avion et de son système industriel à différents niveaux d'abstraction, niveaux correspondants aux différentes phases de spécification et de conception [150, 151]. Ces modèles ne sont que les premiers pas vers une approche plus globale qui reste à définir. Ils pourraient servir à la définition d'une méthodologie de raffinement qui commencerait à un haut niveau d'abstraction et se terminerait au niveau de conception le plus détaillé, ce qui permettrait de pouvoir raisonner à la fois sur la conception de l'artefact et sur la chaîne de fabrication à chaque niveau d'abstraction. Pour le moment, nos modèles étant assez orientés chaîne d'assemblage et industrie aéronautique, il serait donc intéressant de mener un travail de généralisation et d'enrichissement sur la base d'autres cas d'étude.

A ces possibilités de modélisation, nous devons ajouter des outils permettant d'évaluer la performance du couple artefact/production, voire des outils capables de trouver des optimisations. Avec Cédric Pralet et Stéphanie Roussel, nous avons fait un premier pas dans ce sens, dans le cadre de la pointe avant de l'A320, en

1. La pointe avant d'un avion de type A320 correspond à la partie de l'avion qui comprend le poste de pilotage et tout le fuselage jusqu'à la partie, non-comprise, de jonction des ailes.

nous basant sur des technologies d'IA et plus précisément d'ordonnancement [1] [155, 30, 162]. Comme pour la partie modélisation, nous avons, ici, des travaux préliminaires qui demandent à être approfondis et généralisés.

2.2. L'argumentation pour le choix collaboratif

Jusqu'ici, dans l'optique d'organiser des éléments de justification, nous nous sommes servis de travaux issus du domaine de l'argumentation portant uniquement sur la validité du raisonnement. Il existe toutefois un autre pan de travaux qui s'intéressent à évaluer, modéliser et comparer des arguments qui se contredisent, se réfutent ou se soutiennent [17]. De la même façon que nous avons cherché à appliquer les travaux de Stephen Toulmin au monde industriel, nous pourrions utiliser un grand nombre de travaux existants, en les adaptant à des problèmes auxquels nous avons été confrontés.

Un exemple d'application concrète concerne la capture d'une prise de décision technique par un groupe d'experts. Dans le cadre de la création d'artefacts, il est très courant que des experts se réunissent pour prendre une décision ensemble tel qu'effectuer un choix de conception ou valider des résultats de simulations. Garder une trace de tous les arguments qui sont échangés à cette occasion, avec une représentation claire et précise des relations qui les unissent, peut permettre de comprendre, a posteriori, les justifications d'un choix technique [15, 6]. De plus, durant le cycle de vie d'un artefact complexe, les personnes qui ont participé à sa conception peuvent changer d'emploi ou prendre leur retraite [2]. Dès lors, il arrive que plus personne ne sache exactement pourquoi un artefact a été conçu d'une certaine manière par rapport à une autre. Cette perte de connaissance des explications sur le pourquoi des choix qui ont été faits peut devenir particulièrement problématique dans le cas de la réutilisation et de l'adaptation d'anciennes solutions (sachant que 90% des activités de conception se basent sur de la réutilisation [81]).

Pour répondre à ce besoin, avec Laurence Cholvy, nous sommes partis de travaux portant sur la modélisation des débats et nous avons proposé un cadre de modélisation formel pour rendre compte d'une prise de décisions collaborative [149]. Par la suite, j'ai proposé une méthodologie en deux étapes consistant à représenter graphiquement la discussion entre les experts, puis à enregistrer la justification de la décision dans notre cadre formel [146]. Une suite possible pourrait être d'enrichir notre approche de telle façon à évaluer la pertinence de la solution retenue par les experts, voire à proposer une aide à la décision basée sur les différents arguments avancés. De façon plus large, ceci n'étant qu'un exemple d'applications possibles des travaux menés dans le domaine de l'argumentation, il

1. Précisons que la partie optimisation et implémentation de ce travail revient complètement à Cédric Pralet et Stéphanie Roussel.

2. Dans le cadre aéronautique, le cycle de vie d'un avion, de sa conception jusqu'à l'arrêt de sa maintenance, peut dépasser les cinquante ans.

serait intéressant de poursuivre cet effort en regardant dans quels autres domaines nous pourrions appliquer les recherches menées en argumentation.

2.3. De nouvelles représentations pour faire face à la complexité des modèles

Une façon classique de pouvoir appréhender la complexité d'un système, ou d'un artefact, consiste à en proposer une abstraction au travers de modèles. Cette abstraction met de côté les éléments inutiles et reprend les points nécessaires à l'exécution d'une tâche. Un modèle est donc nécessairement conçu dans un but précis. Les modèles conceptuels ne dérogent pas à la règle. Grâce à leurs représentations graphiques, ils permettent de représenter simplement un système sous une forme diagrammatique. Cependant, la complexification constante des systèmes a entraîné parfois la complexification des diagrammes qui les représentent. Ainsi, dans certains projets industriels, il arrive que les modèles conceptuels deviennent illisibles, entrelacs de boites, de flèches et de symboles, et donc difficilement appréhendables par l'être humain. Plutôt que de chercher une nouvelle abstraction de l'abstraction, avec David Bihanic, nous nous sommes interrogés sur ce que pouvait apporter le design d'interface à ce problème. Parce qu'il cherche à amplifier la cognition via la perception, le design d'interface peut aider à rendre le modèle plus intelligible en jouant sur sa représentation visuelle. Fort de ce constat, nous avons proposé de construire une visualisation dédiée à l'utilisateur, propre au contexte d'emploi de chaque modèle, sans changer aucunement le langage de modélisation : en d'autres termes, de rajouter une surcouche graphique qui vient se superposer aux langages de modélisation existants, surcouche qui fait sens pour l'utilisateur et qui est efficiente en termes cognitifs [22, 23, 24].

Rejoint par Max Chevalier, Sophie Dupuy-Chessa, Xavier Le Pallec, Thierry Morineau, Patrice Terrier et Arnaud Banos, nous avons poursuivi ce dialogue interdisciplinaire et cherché quels pouvaient être les apports possibles du croisement des disciplines que sont l'Ingénierie dirigée par les Modèles, l'Ingénierie de la conception de systèmes, l'Ergonomie Cognitive et le Design d'interface [21].

La suite de ces travaux devrait s'attacher à mettre en pratique cette approche sur des cas d'applications concrets et mesurer les gains qu'une visualisation adéquate peut apporter à l'utilisateur en termes de capacité de traitements. Parallèlement le dialogue interdisciplinaire doit être poursuivi afin de comprendre et de caractériser ce qu'est une visualisation « *cognitivement efficace* ».

Remerciements

Viennent maintenant les remerciements. C'est toujours un moment délicat, où l'on a peur d'oublier quelqu'un et où finalement on réalise que chaque personne que l'on a croisée mérite d'être remerciée, car elle a participé d'une façon ou d'une autre à ce que l'on est aujourd'hui.

Un texte n'est rien sans les relecteurs. C'est eux par leurs relectures minutieuses et leurs questions de fond qui ont permis à ce texte d'être, je l'espère, lisible. Je remercie donc tous mes relecteurs, en commençant par le premier, Betty, qui en plus d'avoir lu et relu les premières versions un peu brouillonnes, m'accompagne dans tous les moments y compris les moments de doute et de fatigue. C'est elle qui a rendu ce texte compréhensible pour les non-spécialistes et m'a patiemment aider à écrire et réécrire, m'obligeant à clarifier des passages quelque peu cryptiques. Merci à Florence Sèdes, ma marraine, qui a toujours été là, bienveillante, à m'épauler dans la structuration du texte et qui m'a aidé à y voir plus clair. Merci de m'avoir accompagné à chaque étape que tout candidat traverse dans l'écriture d'un manuscrit d'Habilitation à Diriger des Recherches, y compris face aux étapes administratives ô combien obscures. Merci aussi à toute l'équipe de relecteurs, Claude Cuiller, François Bouissière, Marc Boyer, Mireille Blay-Fornarino, Philippe Molière et Virginie Wiels, pour leur patience et leurs précieux conseils. Merci à mes trois rapporteurs, Régine Laleau, Daniel Le Berre et Lionel Seinturier, tout d'abord pour avoir accepté d'être mes rapporteurs et ensuite pour leurs relectures qui ont permis à ce manuscrit d'aboutir à la version que vous avez entre les mains.

Si ce manuscrit tente de donner une perspective aux travaux de recherches auxquels j'ai participé ces dix dernières années, il est évident que ces travaux, eux, ne relèvent que d'un travail collégial avec d'autres chercheurs. Dès lors, chaque contribution scientifique est une contribution polyphonique, où il serait vain et illusoire de chercher à extraire une quelconque paternité de ma part. Merci donc à tous ceux qui ont accepté de faire un bout de chemin scientifique avec moi. Merci

plus particulièrement, dans l'ordre alphabétique, car aucun ordre n'est possible, à mes coauteurs qui ont tous participé d'une manière ou d'une autre aux travaux présentés dans ce mémoire : David Bihanic, Mireille Blay-Fornarino, Jean-Paul Bodeveix, Frédéric Boniol, François Bouissiere, Max Chevalier, Laurence Cholvy, Claude Cuiller, Kevin Delmas, Rémi Delmas, Pierre-Eric Dereux, David Doose, Clément Duffau, Jean-Claude Dunyach, Stéphane Duprat, Sophie Dupuy-Chessa, Mamoun Filali, Stéphane Kersuzan, Juyeon Kang Choi, Claire Pagetti, Anthony Fernandes Pires, Xavier Le Pallec, Cédric Pralet, Stéphanie Roussel, Sanjiv Sharma, Florence Sèdes, Vincent Tuloup et Virginie Wiels.

Merci à Anthony et Clément qui m'ont donné l'impression de savoir encadrer une thèse, mais ne nous leurrons pas le mérite leur revient en grande partie. A nos séjours à Miami et Tallin, vous avoir vus présenter vos travaux dans des conférences internationales reste pour moi un grand moment. Vos années de thèse ont été un plaisir. J'ai aussi une pensée pour Virginie et Mireille qui m'ont accepté comme co-encadrant et auprès de qui j'ai beaucoup appris. Merci aussi à Stéphane Duprat qui a non seulement rendu la thèse d'Anthony possible, en trouvant un financement, mais a permis de faire une thèse dans d'excellentes conditions à ses côtés. D'ailleurs, merci à Ploc et Rémi de m'avoir couru après au Japon.

Ce mémoire d'HDR n'existerait pas sans ces quelques semaines de printemps 2019 où Flo, Mireille et Régine ont su me motiver et me montrer que cela avait un sens. Merci à toutes les trois, vraiment, sans vous je ne l'aurais pas fait ! Merci aussi à l'ensemble des membres du jury qui ont pris sur leur temps pour venir m'écouter le jour de ma soutenance.

De façon plus large, je remercie la communauté Inforsid, de m'avoir accueilli chaleureusement et avoir toujours eu un regard très positif sur mes travaux. Merci à David Bihanic pour nos nombreuses conversations et pour tes efforts à non seulement m'expliquer ta discipline, mais en plus à comprendre la mienne. Il est loin le temps où nous étions tous les deux thésards et où, de passage à Toulouse, tu avais poussé la porte de mon bureau. Que l'aventure continue, avec encore pleins de choses à faire ensemble. Je voudrais aussi remercier le Réseau National des Systèmes Complexes et plus particulièrement Arnaud Banos qui ont été les premiers à soutenir notre démarche à David et moi. Je me souviens de cette conférence à Paris où, dès le premier repas, Arnaud est venu me voir au moment du repas et m'a présenté à des chercheurs de disciplines très hétérogènes alors que j'étais en territoire complètement inconnu. Grâce à lui, j'ai pu voir de près l'interdisciplinarité que cela soit Rue Lhomond ou lors de cette semaine mémorable à Sète. J'espère que nous aurons de nouveau l'occasion de nous croiser. Merci à Sophie Dupuy-Chessa et Xavier Le Pallec de nous avoir suivis dans cette aventure transdisciplinaire. Et puis il y a aussi tous ces très bons souvenirs ensemble à Porquerolles, Grenoble, Toulouse, etc. Merci à Mamoun Filali de m'avoir fait revenir dans le monde de la recherche, ce jour-là, tu as eu raison d'insister et de me dire de venir travailler avec toi, et à Laurence Cholvy pour

m'avoir fait confiance.

J'ai eu l'occasion de participer à de très gros projets européens. On ne sort pas indemne de ces expériences internationales sans l'aide de gens précieux. Merci à Sanjiv Sharma et Vincent Tuloup d'avoir été présents. Ils ont pris le temps de comprendre ce que je cherchais à faire, même si c'était très loin de leurs préoccupations. Grâce à Vincent, les diagrammes de justification ont pu prendre vie dans 3DExperience. Et puis, merci Jean-Claude Dunyach. Ton soutien indéfectible m'a permis d'affronter la difficulté de mes premiers projets internationaux. J'en profite aussi, pour remercier l'autre facette de Jean-Claude qui avait pris le temps, il y a presque trente ans maintenant, de venir converser plusieurs fois à la radio et dont j'apprécie toujours avec grand plaisir la compagnie.

Je viens historiquement de la logique mathématique et si le passage vers les méthodes formelles s'est fait assez naturellement, je dois préciser que je ne suis pas tombé tout seul dans l'argumentation : on m'a poussé ! Merci à René Jacquart de m'avoir montré les liens possibles entre VV&A et argumentation, une partie de ce manuscrit s'inscrit dans la continuité des conseils que tu as eu la gentillesse de me prodiguer. Merci à Philippe Besnard, de m'avoir ouvert un empire rhétorique et de m'avoir parlé de Toulmin. Merci également à Nicolas Maudet et Wassila Ouerdane pour l'enthousiasme que vous portez à mes bricolages argumentatifs et pour m'avoir fait une place dans votre monde. Merci à Ghilaine Martinez de m'avoir expliqué le travail côté autorité de certification et à Hervé Delseny de m'avoir expliqué les Overarching Properties. Merci à Claire Pagetti et Kevin Delmas de m'avoir embarqué avec eux sur leurs problèmes de certification.

Je les ai déjà remerciés dans le manuscrit, mais n'étant pas du genre à compter, je remercie à nouveau Rémi compagnon de recherche et de trouvailles (et de bien d'autres choses), François et Claude de m'avoir fait confiance et de m'avoir amené dans cette aventure scientifique que nous vivons actuellement et avec eux Stéphanie Roussel, Cédric Pralet, Pierre-Eric Dereux et Stephane Kersuzan (the Magnificent Seven) : tous ces moments passés avec vous sont pour moi un vrai plaisir.

Pour finir, je remercie tous les gens qui ont été là pendant cette année d'écriture et plus particulièrement Philo, Louise, Claire et Philippe qui ont contribué à rendre cette année douce, Manu et Alex qui m'ont fait une place sur la Lune, l'ami Vincent dont les passages me sont toujours précieux et Betty pour trop de choses pour qu'il me soit possible de les compter.

Bibliographie

[1] Wafa Abdelghani, Corinne Amel Zayani, Ikram Amous, and Florence Sèdes. Trust management in social internet of things : A survey. In *Social Media : The Good, the Bad, and the Ugly - 15th IFIP WG 6.11 Conference on e-Business, e-Services, and e-Society, I3E, Proceedings*, volume 9844 of *Lecture Notes in Computer Science*, pages 430–441. Springer, 2016.

[2] Jean-Raymond Abrial. *The B-book - assigning programs to meanings*. Cambridge University Press, 1996.

[3] Selim G Aki and Dorothy E Denning. Checking classification constraints for consistency and completeness. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, pages 196–201. IEEE Computer Society, 1987.

[4] Christopher Alexander, Sara Ishikawa, and Murray Silverstein. *A Pattern Language : Towns, Buildings, Construction*. Oxford University Press, New York, August 1977.

[5] Rob Alexander, Richard David Hawkins, and Tim Kelly. Security assurance cases : Motivation and the state of the art. Technical report, Department of Computer Science, University of York, 2011.

[6] Muhammad Ali Babar and Patricia Lago. Design decisions and design rationale in software architecture. *Journal of Systems and Software*, 82 :1195–1197, 2009.

[7] Kyriakos Anastasakis, Behzad Bordbar, Geri Georg, and Indrakshi Ray. UML2Alloy : A Challenging Model Transformation. In G. Engels, B. Opdyke, D.C. Schmidt, and F. Weil, editors, *ACM/IEEE 10th International Conference on Model Driven Engineering Languages and Systems*, volume 4735 of *LNCS*, pages 436–450, Nashville, USA, 2007. Springer.

[8] Sophie Archambault de Beaune. Aux origines de la division du travail. *Pour la science*, 445, 2014.

[9] Aristote. *Physique, Traduction par Jules Barthélemy-Saint-Hilaire*. Librairie Philosophique de Ladrange, 1862.

[10] Aristote. *Organon, tome 6 : Les Réfutations sophistiques, Traduction par Jules Tricot*. Vrin, 1995.

[11] William Aspray. An interview with Richard Bloch, 1984.

[12] Association for the Advancement of Artificial Intelligence. *Proceedings of the the AAAI Spring Symposium on producing cooperative explanations*, 1992.

[13] Algirdas Avizienis, J-C Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE transactions on dependable and secure computing*, 1(1) :11–33, 2004.

[14] Osman Balci. Verification validation and accreditation of simulation models. In *Proceedings of the 29th conference on Winter simulation*, pages 135–141. IEEE Computer Society, 1997.

[15] Iain Bate. Systematic approaches to understanding and evaluating design trade-offs. *Journal of Systems and Software*, 81 :1253–1271, 2008.

[16] P. Baudin, P. Cuoq, J.C. Filliâtre, C. Marché, B. Monate, Y. Moy, and V. Prevosto. *ACSL Version 1.6*, 2012.

[17] Philippe Besnard and Anthony Hunter. *Elements of Argumentation*. MIT Press, 2008.

[18] Pierre Bieber, Frédéric Boniol, Youcef Bouchebaba, Julien Brunel, Claire Pagetti, Olivier Poitou, Thomas Polacsek, Luca Santinelli, and Nathanaël Sensfelder. A model-based certification approach for multi/many-core embedded systems. In *Proc. of the 9th European Congress Embedded Real Time Software And Systems (ERTS'18)*, 2018.

[19] Pierre Bieber, Frédéric Boniol, Guy Durrieu, Olivier Poitou, Thomas Polacsek, Virginie Wiels, and Ghilaine Martinez. MIMOSA : Towards a model driven certification process. In *Proc. of the 8th European Congress Embedded Real Time Software And Systems (ERTS'16)*, 2016.

[20] Pierre Bieber and Frédéric Cuppens. *Expression of confidentiality policies with deontic logic*, pages 103–123. John Wiley & Sons, Inc., New York, NY, USA, 1994.

[21] David Bihanic, Max Chevalier, Sophie Dupuy-Chessa, Xavier Le Pallec, Thierry Morineau, and Thomas Polacsek. Modèlisation graphique des SI. du traitement visuel de modèles complexes. In *Actes du XXXIème Congrès INFORSID*, pages 99–114, 2013.

[22] David Bihanic and Thomas Polacsek. Models for visualisation of complex information systems. In *16th International Conference on Information Visualisation, IV 2012*, pages 130–135. IEEE Computer Society, 2012.

[23] David Bihanic and Thomas Polacsek. Visualisation de systèmes d'information complexes une approche par « points de vue étendus ». *Journal Studia Informatica Universalis*, 10(1) :235–262, 2012.

[24] David Bihanic and Thomas Polacsek. Stratavis : une technique de visualisation graphique orientée modèle. *Technique et Science Informatiques*, 35(2) :145–174, 2016.

[25] Lenore Blum. Alan turing and the other theory of computation (expanded). In Rod Downey, editor, *Turing's Legacy : Developments from Turing's Ideas in Logic*, volume 42 of *Lecture Notes in Logic*, pages 48–69. Cambridge University Press, 2014.

[26] Barry W. Boehm. Verifying and validating software requirements and design specifications. *IEEE Software*, 1(1) :75–88, 1984.

[27] Barry W. Boehm. A view of 20th and 21st century software engineering. In Leon J. Osterweil, H. Dieter Rombach, and Mary Lou Soffa, editors, *Proceedings of the 28th international conference on Software engineering (ICSE 2006)*, pages 12–29. ACM, 2006.

[28] Frédéric Boniol, Youcef Bouchebaba, Julien Brunel, Kevin Delmas, Claire Pagetti, Thomas Polacsek, and Nathanaël Sensfelder. PHYLOG : a model-based certification framework. In *37th AIAA/IEEE Digital Avionics Systems Conference (DASC)*, 2018.

[29] François Bouissière, Claude Cuiller, Pierre-Eric Dereux, Stéphane Kersuzan, and Thomas Polacsek. Modéliser l'avion et son moyen de production : vers un modèle global pour de la conception simultanée. In *Actes du XXXVème Congrès INFORSID*, pages 77–92, 2017.

[30] François Bouissiere, Claude Cuiller, Pierre-Eric Dereux, Stephane Kersuzan, Thomas Polacsek, Cédric Pralet, and Stephanie Roussel. Co-engineering in aeronautics ? the a320 forward section case study. In *Proc. of the 9th European Congress Embedded Real Time Software And Systems (ERTS'18)*, 2018.

[31] Kari Ann Briski, Poonam Chitale, Valerie Hamilton, Allan Pratt, Brian Starr, Jim Veroulis, and Bruce Villard. Minimizing code defects to improve software quality and lower development costs. Technical report, IBM Software Group, 2008.

[32] J. Browne and J. Zhang. Extended and virtual enterprises similarities and differences. *International Journal of Agile Management Systems*, 1(1) :30–36, 1999.

[33] Jordi Cabot, Robert Clarisó, and Daniel Riera. Verification of uml/ocl class diagrams using constraint programming. In *Proceedings of the 2008 IEEE International Conference on Software Testing Verification and Validation Workshop (ICSTW'08)*, pages 73–80, Washington, DC, USA, 2008. IEEE Computer Society.

[34] Jordi Cabot, Robert Clarisó, and Daniel Riera. On the verification of UML/OCL class diagrams using constraint programming. *Journal of Systems and Software*, 93 :1–23, 2014.

[35] Lewis Carroll. *Symbolic logic : part I, Elementary (fourth edition)*. Macmillan and co., 1896.

[36] Valentin Cassano and Thomas S. E. Maibaum. The definition and assessment of a safety argument. In *25th IEEE International Symposium on Software Reliability Engineering Workshops, ISSRE Workshops, Naples, Italy, November 3-6, 2014*, pages 180–185. IEEE Computer Society, 2014.

[37] Hector-Neri Castañeda. *Thinking and doing : The philosophical foundations of institutions*, volume 7 of *Philosophical Studies Series*. Springer, 1975.

[38] Certification Authorities Software Team. Multi-core Processors - Position Paper. position paper CAST 32-A, Federal Aviation Administration, 2016.

[39] Bruce Chandrasekaran, Michael C. Tanner, and John R. Josephson. Explaining control strategies in problem solving. *IEEE Intelligent Systems*, 4(1) :9–15, 19–24, 1989.

[40] Timothy Chappell. Plato on knowledge in the theaetetus. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2013 edition, 2013.

[41] Peter P. Chen. The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1) :9–36, 1976.

[42] National Research Council Committee on Identifying the Needs of the Forensic Sciences Community. *Strengthening Forensic Science in the United States : A Path Forward*. The National Academies Press, 2009.

[43] Leéo Creuse, Joffrey Huguet, Christophe Garion, and Jérôme Hugues. Spark by example : an introduction to formal verification through the standard c++ library. to appear, 2019.

[44] Fabiano Dalpiaz, Xavier Franch, and Jennifer Horkoff. istar 2.0 language guide. *CoRR*, abs/1605.07767, 2016.

[45] Donald Davidson. I. agency. In Ausonio Marras, Richard N. Bronaugh, and Robert W. Binkley, editors, *Agent, Action, and Reason*, pages 1–37. University of Toronto Press, 1971.

[46] Robert C Davis. *Shipbuilders of the Venetian arsenal : workers and workplace in the preindustrial city*, volume 109. JHU Press, 2007.

[47] Marc J De Vries. Gilbert simondon and the dual nature of technical artifacts. *Techné : Research in Philosophy and Technology*, 12(1) :23–35, 2008.

[48] Kevin Delmas, Claire Pagetti, and Thomas Polacsek. Certification elicitation. submited, 2020.

[49] Rémi Delmas, David Doose, Anthony Fernandes Pires, and Thomas Polacsek. Supporting model based design. In *Model and Data Engineering - Proceedings First International Conference, MEDI*, volume 6918 of *Lecture Notes in Computer Science*, pages 237–248. Springer, 2011.

[50] Rémi Delmas, David Doose, and Thomas Polacsek. Utilisation de techniques SAT/PseudoBool pour la synthèse de modèles corrects par construction dans le cadre IDM. *Génie Lociel*, 97, 2011.

[51] Rémi Delmas, David Doose, Thomas Polacsek, and Anthony Fernandes Pires. IDM : Vers une aide à la conception. In *Actes du XXIXème Congrès INFORSID2011*, pages 147–162, 2011.

[52] Rémi Delmas, Anthony Fernandes Pires, and Thomas Polacsek. A verification and validation process for model-driven engineering. In *Progress in Flight Dynamics, Guidance, Navigation, Control, Fault Detection, and Avionics*, volume 6, chapter 5 Avionics, pages 455–468. EDP Sciences, 2013.

[53] Rémi Delmas and Thomas Polacsek. Formal methods for exchange policy specification. In Camille Salinesi, Moira C. Norrie, and Oscar Pastor, editors, *Advanced Information Systems Engineering - 25th International Conference, CAiSE 2013, Proceedings*, volume 7908 of *Lecture Notes in Computer Science*, pages 288–303. Springer, 2013.

[54] Rémi Delmas and Thomas Polacsek. Exigences de confidentialité et de diffusion concernant les politiques d'échanges d'information. *Génie Logiciel*, 111 :49–53, 2014.

[55] Rémi Delmas and Thomas Polacsek. Critical information diffusion systems. In Tadeusz Morzy, Patrick Valduriez, and Ladjel Bellatreche, editors, *New Trends in Databases and Information Systems - ADBIS 2015 Workshops WISARD*, volume 539 of *Communications in Computer and Information Science*, pages 557–566. Springer, 2015.

[56] Rémi Delmas and Thomas Polacsek. Need-to-share and non-diffusion requirements verification in exchange policies. In Jelena Zdravkovic, Marite Kirikova, and Paul Johannesson, editors, *Advanced Information Systems Engineering - 27th International Conference, iSE 2015, Proceedings*, volume 9097 of *Lecture Notes in Computer Science*, pages 151–165. Springer, 2015.

[57] Rémi Delmas and Thomas Polacsek. Vérification automatique d'exigences pour les politiques d'échange d'information. In *Actes du XXXIIIème Congrès INFORSID*, pages 251–265, 2015.

[58] Rémi Delmas and Thomas Polacsek. Vérification automatique d'exigences pour les politiques d'échange d'information. exigences de diffusion et de non-diffusion d'information. *Ingénierie des Systèmes d'Information*, 21(2) :39–63, 2016.

[59] Dorothy E Denning, Selim G Akl, Mark Heckman, Teresa F. Lunt, Matthew Morgenstern, Peter G. Neumann, and Roger R. Schell. Views for multilevel database security. *IEEE Transactions on Software Engineering*, 13(2) :129–140, 1987.

[60] Edsger W. Dijkstra. The humble programmer. *Commun. ACM*, 15(10) :859–866, 1972.

[61] Edsger W Dijkstra. Programming as a discipline of mathematical nature. *The American Mathematical Monthly*, 81(6) :608–612, 1974.

[62] DoD. DoD Directive 5000.59 : Modeling and Simulation (M&S) Management. Standard, US Department of Defense, 1994.

[63] Clément Duffau. *Justification Factory : from justification requirements elicitation to their continuous production*. PhD thesis, Université Côte d'Azur, 2018.

[64] Clément Duffau, Thomas Polacsek, and Mireille Blay-Fornarino. Support of justification elicitation : Two industrial reports. In *Advanced Information Systems Engineering - 30th International Conference, CAiSE 2018, Proceedings*, volume 10816 of *Lecture Notes in Computer Science*, pages 71–86. Springer, 2018.

[65] Clément Duffau, Thomas Polacsek, and Mireille Blay-Fornarino. Une sémantique pour les patrons de justification. In *Actes du XXXVIème Congrès INFORSID*, 2018.

[66] Phan Minh Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence*, 77(2) :321–358, 1995.

[67] Jeffrey H. Dyer and Harbir Singh. The Relational View : Cooperative Strategy and Sources of Interorganizational Competitive Advantage. *The Academy of Management Review*, 23(4) :660–679, 1998.

[68] Bruce Edmonds. How formal logic can fail to be useful for modelling or designing mas. In Gabriela Lindemann, Daniel Moldt, and Mario Paolucci, editors, *Regulated Agent-Based Social Systems, First International Workshop, RASTA 2002, 2002, Revised Selected and Invited Papers*, volume 2934 of *Lecture Notes in Computer Science*, pages 1–15. Springer, 2002.

[69] David W. Embley, Stephen W. Liddle, and Oscar Pastor. Conceptual-model programming : A manifesto. In David W. Embley and Bernhard Thalheim, editors, *Handbook of Conceptual Modeling - Theory, Practice, and Research Challenges*, pages 3–16. Springer, 2011.

[70] Luke Emmet and George Cleland. Graphical notations, narratives and persuasion : a pliant systems approach to hypertext tool design. In James Blustein, Robert B. Allen, Kenneth M. Anderson, and Stuart Moulthrop, editors, *HYPERTEXT 2002, Proceedings of the 13th ACM Conference on Hypertext and Hypermedia*, pages 55–64. ACM, 2002.

[71] Sextus Empiricus. *Les Hipotiposes pirroniennes, Livre premier, Traduction par Claude Huart*. Librairie De L. Hachette Et Cie, 1725.

[72] Michael E. Fagan. Design and code inspections to reduce errors in program development. *IBM Systems Journal*, 15(3) :182–211, 1976.

[73] Anthony Fernandes Pires. *Amélioration des processus de vérification de programmes par combinaison des méthodes formelles avec l'Ingénierie Dirigée par les Modèles*. PhD thesis, Institut supérieur de l'aéronautique et de l'espace ISAE, 2014.

[74] Anthony Fernandes Pires, Thomas Polacsek, and Stéphane Duprat. Formal software verification at model and at source code levels. In *Model and Data Engineering - 2nd International Conference, MEDI 2012, Proceedings*, volume 7602 of *Lecture Notes in Computer Science*, pages 162–169. Springer, 2012.

[75] Anthony Fernandes Pires, Thomas Polacsek, and Stéphane Duprat. An eclipse plug-in to link modelling and code proof. *Joint Proceedings of Tools, Demos & Posters 27th European Conference Object-Oriented Programming (ECOOP 2013)*, page 23, 2013.

[76] Anthony Fernandes Pires, Thomas Polacsek, Virginie Wiels, and Stéphane Duprat. Behavioural verification in embedded software, from model to source code. In *Model-Driven Engineering Languages and Systems - 16th International Conference, MODELS 2013, Proceedings*, volume 8107 of *Lecture Notes in Computer Science*, pages 320–335. Springer, 2013.

[77] Anthony Fernandes Pires, Thomas Polacsek, Virginie Wiels, and Stéphane Duprat. Vérifier le comportement du code d'un système embarqué à partir de son modèle. *Journal Européen des Systèmes Automatisés*, 47/1-3 :61–75, 2013.

[78] Anthony Fernandes Pires, Thomas Polacsek, Virginie Wiels, and Stéphane Duprat. Use of formal methods in embedded software development : stakes, constraints and proposal. In *Proc. of the 7th European Congress Embedded Real Time Software And Systems (ERTS'14)*, 2014.

[79] Matthew Fisher and Frank C Keil. The illusion of argument justification. *Journal of Experimental Psychology : General*, 143(1) :425–433, 2014.

[80] George S Fishman and Philip J Kiviat. The statistics of discrete-event simulation. *Simulation*, 10(4) :185–195, 1968.

[81] Desmond Fletcher and Pingping Gu. Adaptable design for design reuse. In *Proceedings of the Canadian Engineering Education Association (CEEA).*, pages 514–517, 2005.

[82] Robert W Floyd. Assigning meanings to programs. *Mathematical aspects of computer science*, 19(19-32) :1, 1967.

[83] Maarten Franssen, Gert-Jan Lokhorst, and Ibo van de Poel. Philosophy of technology. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, fall 2018 edition, 2018.

[84] W. Barkley Fritz. The women of eniac. *IEEE Annals of the History of Computing*, 18(3) :13–28, 1996.

[85] Jean H. Gallier. *Logic for Computer Science : Foundations of Automatic Theorem Proving*, chapter 10, pages 448–476. Wiley, 1987.

[86] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns : Elements of Reusable Object-oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[87] David M. Godden and Douglas Walton. Argument from expert opinion as legal evidence : Critical questions and admissibility criteria of expert testimony in the american legal system. *Ratio Juris*, 19(3) :261–286, 2006.

[88] Martin Gogolla, Fabian Büttner, and Mark Richters. Use : A uml-based specification environment for validating uml and ocl. *Sci. Comput. Program.*, 69(1-3) :27–34, 2007.

[89] Adele Goldstine. A report on the ENIAC (Electronic Numerical Integrator and Computer), volume I. technical report, The University of Pennsylvania, Moore School of Electrical Engineering, 1946. Report of Work Under Contract No. W-670-ORD-4926.

[90] Bryce Goodman and Seth R. Flaxman. European union regulations on algorithmic decision-making and a "right to explanation". *AI Magazine*, 38(3) :50–57, 2017.

[91] Jean Goodwin and Alec Fisher. Wigmore's Chart Method. *Informal Logic*, 20(3) :223–243, 2001.

[92] Ingrid Göpfert and Matthias Schulz. Logistics integrated product development in the german automotive industry : current state, trends and challenges. In *Dynamics in Logistics : Third International Conference, LDIC 2012 Proceedings*, pages 509–519. Springer, 2013.

[93] Allison Rose Greenwald. *Learning how to argue : experiences teaching the Toulmin model to composition students*. PhD thesis, Iowa State University, 2007.

[94] Shirley Gregor and Izak Benbasat. Explanations from intelligent systems : Theoretical foundations and implications for practice. *MIS Quarterly*, 23(4) :497–530, 1999.

[95] Yves Gruet. Les coquillages marins : objets archéologiques à ne pas négliger. quelques exemples d'exploitation et d'utilisation dans l'ouest de la france. *Revue archéologique de l'Ouest*, 10(10) :157–161, 1993.

[96] Leo Gugerty. Newell and Simon's Logic Theorist : Historical Background and Impact on Cognitive Modeling. In *Proceedings of the Human Factors*

and Ergonomics Society Annual Meeting, volume 50, pages 880–884. SAGE Publications Sage CA : Los Angeles, CA, 2006.

[97] Denise Gürer. Women in computing history. *SIGCSE Bulletin*, 34(2) :116–120, 2002.

[98] Thomas Haigh and Mark Priestley. Where code comes from : Architectures of automatic control from babbage to algol. *Commun. ACM*, 59(1) :39–44, 2015.

[99] Charles L. Hamblin. *Fallacies*. University paperback. Methuen, 1970.

[100] Maralee Harrell and Danielle Wetzel. *Using Argument Diagramming to Teach Critical Thinking in a First-Year Writing Course*, pages 213–232. Palgrave Macmillan US, New York, 2015.

[101] Jean Hilaire. *Adages et maximes du droit français*. Dalloz, 2015.

[102] Risto Hilpinen. On artifacts and works of art 1. *Theoria*, 58(1) :58–82, 1992.

[103] Risto Hilpinen. Authors and artifacts. In *Proceedings of the Aristotelian Society*, volume 93, pages 155–178. Oxford University Press, 1993.

[104] Charles Antony Richard Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10) :576–580, 1969.

[105] Charles Antony Richard Hoare. How did software get so reliable without proof ? In *FME '96 : Industrial Benefit and Advances in Formal Methods, Third International Symposium of Formal Methods Europe, Co-Sponsored by IFIP WG 14.3, Oxford, UK, March 18-22, 1996, Proceedings*, volume 1051 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1996.

[106] Robert E. Horn. *Infrastructure for navigating interdisciplinary debates : critical decisions for representing argumentation*, pages 165–184. Springer-Verlag, London, UK, 2003.

[107] IEEE. IEEE Standard for Software Verification and Validation. Standard, IEEE Computer Society, 1998.

[108] IEEE. 730-2014 - ieee standard for software quality assurance processes (revision of ieee std 730-2002). Standard, Institute of Electrical and Electronics Engineers , 2014.

[109] International Joint Conferences on Artificial Intelligence. *Proceedings of the IJCAI'93 Workshop on Explanation and Problem Solving*, 1993.

[110] ISO. 9001 :2015 systèmes de management de la qualité – exigences. Standard, International Organization for Standardization, 2015.

[111] ISO/IEC. ISO/IEC 15026 Systems and software engineering – Systems and software assurance – Part 2 : Assurance case. Standard, International Organization for Standardization, 2011.

[112] ISO/IEC/IEEE. ISO/IEC/IEEE International Standard - Systems and software engineering—Vocabulary. Standard, ISO/IEC/IEEE, 2017.

[113] Ralph Henry Johnson and J. Anthony Blair. *Logical Self-defense*. Key titles in rhetoric, argumentation, and debate series. International Debate Education Association, 2006. first edition 1977.

[114] Juyeon Kang. *Morpho-syntactic problems analyzed in an extended categorial grammar : application to Korean and to French with an implementation of a categorial parser*. PhD thesis, Université Paris-Sorbonne (Paris IV), 2011.

[115] Tim Kelly and John McDermid. Safety case construction and reuse using patterns. In *16th International Conference on Computer Safety, Reliability and Security, Safe Comp 1997*, pages 55–69. Springer, 1997.

[116] Timothy Kelly. *Arguing safety : a systematic approach to managing safety cases*. PhD thesis, University of York York, 1999.

[117] Timothy Kelly and Rob Weaver. The Goal Structuring Notation /- A Safety Argument Notation. In *Proceedings of Dependable Systems and Networks 2004 Workshop on Assurance Cases*, 2004.

[118] John Knight. Advances in software technology since 1992. In *National Software and Airborne Electronic Hardware Conference, ser. FAA*, 2008.

[119] Gerald Kotonya and Ian Sommerville. *Requirements engineering : processes and techniques*. Wiley Publishing, 1998.

[120] Peter Kroes. Engineering and the dual nature of technical artefacts. *Cambridge journal of economics*, 34(1) :51–62, 2009.

[121] Peter Kroes and Anthonie Meijers. The dual nature of technical artefacts. *Studies in History and Philosophy of Science Part A*, 37(1) :1–4, 2006.

[122] Mirco Kuhlmann, Lars Hamann, and Martin Gogolla. Extensive validation of OCL models by integrating SAT solving into USE. In Judith Bishop and Antonio Vallecillo, editors, *Objects, Models, Components, Patterns, Proceedings 49th International Conference, TOOLS 2011*, volume 6705 of *Lecture Notes in Computer Science*, pages 290–306. Springer, 2011.

[123] Patrick R Laughlin, Erin C Hatch, Jonathan S Silver, and Lee Boh. Groups perform better than the best individuals on letters-to-numbers problems : effects of group size. *Journal of Personality and social Psychology*, 90(4) :644, 2006.

[124] Daniel Leberre. SAT4J, a SATisfiability library for java, 2004. http ://www.sat4j.org.

[125] Jennifer S Light. When computers were women. *Technology and culture*, 40(3) :455–483, 1999.

[126] Vasco M. Manquinho, Ruben Martins, and Inês Lynce. Improving unsatisfiability-based algorithms for boolean optimization. In *SAT*, pages 181–193, 2010.

[127] Marcos Martinón-Torres, Xiuzhen Janice Li, Andrew Bevan, Yin Xia, Kun Zhao, and Thilo Rehren. Forty thousand arms for a single Emperor : from chemical data to the labor organization behind the bronze arrows of the Terracotta Army. *Journal of Archaeological Method and Theory*, 21(3) :534–562, 2014.

[128] John McCarthy. Modality, si ! modal logic, no ! *Studia Logica*, 59(1) :29–32, 1997.

[129] John McDermid. Support for safety cases and safety arguments using sam. *Reliability Engineering & System Safety*, 43(2) :111–127, 1994.

[130] John Stuart Mill. *Système de logique déductive et inductive. Exposé des principes de la preuve et des méthodes de recherche scientifique. (1843). Traduction française réalisée par Louis Peisse à partir de la 6e édition anglaise de 1865*. Librairie philosophique de Ladrange, Paris, France, 1866.

[131] Joel Mokyr. The Rise and Fall of the Factory System : Technology, firms, and households since the Industrial Revolution. *Carnegie-Rochester Conference Series on Public Policy*, 55(1) :1–45, 2001.

[132] James H Moor. Three myths of computer science. *The British Journal for the Philosophy of Science*, 29(3) :213–222, 1978.

[133] F. Lockwood Morris and Cliff B. Jones. An early program proof by alan turing. *IEEE Annals of the History of Computing*, 6(2) :139–143, 1984.

[134] David Moshman. Reasoning as self-constrained thinking. *Human Development*, 38(1) :53–64, 1995.

[135] David Moshman and Molly Geil. Collaborative reasoning : Evidence for collective rationality. *Thinking & Reasoning*, 4(3) :231–248, 1998.

[136] National Research Council. *Software for Dependable Systems : Sufficient Evidence ?* The National Academies Press, Washington, DC, 2007.

[137] Peter Naur and Brian Randell. Software engineering : Report on a conference sponsored by the nato science committee, garmisch, germany, 7th to 11th october 1968. Technical report, NATO, 1969.

[138] John von Neumann. *The Computer and the Brain*. Yale University Press, New Haven, CT, USA, 1958.

[139] Allen Newell and Herbert A. Simon. The logic theory machine-a complex information processing system. *IRE Trans. Information Theory*, 2(3) :61–79, 1956.

[140] William L Oberkampf, Timothy G Trucano, and Charles Hirsch. Verification, validation, and predictive capability in computational engineering and physics. *Applied Mechanics Reviews*, 57(5) :345–384, 2004.

[141] OMG. Structured Assurance Case Meta-model (SACM). Standard, Object Management Group, 2013.

[142] Naomi Oreskes, Kristin Shrader-Frechette, and Kenneth Belitz. Verification, validation, and confirmation of numerical models in the earth sciences. *Science*, 263(5147) :641–646, 1994.

[143] Oscar Pastor, Marcela Ruiz, and Sergio España. From requirements to code : A full model-driven development perspective. In *Software and Data Technologies - 6th International Conference, ICSOFT. Revised Selected Papers*, volume 303, pages 56–70. Springer, 2011.

[144] Chaim Perelman and Lucie Olbrechts-Tyteca. *Traité de l'argumentation : La nouvelle rhétorique*. UBlire - Fondamentaux. Éditions de l' Université de Bruxelles, 2008 (première édition 1958).

[145] Platon. *Théétète Parménide, Traduction par Émile Chambry*. GF-Flammarion, 1991.

[146] Thomas Polacsek. A process to support and to report collaborative decision. In Giancarlo Fortino, Weiming Shen, Jean-Paul A. Barthès, Junzhou Luo, Wenfeng Li, Sergio F. Ochoa, Marie-Hélène Abel, Antonio Guerrieri, and Milton Pires Ramos, editors, *19th IEEE International Conference on Computer Supported Cooperative Work in Design, CSCWD 2015*, pages 188–193. IEEE, 2015.

[147] Thomas Polacsek. Validation, Accreditation or Certification : a New Kind of Diagram to Provide Confidence. In *10th IEEE International Conference on Research Challenges in Information Science, RCIS*, 2016.

[148] Thomas Polacsek. Diagramme de justification. un outil pour la validation, la certification et l'accréditation. *Ingénierie des Systèmes d'Information*, 22(2) :95–119, 2017.

[149] Thomas Polacsek and Laurence Cholvy. A framework to report and to analyse a debate. In Weiming Shen, Jean-Paul A. Barthès, Junzhou Luo, Peter G. Kropf, Michel Pouly, Jianming Yong, Yunjiao Xue, and Milton Pires Ramos, editors, *Proceedings of the 2011 15th International Conference on Computer Supported Cooperative Work in Design, CSCWD 2011*, pages 84–90. IEEE, 2011.

[150] Thomas Polacsek, Stéphanie Roussel, François Bouissière, Claude Cuiller, Pierre-Eric Dereux, and Stéphane Kersuzan. Towards thinking manufacturing and design together : An aeronautical case study. In *Conceptual Modeling - 36th International Conference, ER 2017, Proceedings*, volume 10650 of *Lecture Notes in Computer Science*, pages 340–353. Springer, 2017.

[151] Thomas Polacsek, Stéphanie Roussel, Cédric Pralet, and Claude Cuiller. Design for efficient production, a model-based approach. In *13th IEEE International Conference on Research Challenges in Information Science, RCIS*, 2019.

[152] Thomas Polacsek, Sanjiv Sharma, Claude Cuiller, and Vincent Tuloup. The need of diagrams based on toulmin schema application : an aeronautical case study. *EURO Journal on Decision Processes*, 6(3-4) :257–282, 2018.

[153] Dominique Potier. Briques génériques du logiciel embarqué. Technical report, Ministère de l'industrie (France), 2010.

[154] Henry Prakken. Formal systems for persuasion dialogue. *Knowledge Eng. Review*, 21(2) :163–188, 2006.

[155] Cédric Pralet, Stéphanie Roussel, Thomas Polacsek, François Bouissière, Claude Cuiller, Pierre-Eric Dereux, Stéphane Kersuzan, and Marc Lelay. A scheduling tool for bridging the gap between aircraft design and aircraft manufacturing. In *Proceedings of the Twenty-Eighth International Conference on Automated Planning and Scheduling, ICAPS*, pages 347–355. AAAI Press, 2018.

[156] Chris Reed and Douglas Walton. Applications of argument schemes. In H. V. Hanson, C. W. Tindale, J. A. Blair, and R. H. Johnson, editors, *Proceedings of the 4th Conference of the Ontario Study of Argumentation (OSSA-2001)*, 2001.

[157] Lesley Rex, Ebony Thomas, and Steven Engel. Applying Toulmin : Teaching logical reasoning and argumentative writing. *English Journal*, 99(6) :55–61, 2010.

[158] Pierre Claude Reynard. Manufacturing quality in the pre-industrial age : finding value in diversity[the author]. *Economic History Review*, 53(3) :493–516, 2000.

[159] Rodney A. Reynolds and J. Lynn Reynolds. *Evidence*, pages 427–446. SAGE Publications, Inc., 2002.

[160] David J Rinehart, John C Knight, and Jonathan Rowanhill. Current practices in constructing and evaluating assurance cases with applications to aviation. Technical report, NASA, 2015.

[161] Patrick J Roache. *Verification and validation in computational science and engineering*. Hermosa, 1998.

[162] Stephanie Roussel, Cédric Pralet, Thomas Polacsek, François Bouissiere, Claude Cuiller, Pierre-Eric Dereux, and Stephane Kersuzan. Un outil d'ordonnancement pour interfacer conception de produit et chaîne de production en aéronautique. In *Proc. congrès annuel de la société Française de Recherche Opérationnelle et d'Aide à la Décision (ROADEF 2018)*, 2018.

[163] Manfred Roza, Jeroen Voogd, and Derek Sebalj. The generic methodology for verification and validation to support acceptance of models, simulations and data. *The Journal of Defense Modeling and Simulation*, 10(4) :347–365, 2013.

[164] John Rushby, Xidong Xu, Murali Rangarajan, and Thomas L Weaver. Understanding and evaluating assurance cases. Technical Report NASA/CR-2015-218802, NASA Langley Research Center, 2015.

[165] SAE. ARP 4754A Guidelines For Development Of Civil Aircraft and Systems. Guidelines, Society of Automotive Engineers International, 2010.

[166] Robert G Sargent and Osman Balci. History of verification and validation of simulation models. In *Proceedings of the 2017 Winter Simulation Conference (WSC)*, pages 292–307. IEEE Press, 2017.

[167] Roberto Sebastiani and Michele Vescovi. Automated reasoning in modal and description logics via sat encoding : the case study of k(m)/alc-satisfiability. *Journal of Artificial Intelligence Research (JAIR)*, 35 :343–389, 2009.

[168] Rémi Sèdes, Florence andDelmas and Thomas Polacsek. Éditorial. *Ingénierie des Systèmes d'Information*, 21(4) :7–10, 2016.

[169] Delavar G Shenas and Sepehr Derakhshan. Organizational approaches to the implementation of simultaneous engineering. *International Journal of Operations & Production Management*, 14(10) :30–43, 1994.

[170] Walter A Shewhart. Nature and Origin of Standards of Quality. *Bell System Technical Journal*, 37(1) :1–22, 1958.

[171] Herbert A Simon. The architecture of complexity. In *Facets of systems science*, pages 457–476. Springer, 1991.

[172] Mathias Soeken, Robert Wille, Mirco Kuhlmann, Martin Gogolla, and Rolf Drechsler. Verifying UML/OCL models using boolean satisfiability. In Giovanni De Micheli, Bashir M. Al-Hashimi, Wolfgang Müller, and Enrico Macii, editors, *Design, Automation and Test in Europe, DATE 2010*, pages 1341–1344. IEEE Computer Society, 2010.

[173] Richard W. Southwick. Explaining reasoning : an overview of explanation in knowledge-based systems. *Knowledge Eng. Review*, 6(1) :1–19, 1991.

[174] J. Michael Spivey. An introduction to Z and formal specifications. *Software Engineering Journal*, 4(1) :40–50, 1989.

[175] Jonette M Stecklein, Jim Dabney, Brandon Dick, Bill Haskins, Randy Lovell, and Gregory Moroney. Error cost escalation through the project life cycle. In *14th Annual International Symposium, International Council on Systems Engineering INCOSE*, pages 1723–1737. INCOSE, 2004.

[176] Einar Stefferud. The logic theory machine : a model heuristic program. Memorandum, Rand Corporation, 1963.

[177] Brian L. Stuart. Programming the eniac [scanning our past]. *Proceedings of the IEEE*, 106(9) :1760–1770, 2018.

[178] William R. Swartout. XPLAIN : A system for creating and explaining expert consulting programs. *Artificial Intelligence*, 21(3) :285–325, 1983.

[179] Frederick Winslow Taylor. *The principles of scientific management (chapter 2, page 59)*. Harper & Brothers, 1911.

[180] Stephen E. Toulmin. *The Uses of Argument*. Cambridge University Press, Cambridge, UK, 2003. Updated Edition, first edition 1958.

[181] Alain Trognon, Martine Batt, and Jennifer Laux. Why is dialogical solving of a logical problem more effective than individual solving ? : A formal and experimental study of an abstract version of Wason's task. *Language and Dialogue*, 1(1) :44–78, 2011.

[182] Raymond Turner. Specification. *Minds and Machines*, 21(2) :135–152, 2011.

[183] Raymond Turner and Nicola Angius. The philosophy of computer science. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, spring 2017 edition, 2017.

[184] Charles R. Twardy. Argument maps improve critical thinking. *Teaching Philosophy*, 27(2) :95–116, 2004.

[185] UK Ministry of Defence. Defence Standard 00-56 Safety Management Requirements for Defence Systems. Standard, UK Ministry of Defence, 2007. Def Stan 00-56 Part 1 Issue 4.

[186] UK Ministry of Defence. Defence Standard 00-42 Reliability and Maintainability Assurance Guide Part 3 : R&M Case. Standard, UK Ministry of Defence, 2008. Def Stan 00-42 Part 3 Issue 3.

[187] Axel van Lamsweerde. *Requirements Engineering - From System Goals to UML Models to Software Specifications*. Wiley, 2009.

[188] Bart Verheij. Artificial argument assistants for defeasible argumentation. *Artificial Intelligence*, 150 :291–324, November 2003.

[189] Pieter E Vermaas and Wybo Houkes. Ascribing functions to technical artefacts : A challenge to etiological accounts of functions. *The British Journal for the Philosophy of Science*, 54(2) :261–289, 2003.

[190] Georg Henrik Von Wright. Deontic logic. *Mind*, 60(237) :1–15, 1951.

[191] Douglas Walton. Practical reasoning and the structure of fear appeal arguments. *Philosophy and Rhetoric*, 29(4) :301–313, 1996.

[192] Douglas Walton. The appeal to ignorance, or argumentum ad ignorantiam. *Argumentation*, 13(4) :367–377, 1999.

[193] Alan Wassyng, Neeraj Kumar Singh, Mischa Geven, Nicholas Proscia, Hao Wang, Mark Lawford, and Tom Maibaum. Can product-specific assurance case templates be used as medical device standards ? *IEEE Design & Test*, 32(5) :45–55, 2015.

[194] Charles B Weinstock, John B Goodenough, and John J Hudak. Dependability cases. Technical report, Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, 2004.

[195] Richard Whately. *Elements of logic, comprising the substance of the article in Enciclopædia Metripolitana : with additions*. William Jackson, 1836.

[196] Ursula Wingate. 'argument!' helping students understand what essay writing is about. *Journal of English for Academic Purposes*, 11(2) :145–154, 6 2012.

[197] Eric Winsberg. Models of success versus the success of models : Reliability without truth. *Synthese*, 152(1) :1–19, 2006.

[198] Eric Winsberg. *Science in the age of computer simulation, Chapter 2*. University of Chicago Press, Chicago, IL, USA, 2010.

[199] L. Richard Ye. The value of explanation in expert systems for auditing : An experimental investigation. *Expert Systems with Applications*, 9(4) :543–556, 1995.

[200] L. Richard Ye and Paul E. Johnson. The impact of explanation facilities in user acceptance of expert system advice. *MIS Quarterly*, 19(2) :157–172, 1995.

[201] Eric S. K. Yu. Towards modeling and reasoning support for early-phase requirements engineering. In *3rd IEEE International Symposium on Requirements Engineering (RE'97)*, pages 226–235. IEEE Computer Society, 1997.