Validation de modèles de simulation

Damien Foures

Université de Toulouse

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

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

Présentée et soutenue le *26 Juin 2015* **par :**

DAMIEN FOURES

Validation de modèles de simulation

JURY

MICHEL COMBACAU	Professeur des Universités	Président du jury
CLAUDIA FRYDMAN	Professeur des Universités	Rapporteur
GREGORY ZACHAREWICZ	Maître de Conférences -HDR-	Rapporteur
VINCENT ALBERT	Maître de Conférences	Examinateur
ALEXANDRE NKETSA	Professeur des Universités	Examinateur
PAUL-ANTOINE BISGAMBIGLIA	Maître de Conférences	Examinateur

École doctorale et spécialité :
EDSYS : Génie Industriel 4200046

Unité de Recherche :
Laboratoire d'Analyse et d'Architecture des Sytèmes du CNRS (UPR 8001)

Directeur(s) de Thèse :
Alexandre NKETSA et *Vincent ALBERT*

Rapporteurs :
Claudia FRYDMAN et *Gregory ZACHAREWICZ*

Remerciements

Mes remerciements iront en premier lieu à vous, curieux, stagiaires, doctorants, chercheurs, qui êtes en train de lire ces quelques lignes, en espérant que la suite de ce document saura vous donner une certaine satisfaction. J'adresse également mes remerciements à tous ces gens, auteurs d'articles ou de livres que je cite dans ce manuscrit, et qui m'ont permis de mieux comprendre ce domaine.

Ces travaux se sont déroulés au sein du Laboratoire d'Analyse et d'Architecture des Systèmes du Centre National de la Recherche Scientifique (LAAS-CNRS) et du groupe Ingénierie Système et Intégration (ISI). Je tiens ici à remercier le directeur du laboratoire Monsieur Jean Arlat et les différents services du laboratoire de m'avoir permis de réaliser cette thèse.

Je remercie Monsieur Hamid Demmou, responsable du groupe ISI, de m'avoir accueilli au sein de son équipe de recherche. J'en profite pour remercier toute l'équipe pour son accueil et sa sympathie durant ces trois années.

Je tiens à remercier Monsieur Michel Combacau de m'avoir fait l'honneur de présider mon jury de soutenance, et d'avoir su trouver les mots justes lors du traditionnel repas de thèse.

Je remercie Madame Claudia Frydman et Monsieur Gregory Zacharewicz d'avoir accepté de rapporter ce travail. Je remercie également Monsieur Paul-Antoine Bisgambiglia (MCF) de l'intérêt particulier qu'il a porté à l'égard de mon travail et d'avoir accepté de participer au jury de soutenance.

Je remercie mon co-directeur de thèse Monsieur Vincent Albert pour ses conseils qui m'ont aidé et orienté dans l'élaboration de cette thèse.

Je suis reconnaissant envers mon directeur de thèse Monsieur Alexandre Nketsa pour la confiance et la liberté qu'il a su m'accorder pendant ces travaux.

Je souhaite également remercier Monsieur Hessam Sarjoughian, et toute l'équipe du Arizona Center for Integrative Modeling and Simulation (ACIMS) de l'Arizona State University, de m'avoir accueilli pendant ces quelques mois. Nos discussions ont sans aucun doute aiguisé mon esprit scientifique.

(*I also wish to thank Mr. Hessam Sarjoughian and all the members of the Arizona Center for Integrative Modeling and Simulation (ACIMS) team, Arizona State University for hosting me during those few months. Our discussions have undoubtedly sharpened my scientific spirit.*)

Je remercie également tous mes collègues du département EEA de leur amitié et leurs précieux conseils.

Les derniers mois de ces travaux se sont déroulés au sein d'un autre laboratoire, le SPE (Science Pour l'Environnement) dans le projet de recherche TIC (Technologies de l'Information et de la Communication). Je tiens ici à remercier le directeur du laboratoire Monsieur Paul-Antoine Bisgambiglia (PR) de m'avoir accueilli au sein de son laboratoire. Je remercie également Monsieur Jean-François Santucci, responsable du projet TIC, ainsi que toute l'équipe, de m'avoir accueilli au sein de leur projet de recherche.

Durant la scolarité, les plus chanceux d'entre nous ont un jour croisé un enseignant pas comme les autres, plus passionnant, plus charismatique, qui a su nous parler, nous rendre curieux, critiques, rigoureux et travailleurs. Tous ces éléments ont été indispensables pour réaliser ce manuscrit. C'est pourquoi je remercie ici Monsieur Francis Demanze, qui reste aujourd'hui encore une source d'inspiration importante dans mon rôle d'enseignant-chercheur.

Je tiens à remercier les collègues et amis de mes différents laboratoires : Merci à Sylvain, le génie des mots croisés en amphi, à qui il n'aura fallu qu'un papier pour briller. Merci à Vikas, qui m'a fait voyagé et grâce à qui je peux aujourd'hui parler avec deux milliards de personnes. Merci aussi d'avoir partagé ton bureau et répondu à mes questionnements philosophiques. Merci à Romain, oreille attentive durant cette rédaction et compagnon de cordée en devenir, à qui je souhaite une même prochaine réussite. Merci aussi à Anaïs, Saurabh (Thanks), Roberto, Manu, Coquinou, Hélène, Laeti, Wani, Abdurrahman (Thanks), Soroosh (Thanks), Jean-Baptiste, Savitha (Thanks), Camille, Moussa, Sylvain, Olivier, Mathilde, Hung, Gauthier, Aurélia et bien d'autres !

Je tiens à remercier plus particulièrement celui qui hier encore était Toulousain, Sven, qui après nos années magiques a su ouvrir le chemin vers les études longues, en me montrant que cela n'était pas réservé qu'aux autres.

Je tiens également à remercier celle qui aussi loin que je me souvienne a été ma plus fidèle amie, Marion, relectrice acharnée de ce manuscrit, sachant déjouer les facéties de la langue française.

Je tiens également à remercier tous mes amis, qui ont su accompagner de rire et de bonheur ces années de thèses avec leurs questions : « Mais tu cherches quoi exactement ? », « C'est quoi en fait ton métier ? » et « Tu soutiens quand au fait ?». Dans le désordre, La tocard Team : Boubou, Croc, Tony. Les grimpeurs : Mathieu, Katia, Vérane, Franck, Laeti, Amalia, Benjamin, GodziMila. Les anciens : Aurélie, Fred, Carole, Marto, Clément. Ceux qui sont partis voir si l'herbe était plus verte ailleurs : John, Hugues, Marion, Hanaé, Sara, Djou, Jean-Hugues, Djo, Yi.

Je tiens également à remercier ma famille qu'elle soit Ariègeoise, Aveyronnaise, Parisienne, Héraultaise, Mauricienne, Basques, Haut-Garonnaise, Tarn-et-Garonnaise, de près ou de loin, vous avez su me soutenir tout au long de ce travail et avez été présents le jour J.

Mes plus profonds remerciements vont à mes parents. Tout au long de mon cursus, ils m'ont toujours soutenu, encouragé et aidé. Ils ont su me donner toutes les chances pour réussir. Qu'ils trouvent, dans la réalisation de ce travail, l'aboutissement de leurs efforts ainsi que l'expression de ma plus profonde gratitude.

Enfin, je remercie Lucille qui a su composer avec mes humeurs et les multiples relectures, tout en me soutenant et en m'encourageant. Notre couple a grandi en même temps que mon projet scientifique, le premier servant de fondation à l'épanouissement du second. Cette thèse et moi te devons beaucoup. Merci.

Le 16 Juillet 2015, à Corte
Damien FOURES.

Table des matières

Table des figures

Introduction générale

Tout élément en interaction avec un autre peut être considéré comme un système. Aujourd'hui, certains des systèmes conçus par l'homme doivent répondre à des problématiques complexes et opérer dans un contexte critique. L'ingénierie système a dû constamment s'adapter pour répondre aux besoins croissants en complexité et en criticité des applications. L'utilisation de modèles a été une des réponses à cette croissance, en permettant de s'abstraire de problématiques plus techniques, et en autorisant l'abord du système selon des points de vue et des objectifs différents. Le modèle permet de décrire ce qui constitue le système, pour mieux comprendre son comportement et son évolution. La première utilisation des modèles était d'ordre contemplative, ils étaient un support à la réflexion et à la communication. L'interaction entre les modèles était alors limitée à la seule force de pensée des ingénieurs.

La disponibilité des outils informatiques dans les années 1950-1960 va rapidement s'intégrer dans le processus d'ingénierie système. Il est alors possible d'implémenter le modèle dans un langage informatique pour simuler son comportement sur un ordinateur afin de l'étudier. Il devient indispensable de s'assurer que tous les acteurs de la simulation aient une compréhension identique du modèle, sans erreur d'interprétation sur sa signification.

Aujourd'hui, la simulation est présente tout au long des processus d'ingénierie, comme moyen d'aide à la décision et de validation des systèmes en cours de développement. De nombreux acteurs, industriels et académiques, s'investissent dans ce domaine émergent portant le nom de Modélisation et Simulation (M&S). On pourra citer la création du *Modeling and Simulation Office* en 1991, connu aujourd'hui sous le nom de *Modeling and Simulation Coordination Office*, le *Navy Modeling and Simulation Management Office* en 1995, ou encore la division *Modélisation-Simulation-Expérimentation* en France.

L'activité de validation des modèles et des simulations est devenue une activité incontournable, pour s'assurer du crédit que l'on peut porter aux résultats fournis par la simulation, en fonction des objectifs d'utilisation, mais aussi de l'environnement d'exécution.

Il est possible de considérer la simulation comme un ensemble de modèles, représentant d'un coté le système à étudier, et de l'autre l'environnement d'exécution et les objectifs de simulation. De ce fait, la simulation ajoute de nouvelles contraintes et regroupe de nombreuses sources potentielles d'erreurs (modèle de l'environnement, objectif de simulation, choix d'implémentation, plateforme d'exécution, etc.). Ces éléments sont susceptibles d'impacter le niveau de crédibilité des résultats de simulation.

Il est primordial d'étudier la relation liant le modèle du système au modèle de l'environnement. C'est dans ce contexte que B.P. Zeigler fait une proposition dans

l'ouvrage "Theory of modeling and simulation" [Zeigler 2000] :

"*The solution lies in a capable tool set that recognizes the influence of modeling objectives and error tolerances, the multidimensional choices of bases for agregation mappings. Formalizing such dimensions of the problem leads to a sophisticated framework that involves concepts such as scope/resolution product, experimental frame, applicability and derivability lattices, and conditions for valid abstractions*". B.P. Zeigler

Sans apporter de réelle solution à cette problématique, B.P. Zeigler soulève ici l'absence de démarche standardisée permettant d'évaluer la crédibilité des résultats de simulation. Il propose de s'intéresser à la relation entre le modèle du système d'intérêt et les objectifs de simulation.

Les travaux de V. Albert, sur l'évaluation de la validité de la simulation dans le cadre du développement des systèmes embarqués [Albert 2009], reprend le principe énoncé par B.P. Zeigler. Il propose un ensemble de règles formelles permettant une mise en correspondance entre objectifs de simulation et domaine d'usage du modèle. Il s'agit ainsi de proposer une approche générale d'évaluation statique de la validité des modèles de simulation.

Cette thèse s'inscrit dans la suite des travaux de V. Albert. Elle propose un apport théorique pour évaluer la validité d'une abstraction, i.e. évaluer si la simulation d'un modèle abstrait et la simulation d'un modèle de référence conduisent au même jugement pour l'objectif considéré. L'idée est de proposer au modèle un ensemble d'informations permettant de préciser son domaine de validité. Ces informations incluent :

- les moyens de controlabilité permettant de stimuler le modèle,
- les moyens d'observabilité permettant d'observer les effets des stimuli d'entrée sur le modèle,
- le type et la structure des données échangées ,
- les traces d'exécutions acceptables.

L'utilisation de ces informations, au travers de métriques, va permettre de statuer sur l'efficacité du cadre expérimental à satisfaire les objectifs de simulation.

La thèse en quelques lignes

Ce sont donc deux axes principaux d'étude qui se dégagent de cette thèse :

- **Une démarche formelle d'élaboration des métriques par une approche ensembliste**, qui permet d'améliorer la crédibilité de la simulation, en évaluant l'efficacité du cadre expérimental vis-à-vis des objectifs de simulation.
- **La description du langage abstrait SiML**, qui permet de guider l'utilisateur de la simulation dans une démarche méthodique pour l'étude de la

validité des modèles de simulation.

Plan du manuscrit

Après cette introduction générale, le premier chapitre définit le cadre de travail de cette thèse et a la vocation d'être accessible au plus grand nombre. Il présente l'ingénierie des systèmes complexes. Il s'intéresse à la définition de systèmes complexes et à leur conception. Il présente en particulier la phase de validation et de vérification des systèmes complexes, nous permettant ainsi de définir précisément le périmètre de notre étude.

Le deuxième chapitre de ce manuscrit propose un état de l'art détaillé autour de la validation et de la vérification. Il présente les concepts associés à la problématique, qui sont la modélisation, la simulation, la validation, la vérification, les relations de dérivabilité ou de morphisme, le cadre expérimental et la notion de compatibilité entre un modèle et son environnement.

Le troisième chapitre s'intéresse à la qualification de la validité d'un modèle de simulation. Après une présentation du principe de crédibilité dans un contexte de M&S, ce chapitre décrit les métriques présentes dans la littérature. Deux approches sont ensuite proposées pour l'élaboration de nouvelles métriques, destinées à l'évaluation de la validité de la simulation. Une première approche est basée sur les automates à interfaces, la seconde approche est basée sur un formalisme de spécification des systèmes à évènements discrets (DEVS [1]). Chacune de ces approches montrera des avantages et des inconvénients dans l'élaboration des métriques.

Le quatrième chapitre décrit la méthodologie d'élaboration des métriques et propose l'élaboration d'un langage dédié à la simulation. Celui-ci permet d'orienter l'utilisateur de la simulation dans la démarche de construction du modèle de simulation. Ce chapitre commence par introduire le principe de méta-modélisation, ainsi que les différentes techniques permettant de décrire la sémantique d'exécution d'un langage abstrait. L'élaboration de ce langage est ensuite décrite en deux temps : tout d'abord avec une description générale du langage, puis avec l'étude détaillée du sous-ensemble, permettant la capture du comportement du couple modèle-cadre expérimental. Pour finir, ce chapitre présente une syntaxe concrète possible avec l'outil ProDevs.

Le cinquième et dernier chapitre de ce manuscrit présente un exemple d'application proposé par Y. Iwasaki et AY. Levy dans [Iwasaki 1994] et permet d'illustrer les chapitres précédents. Cet exemple nous permettant d'appliquer la méthodologie pour l'élaboration des métriques avec l'utilisation du langage SiML. L'injection d'erreurs dans le processus de modélisation nous permettra de montrer le potentiel

1. DEVS : Discrete Event System Specification

de détection des métriques, ainsi que leur force de persuasion, pour améliorer la crédibilité de la simulation.

Ce manuscrit se termine avec une conclusion générale faisant le bilan de notre contribution, et permettant d'ouvrir vers les perspectives qui se dégagent de ces travaux, afin de les améliorer.

La figure 1 propose une méthode de lecture dépendante des affinités avec les différents domaines abordés dans ce manuscrit. Chaque chapitre de cette thèse est composé d'un état de l'art, suivi de la contribution associée et d'une section discussion, pouvant être considérée comme une réflexion sur les limites ou les perspectives liées à la contribution. De manière générale, nous considérons que l'*expert* maîtrise l'état l'art du domaine (IS, V&V de la M&S, IDM, UML) et a une connaissance générale des problématiques associées. Il peut alors se concentrer sur un rappel de la problématique et la contribution associée. Un lecteur *intermédiaire* connaît les fondamentaux du domaine, mais sera dirigé vers les parties plus spécialisées de l'état de l'art avant de poursuivre le chemin de l'*expert*. Le novice, lui, pourra aborder le chapitre dans son ensemble pour une entrée progressive dans la problématique et la contribution.

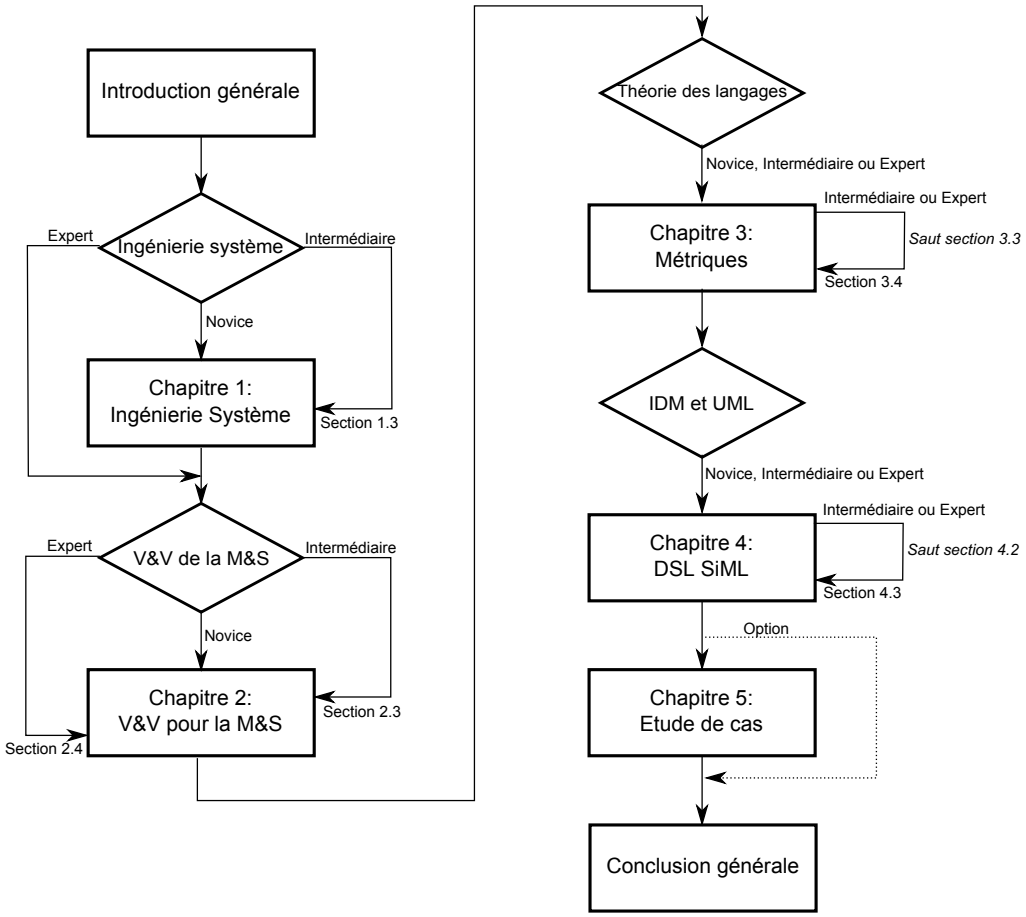


FIGURE 1 – Diagramme de lecture du manuscrit

Chapitre 1

Ingénierie des systèmes complexes et simulation

Sommaire

1.1 Introduction

Ce chapitre est une introduction à l'ingénierie des systèmes (IS) et à l'ingénierie de la simulation. Il donnera un rapide historique de l'IS afin d'en comprendre les motivations et les enjeux. D'abord composée de pratiques peu formalisées, l'IS sera finalement vue à travers le monde comme une discipline à part entière avec ses normes et ses méthodes.
L'ingénierie de la simulation et ses différents aspects seront abordés. Autrefois relayée au second rang, la *M&S* devient aujourd'hui un élément clef pour le développement, la sécurité, la fiabilité et la compétitivité des systèmes. Discipline en relation étroite avec l'IS, elle se voit aujourd'hui attribuer de nouveau enjeux et méthodes qu'elle va devoir maitriser. Nous décrirons ici comment ces méthodes s'insèrent dans l'*IS*.

Pour finir, un ensemble de définitions, de normes et de standards sera proposé comme référence pour la suite de ce document.

1.2 Systèmes complexes

La résolution de problèmes fait partie intégrante de l'évolution de la nature humaine. En imaginant de nouveaux outils, matériels et techniques, l'Homme a su évoluer et apprendre à survivre jusqu'à devenir ce qu'il est aujourd'hui. Si les premiers éléments proposés par l'Homme étaient simples, l'interaction et la composition de ces éléments constituent déjà des "systèmes". La complexité des systèmes n'a cessé de croitre pour résoudre des problèmes de plus en plus complexes. La solution n'est plus apportée par un élément seul mais par l'apparition de propriétés émergentes (cf. 1.2.3), engendrées par l'interaction entre plusieurs éléments. L'Homme a dû se spécialiser et créer un environnement favorable à l'élaboration de solutions à des problèmes complexes.

1.2.1 Systèmes

Il existe un grand nombre de définitions du terme système. Nous allons en voir quelques unes afin de mettre en évidence les termes communs et leurs différences.

L. Von Bertalanffy énonce dés 1956 une définition de système dans "*General system theory*" :

Définition 1. *[...] un système est un ensemble d'éléments en interaction ouvert à leurs environnements [...]. En outre, ils peuvent acquérir qualitativement de nouvelles propriétés émergentes,[...].*
[Von Bertalanffy 1956]

L'INCOSE (International Council On Systems Engineering) définit un système de la façon suivante :

Définition 2. *Un **système** est un ensemble d'éléments interdépendants qui interagissent entre eux de façon organisée et forment un ensemble unique. [INCOSE 2004]*

JP. Meinadier nous présente sa propre définition de système dans le livre "*Ingénierie et intégration des systèmes*" :

Définition 3. *Un **système** est un ensemble composite de personnels, de matériels et de logiciels organisés pour que leur interfonctionnement permette, dans un environnement donné, de remplir les missions pour lesquelles il a été conçu.[Meinadier 1998]*

L'AFIS (Association Française d'Ingénierie Système) possède une autre définition de système :

Définition 4. *Un **système** est décrit comme un ensemble d'éléments en interaction entre eux et avec leur environnement, intégré pour rendre à son environnement les services correspondants à sa finalité. Un système présente donc des propriétés nouvelles résultant des interactions entre constituants : si l'on intègre des éléments pour faire un système, c'est bien pour bénéficier des effets de synergie résultant de leurs interactions. [AFIS 2014]*

Il apparaît clairement que l'interaction est la quintessence des systèmes. C'est elle qui permet l'apparition de propriétés émergentes dépassant celles propres à chaque éléments. Si la définition de l'INCOSE semble vouloir se limiter à cet aspect, JP. Meinadier, L. Von Bertalanffy et l'AFIS évoquent d'autres aspects essentiels des systèmes comme l'*environnement* et la *mission*, deux constituants fondamentaux de cette thèse.

D'autres définitions s'orientent plus vers la notion de produit et de processus, comme celle de l'EIA-632 (Electronic Industrie Alliance) :

Définition 5. *Un **système** est une agrégation de produits nécessaires à la réalisation d'un but donné. Un système est composé de produits nécessaires à la réalisation de la fonction opérationnelle et de produits nécessaires à sa réalisation, sa mise en service, son maintien et son retrait de service [EIA 2003].*

1.2.2 Systèmes complexes et systèmes compliqués

S'il est une distinction essentielle à connaitre pour maîtriser la notion de *système complexe*, c'est la différenciation en *complexe* et *compliqué*.

C'est JL. Le Moigne qui nous précise que l'intelligibilité du compliqué se découvre par simplification alors que l'intelligibilité du complexe se construit par modélisation [Le Moigne 1990].

Cela signifie que le compliqué est lié à une superposition de composants qui conduit à une accumulation de difficultés, alors que la complexité d'un système est davantage inhérente aux combinaisons et interactions nombreuses et variées entre composants qu'aux composants eux-mêmes.

Un système compliqué peut être ramené à ces éléments essentiels : on le « mutile » pour l'analyser. Un système complexe doit être modélisé pour le comprendre, en aucun cas « mutilé ». Une simplification hasardeuse ou une mutilation pourrait détruire son intelligibilité.

Les systèmes complexes existent à l'état naturel : l'interaction des cellules nerveuses dans le cerveau, une colonie de fourmis, un marché économique, sont autant de systèmes complexes. Il existe aussi des systèmes complexes artificiels, les plus proches sont dans notre poche alors que d'autres sont à quelques 18 milliards de kilomètres de notre planète (sonde Voyager 1).

La terme système complexe est défini dans plusieurs travaux [EIA 2003, Sage 2000, IEEE 2005, Haskins 2011, Guillerm 2011].

Définition 6. *Un **système complexe** est un ensemble d'éléments en interaction, organisé pour atteindre un ou plusieurs objectifs donnés. C'est un ensemble intégré d'éléments, de sous-systèmes ou assemblages qui permettent de réaliser un objectif*

défini. Ces éléments comprennent les produits (matériels, logiciels, micrologiciels), processus, personnes, informations, techniques, structures, services et autres supports. [Haskins 2011]

Cette première définition de l'INCOSE, qui fait apparaître l'interaction entre éléments, peut être complétée par la définition donné par G. Wang d'un système complexe où la notion de complexité apparait telle qu'expliquée par JL. Le Moigne :

Définition 7. *Un **système complexe** est un système qui par sa conception ou ses fonctions ou les deux est difficile à comprendre et à vérifier.[Weng 1999]*

1.2.3 Propriétés émergentes

La définition de système et de système complexe a déjà fait apparaître le concept de propriété émergente. Il est intéressant d'énoncer le principe général : *Le système pris dans son ensemble est plus que la simple somme de ses constituants* [Von Bertalanffy 1956]. En d'autres termes : l'interaction de ces constituants amène de nouvelles caractéristiques au système émergent.
L'émergence est un domaine de recherche à part entière qui n'est pas exploré dans ce document. Cette étude sur la validation des modèles de simulation amènera toutefois à observer et traiter des propriétés émergentes.

1.3 L'ingénierie des systèmes

Si la compréhension de systèmes complexes existants n'est pas réservée à un domaine scientifique particulier, il en va différemment pour la conception d'un système complexe qui fait intervenir un domaine expert : L'Ingénierie des Systèmes.

1.3.1 Ingénierie des systèmes

1.3.1.1 Historique et Définition

Le terme "*Ingénierie des systèmes*" (IS) apparait semble-t-il pour la première fois au début des années 30 dans le laboratoire "*Bell Telephone laboratories*". L'idée de l'associer à une discipline à part entière viendra du MIT en 1950 [INCOSE 2013]. L'IS ne sera reconnue officiellement comme discipline qu'en 2002 grâce à l'introduction d'un standard international ISO/IEC 15288 [Haskins 2011]. Pourtant, dès les années 50, l'ingénierie des systèmes va évoluer. La course à l'espace et l'armement atomique menée par les États-Unis dans les années 50, 60 et 70 apporte des exigences en termes de performance, de fiabilité, d'échéance calendaire et de gestion de coût, ce qui contribuera grandement à la création et à l'amélioration des outils et des méthodes. Le domaine doit sans cesse innover pour pallier la complexité grandissante des systèmes.

La figure 1.1 illustre l'augmentation du nombre de mots composant un logiciel embarqué à bord d'un avion entre 1965 et 1993 (données issues de [Montalk 1993]). Sans qu'il existe de corrélation établie entre nombre de mots et complexité d'un

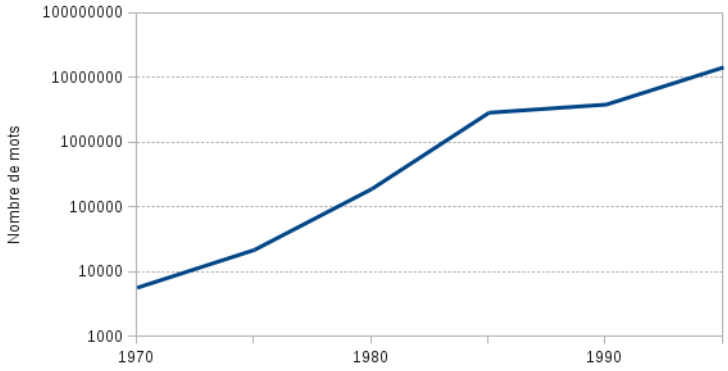


FIGURE 1.1 – Augmentation de la complexité des logiciels avion (Airbus 1965-1993)

logiciel, on imagine les enjeux de l'ingénierie système et la complexité croissante à laquelle elle a du faire face. Fait amusant, la documentation nécessaire pour codifier les exigences de qualité, pour décrire le logiciel, pour définir les essais requis et pour documenter les résultats du logiciel embarqué dans l'A320 est impressionnante : 16 800 pages, qui, imprimées sur du papier standard formeraient une pile aussi haute qu'un homme moyen. A noter qu'environ 90% de cette documentation est en fait là à des fins de qualité ([Montalk 1993]).

On résume communément l'IS comme une approche scientifique interdisciplinaire, dont le but est de formaliser et d'appréhender la conception de systèmes complexes avec succès. Il existe pourtant un nombre important de définition de l'IS, ce paragraphe en présentera une sélection avant d'en synthétiser les idées principales et de donner la définition retenue pour la suite.

Certains comme JP. Meinadier apportent une nuance entre *Ingénierie système* et *Ingénierie de systèmes*.

Définition 8. ***Ingénierie système*** *: Phase de définition du système spécifiant les propriétés attendues et débouchant sur la spécification des constituants et de leurs interfaces. (Cette phase prépare le travail des métiers et l'intégration système.)*

Définition 9. ***Ingénierie de systèmes*** *: Ensemble des activités d'ingénierie système et intégration système auxquelles s'ajoute la coordination des différents métiers mis en œuvre.*

L'AFIS de son coté tend à rassembler les deux termes sans distinction entre phase de définition, d'intégration ou de coordination, et fait intervenir la notion de parties prenantes [1] . C'est la définition qui sera retenue pour cette thèse.

1. Une partie prenante est un acteur (groupe, individu, ou organisme), concerné par tout ou

Définition 10. *L'**ingénierie Système** (ou ingénierie de systèmes) est une démarche méthodologique générale qui englobe l'ensemble des activités adéquates pour concevoir, faire évoluer et vérifier un système apportant une solution économique et performante aux besoins d'un client tout en satisfaisant l'ensemble des parties prenantes. [AFIS 2014]*

L'AFIS décrit plus en détail l'ingénierie système comme un processus :

- Coopératif et interdisciplinaire de résolution de problèmes,
- S'appuyant sur les connaissances, méthodes et techniques issues de la science et de l'expérience,
- Mis en œuvre pour définir, faire évoluer et vérifier la définition d'un système (ensemble organisé de matériels, logiciels, compétences humaines et processus en interaction)
- Apportant une solution à un besoin opérationnel identifié conformément à des critères d'efficacité mesurables,
- Qui satisfasse aux attentes et contraintes de l'ensemble de ses parties prenantes et soit acceptable pour l'environnement,
- En cherchant à équilibrer et optimiser sous tous les aspects l'économie globale de la solution sur l'ensemble du cycle de vie du système.

Comme nous le fait remarquer J. Verries [Verries 2010], il existe des définitions plus personnelles comme celle de D. Hitchins, président de l'INCOSE, en 2007 :

Définition 11. *L'**IS** est l'art et la science de créer des systèmes efficaces, utilisant le système dans son intégralité et dans l'intégralité de la vie opérationnelle du système ; ou l'art et la science de créer une solution optimale à un problème complexe.*

En évoquant l'art, D. Hitchins nous rappelle que l'humain est indissociable du processus d'ingénierie système où l'arbitrage et l'expertise sont parfois irremplaçables dans la recherche de la solution.

Au travers de cet ensemble de définitions, on voit que l'Ingénierie Système est une approche méthodique, s'appuyant sur le retour d'expérience, pour aider le concepteur à obtenir une solution équilibrée entre coût, délai et répondant au mieux aux besoins du client et des différentes parties prenantes.

1.3.1.2 Cycle de développement

L'approche Ingénierie Système passe inévitablement par une description chronologique de la méthode, appelée cycle de développement. Il existe un nombre important de *cycles de développement* et de représentations associées. Les principaux sont : le cycle en V [Forsberg 2005] , le cycle en cascade [Royce 1970] et le cycle en spirale [Boehm 1988] ; mais on pourrait également citer le cycle itératif, le cycle

partie du projet ; dont les intérêts peuvent être affectés positivement ou négativement à la suite de sa réalisation (ou de sa non-réalisation).

semi-itératif et un nombre important de cycles dérivés de ces 5 grandes familles. L'apparition de ces variantes s'explique par des besoins plus spécifiques dans certains domaines (contraintes organisationnelles, adaptation à des outils, etc.).

Cycle en V La figure 1.2 donne un exemple de cycle de développement en V. Le cycle en V est aujourd'hui la méthode de développement la plus utilisée dans l'IS. Si les noms et le nombre d'étapes changent d'un ouvrage à l'autre, le principe général reste identique. Le cycle est composé d'une branche descendante correspondant à une démarche de raffinement (décomposition et définition) et d'une branche ascendante détaillant les phases d'intégration et de vérification.

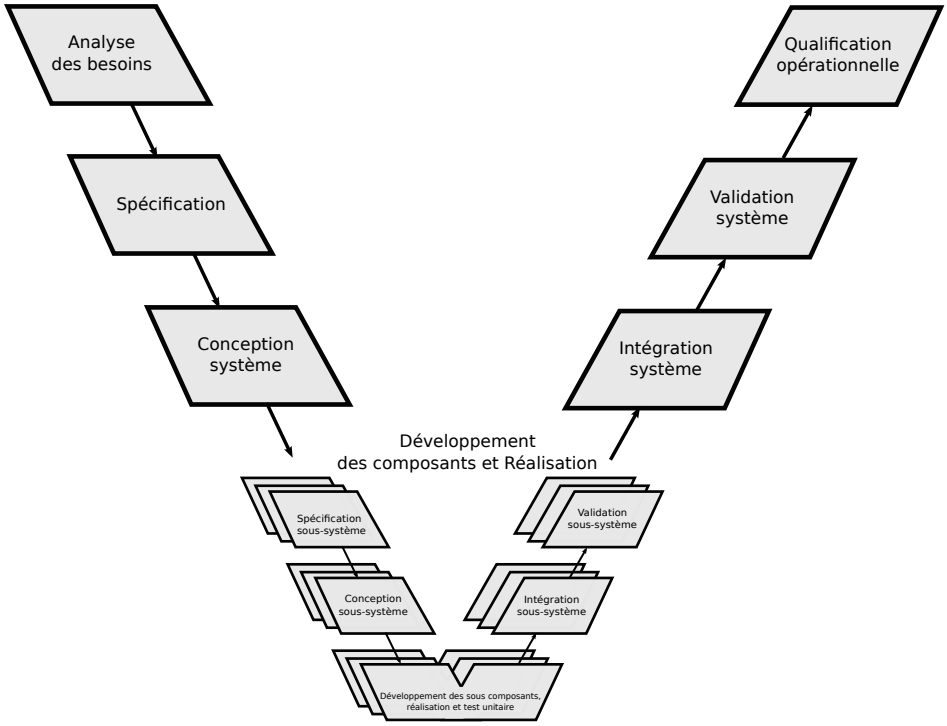


FIGURE 1.2 – Exemple de cycle de développement en V d'un système

Voici de façon synthétique les différentes phases du cycle en V pour décrire un système :

Analyse des besoins : Expression bien formulée des besoins du client et de l'ensemble des parties prenantes ayant un lien avec le système à concevoir.Elle définit la mission du système.

Spécification : La spécification système définit de façon détaillée et rigoureuse les exigences que doivent respecter le système et les sous-systèmes. Elle exprime

au travers de ces exigences ce que doit faire le système et non comment il doit le faire (aucun apport de solution technique). Les exigences peuvent être de deux types. Elles peuvent transmettre un besoin fonctionnel (exigences fonctionnelles) ou un besoin qualitatif (exigences non fonctionnelles). Cette phase constitue un point important pour la réussite d'un projet. La spécification doit éviter toute ambiguïté, tout oubli ou incohérence avec la phase amont. Durant cette phase, est ajouté le plan de validation qui consiste à définir les exigences qui devront être vérifiées, ainsi que la manière dont elles seront vérifiées.

Conception système : Le système va être décomposé de façon itérative en sous-systèmes. Les exigences issues de la spécification sont alors distribuées entre les différents sous-niveaux. On obtient un dossier de conception, composé de spécifications techniques et de l'architecture système. Ce dossier est accompagné du plan d'intégration et des tests d'intégration à effectuer. La validation de cette phase s'effectue aux travers des prototypes, modèle et/ou simulation.

Développement des composants et Réalisation : L'idée générale est ici de reprendre le cycle en V pour les sous-systèmes de façon itérative. On retrouve la spécification sous-système, la conception détaillée, la réalisation, les tests unitaires, l'intégration sous-système et la validation sous-système. Sans être normée, c'est en général le moment où la conception n'est plus assurée par les métiers de l'IS mais par un seul métier (réalisation d'une carte, d'un logiciel, d'une puce, d'une structure...). Il est essentiel d'avoir une définition parfaite des interfaces des composants développés par des équipes différentes afin d'assurer la réussite de l'intégration.

Intégration système : C'est la phase d'assemblage des composants. Chaque composant, après avoir fait l'objet d'une validation, est inséré suivant le plan d'intégration défini en phase de conception. Le nouveau système constitué est alors soumis à une nouvelle phase de validation. Il en résulte le système intégré prêt à être validé à son tour.

Validation système : En suivant le plan de validation défini en phase de spécification, on vérifie exigence par exigence que le système intégré est conforme à ces spécifications. Le système est alors prêt pour la phase de qualification opérationnelle.

Qualification opérationnelle : Parfois appelée recette, c'est la phase de test de la conformité aux besoins opérationnels définis par le client et les parties prenantes en phase d'analyse des besoins.

Le modèle de développement en V n'est pas parfait et souffre de certaines lacunes. La séquentialité et la linéarité du modèle en V ne font pas apparaitre l'ensemble des rétroactions des phases aval vers les phases amont, les cycles de développement intermédiaires, ...
Par ailleurs, l'approche descendante (top down) du cycle en V ne facilite pas l'intégration de composants existants. Il est clair qu'un système complexe est analysé

avec une approche descendante, mais concrètement le système est constitué en majorité de composants existants sur le marché. Le système est donc construit dans une approche ascendante (bottom up) que ne montre pas le cycle en V. L'aspect séquentiel de la méthode oblige à bien concevoir chaque phase du cycle, mais en pratique c'est rarement le cas (mauvaise interprétation des besoins client, incohérence de la spécification, mauvais choix de conception).

Cycle en Spirale La figure 1.3 donne un exemple de cycle de développement provenant des méthodes agiles, la méthode Spirale, qui a été énoncée par B.W. Boehm en 1986 dans un article intitulé «A Spiral Model of Software Development and Enhancement » [Boehm 1986]. Ce modèle reprend les différentes étapes du cycle en V mais permet de réitérer autant de fois que nécessaire le cycle complet, en apportant à chaque itération de nouvelles fonctionnalités, une meilleure réponse aux besoins du client et une meilleure robustesse. L'analyse de risques et l'utilisation de prototypes sont deux autres dominantes de cette méthode, où l'on estime qu'il est plus efficace d'expérimenter avec les utilisateurs (prototypage) puis de rectifier, que de réfléchir longuement pour tenter d'obtenir directement la bonne solution. Cette méthode, très employée dans le génie logiciel n'a encore jamais été utilisée pour les systèmes complexes.

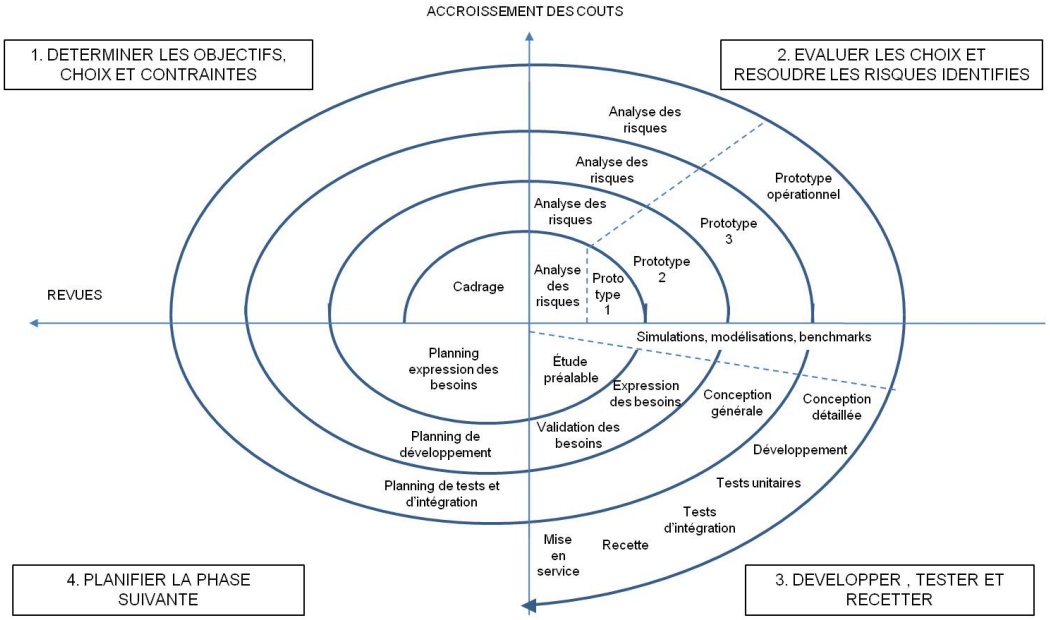


FIGURE 1.3 – Le cycle de développement en spirale selon B. Boehm. (Image extraite de [Vienne 2014])

En résumé, les activités de base pour la réalisation dans l'ingénierie système sont :

1. définir la mission du système,
2. établir les exigences à satisfaire,
3. élaborer une conception,
4. itérer la démarche pour les sous-systèmes,
5. vérifier que les exigences soient respectées,
6. valider que la mission effectuée satisfasse le client.

Le cycle de développement du système est parfois appelé à tort cycle de vie du système, alors que ce dernier fait intervenir d'autres champs d'actions de l'IS. Le cycle de vie ne s'intéresse pas uniquement à la conception du système mais à l'ensemble du cycle de vie du système de la conceptualisation au retrait de service, comme l'illustre la figure 1.4.

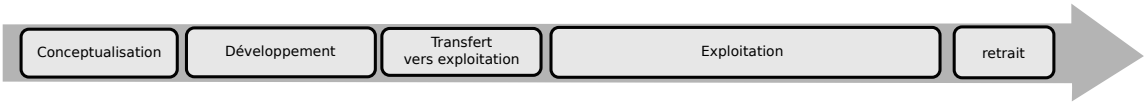


FIGURE 1.4 – Les phases du cycle de vie

La phase de conceptualisation étudie la faisabilité et exprime les besoins. La phase de développement est équivalente au cycle de développement vu précédemment. La phase de transfert vers l'exploitation fait le lien entre conception et utilisation. Elle comprend l'intégration dans l'environnement et la mise en route du système. La phase d'exploitation fait intervenir plusieurs aspects qui sont la fabrication, l'installation, le déploiement, la maintenance et les cycles de réingénierie. La dernière phase rappelle qu'un système va de la naissance à la mort avec le retrait de service.

1.3.1.3 Processus et norme

Quelle que soit la stratégie de développement utilisée pour concevoir un système, les activités sont identiques [Evrot 2008]. Il est donc possible de définir l'IS par un ensemble d'activités en intéraction. C'est ce que l'on appelle un processus.

Définition 12. *Un **Processus** est un ensemble d'activités interactives coordonnées, défini en vue de répondre à la mission par progression d'élément d'entrée en produit.*

L'AFIS fournit une définition détaillée et hiérarchique de la notion de processus :

— Le **processus *normalisé*** : défini dans des normes, il décrit les types d'activités à réaliser et les résultats attendus de ces activités. Les normes résultent d'un certain consensus entre des experts des entreprises les plus avancées.

— Le **processus *défini*** ou ***institutionnalisé*** dans un organisme : résultant souvent de l'ajustement d'une norme aux besoins de l'organisme, il intègre les bonnes pratiques acquises et les méthodes choisies par l'organisme pour réaliser les activités. La conception et l'amélioration des processus institutionnalisés relèvent de l'ingénierie des processus.

— Le **processus *ajusté*** à un projet : c'est une adaptation d'un processus institutionnalisé aux justes besoins du projet. Il prend en compte les contraintes du projet et en consomme des ressources. Cet ajustement est un des éléments de la planification du projet, dont les tâches techniques sont généralement des instances d'activités des processus appliquées aux éléments de l'arborescence des produits à réaliser (constituants du système et produits contributeurs) : chacun de ces éléments doit, en effet, être spécifié, conçu, intégré, vérifié et validé.

Il est important de bien comprendre la notion de processus car les normes [2] d'Ingénierie Système sont définies en terme de processus normalisé.

L'utilisation des normes dans l'ingénierie système revêt un intérêt tout particulier. Si, dans la vie quotidienne, elles peuvent être vues comme un ensemble de contraintes allant à l'encontre de la réalité de terrain, il n'en est pas de même dans l'IS. Les normes de l'IS sont nées d'une capitalisation de savoir-faire et d'expériences des grandes entreprises. Elles reprennent les bonnes pratiques et les échecs industriels passés pour créer ou améliorer les activités censées éviter ces échecs.

Il existe actuellement trois normes disponibles pour décrire les processus de l'IS : l'IEEE-1220 [IEEE 2005], l'EIA-632 [EIA 2003] et l'ISO-15288 [ISO/IEC 2008]. Ces normes s'étendent sur des zones différentes du cycle de vie du système comme l'illustre la figure 1.5.

Il n'est pas établi qu'une norme domine. En effet, elles abordent le cycle de vie de façon d'autant plus approfondie que leur champ est limité. Elles ne sont pas en concurrence mais se complètent. L'AFIS et l'INCOSE affichent la norme ISO 15288 comme étant le référentiel choisi. Ces organismes s'intéressent à l'ensemble des pro-

2. Afin d'éviter toute confusion : Le terme anglais "*standard*" se traduit par "*norme, standard*" en français, ce qui peut parfois prêter à confusion car la sémantique de ces deux termes diffère en francais. Le terme *standard* est défini comme un ensemble de recommandations développées et préconisées par un groupe représentatif d'utilisateurs. De son coté, la *norme* est un document établi par un consensus et approuvé par un organisme reconnu, qui fournit, pour des usages communs et repérés, des règles, des lignes directrices ou des caractéristiques, pour des activités ou leurs résultats, garantissant un niveau d'ordre optimal dans un contexte donné. Nous parlons ici de normes de l'IS et non des standards de l'IS comme on peut parfois le lire.

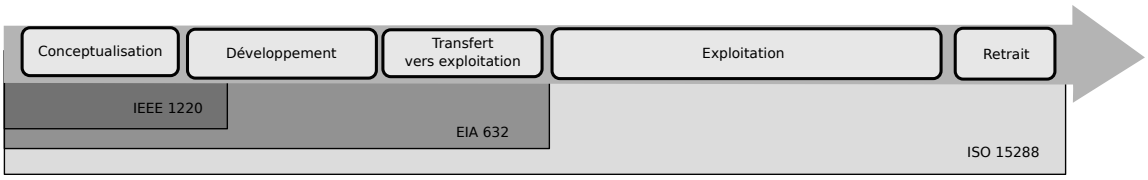


FIGURE 1.5 – Couverture des 3 grandes normes de l'IS

cessus de l'IS, il apparaît logique qu'ils s'appuient sur la norme la plus large.

L'AFIS donne une vision synthétique de ces normes :

Norme IEEE 1220 : Issue du standard militaire MIL STD 499B, cette norme de l'IEEE, dont la version initiale date de 1994, se focalise sur les processus techniques d'ingénierie système allant de l'analyse des exigences jusqu'à la définition physique du système. Les trois processus d'analyse des exigences, d'analyse fonctionnelle et d'allocation et de synthèse, finement détaillés, comprennent chacun leur sous-processus de vérification ou de validation. Le processus d'analyse système a pour but d'analyser dans un cadre pluridisciplinaire les problèmes (conflits d'exigences ou solutions alternatives) issus des processus principaux afin de préparer les décisions. Le processus de maîtrise de l'IS concerne tout particulièrement la gestion technique de l'ingénierie système et la maîtrise de l'information tant du système que du projet.

Norme EIA 632 : Cette norme de l'EIA complète les processus techniques de définition du système en couvrant la réalisation des produits jusqu'à leurs mises en service (transfert vers l'utilisation). De plus, elle incorpore les processus contractuels d'acquisition et de fourniture. Processus techniques et processus contractuels sont encadrés par les processus de management (selon leur forme traditionnelle avec les trois sous-processus de planification, évaluation, pilotage) et par les processus d'évaluation des résultats des activités (processus de vérification vérifiant que l'activité a été bien faite et processus de validation vérifiant que le résultat répond au besoin, les deux justifiant de la conformité, ainsi que processus d'analyse système justifiant des choix réalisés tout au long de la définition et donc de l'optimisation du système).

Norme ISO 15288 : Inspirée sur le plan et la forme par la norme ISO/CEI 12207–AFNOR Z 67-150 (Typologie des processus du cycle de vie du logiciel), cette norme de l'ISO, datant de 2003 et revue en 2008, étend les processus techniques à tout le cycle de vie du système (elle couvre ainsi les processus d'exploitation, de maintien en condition opérationnelle et de retrait de service). La norme s'applique à l'ingénierie des systèmes contributeurs qui ont leurs propres cycles de vie (systèmes de fabrication, de déploiement, de soutien logistique, de retrait de service) : on songe par exemple à l'ingénierie des systèmes de démantèlement et de traitement

des déchets d'une installation nucléaire. Elle complète les processus s'appliquant aux projets par des processus dits d'entreprises, qui ont pour objectif de développer le potentiel de l'organisme d'IS en manageant les domaines communs au profit des projets d'IS.

Sans que le choix d'une norme soit primordial pour nos travaux, il est intéressant d'en établir la position dans un processus normalisé de l'IS.
L'IEEE 1220 apparait comme étant trop spécifique aux premières phases du cycle de vie et l'ISO 15288 est au contraire trop étendue en faisant intervenir les phases d'exploitation et de retrait qui ne sont pas abordées dans cette thèse. L'EIA-632 est donc désignée en fournissant un bon compromis entre couverture du cycle de vie et détails des activités.

L'EIA 632 est décrite au travers de 5 groupes de processus en interaction : *Management technique*, *Acquisition et fourniture*, *Conception Système*, *Réalisation du produit* et *Évaluation technique.*

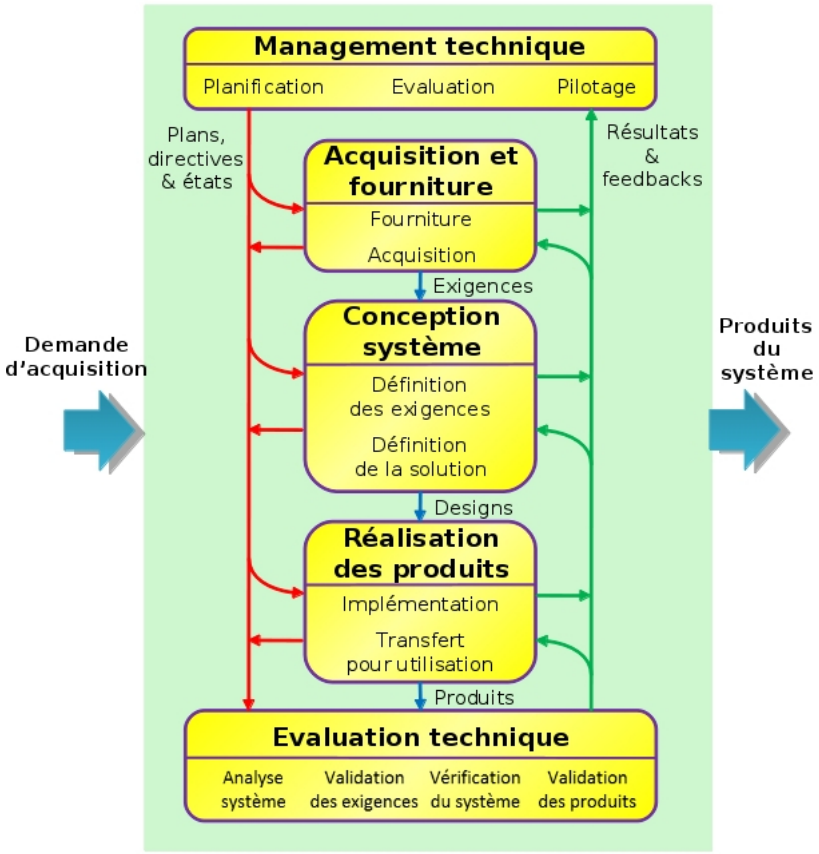


FIGURE 1.6 – Les processus de l'IS (Source : traduction de l'EIA-632)

La figure 1.6 nous offre une représentation des interactions entres les différents processus :

1. Une demande d'acquisition est traitée par le processus de **Fourniture** qui établit un contrat.
2. Les exigences de l'acquéreur (le client) définies dans (1) sont transmises au processus **Conception Système** qui élabore la solution logique puis physique ainsi qu'un ensemble d'exigences techniques associées aux sous-systèmes.
3. Le processus d'**Acquisition** s'occupe alors d'acheter ou de fabriquer (si non disponibles) les sous-systèmes répondants aux exigences spécifiées dans (2).
4. Lorsque les sous-systèmes sont acquis (3), le processus d' **Implémentation** réalise le produit final en suivant le solution de conception choisie en (2). Le processus de **Transfert pour utilisation** le transmet à l'utilisateur après une série de tests et de validations finales.

L'ensemble des interactions entre processus est cadencé et évalué par les processus de **Management technique**. Les processus d'**Évaluation technique** vont, lorsque cela sera nécessaire, analyser le système, valider les exigences et vérifier le système.

Les activités de V&V (Validation et Vérification) s'appliquent à tous les processus et sous-processus intervenant dans les processus techniques d'ingénierie système. Si elles sont techniquement semblables, ces activités traitent deux problématiques dissociées.

La vérification doit montrer qu'une activité a été faite en conformité avec son plan de réalisation et qu'aucune erreur n'a été intégrée dans la définition du système.

La validation doit montrer qu'une activité respecte son objectif et qu'elle répond à la mission pour laquelle elle a été faite. Ces définitions fournies par l'AFIS [AFIS 2014] ne doivent pas être confondues avec la V&V finale d'un système ou d'un sous-système sortant d'une phase de développement utilisant des référentiels différents (cf. 1.4).

1.3.2 L'erreur

Nous nous intéressons ici à quelques exemples de systèmes dont le processus de V&V n'a pas détecté d'erreur et qui pourtant sont allés jusqu'à la panne. Nous discuterons la place de l'erreur dans le processus d'ingénierie des systèmes et les méthodes associées pour détecter leur insertion. Nous pourrons alors extraire l'origine de ces problèmes pour conclure sur l'intérêt des méthodes de V&V et la nécessité d'une méthode de validation des activités de V&V.

1.3.2.1 Exemple d'échecs

Kosmos 419 Nous sommes le 10 mai 1971, après 10 ans et 9 missions infructueuses, tous les espoirs russes pour la conquête de Mars reposent sur la sonde Kosmos 419. Le lanceur Proton K décolle et lance la sonde de plus de 4 tonnes en orbite d'attente. Malheureusement, l'étage supérieur qui devait envoyer Kosmos 419 ne s'allume pas, c'est un nouvel échec. Loin des soucis mécaniques des premiers lancers, c'est une erreur de programmation qui est la cause de cet échec. Le moteur, qui devait envoyer la sonde vers Mars 1,5h après la mise en orbite d'attente, a été programmé pour se déclencher 1,5 années après. Kosmos 419 rentre dans l'atmosphère terrestre et se désintègre deux jours après son lancement.

Vol 501 Le 4 juin 1996 à 9h35, le lanceur Ariane 5 allume ses moteurs pour un vol inaugural. Les systèmes de guidage inertiel (principal et secours) sont mis en route et commencent à mesurer les mouvements de rotation et d'accélération de la fusée. Ce système est principalement composé d'un calculateur, d'un accéléromètre et d'un gyroscope. Il permet de connaître la position, la vitesse et l'inclinaison d'un véhicule. C'est un système classique que l'on retrouve dans les bateaux, les avions, les missiles et les engins spatiaux. Après 37 secondes de vol, les fortes accélérations de la fusée provoquent un dépassement de capacité dans le calculateur du système de guidage inertiel principal, qui se met aussitôt hors service. Le système de guidage de secours (identique à l'autre) subit la même avarie, et s'arrête à la même seconde. Le pilote automatique, qui s'appuie sur les informations provenant du système de guidage inertiel, n'a alors plus aucun moyen de contrôler la fusée. L'échec de la mission est inéluctable. Une enquête est ouverte et montrera que, pour économiser 200 000 $, le Centre national d'études spatiales a demandé à ne pas refaire les simulations de vol puisque le logiciel de navigation était identique à celui d'Ariane 4, connu pour être un lanceur fiable.
C'est la variable d'accélération horizontale du logiciel de navigation qui est la cause de l'autodestruction de la fusée. Certaines variables ne sont pas protégées des valeurs trop élevées puisqu'il est physiquement impossible qu'un débordement se produise. La puissance d'Ariane 5 amène des accélérations 5 fois plus importantes, et ce qui était physiquement impossible avec Ariane 4 le devient avec Ariane 5. La variable codée sur 8 bits n'était pas suffisante pour stocker l'accélération d'Ariane 5. Le calculateur voyait une trajectoire très différente de celle que la fusée suivait. L'ordinateur a alors commandé une violente correction de trajectoire, les tuyères des boosters et du moteur principal ont tourné jusqu'en butée, et la fusée est partie en virage serré. Les boosters ont alors été arrachés et la fusée a commandé son auto-destruction. Des simulations de vol après-coup, utilisant les systèmes de guidage inertiel et l'ordinateur de bord, dans les conditions de vol d'Ariane 5, ont reproduit les événements qui ont conduit à l'explosion de la fusée. La négligence de ces activités de V&V a couté 500 millions de dollars.

Mars Climate Orbiter Le 23 septembre 1999, la sonde Mars Climate Orbiter est mise en orbite. La manœuvre d'insertion en orbite martienne était l'une des phases les plus critiques de la mission. La sonde suit à ce moment là une procédure entièrement automatique et lorsque la manœuvre commence, plus rien ne peut stopper son déroulement. Les télémesures reçues attestent du bon comportement de la sonde. Comme prévu, la sonde passe de l'autre coté de Mars et le signal radio est perdu. Les calculs annoncent un rétablissement de la connexion pour 11h26. A 11h36 la NASA annonce officiellement que le contact avec la sonde est perdu. Si, durant les premières heures, les ingénieurs annoncent plusieurs théories sur la perte de la sonde, ce n'est que quelques heures plus tard que les premières analyses montrent des résultats inattendus. La sonde est passée à peine à 57 km au dessus de la surface de mars, soit 28 km en dessous de la limite critique et à plus de 100 km de la trajectoire prévue. Une rentrée aussi directe dans l'atmosphère engendre un échauffement trop important et mène à la destruction de la sonde. Trois commissions sont chargées de trouver d'où provient cette erreur fatale. Une des commissions se prononce à peine une semaine après la destruction de Mars Climate Orbiter. Il apparaît qu'un problème d'unité dans l'expression d'une force de poussée est la cause du crash. Les ingénieurs de Lockheed Martin Astronautics (Denver dans le Colorado), la firme qui a conçu et fabriqué la sonde martienne, avaient apparemment gardé l'habitude de travailler avec les unités du système anglo-saxons. De leur côté, les ingénieurs du Jet Propulsion Laboratory (Pasadena en Californie) travaillaient depuis des années dans le système métrique. Aucun système de vérification n'a alors été capable de détecter une telle erreur considérée comme trop simple. C'est dans un contexte économique déjà difficile que la NASA doit supporter la perte d'un engin à 126 million de dollars pour une simple erreur de conversion d'unités. Les scientifiques perdent quand à eux le premier satellite de météorologie martien.

Ces échecs, bien que financièrement coûteux, n'ont pas eu de conséquences dramatiques, ce qui n'a malheureusement pas toujours était le cas dans l'histoire de l'IS. L'augmentation de la complexité des systèmes rend la conception de plus en plus difficile et ces erreurs "simples" se retrouvent immergées au sein d'un système complexe, ce qui rend le processus de détection difficile. C'est une problématique qui amène académiques et industriels à réfléchir à de nouveaux processus, méthodes et outils pour améliorer la conception des systèmes. Un travail parallèle s'effectue en améliorant les méthodes de V&V et en perfectionnant les méthodes de développement pour diminuer l'insertion d'erreurs. L'utilisation massive de simulations fidèles et peu couteuses apparaissent.

1.3.2.2 La chaîne de l'erreur

Les faiblesses des méthodes de conception actuelles, qui mènent à la panne, sont en partie connues. Dans [Printz 1998], l'auteur nous rappelle l'origine des erreurs humaines dans les processus informatisés qui sont retrouvés en ingénierie système.

En voici une sélection :

- Erreur de perception, manque de discrimination, distinction fond/forme et/ou sémantique/syntaxe
- Erreur de codage/décodage
- Erreur de représentation et/ou symbologie non adaptée (interopérabilité entre systèmes)
- Représentation, compréhension et interprétation erronées de phénomènes dynamiques et/ou combinatoires
- Impasse, non-exhaustivité, oubli pur et simple
- Acceptation comme vraie d'une hypothèse fausse
- Acceptation comme fausse d'une hypothèse vraie
- Attribution de propriétés inutiles et/ou erronées
- Hypothèse superflue et/ou non appropriée
- Erreur de communication homme - homme, de traduction lors d'un changement de code (contre sens, faux sens, etc.)
- Non respect d'une procédure ou d'une règle
- Non prise de décision en temps voulu
- Action non adaptée au contexte, action contradictoire et/ou antinomique vis à vis d'autres actions
- Itération, répétition inutile d'une action
- Absence d'information qui entraîne une action par défaut non adaptée
- Erreur de raisonnement, raisonnement circulaire
- Défaut ou excès de généralité, abstraction mal construite, définition ambiguë
- Confusion langage / métalangage, concept / méta-concept (mélange de types au sens logique, ou équations de dimensions incohérentes en physique)
- Analogies, associations erronées et/ou non adaptées au contexte
- Changement de tâche fréquent, « papillonnage » , toute perturbation qui augmente la probabilité de confusion et d'interférence

Auxquelles nous rajouterons des problèmes de méthodologie, facilitant l'insertion d'erreurs humaines :

- Multiplication des modèles d'un même système
- Absence d'historique des décisions (problème de traçabilité)
- Méthodes utilisées obsolètes au vu de la complexité des systèmes développés
- Mauvaise traçabilité de l'exigence (modification de l'exigence non répercutée)
- Mauvaise évaluation des délais de développement (pression sur les délais de livraison)
- Extension d'exigences fonctionnelles/ nouveaux besoins
- Mauvaise définition des interfaces
- Manque de retour d'expériences
- Pari technologique utopique
- Augmentation du nombre de sous-traitants
- Division des équipes de développement

— Réutilisation de composants
— Mauvaise gestion de versions (modèle, logiciel)
— Utilisation de patchs pour pallier les erreurs
— ...

Loin d'être exhaustive, cette liste illustre quelques uns des vecteurs d'erreurs possibles en IS. Ces travaux de thèse chercheront à modérer certaines de ces erreurs dans le processus de modélisation de la simulation.

JF. Pradat-Peyre nous rappelle la chaine de l'erreur et l'apparition d'une panne dans Pratique Des Tests Logiciels [Pradat-Peyre 2009]. La figure 1.7 reprend le cycle de vie présenté en figure 1.4 afin d'y associer les propos de l'auteur.

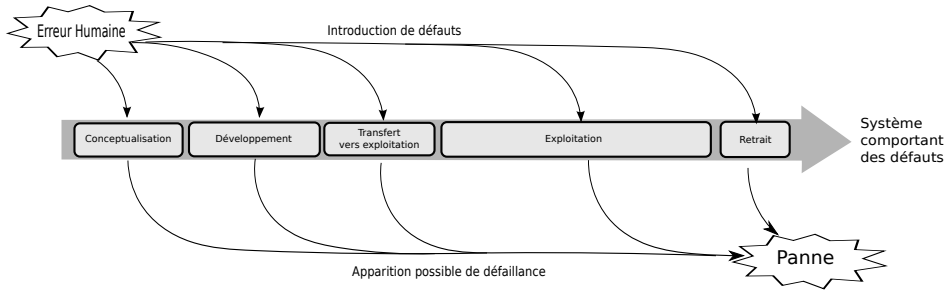


FIGURE 1.7 – La chaine de l'erreur

La figure 1.7 montre que l'apparition d'une erreur humaine [3] va insérer un défaut dans le système. Il est important de noter que tout défaut n'engendre pas une défaillance. En effet, il n'est pas rare qu'un système fonctionne parfaitement malgré la présence d'erreurs en son sein. L'environnement n'amène tout simplement pas le système vers une zone défaillante. Le système pourra alors nous sembler robuste. Cette constatation explique pourquoi la réutilisation de composant doit être effectuée avec une grande prudence, même pour un composant connu pour avoir un fonctionnement parfait. En effet, le nouvel environnement peut mener le composant vers une zone de défaillance comme l'a montré le vol 501 (cf. 1.3.2.1).

JF. Pradat-Peyre insiste sur la différence entre défaillance et panne. Une défaillance se traduit par un résultat inattendu (message non reçu, calculs erronés,etc.) mais le système peut continuer à fonctionner normalement. Ce fonctionnement normal peut être dû au hasard ou à un dispositif de tolérance aux fautes. Une panne survient toujours après une défaillance. La panne a un caractère plus direct et mène le système en mode dégradé, le système perd certaines, voire l'ensemble, de ces fonctionnalités.

3. L'ergonome F. Vanderhaegen [Vanderhaegen 2003] donne un taux de 5 à 10 erreurs par heure d'activité effective.

Au travers des exemples précédents, nous avons montré que les activités de V&V sont indispensables mais faillibles. Il existe effectivement un taux résiduel de défauts dans les systèmes.

Les métiers du logiciel nous fournissent une idée du taux d'erreur résiduel lors d'un développement logiciel par KLS [4].(voir tableau 1.1)

Logiciel	**Défaut résiduel**
Grand public	5 à 10 par KLS
NASA : embarqué	0,1 par KLS (1400 KLS)
NASA : sol	0,5 par KLS (700 KLS)

TABLE 1.1 – Le taux de défaut résiduel par KLS (Source [Printz 2012]))

Les experts du domaine logiciel s'accordent à dire qu'un code source qui compile sans erreur et n'ayant subi aucune activité de V&V possède en moyenne 80 à 100 défauts par KLS. Après avoir passé l'ensemble des processus de V&V, le code possède en moyenne 1 à 2 défauts par KLS. Grâce à un processus de V&V plus développé, les logiciels critiques arrivent à un taux résiduel de 0,1 défaut par KLS.

Il ne semble pas exister de chiffres spécifiques à l'IS, mais les données du domaine logiciel peuvent servir de base pour imaginer le taux de défauts résiduel présent dans un système complexe à la mise en exploitation. Il faut toutefois rester prudent sur l'utilisation de ces données, qui, pour la plupart, se basent sur des études faites au début des années 1990 ([Beizer 1990], [Cusumano 1995],[Boehm 1987]), époque qui peut être considérée comme le Moyen-âge en terme de méthodes logicielles.

Il existe un écart conséquent, en terme de défaut résiduel, entre un logiciel grand public, un logiciel sol et un logiciel embarqué. Cet écart provient de l'effort mis en place pour valider et vérifier le logiciel. Les critères de terminaison peuvent être très variés (nombre d'erreurs détectées, estimation type COCOMO, etc.). Malgré cela, le principe reste le même. Plus le logiciel/système sera critique, plus l'effort sera important (voir figure 1.8). L'obtention d'un taux d'erreur résiduel nul (système parfait) demandera un effort infini.

Le travail de détection d'erreur représente selon les sources [Brooks Jr 1995, Paulish 2001, Printz 2013] 20 à 30 % du coup de développement, et passe à plus de 50% pour les parties critiques . L'activité de V&V représente alors plus de la moitié du temps de développement du système.

4. KLS pour Kilo Ligne Source, ce qui représente le nombre de milliers d'instructions que contient l'application.

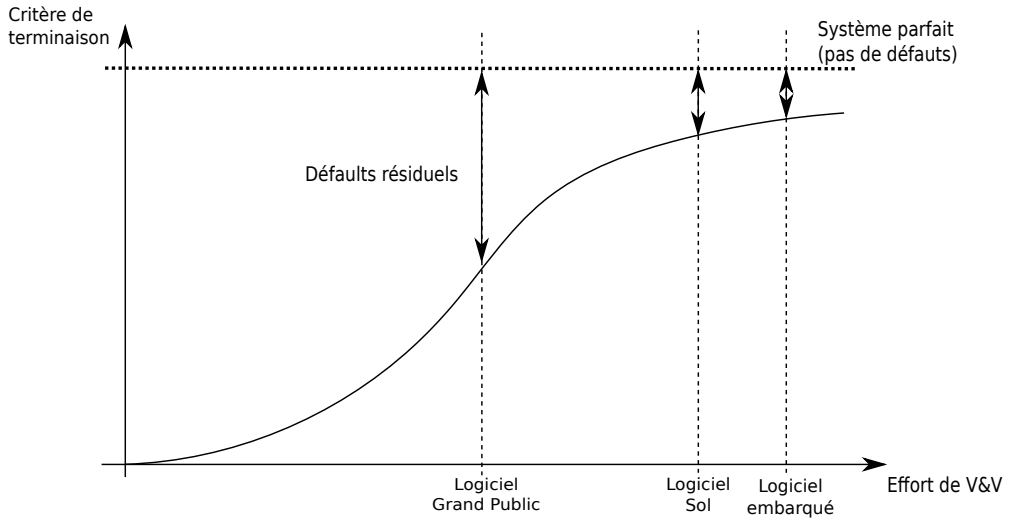


FIGURE 1.8 – Défauts résiduels dans le logiciel

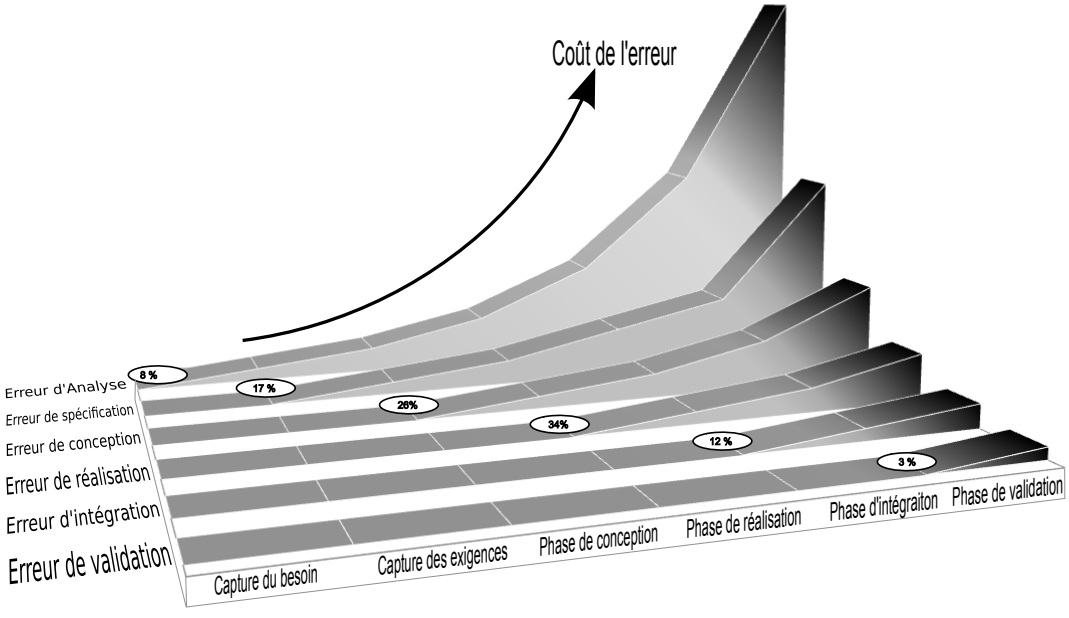


FIGURE 1.9 – Coût de l'erreur en fonction de la phase d'insertion (avec % d'insertion) et de la phase de détection.

En reprenant les phases du cycle en V (cf. 1.3.1.2), la figure 1.9 représente l'évolution du coût d'une erreur lors d'un projet en fonction de la phase d'insertion

de l'erreur et de la phase de détection. Il apparaît également sur cette figure le pourcentage d'insertion d'erreurs en fonction de la phase du cycle de développement [Beizer 1990]. On observe que près de la moitié des erreurs sont insérée avant la phase de réalisation et que le coût d'une erreur non détectée augmente exponentiellement. Il est donc primordial que la détection de ces défauts se fasse aussi tôt que possible durant le cycle de développement du système et donc que les méthodes de V&V s'adaptent à ce bilan. Autre constatation, la phase de validation insère également de l'erreur. B. Beizer annonce dans la même étude un taux d'insertion d'erreurs de 3% durant la phase de validation.

L'amélioration des méthodes de V&V est donc essentielle pour améliorer les systèmes et répondre ainsi à l'ensemble des exigences (coût, qualité, fonctionnalités, délais).

1.4 Validation & Vérification

L'activité de V&V occupe une place essentielle en ingénierie système. Présente tout au long du cycle de développement, c'est une activité coûteuse qui demande une prise en compte particulière. Dans ce paragraphe, nous montrerons que la V&V va au-delà de la validation et de la vérification du système final. Pour améliorer la détection des fautes, les activités de V&V concernent aujourd'hui toute la chaîne de développement ("produit-final", activités, tâches, etc.), l'activité de V&V fait elle-même l'objet d'un processus de validation et de vérification. Nous présenterons ensuite deux stratégies essentielles de la V&V que sont le Test et la M&S.

L'EIA-632 [EIA 2003] donne les définitions suivantes :

Définition 13. ***Vérification*** *: Confirmation par examen et collecte d'évidences objectives que les exigences à partir desquelles un "produit final"*[5] *est construit, codé, ou assemblé, sont satisfaites.*

Définition 14. ***Validation*** *: Confirmation par examination et collecte d'évidences objectives que les exigences à partir desquelles un "produit final" ou l'agrégation de "produits finaux" fonctionne comme attendu par le client dans son environnement opérationnel.*

Les définitions des termes validation et vérification sont régulièrement sujettes à discussion. Ces termes ont une définition différente suivant le groupe concerné ou le domaine d'application. Il est fréquent de voir ces deux termes regroupés sous l'appellation V&V afin d'éviter toute ambiguïté. Les activités pouvant être très proches, elles sont parfois confondues. Nous allons voir ici qu'elles traitent de manière relativement semblable deux problématiques très différentes tout au long du développement du système.

5. Traduction de l'anglais *end-product*, partie du système fournissant des capacités opérationnelles et livrée à un acquéreur.

La figure 1.10 reprend, en version simplifiée, le cycle de développement d'un système (voir 1.3.1.2) et illustre les différentes activités de validation et de vérification. Les questions sous-jacentes à chacune des activités apparaissent sur le schéma de manière non-formelle.

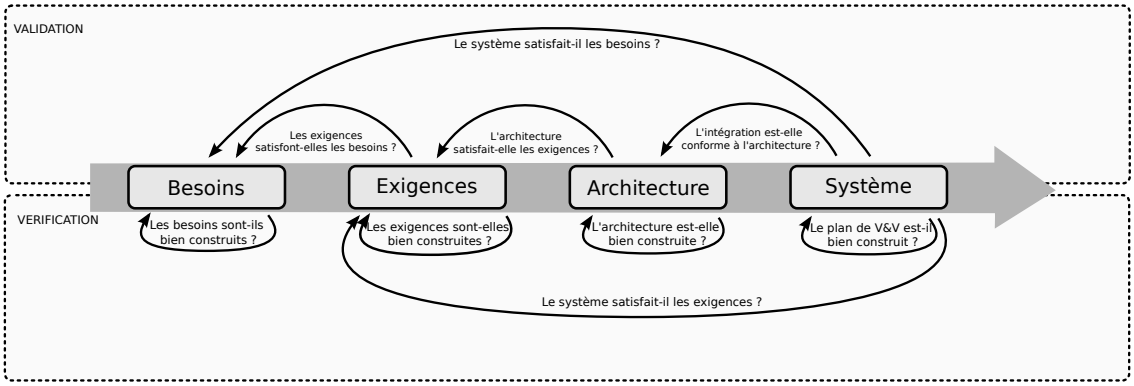


FIGURE 1.10 – Les activités de V&V durant le cycle de développement.

Il est possible d'extraire une version générique des activités de V&V, applicable quelle que soit l'étape du cycle de développement (voir figure 1.11).

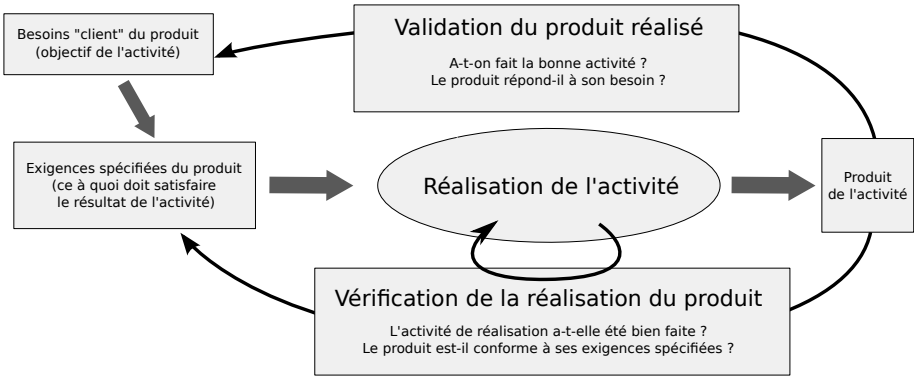


FIGURE 1.11 – Schéma générique des activités de V&V (Source [IVVQ 2014])

Les activités de vérification [6] cherchent à s'assurer que la réalisation de l'activité a été bien faite, que le produit généré correspond à ses exigences de réalisation sans introduire de défauts. La vérification peut avoir lieu tout au long de l'activité pour confirmer que la réalisation se dirige convenablement vers le produit. Les méthodes de vérification sont variées : preuve, test, inspection et simulation.

Les activités de validation [7] cherchent à confirmer que la réalisation de l'activité répond à son objectif en vérifiant la conformité du produit de l'activité avec le besoin

6. B. Boehn [Boehm 1987] l'a résumé à la question : Are we building the product right ?
7. B. Boehn [Boehm 1987] l'a résumé à la question : Are we building the right product ?

du "client". La validation va donner la confirmation que l'on se dirige vers le bon produit. Les méthodes de validation sont similaires aux méthodes de vérification : modélisation, preuve, test, inspection et simulation.

Il est à noter que les opérations de V&V sont planifiées et suivies au même titre que les autres activités. C'est le rôle du management technique de définir une stratégie de V&V, d'optimiser et de justifier le rapport couverture/coût, de planifier l'activité et en assurer la réalisation. Le contrôle qualité s'assurera que ces opérations sont planifiées, exécutées conformément au plan d'exécution et techniquement correctes.

Les activités de validation et de vérification sont nombreuses et variées, il serait difficile et peu utile pour notre étude d'en faire une liste exhaustive. Le tableau 1.2 fournit néanmoins quelques exemples de ces activités tout au long du cycle de développement, et illustre quelques-unes des tâches que doit planifier le management technique. La ligne V&V propose deux exemples d'activités assurées par le contrôle qualité.

Etape	**Validation**	**Vérification**
Exigences	S'assurer que les exigences de besoins répondent correctement au besoin final des utilisateurs.	S'assurer qu'une exigence porte sur le quoi et pas sur le comment.
Conception	S'assurer de l'absence d'incompatibilité de l'architecture logique avec les exigences non directement associables à l'aspect fonctionnel du système.	S'assurer que les interactions inter-fonctionnelles sont dûment identifiées et caractérisées.
V&V	S'assurer que le document de preuve satisfait au besoin exprimé par le document parent auquel il répond, en terme de conformité aux objectifs, de complétude, de faisabilité et d'optimisation.	S'assurer que les dossiers de preuves rassemblent les configurations appliquées, les procédures appliquées, les durées, etc.
Système	Fournir le dossier de preuves de satisfaction du besoin exprimé par le client.	S'assurer que le système est conforme aux spécifications.

TABLE 1.2 – Exemples d'activités de validation et de vérification selon la phase de développement

Le management technique possède plusieurs méthodes à sa disposition et choisira, selon la criticité, une ou un ensemble des méthodes de V&V tel que : le Test, la Preuve, l'Inspection, la Modélisation et la Simulation. Nous présenterons ici le Test et la M&S, qui sont deux activités majeures pour réaliser les activités liées à la V&V.

1.4.1 Le Test

Si le test semble avoir été délaissé pendant quelques années, il est aujourd'hui un des acteurs majeurs de la V&V et trouve un support particulier auprès des domaines du logiciel. Sans chercher à faire un état de l'art du test, ce paragraphe présentera rapidement le test, au travers de définitions, de la classification des tests et s'intéressera à ses limites. Ceci permettra de mieux appréhender la M&S et sa position vis-à-vis du test.

Il n'existe pas de définition unique du test dans la littérature. Voyons ici plusieurs définitions qui vont nous permettre de mieux comprendre les principes associés au test.

Définition 15. *Le **test** est une activité destinée à déterminer si l'évaluation d'une caractéristique ou d'une aptitude d'un programme ou d'un système donne les résultats requis. [Hetzel 1988]*

Définition 16. *Le **test** est un ensemble d'activités ayant pour but d'identifier des défaillances dans un logiciel ou un système et d'en évaluer le niveau de qualité, afin de permettre la satisfaction des utilisateurs. C'est un ensemble de tâches aux finalités bien définies. [Bernard 2008]*

Définition 17. *Le **test** est un ensemble d'un ou plusieurs **cas de tests**.*
[Van Veenendaal 2008]

Définition 18. *Un **cas de test** est un ensemble de valeurs d'entrée, de préconditions d'exécution, de résultats attendus et de postconditions d'exécution, développées pour un objectif ou une condition de test particulier, tel qu'exécuter un chemin particulier d'un programme ou vérifier le respect d'une exigence spécifique. [Van Veenendaal 2008]*

Le test reste étroitement lié au "produit-fini", ceci pour des raisons historiques. Le test était en effet la dernière étape sur le "produit-fini" pour vérifier les aptitudes du système, qui est aujourd'hui appelé *test système*. Le test s'adapte à l'utilisation massive de modèle et propose le model based testing (MBT)[Apfelbaum 1997], permettant la génération de test automatique à partir de la modélisation du système sous test. Sans qu'il existe de classification officielle, ou cohérente dans la littérature, une classification du test émerge ([Bernard 2008] ,[Calimet 2009] ou [Felix 2009]). Nous retiendrons la classification proposée par S. Calimet dans "Les Tests : L'état de l'art" [Calimet 2009].

Les modes d'exécution :

Manuel : Les tests sont exécutés à la main, on exécute un jeux de tests et on compare les résultats fournis avec ceux attendus. Méthode à proscrire dès que le système devient complexe, car lente et fastidieuse, avec un fort taux d'insertion d'erreur.

Automatique : Les tests sont en majorité gérés par un outil. Le testeur doit souvent effectuer un travail complémentaire pendant la phase de conception pour aider l'outil de test.

Les modalités :

Statique : Il n'y a pas d'exécution, c'est une tâche manuelle effectuée par le testeur (relecture de code, avis d'expert, etc.).

Dynamique : Le système est exécuté afin de comparer résultats attendus et fournis. Il a l'avantage d'être plus proche de la réalité et à la portée des programmeurs dans la mise en œuvre.

Les méthodes :

Structurelle : Parfois appelée boîte blanche, c'est une analyse en profondeur. Le testeur s'intéresse au choix de conception, d'architecture, ou de mise en œuvre.

Fonctionnelle : Ce n'est pas le système final qui est sous test mais une spécification (souvent formelle) de la fonction qui est testée. Cette méthode permet de mettre en place des tests très en amont de la phase de conception.

Les niveaux :

Unitaires : Ces tests, parfois appelés tests de composant, s'assurent que chaque composant du système est conforme à sa spécification et est prêt à entrer en phase d'intégration.

D'intégration : Il a pour but de mettre en avant les défauts d'interfaces ou d'interactions entre composants.

Système : C'est un test final parfois appelé "big-bang". Le système complet, matériel et logiciel, est sous test.

De non régression : Il vise à mettre en avant que toute correction d'erreur n'a pas engendré d'autre erreur. Le but est de rejouer l'ensemble des tests déjà exécutés pour les parties "connectées" à la partie traitée.

Les types :

Ils sont nombreux : *test de sécurité, test de sûreté, test d'ergonomie, test de performance, test de robustesse, test fonctionnel, etc.*

Cette classification est complémentaire. C'est l'utilisation conjointe des différents types, niveaux, méthodes, modalités et modes qui permet d'obtenir un bon processus de test. On voit apparaître aux travers de cette classification et des définitions précédentes que l'objectif du test n'est pas de montrer le bon fonctionnement du système mais de mettre en évidence les erreurs. Le test est cependant limité par le phénomène de l'explosion combinatoire. Un test peut être potentiellement infini.

L'addition de deux nombres entiers est un exemple permettant d'illustrer le problème de l'explosion combinatoire. Les deux entiers codés sur la valeur la plus courants sur un ordinateur personnel, soit 32 bits (voir figure 1.12), un test naïf pourrait être de tester chaque combinaison en entrée et d'en vérifier le résultat. En évaluant un couple de valeurs toutes les 10 ns, il faudrait plus de 5000 ans à un ordinateur personnel moyen pour tester l'ensemble des solutions.

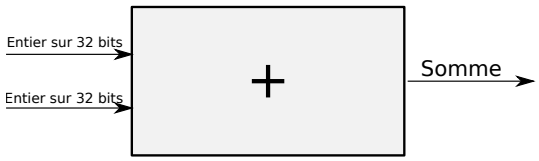


FIGURE 1.12 – Additionneur

Voici un autre exemple bien connu pour son explosion combinatoire. Le problème du voyageur de commerce [Shmoys 1985] (figure 1.13) est un problème en apparence simple. Le voyageur commence par une ville quelconque et termine en retournant à la ville de départ. Les distances entre les villes sont connues. Il faut alors trouver le chemin minimisant le trajet. Même si cet exemple ne traite pas directement une problématique de test, il montre qu'un problème en apparence simple peut mener à une explosion combinatoire.

Nb villes	Nb possibilités	Temps de calcul
5	12	12 μs
10	181440	0,18 ms
25	310 E+21	9,8 milliard d'années

TABLE 1.3 – Nombre de possibilités de chemins et de temps de calcul en fonction du nombre de villes (1 μs par possibilité)

Ici, avec une complexité en $O(n!)$ avec n le nombre de villes à visiter (voir tableau 1.3), il faut presque 10 milliards d'années pour trouver la solution optimale avec 25

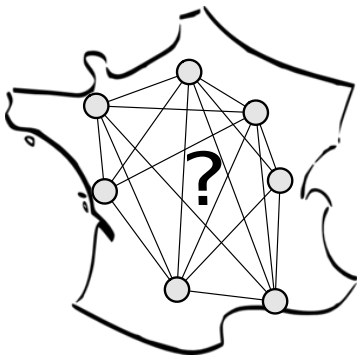


FIGURE 1.13 – Problème du voyageur de commerce

villes sur un ordinateur personnel standard. Les méthodes actuelles permettent de contenir l'explosion combinatoire grâce au calcul à haute performance, avec par exemple le supercalculateur Milkyway-2 de l'Université chinoise de technologie de défense et ses 3 120 000 cœurs de calcul pour 33,86 PFLOPS [8] [Liao 2014]. Même si l'amélioration des calculateurs et des algorithmes de résolution du problème du voyageur de commerce ont permis de trouver en 2001 la solution optimale pour 15112 villes [Applegate 2001], ces "solutions" sont extrêmement coûteuses, réservées à certaines problématiques (phénomènes climatiques, trajectoire missile, développement de médicament, etc.) et en définitive toujours rattrapées par l'explosion combinatoire.

A cause de l'explosion combinatoire, le test exhaustif est généralement impossible. Il est donc essentiel de tester de façon "correcte". L'utilisation de méthodologies, de stratégies, de critères de couverture sont autant de facteurs essentiels. C'est avant tout une activité préventive qui cherche à detecter les fautes aussi tôt que possible dans le cycle de développement.

D'autres méthodes, comme la preuve ou le model-checking, sont basées sur l'exhaustivité mais leur mise en œuvre est plus difficile et limitée par la taille du système. Ces deux méthodes ne s'exécutant pas sur le système réel mais se basant sur la vérification de propriétés sur un modèle, se pose la question de la distance entre modèle et système réel. On voit aujourd'hui apparaître de nouvelles méthodes de tests comme le "test basé modèle", qui amènent à penser que la M&S et le Test (dans leurs définitions historiques) sont en train de fusionner ou tout du moins sont aujourd'hui capables de s'apporter mutuellement des solutions aux problématiques rencontrées.

8. FLOPS est un acronyme signifiant "FLoating point Operations Per Second". Le nombre de FLOPS est une mesure commune de la vitesse d'un système informatique. Le P signifie Péta soit 1 PFLOPS = 10^{15} FLOPS. En 1938 le premier "supercalculateur" avait une puissance de 1 FLOPS.

1.4.2 La M&S

M. Minsky [Minsky 1968] a résumé ce qu'était un modèle de la façon suivante :
"Pour un observateur A, M est un modèle de l'objet O, si M aide A à répondre aux questions qu'il se pose sur O".

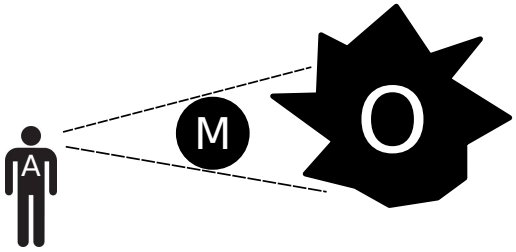


FIGURE 1.14 – Vision abstraite du système

A travers ce résumé M.Minsky donne les clés pour comprendre ce qu'est réellement un modèle, son intérêt et ses limites.

1.4.2.1 S'abstraire de la réalité

Un modèle est donc une représentation abstraite d'un système, qui doit permettre à un observateur donné de répondre aux questions qu'il se pose à propos de ce système. Le modèle doit être capable de donner une réponse identique à celle qu'aurait donnée le système réel. Il n'existe pas une définition unique pour la notion de modèle dans la littérature. B. Combemale nous propose une définition regroupant les travaux de l'OMG [OMG 2003], de J. Bézivin [Bézivin 2001] et de E. Seideiwitz [Seidewitz 2003], que nous retiendrons pour la suite de cette thèse.

Définition 19. *Un **modèle** est une abstraction d'un système, modélisé sous la forme d'un ensemble de faits construits dans une intention particulière. Un modèle doit pouvoir être utilisé pour répondre à des questions sur le système modélisé.*

La modélisation est motivée par le besoin de s'abstraire de la réalité pour apporter une représentation compréhensible et une meilleure communication autour du système et de l'environnement dans lequel le système opère. Si le modèle a été longtemps contemplatif, il peut aujourd'hui être dynamique. Les modèles contemplatifs sont limités, que ce soit par l'expressivité du formalisme utilisé ou par la complexité du modèle. Il est alors possible d'exécuter le modèle à travers le temps pour observer le comportement dynamique du modèle et ainsi conclure sur le système. On parle alors de simulation comme nous le rappelle E. Ramat :

Définition 20. *La **simulation** est la reproduction du comportement dynamique d'un système réel s'appuyant sur un modèle afin d'arriver à des conclusions applicables au monde réel.[Ramat 2006]*

La simulation est forcement liée au concept de simulateur. C'est l'augmentation de la puissance de calcul [9] et la baisse du coût des calculateurs qui font de la simulation une méthode attractive dans un projet. Elle permet d'améliorer la confiance dans la solution proposée avant de lancer des tests physiques parfois onéreux. Elle permet d'évaluer à un coût maitrisé les solutions proposées, de s'adapter à l'avancement du projet et d'intégrer dans le processus de simulation des composants réels si ceux-ci sont disponibles. La simulation est reproductible et peut amener le "système" en condition extrême (pannes, températures max, contraintes mécaniques, etc.) en toute sécurité.

L'engouement pour la modélisation et la simulation par les industriels est notable et ne cesse de croitre. Le constructeur automobile Peugeot a, pour son modèle, la Peugeot 208, utilisé deux fois moins de tests physiques que la 207, en laissant une place plus importante à la simulation [Clapaud 2012].

1.4.2.2 Cycle de développement et simulation

La M&S est présente tout au long du cycle de vie du système :

— Conceptualisation : aide à la comparaison de solutions envisageables, aide à la faisabilité du projet.
— Conception : améliorer la confiance dans la solution proposée avant de passer à la phase de développement, évaluer si les exigences peuvent être respectées, si la solution peut être améliorée.
— Développement : aide au développement du système, aide à la validation du comportement du système à chaque étape du cycle de développement.
— Transfert vers l'exploitation : Entraînement à l'utilisation du système pour les futurs opérateurs du système.
— Exploitation : aide à la réparation, à l'amélioration ou au diagnostic du système après sa mise en exploitation.
— Retrait : aide à la validation du scénario de retrait avec prise en compte des mises à jours ou des pannes subies durant l'exploitation.

On peut voir que la M&S est un atout important tout au long du cycle de vie du système. Son objectif premier est de conforter les équipes sur les solutions élaborées. Elle se décline en quatre phases en fonction de l'avancement du projet, pour s'adapter au mieux aux besoins des équipes de M&S. Le passage d'une plateforme de simulation à l'autre demandera un effort plus ou moins important en fonction du type de systèmes développé et des tests que l'on veut réaliser. La Figure 4 illustre la chronologie d'apparition des plateformes. Il est difficile d'établir un emplacement précis pour chaque plateforme. Suivant les habitudes de l'équipe de développement et le type de systèmes à déployer, ces phases seront amenées à glisser, et ainsi ap-

9. A titre anecdotique : un Ipad 2 sortie en 2010 à une puissance de calcul équivalente au plus puissant supercalculateur de 1985, le Cray-2 avec 1,7 GFLOPS. [Crothaus 2011]

paraître plus ou moins tôt dans le cycle de développement. Nous retiendrons la classification proposé par J. Eickhoff dans [Eickhoff 2009] :

— 1-MIL : "Model in the loop" consiste à récupérer un modèle décrivant le comportement du système pour le valider par simulation. Le modèle du système est couplé à un modèle de l'environnement afin de voir si le modèle satisfait les principales exigences du système.
— 2-SIL : "Software in the loop" consiste à récupérer un programme qui implémente le modèle dans le langage cible. Cette implémentation peut amener des biais dus à des contraintes de mise en œuvre du modèle. Une nouvelle phase de validation et/ou vérification est nécessaire pour s'assurer de l'équivalence sémantique.
— 3-PIL : "Processor in the loop" consiste à valider l'équivalence comportementale du programme après intégration dans le processeur cible. L'environnement reste simulé.
— 4-HIL : "Hardware in the loop" consiste à utiliser le contrôleur physique final. L'environnement reste simulé mais répond maintenant en temps réel.

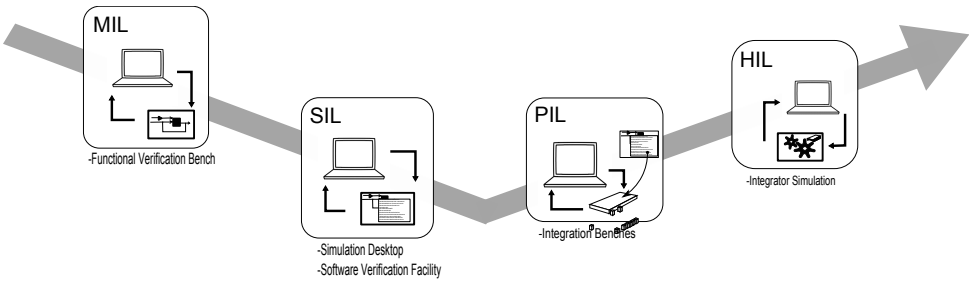


FIGURE 1.15 – Chronologie des plateformes de simulation

La simulation des phases amont du projet (analyse de budget, analyse de faisabilité, etc.) ainsi les simulations après mise en service (support opérationnel,retrait, etc.) n'apparaissent pas dans cette description.

Comme nous l'avons vu, l'utilisation de ces plateformes s'intègre dans le cycle de développement du système en participant au processus de V&V du système. Ces plateformes étant elles-mêmes des systèmes complexes, elles possèdent leur propre cycle de développement et leur propre processus de V&V. La figure 1.16 donne une idée de la complexité possible pour l'élaboration d'une plateforme de simulation. Ici, la plateforme HIL de l'Airbus A350. Cette structure nommée IronBird comprend un ensemble de capteurs et d'actionneurs identiques à ceux du système réel. L'IronBird prend le nom d'Avion zero lorsqu'il est couplé au simulateur.

FIGURE 1.16 – Exemple de plateforme de simulation complexe : HIL, IronBird de l'Airbus A350

1.4.2.3 Motivation de Validation et Vérification de la simulation

L'intérêt pour la simulation, la baisse des coûts des calculateurs et la complexité grandissante des systèmes sont autant de facteurs qui amènent les utilisateurs à demander toujours plus à la simulation, en constituant des modèles avec des objectifs de simulation plus complexes. La figure 1.17 nous montre l'augmentation de cette complexité au travers de l'augmentation du nombres de composants, du nombre de lignes de code et du nombre de connexions nécessaires pour un simulateur avion chez Airbus entre 1989 et 2002.

Pour que les domaines de l'ingénierie et les équipes de développement accordent de l'importance à ces résultats de simulation, il est primordial de valider ces résultats et de définir leur degré de confiance. Il est donc nécessaire que la M&S, au même titre que le système, possède sa propre phase de V&V. Cela signifie assurer la V&V de ce que nous appelons le produit de simulation, soit l'ensemble constitué par : les objectifs de simulation, le modèle du système, le modèle d'environnement, le scénario de simulation, le modèle exécutable et une plateforme d'exécution. Cet ensemble fera l'objet d'une description détaillé dans la suite de cette thèse.

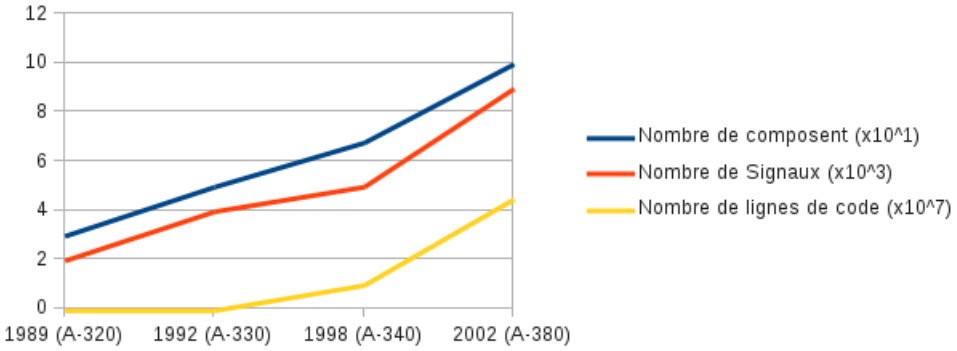


FIGURE 1.17 – Augmentation de la complexité d'un simulateur avion (Airbus 1989-2002) (Source [Albert 2012])

1.5 Résumé

Ce chapitre a posé les bases fondamentales nécessaires à la compréhension de la problématique abordée dans cette thèse. A travers l'historique de l'ingénierie système, nous avons pu voir l'amélioration des méthodes et des processus de développement, allant jusqu'à la normalisation du processus d'ingénierie système, ceci dans le but de rendre possible l'élaboration de systèmes toujours plus complexes. Cette complexité associée à une criticité importante des systèmes a amené l'erreur humaine à devenir fatale. Nous avons vu qu'il était essentiel d'intégrer des méthodes de validation et de vérification performantes permettant d'obtenir un niveau de couverture acceptable. La présentation du Test et de la M&S a permis de faire une introduction générale à ces deux domaines, source de la problématique et support de la solution proposée dans cette thèse. Leurs complexités amènent aujourd'hui à opter pour une approche méthodique et documentée conduisant à établir un processus de V&V dédié divergent légèrement de celui proposé dans le processus d'ingénierie système, en s'éloignant de l'aspect artisanal encore très présent en pratique.

Chapitre 2

V&V pour la M&S

Sommaire

2.1 Introduction

Ce chapitre vient poser la problématique et définir les objectifs de contribution dans un cadre théorique, celui de la M&S. Nous nous intéresserons dans un premier temps à établir de manière détaillée ce qu'est un modèle de simulation, quels sont les constituants du monde de la simulation et comment développer un produit de M&S satisfaisant. Après avoir présenté les différentes entités de la M&S , nous nous focaliserons sur la problématique de compatibilité entre les éléments de la M&S. C'est ici que nous présenterons le rôle d'un des éléments fondamentaux de l'approche proposée, le cadre expérimental. Ce chapitre nous amènera à discuter de la crédibilité des résultats obtenus lors d'une simulation et de la réelle nécessité d'une compatibilité dynamique totale entre cadre expérimental et modèle.

2.2 État de l'art sur la V&V appliquée aux modèles de simulation

Afin de s'intéresser plus en détail à la V&V des modèles de simulation et aux solutions existantes, il est important de connaître la méthodologie d'élaboration

d'une M&S, sa composition, les acteurs associés et leurs attentes.

C'est le principe même de modélisation qui est la cause de notre problématique. Comme nous l'avons vu dans le chapitre précédent (cf. 1.4.2.1), un modèle est une abstraction plus ou moins simplifiée de la réalité. V. Albert [Albert 2009] nous propose une analogie intéressante entre un *modèle* tel qu'entendu par la communauté de la M&S et la notion de *théorie* telle que définie dans la méthode scientifique [Godfrey-Smith 2009]. Une théorie cherche à expliquer comment un ensemble de faits réels interagissent entre eux. Une théorie est réfutable grâce à un ensemble d'observations empiriques ou au contraire il est possible d'améliorer sa crédibilité au travers d'expérimentations. Elle doit avoir la capacité de prédire les observations des faits réels pour lesquels elle a été élaborée. De la même façon, le concepteur d'un modèle (modélisateur) va formuler un ensemble d'hypothèses afin de définir sa vision du système réel. Le modèle ainsi créé représente alors une abstraction du système réel tel que le modélisateur le perçoit. Afin de tester ses hypothèses, le modélisateur va établir un ensemble d'expérimentations. La simulation est l'expérimentation du modèle permettant de valider ou de réfuter les théories formulées lors de la modélisation.

La notion d'abstraction est essentielle à notre problématique. Nous avons montré qu'il existe une relation d'abstraction entre monde réel et modèle. La relation d'abstraction est également réalisable entre modèles. Il est en effet possible qu'un ensemble d'hypothèses associées à un modèle ne soit pas nécessaire pour permettre à la simulation de répondre de façon équivalente aux questions posées. Un modèle trop complexe va amener un temps de développement et un effort d'expérimentation inutilement dépensés pour répondre à la question. Un modèle trop simple ne permettra pas de répondre à la question.

2.2.1 Modèle conceptuel

La communauté de la M&S différencie ce qui est appelé modèle conceptuel du modèle de la simulation. R.G. Sargent [Sargent 2005] nous rappelle cette différence au travers de la figure 2.1. Il y expose de manière plus générale les relations entre monde réel et monde de la simulation tout en donnant les liens de V&V entre les différentes entités.

Pour R.G. Sargent, le monde réel est constitué de deux entités : le système et les données du système. C'est au travers de l'expérimentation que le système réel devient une source de données, et c'est à partir de ces données qu'est émis un ensemble d'hypothèses permettant la mise en place d'une théorie. Les théories sont à la frontière entre monde réel et monde simulé. C'est à partir de ces théories et des objectifs d'expérimentation que l'on crée le modèle conceptuel. C'est à partir de ce modèle conceptuel qu'est construit, à travers une suite de spécifications et d'implémentations, le modèle de simulation. On retrouve des étapes de validation et de vérification dans l'ensemble du monde simulé. Dans le chapitre précédent (1.4), nous avons vu l'intérêt de la V&V et les définitions qui y étaient associées lors du

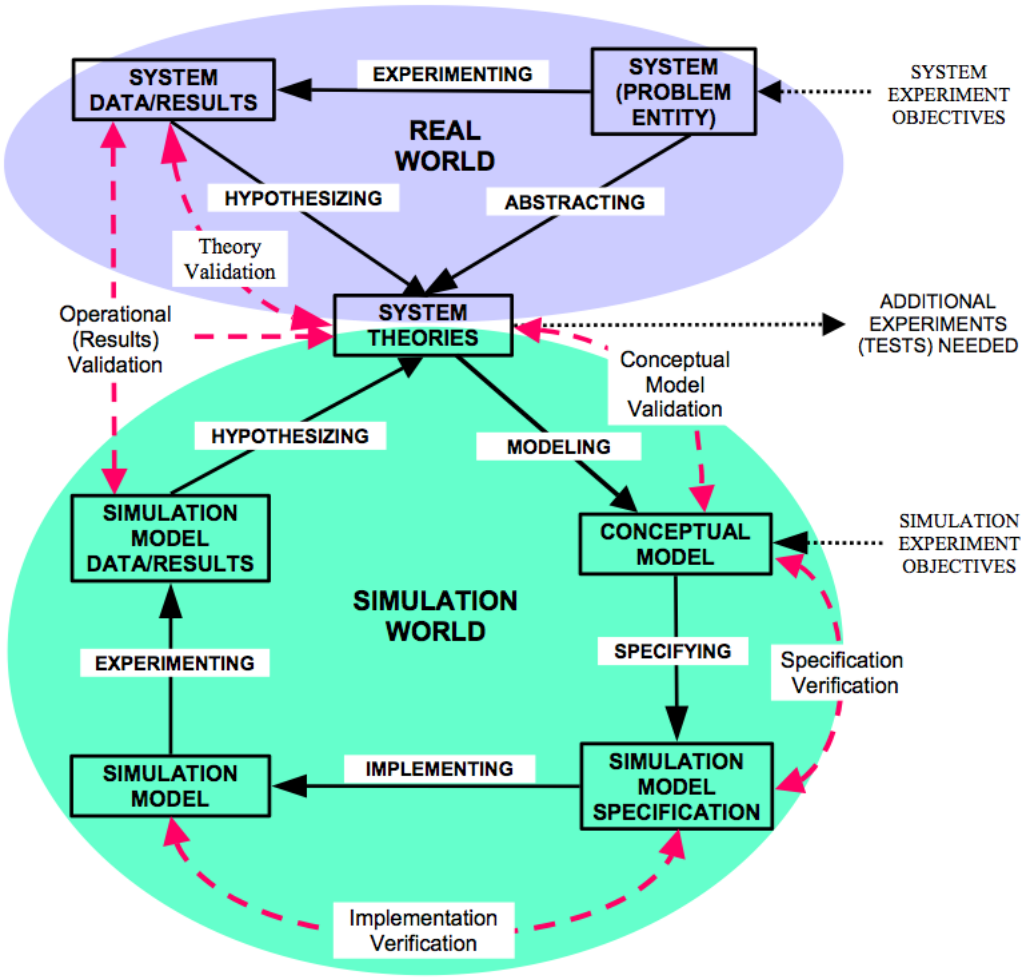


FIGURE 2.1 – Modèle de R.G. Sargent des interactions entre monde réel et monde de la simulation avec les liens de V&V

cycle de développement d'un produit . Les besoins spécifiques de la communauté de la M&S ont amené à une redéfinition du terme validation et à l'ajout du terme accréditation [1] spécifique à la M&S pour former la VV&A.

L'auteur fait apparaître trois étapes de validation :

— **la validation du modèle conceptuel** qui détermine si les théories et les hypothèses du modèle conceptuel sont conformes à celles exprimées dans les théories du système. Elle assure également que le modèle conceptuel est "raisonnable" vis-à-vis du but expérimental.

1. L'accréditation est une activité permettant de certifier officiellement que le modèle ou la simulation et les données associées sont acceptables pour une utilisation précise [DoD 1994].

— **la validation opérationnelle** qui détermine si le comportement de la sortie du modèle a une précision suffisante par rapport à l'application prévue du modèle.

— **la validation des théories** qui consiste (si les données sur le système réel sont disponibles) à comparer les données d'expérimentation sur le système avec les résultats fournis par la simulation.

Au travers de cette illustration, R.G. Sargent nous montre que la définition de validation au sens entendu par la communauté M&S diffère de celle vue au chapitre 1 dans la communauté de l'ingénierie système. Ici, la validation cherche à s'assurer que le produit de simulation est capable d'atteindre les objectifs d'expérimentation prédéfinis. La validation va également chercher à savoir si les résultats fournis par la simulation sont suffisamment proches du système réel pour l'objectif d'utilisation fixé. La validation comporte alors deux aspects, l'approche classique de satisfaction du client et une nouvelle notion de fidélité de la simulation.

Si cette thèse porte le titre de *Validation des modèles de simulation*, il ne faut y voir qu'un abus de langage destiné à une compréhension rapide par le plus grand nombre. Le titre pourrait être "*Validation des modèles conceptuels de simulation*", puisque ces travaux s'intéressent essentiellement à la validation du modèle conceptuel.

La communauté s'accorde sur le rôle du modèle conceptuel dans la M&S ([Sargent 2005], [Pace 1999], [DMSO 2000]). Le modèle conceptuel doit être le premier élément de conception pour une communication claire et complète entre les développeurs de la simulation et les utilisateurs de la simulation. D.K. Pace propose dans [Pace 1999] un modèle conceptuel constitué du contexte de simulation, des éléments de simulation et des concepts de simulation.
Le *contexte de simulation* doit définir le problème et son domaine. Il permet au développeur de la simulation de construire la simulation correctement. Les *éléments de simulation* reprennent un ensemble d'informations (hypothèses, contraintes, algorithmes, interactions, etc.), permettant de décrire chaque entité de la simulation. Les *concepts de simulation* expliquent comment les exigences de M&S décrites dans le contexte de simulation sont transformées en spécifications détaillées permettant d'atteindre les objectifs d'utilisation.

2.2.2 Exigences de la M&S

Les exigences de la M&S sont essentielles pour établir le modèle conceptuel. Elles décrivent les besoins de l'utilisateur de la simulation en termes de propriétés et de comportements du système [2]. Les exigences s'intéressent également au niveau de fidélité, de résolution, d'exactitude ainsi qu'au niveau de confiance attendus pour ces exigences.

2. Le terme système est utilisé ici au sens décrit par R.G. Sargent (c-à-d l'entité du problème).

Une fois les exigences de la M&S établies, elles sont analysées pour définir les critères d'acceptabilité. L'évaluation de ces critères est délicate et la plupart du temps fait appel à un expert. L'évaluation du critère d'acceptabilité va définir l'effort nécessaire pour développer un produit de simulation acceptable. Si le critère est sous-évalué, le risque d'utilisation de la simulation augmente. Si le critère est sur-évalué, l'effort de développement augmente de façon inutile.

2.3 Théorie de la M&S

La modélisation et la simulation sont depuis plusieurs dizaines d'années très utilisées dans le milieu industriel, comme dans celui de la recherche. Il a donc été indispensable de définir une base de réflexion commune à l'ensemble des acteurs participant à l'élaboration d'un produit de simulation.

2.3.1 Modélisation des systèmes dynamiques

Nous avons jusque-là présenté la modélisation et la simulation comme une solution au processus de V&V pour l'ingénierie système et comme un processus nécessitant lui-même de faire l'objet d'un processus de V&V. Il convient de présenter à présent les concepts manipulés pour modéliser et simuler un système dit "classique" (causal et déterministe), évoluant au cours du temps, appelé *système dynamique*.

La théorie systémique développée à la fin des années 50 ([Von Bertalanffy 1956]) a apporté des solutions à l'élaboration et l'étude des systèmes. De façon élémentaire, un système peut être vu comme :

— *Une boîte noire*, seul le comportement externe du système est observable.
— *Une boîte blanche*, la structure interne du système est observable, on accède à son état interne et sa dynamique.

Au plus haut niveau d'abstraction, le système n'est connu qu'au travers de son comportement externe (E/S), on occulte (volontairement ou non) le comportement interne du système. La prédiction des sorties en fonction des valeurs d'entrées n'est pas possible. Au contraire, le système boîte blanche permet l'accès à la structure interne qui peut être présentée selon trois éléments : l'*état interne* regroupant un ensemble de variables appelées *variables d'état* ; une *fonction de transition d'état* qui définie le mécanisme de changement d'état, c'est la description des changements de valeurs des *variables internes* ; une *fonction de sortie* permettant de générer les sorties en fonction de l'état interne.

B.P. Zeigler [Zeigler 2000] nous propose une classification des systèmes selon trois types :

— Les *systèmes continus* : Les *variables d'état* peuvent prendre une infinité de valeur dans un intervalle de temps fini, on dit que l'espace d'état est continu. La base de temps est continue.

— Les *systèmes à temps discret* : La base de temps est discrète ; les variables d'état peuvent prendre des valeurs continues ou discrètes.
— Les *systèmes à évènements discrets* : La base de temps est continue ; les variables d'état peuvent prendre des valeurs continues ou discrètes. Il se différencie du système continu par le calcul de l'état qui n'a lieu que si un événement se produit.

Pour mieux illustrer cette classification observons le tableau 2.1 qui nous montre les différents types d'évolution d'un système pour une variable Y.

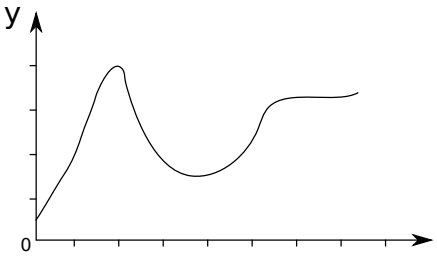
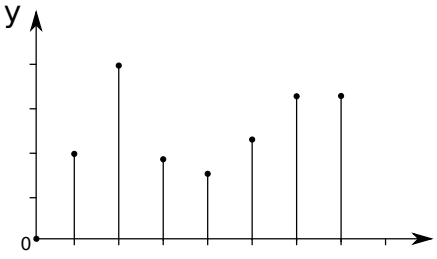
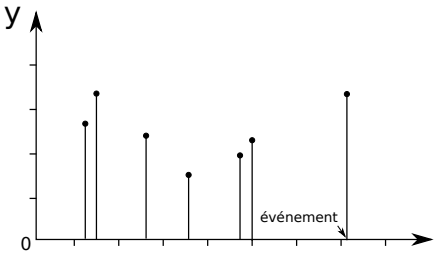
Type de systèmes	Evolution des variables	Remarques
Systèmes continus		temps et espace continus, généralement modélisés par des équations différentielles
Systèmes à temps discret		axe du temps discrétisé de façon constante, les variables peuvent prendre des valeurs continues ou discrètes
Systèmes à événements discrets		axe de temps continu, mais la variable d'état change à un moment particulier (l'apparition d'un événement) dans un espace d'état continu ou discret.

TABLE 2.1 – Evolution des variables suivant le type de système

Cette étude s'intéresse exclusivement aux systèmes à temps discret et aux systèmes à évènements discrets. Par la suite nous nous attacherons à décrire ces systèmes au travers des formalismes que nous utiliserons pour élaborer les produit de M&S.

2.3.2 Entités de la M&S

La M&S devenant une discipline à part entière, il fallait la théoriser. La contribution majeure en faveur de la théorisation de la M&S a été sans aucun doute celle

de B.P. Zeigler en 1976, avec son ouvrage intitulé "*Théorie de la modélisation et de la simulation*" [Zeigler 1976], révisé en 2000 [Zeigler 2000]. Cet ouvrage nous donne les fondamentaux de la conception d'un produit de simulation nous permettant de positionner clairement notre problématique et les solutions envisagées. De plus, ce cadre théorique va nous permettre d'aborder notre problématique en utilisant une terminologie précise en adéquation avec le reste de la communauté de la M&S.

Une des contributions majeures décrites par B.P. Zeigler est l'établissement du "framework" de la M&S avec ces différentes entités (voir figure 2.2). Cet aspect méthodologique fondamental va servir de base à la suite de notre étude, il est donc essentiel de présenter en détail ses différents constituants et de bien comprendre leurs rôles.

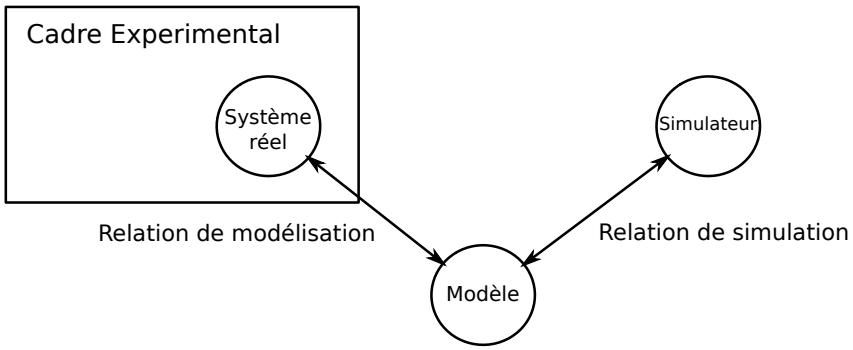


FIGURE 2.2 – Processus de M&S (B.P. Zeigler)

Le système source

Le système source est l'élément réel ou virtuel que l'on souhaite modéliser. Il est vu par l'auteur comme la source de données observables qui permet d'établir "la base de donnée comportementale".

Le cadre expérimental

Le cadre expérimental est une spécification des conditions d'observation du système et des objectifs de simulation. Il peut être vu comme un système qui interagît avec le système d'intérêt permettant d'obtenir les données d'intérêt sous des conditions données. Suivant le point de vue ou le niveau de détail désiré, il est possible que plusieurs cadres expérimentaux soient développés pour un même système (modèle ou système source). Un même cadre expérimental peut quant à lui être appliqué à différents systèmes.

L'auteur propose un cadre expérimental divisé en trois sous-composants : le générateur qui injecte un ensemble de vecteurs d'entrée au système, l'accepteur qui surveille si les conditions expérimentales souhaitées sont respectées et le transducteur qui observe et analyse les vecteurs de sortie du système.

Comme nous l'avons vu, un modèle est toujours valide au regard d'un objectif donné. Le cadre expérimental peut être vu comme une formulation opérationnelle des objectifs, il est donc par nature l'élément essentiel qui permet d'établir si oui ou non un modèle est valide.

Le modèle

Le modèle est pour l'auteur une représentation du système détaillé, correspondant à un ensemble d'instructions, lois, équations et contraintes, qui dicte son comportement aux E/S, ainsi que la manière dont les différents composants sont connectés.

Le simulateur

Le simulateur désigne ici l'entité capable d'exécuter les spécifications décrites au travers du modèle et du cadre expérimental. Dans l'approche proposée par B.P. Zeigler, un des principes fondamentaux du processus de M&S repose sur la séparation entre modèle et simulateur. Cette séparation amène plusieurs avantages. Le modèle est indépendant du simulateur, il peut donc s'exécuter sur différents simulateurs. Le simulateur peut faire l'objet d'un processus de V&V indépendant. La complexité du modèle peut se mesurer au travers des ressources nécessaires au simulateur pour permettre l'exécution correcte du modèle.

La relation de modélisation

Le lien entre modèle, système source et cadre expérimental est appelé "relation de modélisation". Cette relation s'intéresse à la correspondance entre le modèle et le système source dans le contexte du cadre expérimental pour établir la validité de l'expérience.

La relation de simulation

Le lien entre modèle et simulateur est appelé "relation de simulation". C'est cette relation qui s'intéresse à la validité du simulateur par rapport au modèle. On s'assure que le simulateur n'introduit pas de biais dans le processus de simulation, soit en d'autres termes, que le simulateur est capable d'exécuter le modèle tel qu'il est décrit. Cette relation a fait l'objet de réflexions "annexes" au cours de cette thèse. Ces réflexions seront discutées dans le chapitre 4

2.4 Compatibilité entre cadre expérimental et modèle

Cette section s'intéresse à une relation fondamentale de la M&S : la relation de compatibilité. B.P. Zeigler nous la présente dans [Zeigler 1976] comme deux relations distinctes. Il nomme "applicabilité" la relation qui détermine si un cadre expérimental est applicable au modèle. A l'inverse il nomme "accommodation" la relation qui

détermine si un modèle s'adapte à un cadre expérimental. Cette différence s'argumente par l'utilisation de "banques" de modèles et de cadres expérimentaux. Leurs réutilisations posent la question de compatibilité de façon unidirectionnelle. De manière générale, nous regroupons ces relations sous la problématique de compatibilité.

B.P. Zeigler introduit les concepts de dérivabilité et de morphisme (ou homomorphisme). La relation de dérivabilité est définie entre les cadres expérimentaux. Cette relation établit qu'un cadre expérimental est plus restrictif qu'un autre (observe moins de sorties, génère plus d'entrées). La relation de morphisme, quant à elle, s'intéresse à la relation entre un modèle "concret" et un modèle "abstrait". L'auteur désignant par "concret" un modèle davantage capable, c'est à dire pouvant être utilisé par un grand nombre de cadres expérimentaux. L'auteur désigne par "abstrait" un modèle se substituant à un modèle "concret". Il est essentiel de noter qu'un modèle "abstrait" doit être aussi capable qu'un modèle "concret" pour un cadre expérimental donné.

Les relations d'applicabilité et d'accommodation sont des relations essentielles qui permettent d'établir si oui ou non l'application de M&S est capable d'atteindre l'objectif d'utilisation. Afin de mieux comprendre ces notions, la figure 2.3 illustre les relations de morphisme, de dérivabilité, d'accommodation et d'applicabilité.

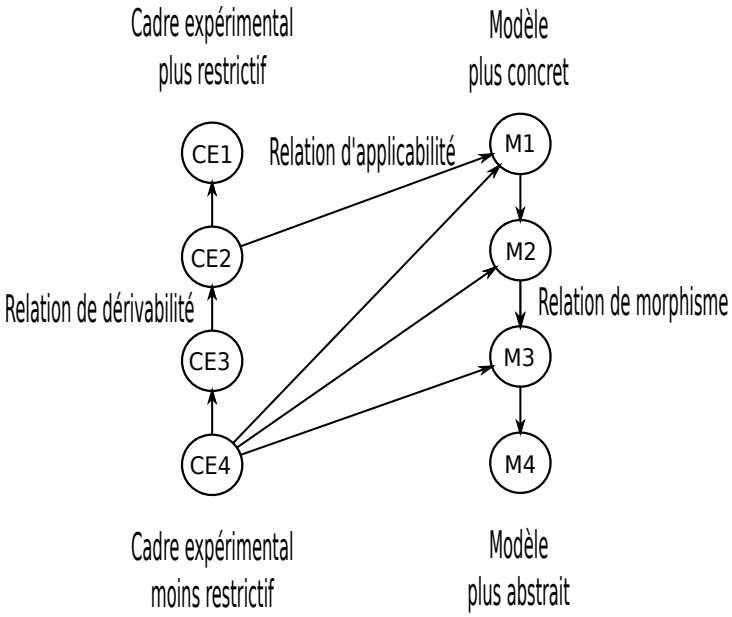


FIGURE 2.3 – Les relations modèle/cadre expériemental [Zeigler 1976].

On peut voir sur cette figure que le cadre expérimental CE4 est le moins restrictif et qu'il est applicable au modèle M3,M2 et M1. CE2 qui est un cadre expérimental dérivé de CE4 par l'intermédiaire de CE3 est applicable à M1, donc par construction CE3 [3] et CE4 sont applicables à M1. M4 est trop abstrait pour le cadre expérimental

3. Les relations d'applicabilité de CE3 ne sont pas représentées afin d'alléger le graphe, de même

proposé, il ne peut s'accommoder aux cadres expérimentaux proposés. CE1 est quant à lui trop restrictif pour les modèles proposés.

2.4.1 Dérivabilité et morphisme : une histoire d'abstraction

S'il suffit d'un couple "modèle/cadre expérimental" pour poser la problématique de compatibilité, les mécanismes de morphisme et de dérivabilité apportent à cette question une autre dimension. Comme nous l'avons déjà vu, un modèle abstrait est aussi valide qu'un modèle concret pour un objectif donné, il doit donc être également applicable. Les mécanismes d'abstraction sont nombreux et variés. V. Albert nous propose dans [Albert 2009] une taxonomie d'abstraction dédiée à la M&S permettant d'établir des règles de compatibilité entre modèle et cadre expérimental, c'est donc tout naturellement que nous adopterons la taxonomie qu'il propose.

Après avoir étudié les différentes taxonomies proposées par la communauté de la M&S ([Frantz 1995], [Zeigler 2000]) mais également celles proposés par la communauté de l'intelligence artificielle ([Weld 1992], [Falkenhainer 1991], [Iwasaki 1993], etc) , l'auteur suggère une taxonomie adaptée au domaine de la M&S et plus particulièrement à notre contexte (les systèmes embarqués et les systèmes hybrides).

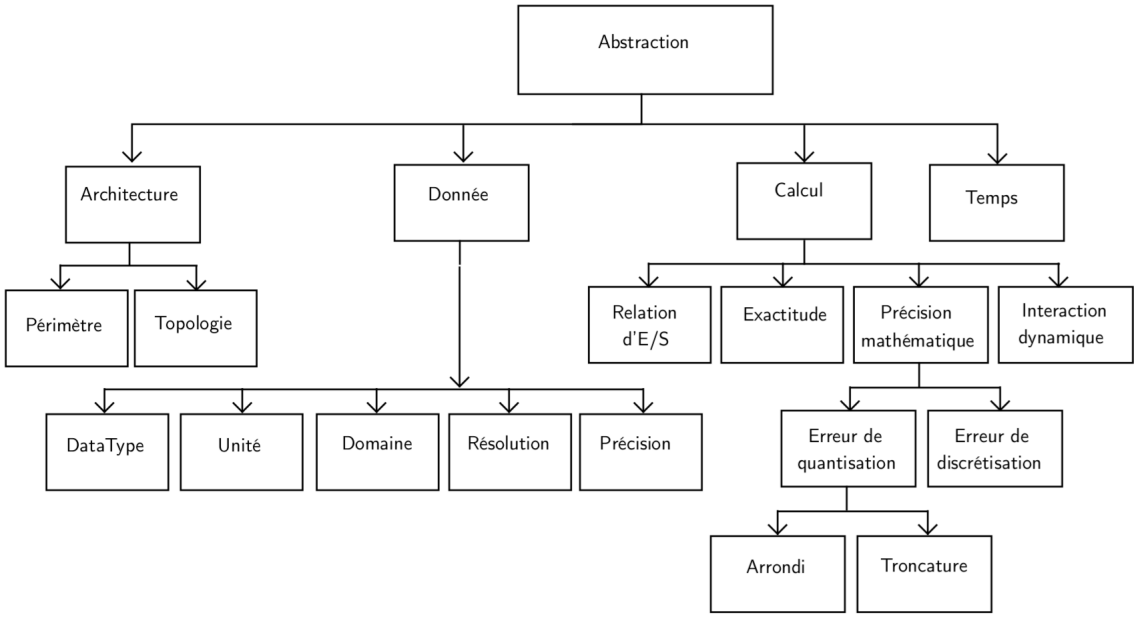


FIGURE 2.4 – Taxonomie d'abstraction de modèle [Albert 2009].

Comme l'illustre la figure 2.4, l'auteur propose une taxonomie d'abstraction selon quatre axes : l'architecture, les données, les calculs et le temps.

que les relations d'accommodations

Afin d'illustrer ces différentes abstractions nous considèrerons le modèle "position du soleil" (voir figure 2.5). Ce modèle fournit l'emplacement du soleil à une date donnée (azimut, élévation) en fonction des coordonnées d'observation (longitude, latitude) : $Latitude \times Longitude \times Date \rightarrow Azimut \times Elevation$. Nous verrons alors comment l'abstraction peut influencer ce modèle en fonction du type d'abstraction choisi, les abstractions pouvant bien évidement se combiner.

FIGURE 2.5 – Le modèle "position du soleil"

Abstraction sur l'architecture

Il est ici question d'abstraction associée au périmètre et à la topologie. Pour le modèle "position du soleil", les paramètres d'entrées "longitude", "latitude", "date" et les paramètres de sortie "azimut" et "élévation" sont des paramètres dit *exogènes*. Ils sont externes au système et sont donc commandables et/ou observables par l'utilisateur. C'est l'environnement du système. Imaginons maintenant un modèle qui étudie la production électrique d'un panneau solaire en fonction de sa position (Inclinaison et Orientation) et de sa surface ($Surface \times Inclinaison \times Orientation \times Date \rightarrow Production$). La position du soleil bien que nécessaire n'a plus besoin d'être observable et devient un paramètre *endogène* (interne au système) de même que les informations concernant la latitude et la longitude puisque le panneau possède une position fixe. Le modèle "position du soleil" est agrégé pour former le modèle "Production d'énergie Panneaux solaire" comme l'illustre la figure 2.6.

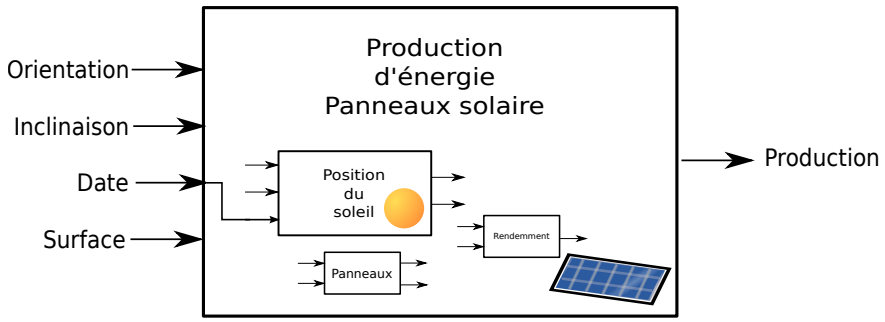


FIGURE 2.6 – L'agrégation de fonctions : paramètres endogènes, paramètres exogènes.

Abstraction sur les données

Les abstractions associées aux données sont nombreuses : unité, type, domaine, type abstrait (information véhiculée par la donnée), résolution, précision. La coercition de type (changement de type pour une variable) est fréquemment utilisée. Par exemple la position du soleil peut être donnée en degré, en radian ou en encore en grade. A un niveau plus concret, on pourra choisir le type de la variable codant la position du soleil (entier, flottant, string, etc.). Il est également possible de donner une information qualitative sur la position du soleil. On identifie alors les variables de position à des zones, on dit que l'on partitionne le domaine de la variable en espace de quantités. Comme le montre la figure 2.7, il est possible de définir la position du soleil suivant un espace de quantités, par exemple : "zone de lever", "zone de culmination supérieure", "zone de coucher", "zone de culmination inférieure". Il est même possible d'imaginer un niveau plus abstrait en agrégeant les états pour obtenir deux états : "jour" et "nuit".

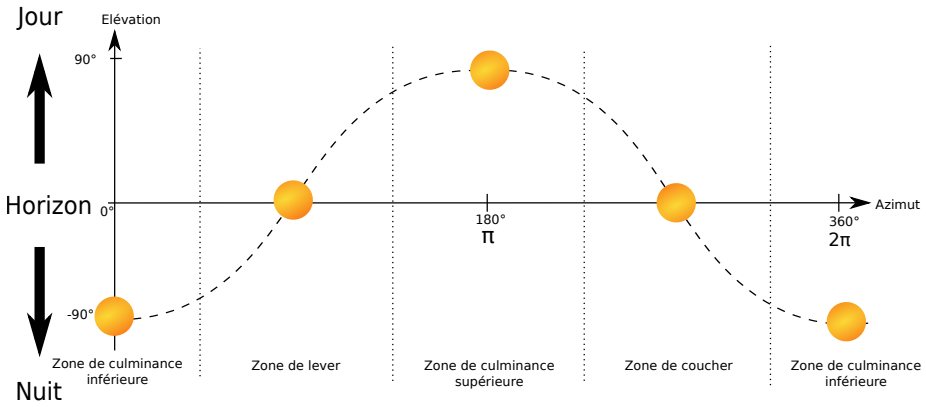


FIGURE 2.7 – Espace de quantités possibles permettant de définir la position du soleil

Suivant l'objectif de simulation, il n'est pas forcement nécessaire de connaître avec précision la position de l'astre, une simple information sur sa présence peut se révéler suffisante. On imagine aisément l'avantage de telles abstractions sur le modèle.

Les techniques d'abstractions ont amené V. Albert à définir les informations nécessaires permettant d'indiquer la qualité d'une donnée afin d'établir le niveau de compatibilité entre cadre expérimental et modèle :

— Une unité : Pour notre modèle "position du soleil", il est possible de donner l'azimut en degrés.
— Un domaine : C'est le domaine de valeur que peut prendre la variable. Par exemple l'azimut un 31 décembre à Libreville (sur l'équateur) nécessite un

domaine [100° ; 250°], la même étude à Toulouse nécessite un domaine plus important [0° ; 360°].

— Une résolution : C'est la distance entre deux valeurs successives que peut fournir la variable.

— Une précision : C'est la distance entre la valeur fournie et la valeur "réelle". Imaginons que notre modèle ait une précision de 10°. Avec une résolution de 1° [4], le modèle nous fournit un azimut à 183°, cela signifie que la position calculée est entre 178° et 188° (sans distinction possible). Le modèle devra être validé en prenant en compte la précision.

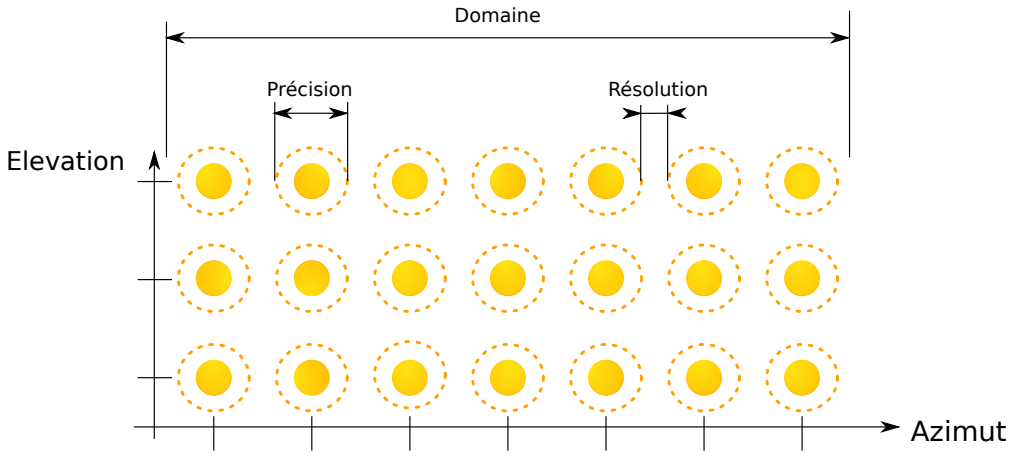


FIGURE 2.8 – Propriétés sur les données

Abstraction sur les calculs

V. Albert s'intéresse ici à l'exactitude du calcul. Il est possible de simplifier les calculs pour une étude donnée. Nous pouvons utiliser le modèle "position du soleil" n'importe quel jour de l'année en fournissant la date afin de pallier l'obliquité [5] terrestre. Pour une étude à court terme (quelques jours) de la position du soleil, l'obliquité (O) de la terre peut être considérée comme un phénomène n'ayant pas d'influence significative pour l'objectif de simulation. La variable *date* peut alors être une constante endogène au système. Il est également possible d'améliorer l'exactitude des calculs en ajoutant certains phénomènes comme la précession (P), la nutation (N) ou la variation d'obliquité de la terre qui sont des phénomènes "lents" [6].

4. La résolution est ici plus petite que la précision, cela peut sembler déroutant, c'est simplement que la sensibilité du modèle est de 1° alors que sa "précision de calcul" est de 10°. Cela se traduirait sur la figure 2.8 par des points rapprochés avec des cercles en pointillés englobant un ensemble de points autour.

5. angle formé entre l'axe de rotation de la terre et la perpendiculaire à son plan orbital

6. La précession a un cycle de 25 760 ans, la nutation un cycle de 18.6 ans et l'obliquité varie entre 22.1° et 24,5° tous les 41 000 ans .

Il existe d'autres phénomènes d'abstraction associés au calcul, comme l'erreur d'arrondi et la troncature que nous n'illustrerons pas ici.

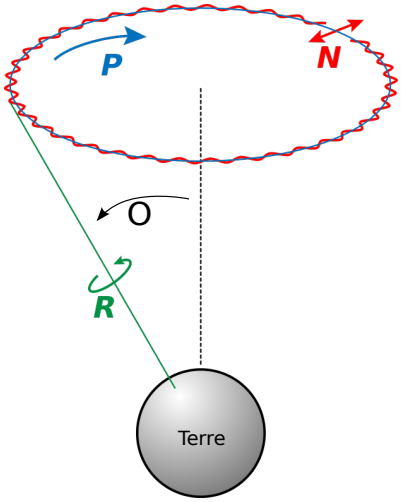


FIGURE 2.9 – Quelques phénomènes jouant sur l'exactitude de la position relative du soleil pour un observateur terrestre. (P)récessions, (N)utation, (O)bliquité, (R)otation.

Abstraction sur le temps

Avant toute chose il est important de rappeler qu'en simulation, le temps peut être attaché à différents concepts. Il existe le temps que nous connaissons tous, appelé temps de "l'horloge murale". C'est le temps du monde réel, il n'est pas possible d'agir dessus. Le temps du simulateur est le temps qui est représenté ; chaque unité de temps du simulateur est associée au temps de l'horloge murale (e.g. 1 ut pour 1 h de "temps horloge murale" pour représenter un système à dynamique "lente" ; 1 ut pour 1 ps de "temps horloge murale" pour un système à dynamique "rapide"). Si le temps du monde réel a une précision infinie, le temps du simulateur doit lui utiliser une représentation mathématique, appelée temps mathématique, qui est sujette aux mêmes abstractions que les données (résolution, domaine, précision). Le temps de simulation fournit le temps "horloge murale" nécessaire pour effectuer la simulation ; il est lié à la performance du simulateur. Reprenons notre utilisateur, il désire maintenant simuler la position du soleil tel qu'on verrait le 31 décembre 2014 dans le ciel toulousain entre 4h le matin et 18h le soir (voir 2.10). Il décide de discrétiser la position du soleil pour connaître celle-ci à chaque heure. Il choisit donc une représentation mathématique discrétisée du type $t \in [4, 18]$ avec $t \in \mathbb{N}$. La figure 2.10 nous permet d'illustrer les trois concepts de temps. Le temps du simulateur est une représentation mathématique discrète du temps réel du 31 décembre 2014 entre 4h00 et 18h00, chaque unité de temps représente 1 h de "temps horloge murale". Cette simulation a été effectuée le 12 mars 2014 entre 15h02 et 15h04 (temps horloge

murale), le temps de simulation est donc de 2 minutes. Si l'utilisateur choisit un pas de temps du simulateur plus petit (e.g. position du soleil toutes les secondes) le temps de simulation pour un même calculateur sera augmenté, le temps du simulateur sera le même (31 décembre entre 4h00 et 18h00).

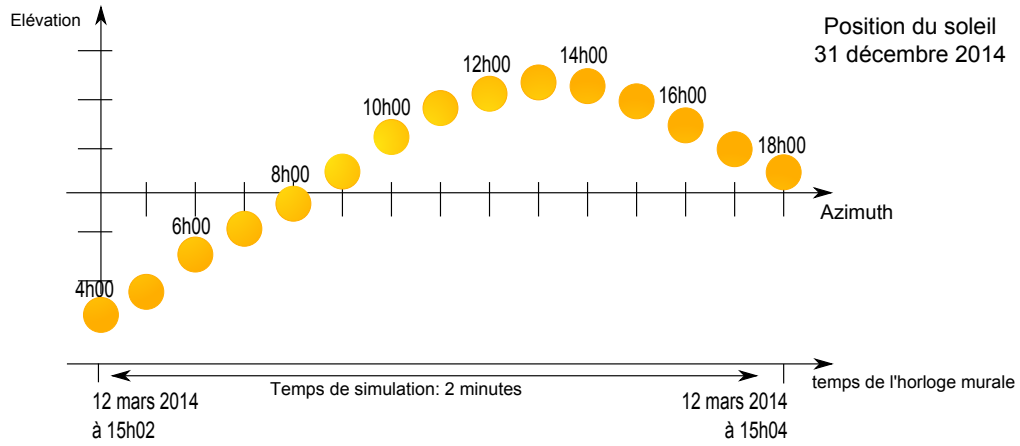


FIGURE 2.10 – Les trois concepts de temps

2.4.2 Compatibilité

L'applicabilité, comme l'accommodation, peuvent s'étudier de manière statique ou dynamique. La compatibilité statique cherche à s'assurer que la "connexion" entre modèle et cadre est suffisante. On s'intéresse au périmètre du produit de simulation. Cette compatibilité n'a pas besoin d'être totale pour permettre d'atteindre l'objectif de simulation. La compatibilité dynamique s'assure que cadre et modèle sont capables d'atteindre ensembles les objectifs de simulation. La compatibilité statique doit toujours précéder la compatibilité dynamique.

2.4.2.1 Étude du périmètre

La compatibilité statique a fait l'objet d'une étude approfondie dans les travaux de V. Albert [Albert 2009] où l'auteur étudie la compatibilité du périmètre du produit de simulation de façon formelle. L'auteur y étudie l'ensemble des critères d'applicabilité associés aux règles d'abstraction que nous avons mentionnées dans la section 2.4.1. Il y définit des règles sur l'information, la topologie, le domaine, l'unité, la résolution et la précision. De manière générale quelle que soit la règle, le modèle doit posséder un périmètre "plus capable" ou "autant capable" que le périmètre du cadre expérimental. La figure 2.11 reprend la représentation du produit de simulation tel que donné par l'auteur, où l'on voit apparaître le modèle et le cadre expérimental. Ce tableau montre au travers des relations E/S la problématique de compatibilité. Notons que les entrées non-utilisées des modèles n'influencent pas les

sorties utilisées du modèle pour que les résultats proposés soient tangibles. Le produit (2) n'est pas compatible puisque le cadre expérimental cherche à commander et observer le modèle au travers de la sortie y3 et de l'entrée x3 (voir figure 2.11) qui ne sont pas accessibles par le modèle. Le produit (1) est compatible puisque il ne cherche à commander que x1 et x2 et observer que y1 et y2. Ne pas commander x3 revient à ne pas explorer certaines parties du modèle, ne pas observer y3 n'impacte pas le comportement du modèle. Le produit est donc possible.

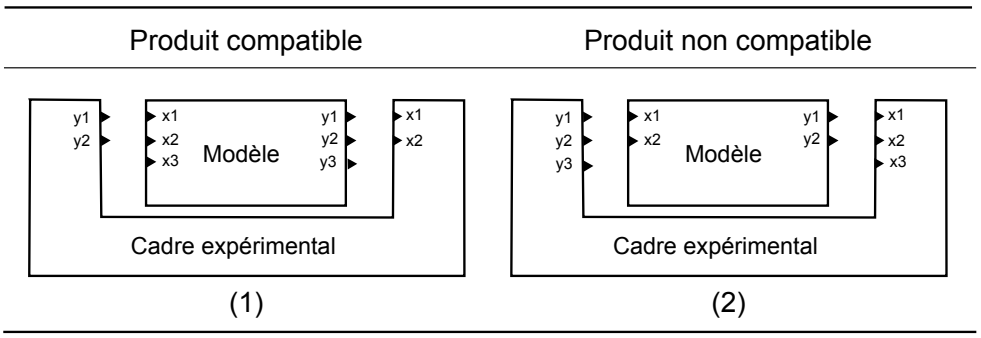


FIGURE 2.11 – Compatibilité du périmètre du produit de simulation : relation E/S

Lorsque cette compatibilité est assurée, il est possible de passer à la phase de simulation à proprement parler et donc à l'étude de la compatibilité dynamique.

2.4.2.2 Étude des trajectoires

Afin d'étudier la compatibilité dynamique, il devient nécessaire de présenter formellement la notion de système et de trajectoire. Cette définition formelle est basée sur la théorie systémique introduite à la section 2.3.1.

Définition 21. *Un **système** est un 8-uplet tel que :*

$$Sys =< T, X, \Omega, Y, Q, \Delta, \Lambda, \Gamma > avec$$

- T la base de temps,
- X l'ensemble des valeurs d'entrées,
- Y l'ensemble des valeurs de sorties,
- Ω est l'ensemble des segments d'entrées acceptables tels que $\Omega \subseteq (X, T)$,
- Q l'ensemble des états,
- $\Delta : Q \times \Omega \to Q$ la fonction de transition,
- $\Lambda : Q \times X \to Y$ la fonction de sortie.
- Γ [7] l'ensemble des segments de sortie qui peuvent être générés tels que $\Gamma \subseteq (Y, T)$,

7. Γ est inséré pour alléger la notion et les explications futures. Il n'est en rien obligatoire à la définition d'un système.

La base de temps T peut prendre trois formes : continue, discrète, partiellement ordonnée, comme le montre la figure 2.12, permettant de modéliser les différents types de systèmes dynamiques (voir 2.3.1).

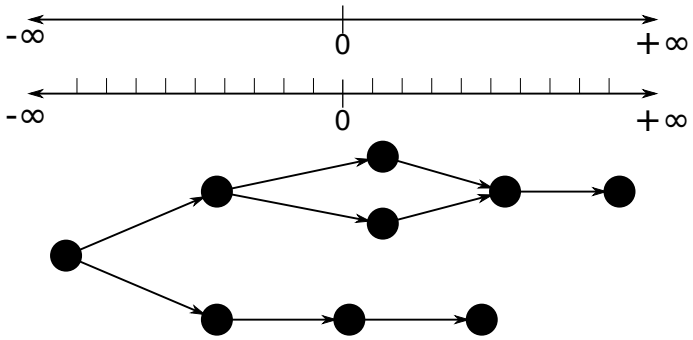


FIGURE 2.12 – Base de temps continue, discrète ou partiellement ordonnée [Zeigler 1976]

.

Définition 22. *Une **trajectoire** est une fonction qui associe à un temps $t \in T$ un ensemble d'entrées, de sorties ou d'états (notons le A) du modèle tel que :*

$$f : T \to A$$

Il est possible d'établir une restriction de la fonction trajectoire pour un intervalle de temps $< t_1, t_2 >$, aussi appelé segment :

$$\omega :< t_1, t_2 > \to A \text{ ou } \omega_{<t_1,t_2>}$$

Si deux segments $\omega_{1<t_1,t_2>}$ et $\omega_{2<t_3,t_4>}$ sont contigus ($t_2 = t_3$) il existe un segment $\omega_{<t_1,t_4>}$ construit par concaténation $\bullet$ tel que :

$$\omega_1 \bullet \omega_2(t) = \omega_1(t) \text{ pour } t \in < t_1, t_2 >$$
$$\omega_1 \bullet \omega_2(t) = \omega_2(t) \text{ pour } t \in < t_3, t_4 >$$

Un système est donc défini par un ensemble d'états Q qui réagissent à un ensemble de segments d'entrée ω sur X, ordonnés selon une base de temps T. La fonction de transition Δ définit l'état suivant du système en fonction du segment d'entrée w et de l'état courant $q \in Q$. La fonction de sortie Λ définit une trajectoire de sortie γ sur Y en fonction de l'état présent q et de la trajectoire d'entrée ω.

La figure 2.13 illustre l'application d'un segment d'entrée $\omega_{<t_a,t_b>}$ sur un système à événements discrets. Il est possible d'imaginer appliquer au système l'ensemble des segments d'entrées acceptables Ω permettant alors d'observer Γ, l'ensemble des sorties que peut générer le système [8] comme l'illustre la figure 2.14.

8. Suivant les propriétés du système (base de temps, variables d'entrées, variables de sorties, etc.), l'application de Ω sur le système va générer des trajectoires de natures différentes. Par souci de

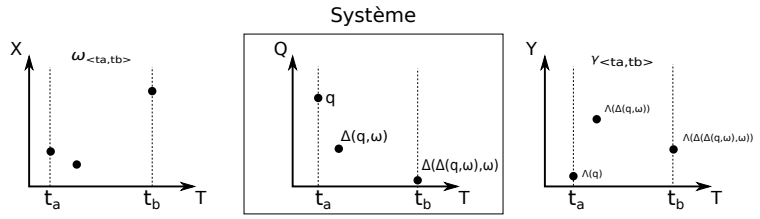


FIGURE 2.13 – Application d'un segment d'entrée à un système dynamique

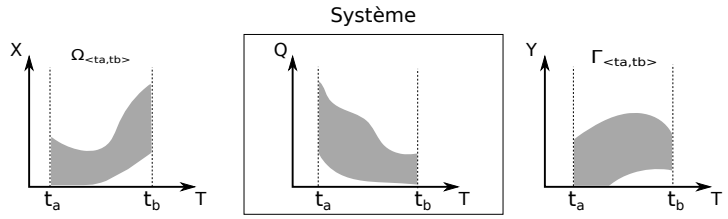


FIGURE 2.14 – Application de l'ensemble $\Omega_{<t_a,t_b>}$

.

Le cadre expérimental et le modèle peuvent être vus comme des systèmes. Nous en proposons les définitions suivantes :

Définition 23. *Un **cadre expérimental** noté EF*[9] *est un 8-uplet tel que :*

$$EF =< T_{EF}, X_{EF}, \Omega_{EF}, Y_{EF}, Q_{EF}, \Delta_{EF}, \Lambda_{EF}, \Gamma_{EF} >$$

Définition 24. *Un **modèle** est un 8-uplet tel que :*

$$M =< T_M, X_M, \Omega_M, Y_M, Q_M, \Delta_M, \Lambda_M, \Gamma_M >$$

où $T, X, \Omega, Y, Q, \Delta, \Lambda$ et Γ sont définis tels que précédemment.

La compatibilité dynamique entre cadre expérimental et modèle peut s'observer au travers des fonctions Ω et Γ.

Proposition 1. *La **compatibilité dynamique** entre les entrées du modèle M et les sorties du cadre expérimental EF est assurée si :*

$$\Gamma_{EF<t_a,t_b>} \setminus \Omega_{M<t_a,t_b>} = \emptyset \tag{2.1}$$

simplification, les illustrations considèreront tous les systèmes comme ayant des ensembles continus d'entrées, d'états et de sorties. Ces ensembles peuvent donc prendre une infinité de valeurs dans un domaine donné. Les modèles sont supposés vérifiés et donc bien construits. Tous les états sont atteignables par la fonction Ω donc $(Q \times \Omega)\Delta Q = \emptyset$ (Δ représente ici la différence symétrique : $A\Delta B = (A \cup B^c) \cap (A^c \cup B)$). Les critères de compatibilité statique établis dans [Albert 2009] sont supposés respectés.

9. EF pour *Experimental Framework*

La figure 2.15 illustre la superposition de l'ensemble Ω_M et de l'ensemble Γ_{EF}. Il apparaît ici que le cadre expérimental et le modèle sont compatibles dans l'intervalle $[t_a, t_b]$. Nous sommes donc dans le cas où $\Gamma_{EF<t_a,t_b>} \subseteq \Omega_{M<t_a,t_b>}$. Nous pouvons dire que l'ensemble des stimuli générés par le cadre expérimental est accepté par le modèle. Le cadre expérimental n'explore qu'une partie des entrées acceptées par le modèle.

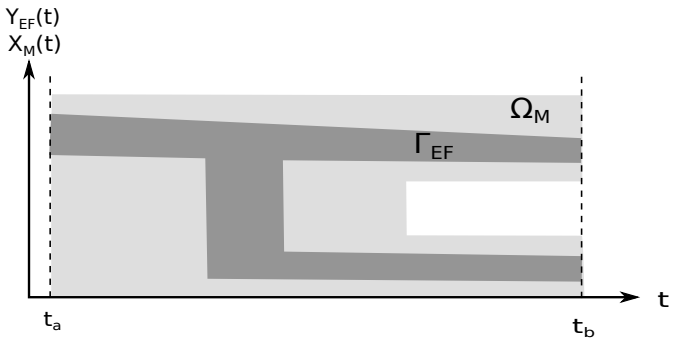


FIGURE 2.15 – Ensemble des segments E/S pour l'étude de la compatibilité dynamique entre X_M et Y_{EF} dans l'intervalle $[t_a, t_b]$.

Il est possible d'observer une compatibilité totale, c'est à dire que le cadre expérimental va explorer l'ensemble des valeurs d'entrées possibles $\Omega_{M<t_a,t_b>} = \Gamma_{EF<t_a,t_b>}$.

Une incompatibilité va se caractériser par l'apparition d'un sous ensemble Ψ tel que $\Gamma_{EF<t_a,t_b>} \setminus \Omega_{M<t_a,t_b>} = \Psi$ comme l'illustre la figure 2.16. La section 2.5.1 va discuter de cette incompatibilité et des conséquences associées sur le produit de simulation. Nous verrons qu'elle n'est pas toujours synonyme d'échec de l'expérimentation.

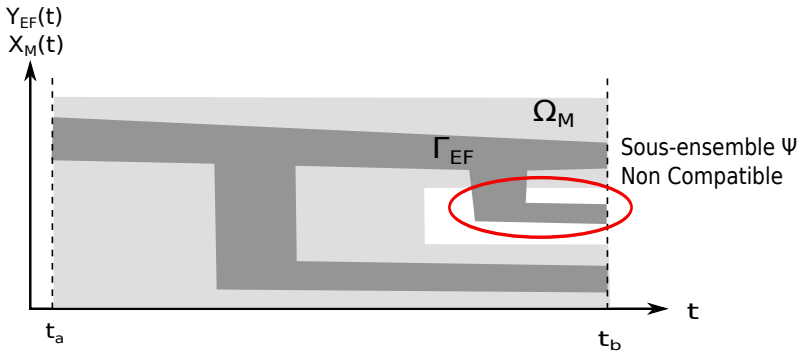


FIGURE 2.16 – Non compatibilité dynamique entre X_M et Y_{EF} dans l'intervalle $[t_a, t_b]$

.

Une réflexion similaire est faite sur les ensembles Ω_{EF} et Γ_M. Il s'agit ici de s'intéresser aux résultats de simulation attendus et donc à la compatibilité entre les trajectoires de sorties du modèle et les trajectoires d'entrées du cadre expérimental.

Proposition 2. *La **compatibilité dynamique** entre les entrées du cadre expérimental EF et les sorties du modèle M est assuré si :*

$$\Omega_{EF<t_a,t_b>} \setminus \Gamma_{M<t_a,t_b>} = \emptyset \tag{2.2}$$

2.5 Discussions

2.5.1 Est-il nécessaire d'obtenir une compatibilité totale ?

La compatibilité totale peut sembler nécessaire, elle exprime dans un certain sens une maîtrise du modèle par l'utilisateur de la simulation. Cependant, entre les objectifs d'expérimentation, les règles de compatibilité statique et la réutilisation de modèle (bibliothèque de modèle), il n'est pas toujours idéal, nécessaire ou même possible d'obtenir une compatibilité totale des trajectoires d'entrées ou de sorties. La compatibilité, qu'elle soit dynamique ou statique, ne doit donc pas chercher à être totale pour un modèle donné mais doit chercher à obtenir une compatibilité relative à un objectif de simulation donné.

Par définition, le cadre expérimental permet d'exprimer les objectifs d'expérimentation. Le générateur décrit les trajectoires d'entrées fournies au modèle. Le transducteur cherche à observer les trajectoires de sorties du modèle. Une exploration convenable serait alors un ensemble Γ_{EF} défini par le générateur qui permet au modèle de fournir un ensemble de trajectoires de sorties noté $\Gamma_{M\setminus\Gamma_{EF}}$ couvrant parfaitement les trajectoires Ω_{EF} attendues par le transducteur comme l'illustre la figure 2.17.

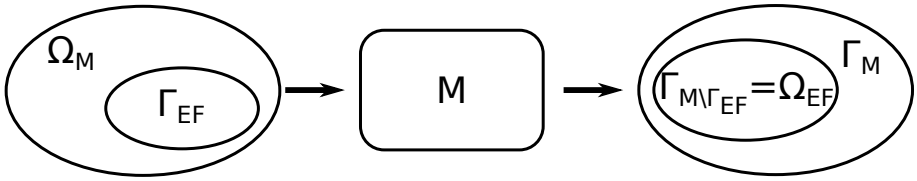


FIGURE 2.17 – Exploration complète pour un objectif de simulation défini par le couple générateur transducteur.

Proposition 3. *La **compatibilité dynamique** pour un objectif de simulation décrit par un couple générateur/transducteur est assurée si :*

$$\Gamma_{M\setminus\Gamma_{EF}} = \Omega_{EF} \tag{2.3}$$

Deux trajectoires d'entrées différentes peuvent amener le système à générer une trajectoire de sortie unique. En effet, deux segments ω_1 et $\omega_2 \in \Omega_{M<t_a,t_b>}$ génèrent en sortie du modèle une trajectoire de sortie unique $\gamma_1 \in \Gamma_{M<t_a,t_b>}$ s'il existe deux fonctions de sorties δ_1 et $\delta_2 \in \Delta_M$ définies telles que $\delta_1(q_i, \omega_1) = \delta_2(q_j, \omega_2)$ avec $\omega_1(x_i, t_i) \neq \omega_2(x_j, t_i)$. Il existe donc plusieurs ensembles de départ Ω_M pour un même ensemble d'arrivée Γ_M, le système peut donc être vu comme une application surjective. En d'autres termes, il existe plusieurs restrictions du modèle par le cadre expérimental permettant d'obtenir une couverture des objectifs de simulations définis au travers du transducteur.

Cette analyse peut s'apparenter à une analyse en boîte noire, où seul le comportement E/S est étudié. La comparaison directe entre les ensembles de trajectoires telle que proposée avec la propriété 3 est suffisante si l'ensemble des hypothèses est exprimé à l'aide du couple générateur/transducteur. Nous considérons qu'il est nécessaire d'exprimer les hypothèses selon plusieurs vecteurs. En effet, le formalisme utilisé pour décrire le cadre expérimental (générateur, transducteur) peut avoir une expressivité limitée, ce qui aurait pour effet de rendre difficile, lourd, voir impossible, l'expression de certaines hypothèses de simulation. Nous pensons que l'expression par l'utilisateur des hypothèses par différents vecteurs amène une certaine redondance qui peut permettre la détection d'incohérences dans la simulation permettant de juger plus efficacement le modèle de simulation.

Il est donc nécessaire d'étudier plus en détail la couverture du cadre sur le modèle afin d'établir plus précisément si le modèle de simulation est valide ou non, ce que nous ferons au travers du chapitre 3.

2.5.2 Quel intérêt porter aux résultats fournis par la V&V ?

Cette section cherche à apporter un regard critique sur les résultats fournis par la V&V et donc sur le produit de M&S et sa *crédibilité*. Il est en effet difficile de déterminer où est située l'erreur entre le modèle, la spécification et le cadre expérimental. Cette discussion n'a pas pour objectif de donner une liste exhaustive des problèmes que l'on peut rencontrer lors du processus de V&V, mais simplement d'amener le lecteur à porter une réflexion sur la crédibilité des résultats de V&V en général en s'appuyant sur l'exemple de la M&S telle que nous l'avons décrite. Le tableau 2.2 nous propose différents cas d'expérimentation que peut rencontrer l'utilisateur de la simulation et les résultats associés. Chaque résultat est obtenu avec une paire cadre-expérimental/modèle différente. L'objectif de la simulation est de s'assurer de la présence de la propriété *P1* dans le modèle pour un cadre expérimental donné.

► 1 (Vrai-Positif) : C'est en quelque sorte le cas nominal, la spécification est correcte, le modèle et le cadre expérimental sont eux aussi corrects. L'utilisateur de la simulation peut utiliser les résultats de simulation sans crainte.

► 2 (Vrai-Négatif) : Dans ce cas le modèle est faux. Le cadre expérimental est correct, mais il est impossible de trouver la propriété attendue dans le système (P1

n'est pas présente). Le modèle n'est pas consistant avec sa spécification. L'utilisateur de la simulation peut utiliser les résultats de simulation et donc demander à modifier le modèle ou le cadre expérimental sans discrimination possible. Une analyse pourrait permettre de détecter ici une erreur dans le modèle.

▶ 3 (Faux-Positif) : C'est un cas critique, l'utilisateur de la simulation va valider le modèle qui n'est pourtant pas consistant avec sa spécification (P2 à la place de P1). L'utilisateur de la simulation peut par exemple mal interpréter la spécification et développer un cadre expérimental qui n'est pas en accord avec l'objectif réel de simulation (cherche à valider P2 en lieu et place de P1). L'utilisateur de la simulation va utiliser ces résultats de simulation et valider un modèle non-conforme.

▶ 4 (Faux-Négatif) : Ici, le cadre expérimental ne permet pas d'explorer convenablement le modèle. La propriété à valider est présente dans le modèle mais non explorée et donc considérée comme non présente. L'utilisateur de la simulation va annoncer un modèle non conforme ou un cadre expérimental non conforme sans pouvoir discriminer l'un ou l'autre. Une analyse pourrait permettre de détecter ici une erreur dans le cadre expérimental

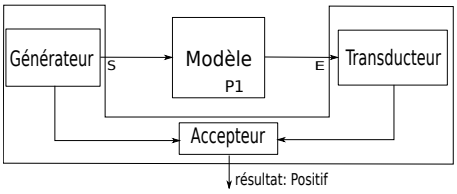
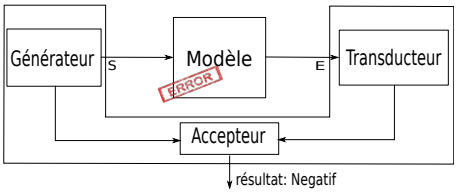
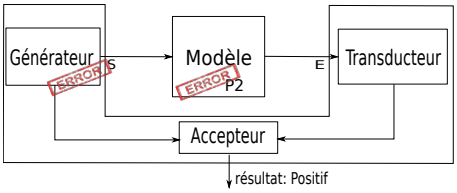
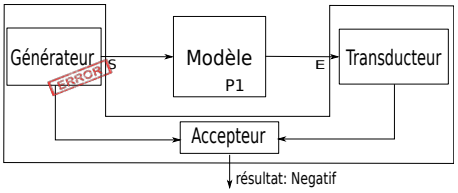
Résultat	**Positif**	**Négatif**
Vrai		
Faux		

TABLE 2.2 – Résultat possible de V&V

Il est évident que certains des cas présentés ici sont plus critiques que d'autres. La simulation ne fournit que le résultat "positif" ou "négatif", l'utilisateur de la simulation va devoir se créer une opinion pour décider si ce résultat est "vrai" ou "faux".
L'apparition d'un résultat négatif, qu'il soit "vrai-négatif" ou "faux-négatif", apporte la certitude qu'il y a eu insertion d'erreurs. S'il est impossible de définir la position de l'erreur, l'utilisateur de la simulation est pourtant assuré de son existence. La décision est simple et la criticité est faible.
Un résultat positif peut être soit un "vrai-positif", soit un "faux-positif" sans discri-

mination possible. Aussi paradoxal que cela puisse paraître, un résultat positif est donc un cas critique puisque l'utilisateur de la simulation doit prendre une décision fondée simplement sur la crédibilité qu'il accorde au produit de M&S. C'est un problème bien connu dans d'autres domaines comme le Test, ou le Model-checking, où il est impossible de démontrer l'absence d'erreurs et non leur présence.

La problématique de cette thèse est donc d'améliorer le crédit qu'accorde l'utilisateur de la simulation au produit de simulation en proposant des méthodes et des métriques adaptées.

2.6 Résumé

Nous avons pu voir dans ce chapitre les principes liés à la V&V du modèle de simulation. Nous avons discuté de la notion de modèle conceptuel et de son rôle spécifique dans le domaine de la M&S. En présentant la théorie de la M&S vue par B.P. Zeigler nous avons pu poser la problématique d'application et d'accommodation que nous regroupons sous le terme de compatibilité. Il a été proposé d'étudier la compatibilité dans son aspect dynamique afin de compléter l'étude fournie par V. Albert sur la compatibilité du périmètre entre cadre expérimental et modèle. Les notions de morphisme et de dérivabilité ont été introduites au travers d'illustrations nous permettant de mieux aborder le rôle de la compatibilité.

Ce chapitre se termine sur deux discussions. La première discussion nous permet d'établir qu'une compatibilité totale n'était pas nécessaire et qu'il était même préférable d'établir le niveau de compatibilité vis-à-vis de l'objectif de simulation et non des capacités respectives du cadre et du modèle. La seconde discussion aborde la notion crédibilité du résultat, montre qu'il est difficile d'attribuer une réponse binaire "vrai/faux" à un résultat. Cette conclusion va nous amener à chercher à proposer à l'utilisateur de la simulation un ensemble d'outils associé à l'étude de la compatibilité dynamique lui permettant d'améliorer le niveau de crédibilité qu'il accorde à son produit de simulation pour répondre à l'objectif d'expérimentation.

Chapitre 3

Qualification de la validité d'un modèle de simulation

Sommaire

3.1 Introduction

Il est nécessaire d'améliorer la crédibilité qu'associe l'utilisateur de la simulation à son expérimentation et aux résultats qu'il fournît. Pour cela, nous allons étudier au cours de ce chapitre la compatibilité dynamique entre un cadre expérimental et un modèle, autrement appelée : étude de la relation d'applicabilité et d'accommodation. Après une courte introduction sur la notion de crédibilité, nous aborderons,

dans une première section l'état de l'art sur les métriques permettant l'évaluation de modèle dans un contexte de simulation. La deuxième section fournira les bases théoriques de la combinatoire des mots et de la théorie des langages sur lesquelles s'appuient ces travaux. Les sections suivantes seront dédiées à l'étude formelle de la compatibilité cadre expérimental/modèle. Notre étude, divisée en deux parties, se focalisera, dans un premier temps, sur l'étude de la compatibilité dynamique entre modèle et cadre expérimental, en utilisant un formalisme de spécification proposant une base de temps "partiellement ordonnée", appelé automates à interface. La seconde partie de l'étude étendra ces automates à interfaces avec l'ajout d'une base de temps continue. Nous utiliserons alors un formalisme de spécification à événements discrets, Discret Event System Specification (DEVS). Ces deux parties seront mises en correspondance avec l'ingénierie système et la M&S [Zeigler 1976]. Une dernière section discutera du réel apport des métriques, des bénéfices à retirer des deux approches proposées, ainsi que de l'importance des hypothèses dans la modélisation.

3.1.1 Crédibilité du produit de M&S

Évaluer la validité d'un modèle ou d'une simulation devient de plus en plus difficile et représente un enjeu réel et immédiat [Lin 2013]. Comme nous l'avons vu, la complexité des simulations ne cesse d'augmenter et il devient difficile, voire impossible, pour l'utilisateur de la simulation, de démontrer la crédibilité des résultats de simulation. Il devient alors nécessaire de proposer à l'utilisateur de la simulation un ensemble d'outils et de méthodes adéquat.

Dans [Liu 2005], l'auteur nous propose de définir la crédibilité de la façon suivante. :

Définition 25. *La **crédibilité d'un modèle** ou **d'une simulation** est l'expression du degré avec lequel quelqu'un est convaincu que le modèle ou la simulation est adapté à un usage prévu.*

Nos travaux cherchent donc à fournir à l'utilisateur de la simulation un ensemble de métriques lui permettant de s'assurer avec plus de conviction que sa simulation est adaptée à l'usage prévu, et ainsi à améliorer la crédibilité qu'il accorde aux résultats.

Il existe différents travaux se rapportant à la crédibilité de la M&S. La communauté s'accorde à dire que les métriques de crédibilité pour un produit de M&S sont basées sur cinq métriques fondamentales : la validité, l'exactitude, la fiabilité, l'utilisabilité et l'interopérabilité. Le DoD (Department of Defense) propose d'établir les différents liens entre métriques de crédibilité et VV&A (voir figure 3.1)[DoD 2000].

On constate alors que la validation peut apporter de la crédibilité au travers de l'évaluation de la validité et de l'interopérabilité. La validité est ici vue comme la capacité du modèle/ de la simulation à déduire ou calculer suffisamment correctement un ensemble d'éléments permettant de répondre à une demande spécifique.

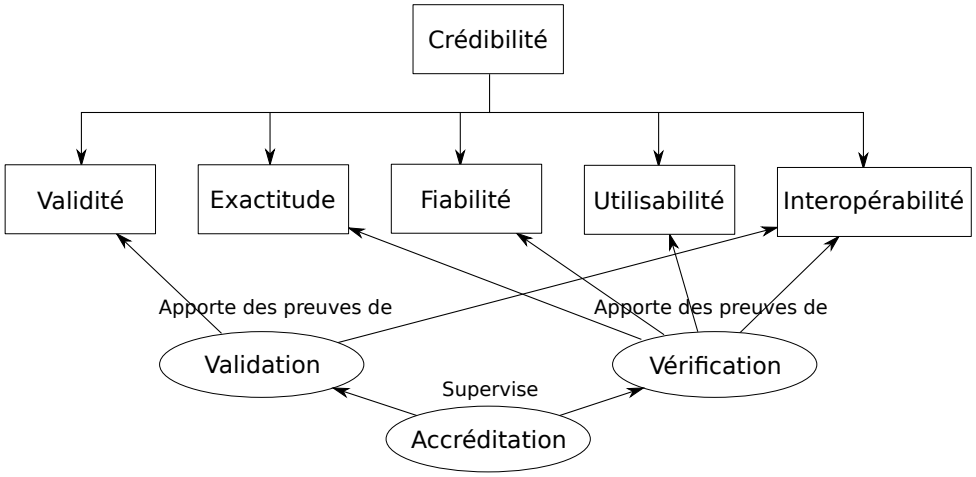


FIGURE 3.1 – Dépendances entre métriques de crédibilité et VV&A.

L'interopérabilité peut être vue comme la capacité du modèle/ de la simulation à fournir ou à accepter des services des autres modèles/simulations afin de fonctionner ensemble de façon efficace, pour répondre à une demande spécifique.

De nombreux travaux ([Gross 1999], [DoD 2000], [Sargent 2000], [Liu 2005]) s'accordent à dire que la validation est la source la plus importante d'évaluation de la crédibilité d'un produit de M&S. Nos travaux s'inscrivent dans cette logique, en proposant un ensemble concret de métriques associées à la validation du modèle conceptuel, à travers l'étude de la validité et de l'interopérabilité. A notre connaissance, il n'existe pas à ce jour de travaux [1] proposant des métriques dédiées à l'étude de la compatibilité du produit de simulation. C'est pourquoi nous élargirons l'état de l'art aux différents domaines proposant des métriques d'évaluation de modèle.

3.1.2 Une première idée de la notion de métrique et de distance

Nous considérons, dans l'état actuel de nos connaissances, qu'il n'existe pas de travaux proposant d'étudier la relation de compatibilité dynamique ou tentant de la mesurer. Nous proposons ici une présentation rapide et informelle de la notion de métrique, qui sera abordée de manière formelle et détaillée dans la suite de ce chapitre.

Les métriques (au sens logiciel du terme) peuvent être vues comme un moyen de connaître la distance entre deux points. Assigner une "distance" entre le comportement prévu d'une simulation et le comportement réel d'une simulation peut être discutable puisque le comportement prévu est rarement modélisé. Habituellement, l'utilisateur de la simulation définit un ensemble de stimuli et effectue une analyse

1. A l'exception des travaux de V. Albert qui dirige cette thèse

des résultats fournis. Par manque de formalisme, la mesure de cette distance devient difficile, voir impossible. L'utilisation d'un cadre expérimental va permettre de capturer de manière formelle les objectifs de simulation, les hypothèses et les contraintes. Ceci va nous permettre de détecter les incohérences et de mesurer la distance entre le comportement de la simulation et le comportement prévu par l'utilisateur de la simulation. Cette distance peut être partiellement décrite par l'étude de le compatibilité du périmètre (voir chapitre 2). Cependant, l'étude statique ne permet pas de détecter de distances associées à une incompatibilité dynamique.

Ces mesures doivent être, pour l'utilisateur, un outil l'aidant à juger la confiance qu'il peut accorder à la simulation. Il est possible de représenter le principe général de distance entre comportement de la simulation et comportement attendu de la simulation par l'illustration de la figure 3.2. La partie grisée représente l'ensemble des comportements possibles du modèle (segment que peut générer le modèle pour un ensemble d'hypothèses d'utilisation). Le couple générateur/transducteur n'est pas encore appliqué. L'utilisateur de la simulation prévoit, à partir du même ensemble d'hypothèses, une trace ou un ensemble de traces d'exécution du modèle, qu'il définit à partir des élément observables et commandables du modèle. Un écart entre les possibilités réelles du modèle et celles prévues par l'utilisateur peut alors apparaître. Cette distance peut être positive, négative ou nulle. Le modèle est alors "moins capable" que prévu (cas de la figure 3.2), "plus capable" ou "autant capable". La détection de cet écart est primordiale. Si cette distance n'est pas détectée, cela peut mener à une mauvaise interprétation des résultats de simulation par l'utilisateur, et donc potentiellement à la catastrophe (cf. chapitre 1).

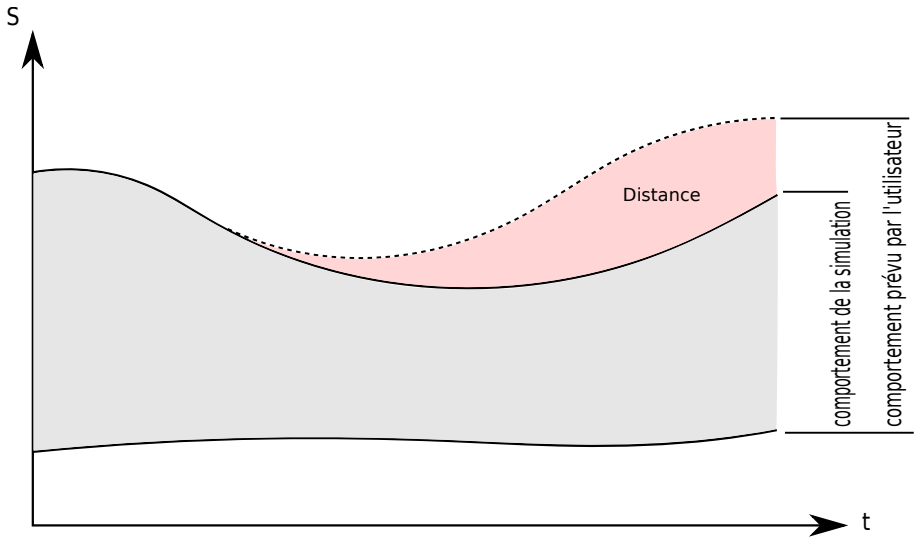


Figure 3.2 – Principe général de trace et distance

Dans les faits, une métrique seule n'est pas capable de mesurer cette distance,

qui est fonction de nombreux paramètres (hypothèses, objectifs, variables d'intérêt, etc.). Notre approche va consister à définir un ensemble de métriques permettant d'évaluer cette distance par composition, et ainsi de donner à l'utilisateur la possibilité d'évaluer son expérimentation et la crédibilité qu'il associe aux résultats. Ce principe peut être limité à un sous-ensemble de propriétés. L'intérêt n'est alors plus porté sur l'ensemble des traces mais uniquement sur un sous-ensemble voire sur certains états particuliers (figure 3.3).

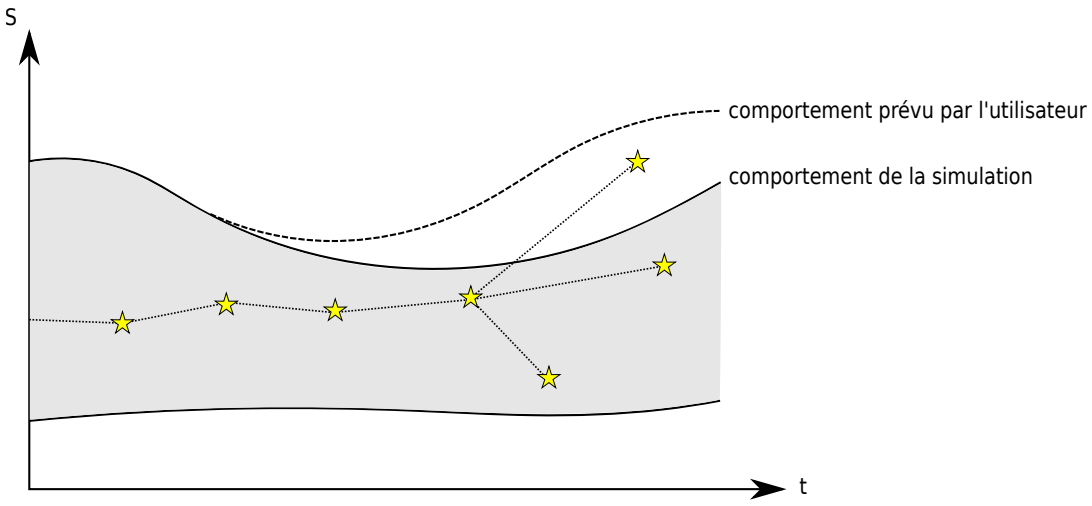


FIGURE 3.3 – Principe limité de trace et distance

Essayons maintenant de mettre en évidence, toujours de manière informelle, les principes que nous allons associer pour mesurer cette distance. L'utilisateur dispose d'un modèle qu'il va utiliser pour valider le système sous test. Il est donc sous entendu que le modèle sera stimulé dans un objectif particulier, sous un ensemble d'hypothèses déterminé. Comme indiqué précédemment, l'évolution d'un modèle au cours du temps peut être analysée comme un ensemble de segments. Ces segments peuvent être alors divisés en "zones" (sous ensemble d'états), chaque "zone" étant associée à une propriété respectée par les états (ici : 1, 2 et 3) (voir figure 3.4).

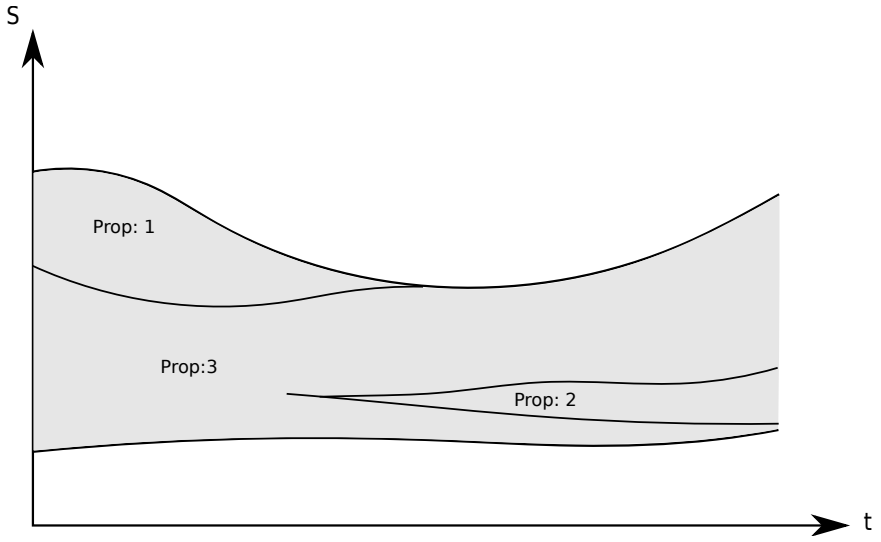


FIGURE 3.4 – Ensemble des segments de propriétés d'un modèle

L'utilisateur va alors définir un couple générateur/transducteur, qui va permettre d'établir les conditions d'expérimentation pour atteindre l'objectif de simulation. Le cadre expérimental est une hypothèse forte sur le modèle, permettant de commander et d'observer le système d'intérêt afin de répondre à l'objectif de validation. Ce cadre expérimental va contraindre le modèle qui sera alors restreint à un sous-ensemble de traces. C'est ici que peut apparaître une distance entre le comportement réel de la simulation et celui attendu.

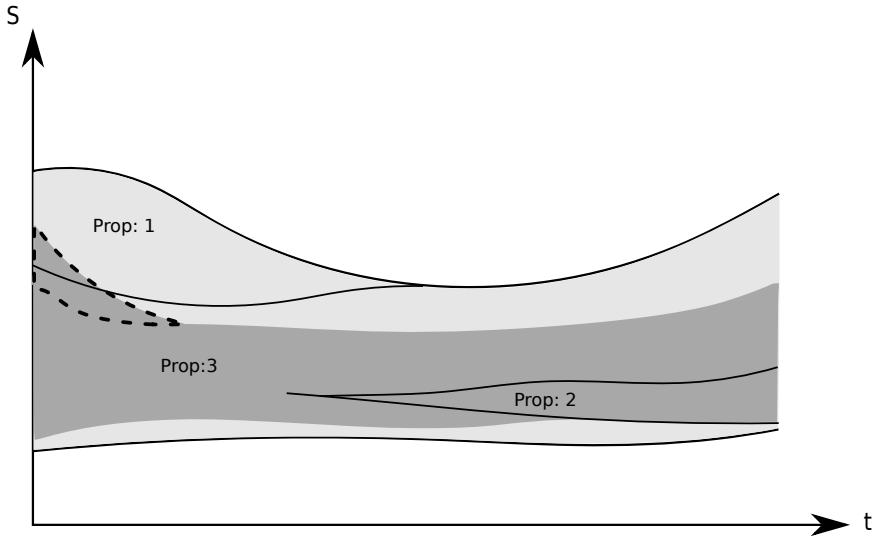


FIGURE 3.5 – Cadre appliqué au modèle

Sur l'exemple de la figure 3.5, l'utilisateur a pour objectif d'explorer la partie "*Prop : 2*". La partie grise (gris clair + gris foncé) représente les segments du mo-

dèle "partiellement contraint" qu'il est possible d'explorer. Nous considérons comme "partiellement contraint" un modèle dont seul un sous-ensemble du cadre expérimental (un sous-ensemble d'hypothèses et de contraintes statiques) est appliqué. A l'inverse, nous appelleront un modèle "totalement contraint" un modèle où le cadre expérimental dans son ensemble (générateur, transducteur, ensemble des hypothèses) est appliqué. La partie en gris foncé de la figure 3.5 représente les segments explorés du modèle "totalement contraint". Nous observons ici que l'utilisateur de la simulation a exploré toute la zone "*Prop 2*" : la distance au regard de cet objectif est donc nulle. L'objectif et l'exemple étant basiques, il n'est pas possible de mieux définir la distance. Le cadre expérimental et le modèle sont compatibles, ce couple permet d'atteindre l'objectif.

Imaginons maintenant que l'utilisateur de la simulation ait comme objectif l'exploration totale de la propriété *Prop 2*, sans jamais explorer *Prop 1*. L'application du même cadre expérimental va cette fois montrer une distance entre le comportement prévu par l'utilisateur de la simulation et le comportement réel. La zone en pointillés sur la figure 3.5 est une zone non prévue par l'utilisateur mais qui est accessible par application du couple générateur/transducteur. Il est essentiel de comprendre que cette zone représente une erreur humaine, celle de l'utilisateur de la simulation. Les causes de cette distance peuvent être très nombreuses : manque d'hypothèses, mauvaise compréhension du fonctionnement du modèle, erreur de modélisation et bien d'autres. La détection n'est ici possible que si l'utilisateur précise qu'il souhaite explorer entièrement *Prop 2* sans explorer *Prop 1*. Délimiter a priori précisément cette zone en pointillés est impossible (cf. 2.5.2). Elle pourra être détectée et minimisée par l'utilisation de méthodes formelles ou encore de méthodes expertes. Dans notre cas, nous pourrons détecter la zone associée à *Prop 1*. Nous pensons que l'utilisation conjointe d'hypothèses, de propriétés et, de manière plus générale, du cadre expérimental, fournit des vecteurs différents, permettant à l'utilisateur d'exprimer avec une redondance partielle des informations nécessaires à l'expérimentation. Ceci permet d'une part, de limiter l'erreur en facilitant la détection des incohérences et d'autre part, de fournir une approche automatisable et intégrable dans une démarche d'ingénierie.

Cette présentation nous a permis d'énoncer de façon informelle et parfois approximative la philosophie que nous allons présenter et développer dans ce chapitre pour établir les métriques.

Les types de métriques envisageables sont nombreux, il ne s'agissait ici que d'une réflexion préliminaire, que nous allons compléter et formaliser dans ce chapitre.

3.2 État de l'art sur les métriques d'évaluation des modèles

La littérature propose un grand nombre de métriques permettant d'évaluer les modèles. C'est le génie logiciel, et plus particulièrement le domaine "qualité logiciel", qui envisage en premier des métriques pour l'évaluation des modèles [Akiyama 1971]. Ce domaine s'intéresse à différentes problématiques : la complexité, la compréhensibilité, la maintenabilité et la stabilité des modèles. Les propositions de métriques du génie logiciel s'appuient essentiellement sur la structure des modèles. Dans [Chidamber 1991], S.R. Chidamber définit un ensemble de métriques permettant de juger la complexité, la maintenabilité et la ré-utilisabilité des modèles orientés objets. Il s'intéresse par exemple au nombre de couplages entre classes ou encore à la profondeur d'héritage des méthodes. De très nombreux auteurs vont par la suite contribuer à améliorer ces métriques : on peut citer [Brito e Abreu 1996], [Genero 2000], ou encore [Gronback 2004] qui travaillent essentiellement à l'évaluation des modèles orientés objets et plus particulièrement aux modèles développés avec le langage UML [Rumbaugh 1999]. Nous considérons que ces mesures de complexité, de compréhensibilité, de maintenabilité et de stabilité sont essentielles pour améliorer la confiance de l'utilisateur de la simulation dans son expérimentation. Ces mesures sont autant d'indicateurs de la qualité de la modélisation et donc de la crédibilité des résultats fournis par ces modèles. Les travaux proposant des métriques adaptées aux approches basées composants, comme [Narasimhan 2009] ou encore [Cai 2000], pourront être utilisés pour nous permettre de proposer à terme un outil et une méthode complète.

Toujours dans le domaine logiciel, un ensemble de métriques pour l'analyse de la réutilisabilité contextuelle de composants logiciels a été proposé récemment ([Washizaki 2003], [Narasimhan 2004], [Bhattacharya 2005], [Wu 2008],[Gui 2009], etc.). Cette tendance, motivée par l'apparition de composants logiciels open source, et la volonté économique des entreprises de réutiliser des modèles déjà développés, fournit un nombre important de métriques.

La prise en compte du contexte donne à ces travaux une importance particulière, puisque la notion de cadre expérimental est assimilable au contexte d'expérimentation, comme nous l'avons déjà vu. Dans [Wu 2008], les auteurs proposent d'étudier la réutilisabilité d'un modèle open-source pour un nouveau contexte. Ils s'intéressent à la complexité de l'interaction entre le composant et son contexte. La prise en compte d'un contexte spécifique donne lieu à une étude du comportement. Le composant est alors décrit grâce à un modèle UMD [2] étendu permettant de définir le comportement. Le modèle est représenté grâce à un graphe dirigé $M = \{N, T, P\}$, où N est un ensemble de nœuds tel que $N = \{\{E, S\} \cup F \cup C\}$, T est l'ensemble des transitions, et P une fonction qui associe à chaque transition un nombre d'appel de fonction (noté l) et une probabilité (notée p) telle que $P :< x, y >\in T \rightarrow (l, p)$. Les auteurs proposent

2. UMD : Usage and Dependency Model

trois métriques, dont CBDS [3] qui va mesurer le niveau de dépendance du modèle vis-à-vis du contexte pour lequel il a été initialement développé. $CBDS = \frac{m}{n}$ avec n le nombre de chemins sans boucle entre S et E, et m le nombre de chemins passant par un noeud de contexte C. Une seconde métrique CSD [4] va mesurer la difficulté à substituer le contexte du composant $CSD = \sum_{t=\langle x \in F, y \in C \rangle} \frac{t.l}{t.p} + \sum_{t=\langle x \in C, y \in F \rangle} \frac{t.l}{t.p}$. Le but est ici d'évaluer à quel point le contexte pour lequel le composant a été développé est imbriqué dans son comportement. Il permet d'évaluer l'effort nécessaire à la création d'un nouveau contexte adapté au modèle. Ces métriques ne sont donc pas capables de répondre à notre problématique, mais peuvent être envisagées pour une étude de l'effort nécessaire pour la création du contexte de simulation (cadre expérimental).

Dans la communauté M&S, D. Karlsson s'intéresse à l'amélioration de la couverture des simulations [Karlsson 2008]. Il propose alors plusieurs métriques de couverture. En utilisant les réseaux de Petri, l'auteur calcule le nombre de places ou de transitions tirées dans un réseau de Petri pour un ensemble de stimuli donné. Dans le but de valider le modèle, l'auteur propose une méthode en deux phases. Dans un premier temps (phase 1), il génère un ensemble de stimuli, et si une trop grande partie du modèle n'est pas couverte, il utilise alors des algorithmes issus des méthodes formelles (model-checking) pour explorer des zones non atteintes par le cadre expérimental (phase2), puis reprend la phase 1. L'auteur cherche ici à valider le modèle en combinant les avantages du model-checking et de la simulation, pour limiter le temps total de validation. Même si les métriques proposées ne sont pas dédiées à la validation du produit de simulation, nous verrons que l'utilisation de métriques de couverture est également intéressante pour notre problématique.

G. Pappas et A. Girard proposent, dans [Girard 2005], un ensemble de métriques dédié à l'étude de l'abstraction des systèmes. Les auteurs fournissent une plate-forme dédiée aux systèmes continus et discrets, puis étendue aux systèmes hybrides dans [Girard 2008], où un ensemble sophistiqué de métriques permet de quantifier la qualité de l'approximation faite entre deux systèmes. Leur travail s'assimile à l'étude de la validation de l'abstraction entre deux modèles, que nous avons introduit ici au même titre que l'étude de la relation de dérivabilité et de morphisme. Comme nous l'avons vu dans le chapitre 2, l'étude de ces relations, bien qu'essentielle pour améliorer le domaine de la M&S, répond à des problématiques différentes de l'étude de la relation d'applicabilité et ne sera donc pas présentée ici.

La notion de métrique pour la validation d'un modèle de simulation apparaît également dans [Sewell 2006] : le modèle, un simulateur de mastoïdectomie [5], est validé en mesurant l'écart de résultats suite à l'application de deux contextes différents, un opérateur expérimenté (professeur), et un novice (interne). Les métriques

3. CBDS : Component Backward Dependency Strength
4. CSD : Context Substitution Degree
5. La mastoïdectomie est une opération qui consiste à retirer une partie de l'os qui se trouve derrière l'oreille.

proposées cherchent à comparer la quantité de zone saine touchée durant l'opération par un débutant et par un expert. Nous considèrerons l'utilisation de ces "métriques" comme une démarche de validation basée sur l'expertise et le prototypage.

On observe également que l'utilisation de métriques est dérivée dans plusieurs domaines dont celui de la simulation pour l'ingénierie système, où apparaît le terme "méta-métrique" ([Meyer 1997], [Berger 2013]). Ces métriques sont dédiées à la gestion de projet, elles fournissent tout au long du développement des données quantifiées permettant d'optimiser la gestion du projet pour obtenir un produit de meilleure qualité. Nous ne considèrerons pas ces métriques dans notre approche. Cependant, nous verrons que la crédibilité que porte l'utilisateur sur les résultats fournis par la simulation peut apporter aux méta-métriques un élément supplémentaire permettant de juger de l'avancement du projet et donc d'optimiser l'allocation de ressources.

3.3 Rappel : Combinatoire des mots et langages

Cette section présente certains concepts de base de la théorie des langages et de la théorie des automates, essentiels à la définition formelle des métriques.

La combinatoire des mots est une branche de l'informatique théorique qui verra sa version moderne se développer dans les années 80, avec [Lothaire 1983] (version corrigée [Lothaire 1997]).

Définition 26. *Un **alphabet**, noté Σ, est un ensemble non vide de symboles, appelés "lettres", "étiquettes" ou "évènements". Dans la suite, nous retiendrons le terme "événement".*

Exemples : Le binaire : $\{0, 1\}$; L'alphabet latin : $\{a, b, c, ..., y, z\}$

Définition 27. *Un **mot**, noté m, est une suite ordonnée de symboles d'un alphabet. Le mot vide est noté ε.*

Exemples : ε, 0, 011, 0100110 sont des mots définis à partir de l'alphabet binaire.

L'ensemble des mots sur Σ est noté Σ^*. La longueur d'un mot m est le nombre de symboles composant ce mot ; on le note $|m|$. Ainsi : $|10| = 2$ et $|bonjour| = 7$. Le mot vide ε a une longueur de 0, ainsi $|\varepsilon| = 0$

Définition 28. *Un **langage** est un ensemble (fini ou infini) de mots construit sur un alphabet Σ. On le note $L(\Sigma)$, avec $L(\Sigma) \subseteq \Sigma^*$.*

Définition 29. *On définit la **fermeture préfixe** de L telle que : $Pref(L) = \{m_1 \in \Sigma^* | \exists m_2 \in \Sigma^*, m1.m2 \in L\}$*

Les langages sont des ensembles, il est donc possible d'utiliser les opérations ensemblistes habituelles. Nous allons présenter ici celles qui possèdent un intérêt particulier pour l'élaboration de nos métriques.

Définition 30. *L' **étoile de Kleene** est un opérateur unaire tel que :* $S^* = \bigcup_{i \geq 0} S^i$

Alors, les mots de S^* constituent l'ensemble des mots obtenus par concaténation d'un nombre i arbitraire de mots de S.
Exemple : $S = \{aa, b\} \Rightarrow S^* = \{\varepsilon, aa, b, aab, bb, aaaa, aaaaaabbaa, ...\}$

Définition 31. *L' **opération d'intersection** permet de faire apparaître la partie commune de deux langages. L'intersection de deux langages est un langage.* $L(\Sigma_1) \cap L(\Sigma_2) = \{m | m \in L(\Sigma_1) \wedge m \in L(\Sigma_2)\}$

Définition 32. *L'**opération de complémentarité** :*
$\overline{L_i}(A) = \{m | m \in \{L(A)\} \wedge m \notin L_i(A)\}$

Cet ensemble de définitions élémentaires servira de base tout au long de ce chapitre, pour présenter les différents formalismes utilisés et proposer une définition formelle des métriques.

3.4 Étude de la compatibilité : Approche automates à interface

3.4.1 Automates à interface

Les automates à interface (*Interface Automata*) introduis par L. Alfaro et T. Henzinger dans [De Alfaro 2001], sont des systèmes de transition possédant 3 types d'actions : action d'entrée (?), action de sortie (!) et action interne (;). Développés initialement pour étudier la composition des systèmes dans une approche optimiste, leur différentiation des éléments observables et commandables, ainsi que leur spécification légère, font des automates à interface de bon candidats pour l'étude de la compatibilité entre cadre expérimental et modèle.

Nous n'utiliserons pas la notation proposée par les auteurs, mais une notation proche de celle définie par B.P. Zeigler pour les automates DEVS, afin de maintenir une notation cohérente entre les différents formalismes et de voir rapidement leurs similitudes. Les automates à interface ont une syntaxe identique aux automates à entrée-sortie (input/output automata) développés par N. Lynch et M.R. Tuttle [Lynch 1988]. La différence pour un automate à interface provient de la non-nécessité d'activer les entrées dans chaque état du système. Nous verrons que cette particularité fait des automates à interface de bons candidats pour l'élaboration de systèmes complexes et leurs simulations.

Définition 33. *Un **automate** est un 6-uplet tel que :*

$$A = \langle X, Y, S, s_0, \delta_x, \delta_y \rangle$$

où,
— X est un ensemble fini d'évènements d'entrées ;
— Y est un ensemble fini d'évènements de sorties ;

— S est un ensemble fini d'états, $s_0 \in S$ l'état initial ;
— $\delta_x : S \times X \to S$ est la *fonction de transition d'état externe* qui définit comment un événement change l'état de l'automate ;
— $\delta_y : S \to Y^\phi \times S$ est la *fonction de sortie et de transition d'état interne*, avec $Y^\phi = Y \cup \{\phi\}$ et $\phi \notin Y$ qui définit *l'évènement nul.* [6]

A titre d'exemple, considérons un système classique type "distribution de boissons" : ici un système permettant l'élaboration de café . Après avoir inséré un jeton, l'utilisateur choisit entre *long* (choix1) et *court* (choix2), pour générer un évènement servi au distributeur, et attend la fin du service (signal fin, envoyé par le distributeur). L'utilisateur peut récupérer son jeton s'il n'a pas fait de choix en demandant un *retour*. L'introduction d'un second jeton par l'utilisateur génère une erreur et restitue automatiquement le jeton.

Ceci nous donne comme spécification formelle :
$X = \{jeton, choix1, choix2, retour, fin, retourjeton\}$;
$Y = \{erreur, retourjeton, service\}$;
$S = \{repos, 1credit, 2credit, retour, court, long, pret\}$; $s_0 = repos$;
$\delta_x(repos, jeton) = 1credit$;
$\delta_x(1credit, jeton) = 2credit$;
$\delta_x(1credit, choix1) = court$;
$\delta_x(pret, fin) = repos$;
$\delta_x(retour, retourjeton) = (1credit)$;
$\delta_x(1credit, choix2) = long$;
$\delta_x(1credit, retour) = repos$;
$\delta_y(2credit) = (erreur, retour)$;
$\delta_y(court) = (service, pret)$;
$\delta_y(long) = (service, pret)$.

Sa représentation graphique est donnée figure 3.6. Avec la syntaxe concrète choisie, les états sont représentés par des cercles étiquetés et les transitions par des arcs étiquetés par une action (entrée ?/ sortie !/ interne ;). L'état initial est repéré par une flèche sans action. Les états dit "marqués" sont représentés par un double cercle.

3.4.1.1 Langages d'un automate :

Intéressons-nous d'abord aux fonctions de transition δ_X et δ_Y qui ont été définies dans la section "Automate à interface" comme étant respectivement [7] : $\delta_x : S \times X \to S$ et $\delta_y : S \to Y^\phi \times S$. On peut alors définir la fonction de transition d'état

6. δ_y peut être divisée en deux sous-fonctions : la fonction de sortie $\lambda : S \to Y$ et la fonction de transition interne $\delta_{int} : S \to S$

7. Bien que lourde, la distinction entre les fonctions de transition δ_X et δ_Y sera conservée puisqu'elle permettra d'établir certaines métriques lors de cette étude.

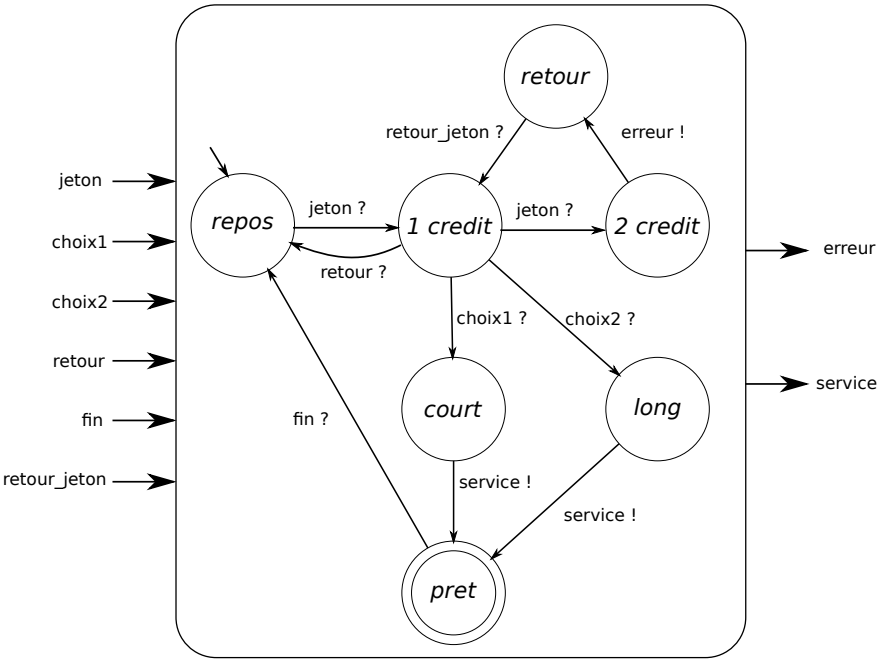


FIGURE 3.6 – Automate à interface : Machine à café

externe étendue telle que $\delta_X^* : S \times X^* \to S$ avec $\delta_X^*(s_i, m_k) = s_j \Leftrightarrow \delta_X(s_i, x1) = s_{i+1}, \delta_X(s_{i+1}, x2) = s_{i+2}...\delta_X(s_{j-1}, x_n) = s_j$

Il est également possible de définir la fonction de transition d'état interne étendue : $\delta_Y^*(s_i) = (m_l, s_j) \Leftrightarrow \delta_Y(s_i) = (y_1, s_{i+1}), \delta_Y(s_{i+1}) = (y2, s_{i+2})...\delta_Y(s_{j-1}) = (y_n, s_j)$

Tout comme δ_Y, δ_Y^* peut être divisée en deux sous-fonctions ; la fonction de sortie étendue $\lambda^* : S \to Y^*$ et la fonction de transition interne étendue $\delta_{int}^* : S \to S$, avec :

$$\lambda^*(s_i) = m_l \Leftrightarrow \lambda(s_i) = y_1, \lambda(s_{i+1}) = y2...\lambda(s_{j-1}) = y_n \tag{3.1}$$

$$\delta_{int}^*(s_i) = s_j \Leftrightarrow \delta_{int}(s_i) = s_{i+1}, \delta_{int}(s_{i+1}) = s_{i+2}...\delta_{int}(s_{j-1}) = s_j \tag{3.2}$$

On utilisera :

$$\delta_Z^* : S \times m \to S \tag{3.3}$$

tel que : $\exists \delta_Z^*(s_i, m) = s_j \in L(A)$ ssi $\big(\exists \delta_{int}^*(s_i) = s_j \wedge \exists \lambda^*(s_i) = m\big) \vee \delta_X^*(s_i, m) = s_j; \delta_X^*, \delta_{int}^*, \lambda^* \in A; s_i, s_j \in S$

On peut alors définir s'il existe un "chemin" m entre s_i et s_j dans l'automate.

Exemple : jeton ?jeton ? et choix2 ?service !fin ? font partie du langage de l'automate figure 3.6.

Définition 34. *Le **langage reconnu** par un automate A est constitué de l'ensemble des mots m tel que $\delta_Z^*(s_0, m) \in S$ est définie.*

$$L(A) = \{m \in \Sigma^* | \exists \delta_Z^*(s_0, m) \in A\}$$

Exemple : jeton ?jeton ? fait partie du langage reconnu par l'automate figure 3.6.

Définition 35. *Le **langage marqué de l'automate** est défini comme l'ensemble des mots m reconnus par un automate A provoquant une évolution à partir de l'état initial s_0 vers un état marqué S_{φ_i}* [8]*, noté $L_{mark\varphi_i}(A)$. On retrouve aussi l'appellation de "chaîne associée au chemin succès" en théorie des automates*

$$S_{\varphi_i} \in S_A \text{ l'ensemble des états qui satisfont la propriété } \varphi_i \text{ dans } A.$$

$$L_{mark\varphi_i}(A) = \{m \in (Z)^* | \delta_Z^*(s_0, m) \in S_{\varphi_i}\}$$

Cette définition a été adaptée à notre problématique, puisque le langage marqué ne cible plus ici les états finaux mais des états respectant une propriété ou un ensemble de propriétés ϕ. Nous discuterons plus en détails le principe d'état marqué dans le chapitre 4.

Exemple : le mot jeton ?choix1 ?service ! fait partie du langage marqué de l'automate figure 3.6.

3.4.2 Automates à interface pour l'IS et la M&S

L'exemple de la figure 3.6 peut également être représenté avec les automates à entrée-sortie. Ce sont les méthodes employées en Ingénierie Système et en M&S qui vont nous pousser à préférer les automates à interface et leurs évènements internes.

Les systèmes complexes sont développés de manière à constituer un ensemble de composants et de sous-composants qui seront intégrés pour former le système (voir chapitre 1). L'utilisateur de la simulation va alors s'intéresser, en fonction de l'avancement du projet et des objectifs d'expérimentation, à différents composants et sous-composants, faisant d'un composant parfois un élément de l'environnement, parfois un élément du modèle. Dans la figure 3.7, le modèle M2 se retrouve dans un premier temps dans l'environnement pour l'étude de M1, puis est "couplé" à M1 pour l'étude de M12.

Il est possible de connecter deux *automates* à interface pour créer un "couple" d'automates, permettant ainsi de retrouver les structures hiérarchiques classiquement utilisées dans l'IS. La définition donnée reprend celle proposée par B.P. Zeigler dans [Zeigler 1976] pour les automates DEVS, en l'adaptant aux automates à interface.

8. S_φ est dit marqué (ou état marqué) : ce mot est habituellement utilisé pour désigner un état final, mais il est ici utilisé pour "marquer" un ensemble d'états validant une propriété i notée φ_i

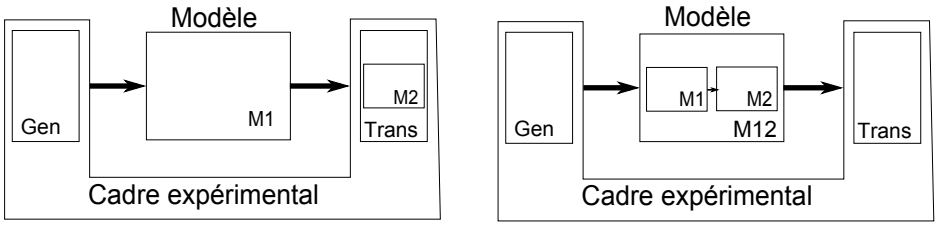


FIGURE 3.7 – Frontière cadre expérimental-modèle

Définition 36. *Un **automate couplé** est un 6-uplet tel que :*

$$C = \langle X, Y, D, C_{xx}, C_{xy}, C_{yy} \rangle$$

- X(res. Y) est un ensemble fini d'évènements d'entrées (resp. de sorties), tel que $X \cap Y = \varnothing$.
- $D = \{M_i\}$ est un ensemble fini de noms de sous-composants.
- $\{M_i\}$ est un ensemble de sous-composants. M_i peut être un composant atomique ou couplé.
- $C_{xx} \subseteq X \times \bigcup_{M_i \in D} X_i$ est l'ensemble des entrées du couplage.
- $C_{xy} \subseteq \bigcup_{M_i \in D} X_i \times \bigcup_{M_j \in D} Y_i$, est l'ensemble des relations internes du composant couplé.
- $C_{yy} = \bigcup_{M_i \in D} Y_i \rightarrow Y \cup \{\varepsilon\}$, est l'ensemble des sorties du composant couplé.

En reprenant notre exemple 3.6, il est possible de coupler cet automate avec un autre automate modélisant le système d'éjection de jeton. Celui-ci va, lorsqu'il reçoit un signal d'erreur, déclencher le signal d'éjection et valider l'éjection par un signal retourjeton (voir figure 3.8). Sans détailler l'automate "Ejection" (trivial), la définition du couplage serait alors :
$X = \{jeton, choix1, choix2, retour, fin\}$;
$Y = \{ejection, service\}$;
$D = \{"selection", "ejection"\}$;
$C_{xx} = \{ \big((Selectionavecjeton, jeton), (Selection, jeton)\big), ...\}$;
$C_{xy} = \{\big((Selection, erreur), (Ejection, erreur)\big), ...\}$;
$C_{yy} = \{\big((Ejection, ejection), (Selectionavecejection, ejection)\big), ...\}$

Peut-être du fait de son évidence, le couplage des automates à interface n'a pas été défini par L.Alfaro et T.Henzinger. Ceux-ci ont proposer l'élaboration d'un nouveau modèle à partir d'un couple d'automates, en utilisant le principe de composition d'automates. Au lieu d'une vision hiérarchique, nous sommes ici dans le cadre d'une représentation à plat. Cette vision n'est pas idéale pour la modélisation et la compréhension du système et de ses interactions. Elle trouve par contre un intérêt particulier pour la simulation et l'étude de la compatibilité entre cadre expérimental et modèle.

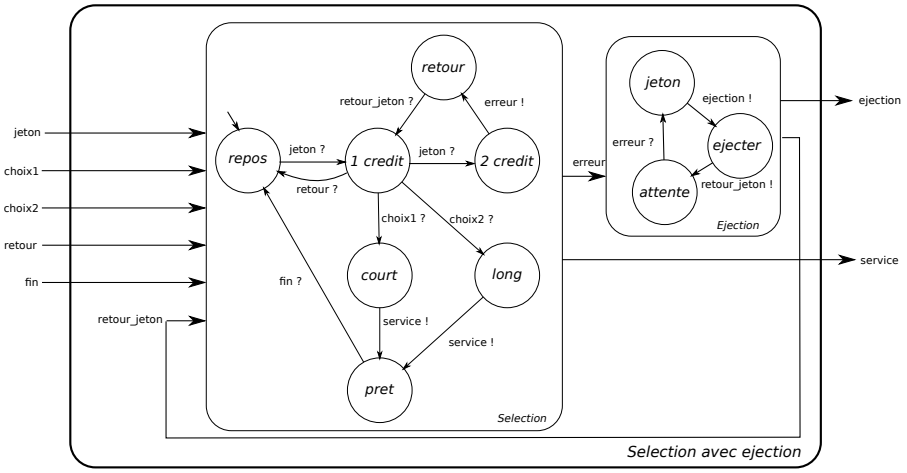


FIGURE 3.8 – Couple d'automate

L'opération de composition permet d'aborder de manière formelle l'étude de la compatibilité dynamique entre cadre expérimental et modèle. L'opération de composition permet d'obtenir un automate résultant, image de la simulation. Le langage de cet automate, appelé *langage de composition*, expose l'ensemble des traces d'exécutions possibles après application du cadre expérimental au modèle. Le langage marqué peut interpréter une hypothèse ou un objectif. La combinaison de l'étude du langage de composition, du langage reconnu par l'automate et du langage marqué va permettre de fournir à l'utilisateur des outils visant à améliorer sa confiance dans le produit de simulation.

3.4.2.1 Produit de composition

Il existe trois produits de composition pour les automates : *Synchrone*, *Asynchrone* et *Parallèle*. Tous trois concernent la composition de deux automates (A_1, A_2) qui permet d'obtenir un nouvel automate, appelé *automate résultant* (A_{12}).

- ***Composition synchrone*** ($\otimes$) : Le produit synchrone ne fait apparaître que les transitions communes aux deux automates. Seul l'ensemble appartenant à l'intersection des alphabets est pris en compte.
- ***Composition parallèle***($//$) : Comme le produit synchrone, cette composition, parfois appelée "produit cartésien", fait apparaître les transitions communes aux deux automates, mais elle permet également l'évolution des automates de façon individuelle sur l'ensemble qui n'appartient pas à l'intersection des alphabets.
- ***Composition asynchrone***($\circledast$) : Cette composition fournit un automate équivalent à celui obtenu par composition parallèle mais qui contient également les évolutions individuelles des deux automates pour les évènements qui appartiennent à l'intersection des alphabets.

Il résulte de ces trois opérateurs des automates différents. La figure 3.9 montre les automates résultants [9] de chaque opération de composition entre "A_1" et "A_2"($A_{1\otimes 2}$, $A_{1//2}$, $A_{1\circledast 2}$).

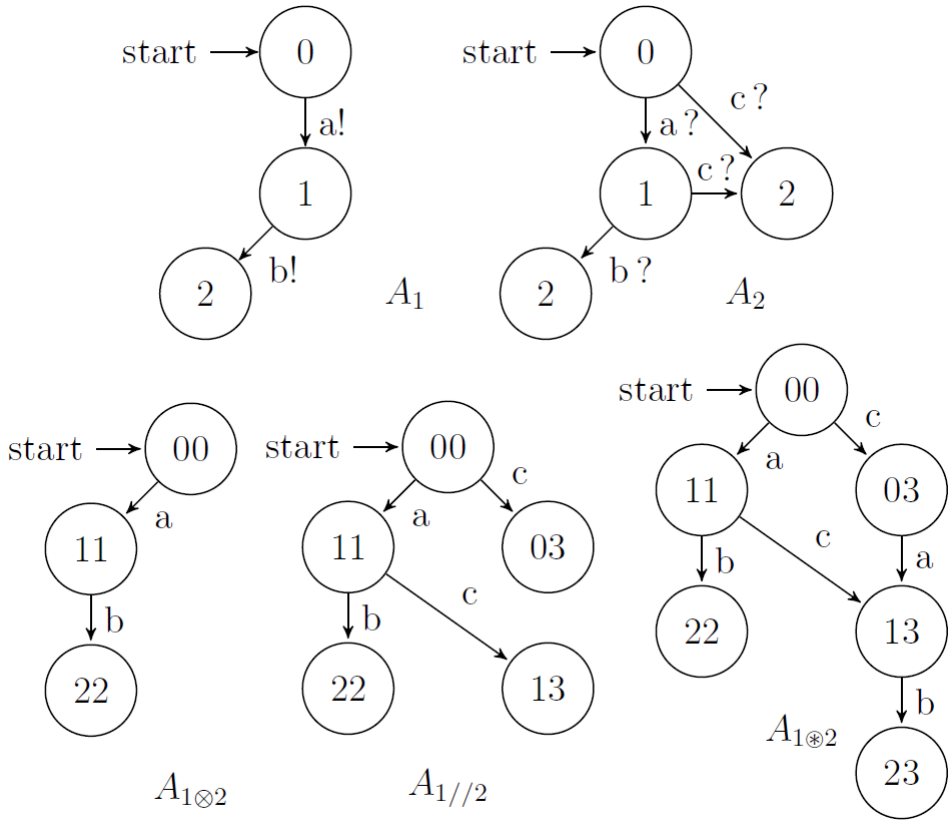


FIGURE 3.9 – Trois opérations de composition

La disparité de ces opérateurs de composition apporte des solutions différentes, mais certaines paraissent plus convaincantes que d'autres pour aider à répondre à notre problématique. Les conditions associées au produit synchrone n'autorisent pas l'automate à évoluer seul ("c" n'est pas pris en compte dans $A_{1\times 2}$ puisqu'il est hors alphabet commun). L'utilisateur de la simulation doit alors générer et observer l'ensemble des évènements permettant d'atteindre l'objectif de simulation. Nous trouvons cette contrainte trop importante pour l'utilisateur de la simulation. Le produit asynchrone est également écarté à ce stade de l'étude, puisqu'il voit les deux automates comme pouvant évoluer indépendamment l'un de l'autre, sans contrainte de communication. Le produit parallèle s'impose comme étant l'opérateur le plus

9. Si l'on se rapporte à la définition, l'opération de composition fait également apparaître un ensemble d'états non atteignables depuis l'état initial. Nous ne montrerons ici que les états atteignables depuis l'état initial.

intéressant, combinant un avancement synchrone entre les deux automates tout en permettant d'avancer de façon autonome lorsque le vocabulaire n'est pas commun. L'absence de vocabulaire commun entre deux automates peut provenir soit d'un choix de modélisation (utilisation d'évènements internes), ce qui est l'utilisation normale des automates à interfaces, soit d'un objectif d'expérimentation particulier, c'est à dire que l'utilisateur de la simulation ne juge pas nécessaire de contrôler (resp. observer) certaines entrées (resp. sorties) du modèle. C'est cette seconde utilisation qui va poser problème durant le produit de composition. Il est donc nécessaire de définir une restriction de l'automate afin de pouvoir réaliser un produit de composition cohérent avec l'objectif d'utilisation. Cette "restriction" d'automates que l'on appellera *bouchonnage*, va permettre de rendre interne une action, qui est à l'origine définie comme externe lors de la modélisation.

Dans notre approche, l'utilisateur de la simulation ne peut pas modifier directement le modèle qu'il utilise. Nous considérons à travers cette contrainte forte que l'utilisateur de la simulation ne peut effectuer que des actions externes sur le modèle. Ces actions peuvent avoir des répercussions sur la structure et le comportement interne du modèle.
L'exemple de la figure 3.10 montre l'effet du *bouchonnage* sur le modèle lorsqu'un utilisateur ne désire pas contrôler b, d et e. Une application directe de l'opérateur de composition parallèle entre cadre et modèle tel que proposé dans la figure 3.10 aurait pour effet un "blocage" après l'action "a", fournissant un automate à deux états (00 et 11). L'utilisateur ne cherche pas à contrôler "b, d, e" (pas d'intérêt à connaître le chemin emprunté entre $\{de\}$ et $\{e\}$). Nous effectuons une restriction permettant aux actions d'entrées de devenir des évènements internes, fournissant ainsi une composition cohérente avec les objectifs d'utilisation.

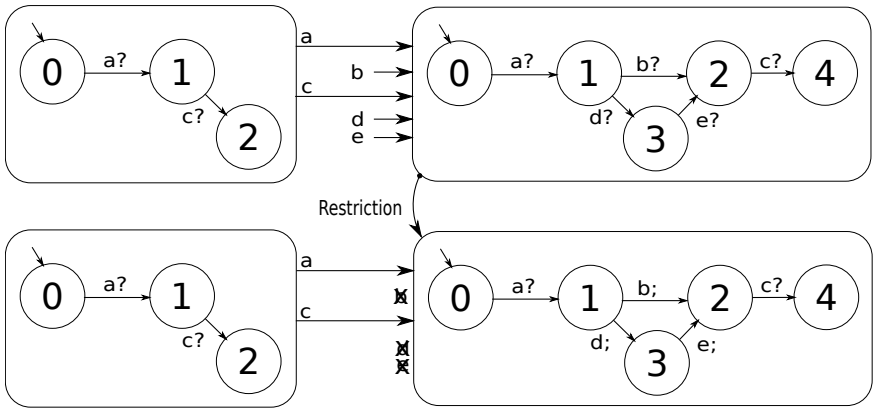


FIGURE 3.10 – Bouchonnage des entrées d'un automate à interface

Définition 37. *L'**Opération de restriction** permet de redéfinir comme interne un sous ensemble d'actions d'entrée ou d'actions de sortie non utilisées lors d'une*

expérimentation. Soit, la restriction d'un automate $A = \langle X, Y, S, s_0, \delta_x, \delta_{int}, \lambda \rangle$ *à un sous-ensemble d'entrées/sorties* $Z' = X' \cup Y'$ *tel que* $X' \subseteq X; Y' \subseteq Y$ *appartient à l'automate* $A' = \langle X', Y', S, s_0, \delta_{x'}, \delta'_{int}, \lambda_{y'} \rangle$ *avec :*

- $\delta'_{int} : S \to S$ *tel que* $\delta'_{int}(s) = \delta_x(s, x)$ *si* $x \in X \setminus X'$ *et* $(s, x) \in \delta_x$ *;* $\delta'_{int}(s) = \lambda(s)$ *si* $y \in Y \setminus Y'$ *et* $(s) \in \lambda$ *;* $\delta'_{int}(s) = \delta_{int}(s)$ *si* $(s) \in \delta_{int}$
- $\delta_{x'} : S \times X' \to S$ *avec* $\delta_{x'}(s, x') = \delta_x(s, x), \forall x' \in X'; (s, x) \in \delta_x$
- $\lambda' : S \to Y'$ *avec* $\lambda'(s) = \lambda(s), \forall y \in Y'; (s) \in \lambda$

Définition 38. *Le* ***langage de composition à deux automates*** *est défini comme l'ensemble des mots m reconnus par deux automates distincts A et B provoquant une évolution à partir de leur état initial respectif* s_{A_0} *et* s_{B_0}*, noté* $L_C(A//B)$*. La composition // n'est pas une opération commutative. Les langages de composition résultants de A//B et B//A sont donc différents :*

$$L_C(A//B) = \{m \in (\Sigma_A^* \cap \Sigma_B^*) \cup Z_A^* | \delta_{Z_A}^*(s_{A_0}, m) \in S_A\} \tag{3.4}$$

$$L_C(B//A) = \{m \in (\Sigma_B^* \cap \Sigma_A^*) \cup Z_B^* | \delta_{Z_B}^*(s_{B_0}, m) \in S_B\} \tag{3.5}$$

Si les automates ne comportent pas d'évènements internes $\delta_{int} = \emptyset$*, il est alors possible de simplifier ces équations :*

$$L_C(A//B) = L(A) \cap L(B) \tag{3.6}$$

avec

$$L_C(B//A)) = L_C(A//B) \tag{3.7}$$

Il devient simple de définir un langage de composition pour l'approche proposée, à savoir : générateur, modèle, transducteur. En reprenant les descriptions énonçant les principes du cadre expérimental données en section 2.3.2, il est possible de définir un générateur et un transducteur vus par le modèle :

Définition 39. *Un* ***générateur*** *est un 4-uplet tel que :*

$$G = \langle Y_G, S_G, s_0G, \delta_yG \rangle$$

Définition 40. *Un* ***modèle*** *est un 6-uplet tel que :*

$$M = \langle X_M, Y_M, S_M, s_0M, \delta_xM, \delta_yM \rangle$$

Définition 41. *Un* ***transducteur*** *est un 5-uplet tel que :*

$$T = \langle X_T, S_T, s_0T, \delta_xT, \delta_{intT} \rangle$$

avec $X, Y, S, s_0, \delta_x, \delta_{int}, delta_y$ tels que définis précédemment. Notons que cette définition diverge de celle habituellement utilisée en théorie des automates [Sakarovitch 2003].

Définition 42. *Le* ***langage de composition de M*** *pour une approche GMT peut être défini comme :*

$$L_C(G//M//T) = \{m \in (X_M \cap Y_G) \cup (Y_M \cap X_T) \cup Z_M | \delta_Z M(s_0 M, m) \in S_M\}$$

Définition 43. *Le* ***langage de composition de G*** *pour une approche GMT peut être défini comme :*

$$L_C(G//M) = \{m \in (X_M \cap Y_G) \cup Z_G | \delta_Z G(s_0 G, m) \in S_G\}$$

Définition 44. *Le* ***langage de composition de T*** *pour une approche GMT peut être défini comme :*

$$L_C(M//T) = \{m \in (Y_M \cap X_T) \cup ZT | \delta_Z T(s_0 T, m) \in S_T\}$$

A ce niveau de description, on observe une dissymétrie entre les définitions des langages de composition. Ceci est du à l'absence d'interaction entre le générateur et le transducteur.

3.4.3 Arbre de décomposition, composition et simulation

Comme nous l'avons vu en introduction de ce chapitre, il est nécessaire de connaître l'ensemble des comportements possibles du modèle pour étudier le niveau de compatibilité dynamique entre un modèle et un cadre expérimental. En théorie des graphes, il est possible d'effectuer une décomposition de l'automate, dite décomposition en arbre, qui permet de connaître tout ou partie du comportement de l'automate. Cette décomposition a été proposée par P. Seymour et N. Robertson dans leur série sur les mineurs des graphes [Robertson 1991]. Elle va nous permettre de répondre à plusieurs problématiques rencontrées lorsque l'on utilise un automate pour définir un produit de simulation. Un automate peut posséder des mots infinis. Si, par construction, une boucle existe dans l'automate, alors il possède une séquence infinie d'actions formant un ensemble de mot infinis. Dans l'exemple de la figure 3.11, l'automate permet de construire le mot "La voiture roule très* vite"

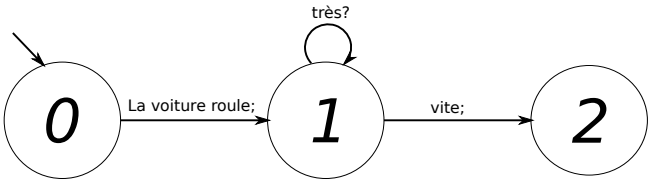


FIGURE 3.11 – Automate à mots infinis

Dans un contexte de simulation, plusieurs éléments vont s'opposer à ce caractère infini pouvant mener à l'explosion combinatoire, et donc s'opposer à un passage à l'échelle. L'opposition la plus basique vient de la notion bien connue des utilisateurs de la simulation et présentée au chapitre 2 : le temps de simulation.

L'application d'un cadre expérimental va également venir limiter l'exploration du modèle. Si nous reprenons l'exemple précédent (figure 3.11), l'application d'un cadre expérimental, ne générant qu'une action de sortie "très !", va fournir un automate résultant capable de nous donner un mot ou un préfixe du mot "La voiture roule très vite". Par contre, un cadre expérimental qui génère une boucle infinie d'actions de sortie "très !" permettra d'obtenir un automate résultant identique au modèle, et donc comportant des mots infinis, ce qui traduit une intention de simulation infinie. Si maintenant nous étudions la compatibilité dynamique entre ce cadre et ce modèle, il est possible de dire qu'ils sont compatibles puisque le modèle reçoit une action attendue "très". Cependant, le modèle est capable de recevoir une infinité de fois l'action "très". Comment définir alors le niveau de compatibilité entre le modèle et le cadre ? L'objectif est-il de n'envoyer qu'une seule et unique fois "très" ? Pour comprendre la difficulté de cette question il est nécessaire de garder à l'esprit que ces questions se posent dans un contexte d'étude de systèmes complexes. Afin de répondre de façon générique à cette question, nous allons utiliser la notion de profondeur d'exploration. Le comportement d'un modèle étant potentiellement infini, il est envisageable de le limiter par dépliage à une profondeur donnée. L'utilisateur va devoir proposer une profondeur maximale d'exploration, de la même façon qu'il fournit aujourd'hui le temps de simulation. Avec une base de temps "partiellement ordonnée", cela revient à établir le nombre maximal d'actions possibles depuis l'état initial de chaque modèle.

La simulation des systèmes complexes nécessite, la plupart du temps, l'envoi de types plus complexes au modèle que de simples évènements (utilisation de paramètres). Il y a donc, lors de la communication entre modèle et cadre expérimental, un échange de données [10]. Ceci augmente la combinatoire de façon significative. Selon le modèle sous test, la nécessité d'établir des hypothèses sur le modèle afin de limiter son exploration sera nécessaire, voire obligatoire. La notion d'hypothèse va prendre un rôle important dans notre approche et fera l'objet d'un développement plus détaillé dans la suite de ce document.

3.4.4 Proposition de métriques génériques d'évaluation pour les automates à E/S

Après avoir défini les différents langages d'intérêt obtenus grâce au couple cadre-expérimental/modèle, cette section propose une analyse de ces langages permettant d'établir le niveau de crédibilité qu'il est possible d'y associer.

Analyse du langage reconnu :

Une analyse du langage reconnu permet d'établir s'il y a respect des hypothèses statiques.

10. Comme cela a été présenté dans le chapitre 2, la communication n'est possible que si le "canal" de communication respecte un ensemble de contraintes (cf. compatibilité du périmètre).

Exemple : Jamais un événement x_i ou une suite d'événements m_j. Il est alors simple de vérifier qu'il n'existe pas de mot équivalent dans le langage reconnu de l'automate.

Soit $L_F(A)$ le langage qui décrit l'évènement x_i ou la suite d'évènement m_j non acceptable dans l'automate A, on définit :

$$L_{resp} = L_F(A) \cap L(A) \tag{3.8}$$

Si $L_{resp} = \{\varepsilon\}$ alors les hypothèses sont respectées, sinon L_{resp} représente l'ensemble des traces qui ne respecte pas la propriété.

L'analyse du langage reconnu seul est assimilable aux techniques de V&V actuelles. Son utilisation, associée à d'autres métriques, permet de détecter d'où provient la faute dans la simulation. Certaines méthodes d'analyse formelles, comme le model-checking, peuvent venir jouer un rôle important dans l'amélioration de la confiance de l'utilisateur.

Analyse du langage marqué :

Le langage marqué de l'automate permet d'établir quels sont les événements exigés pour atteindre l'ensemble des états respectant une hypothèse ou un objectif. Il est alors possible d'établir s'il y a incohérence entre hypothèses statiques, modèle et objectifs.

$$L_{inc} = L_F(A) \cap L_{mark}(A) \tag{3.9}$$

Si $L_{inc} = \{\varepsilon\}$ alors les hypothèses sont respectées sinon L_{inc} représente l'ensemble des incohérences ne permettant pas d'accéder *a priori* aux propriétés dans le respect des hypothèses statiques.

Analyse du langage de composition :

L'analyse du langage de composition, au même titre que celle du langage reconnu par l'automate, permet d'établir s'il y a respect des hypothèses sur événements. Cette analyse permet de s'intéresser en particulier aux hypothèses définies dans le cadre expérimental (générateur, transducteur).

Il existe certains cas particuliers dans l'analyse du langage de composition. Il est important de rappeler ici au lecteur la discussion 2.5.2. Il est donc primordial de lire l'interprétation de ces cas avec prudence. Même si toutes les expérimentations n'entreront pas dans ces cas particuliers, il est intéressant de les énumérer car elles peuvent améliorer de façon significative la confiance de l'utilisateur :

Soit un modèle M :

— si $L_C(G//M//T) = L(M)$, alors le modèle est totalement exploré. L'utilisateur de la simulation a défini un cadre expérimental permettant d'explorer tout le modèle. Les contraintes amenant à un tel résultat prêtent à penser que le cadre expérimental est bien construit. La confiance de l'utilisateur de la simulation peut augmenter.

— si $L_C(G//M//T) = Pref\big(L_{mark}(M)\big)$, alors l'excitation du modèle est dite minimale. Le cadre expérimental va explorer le comportement nécessaire et suffisant du modèle pour remplir les objectifs d'expérimentation. Cela sous-entend une bonne connaissance du modèle. Cela prête à penser que le cadre expérimental est bien construit. La confiance de l'utilisateur de la simulation peut augmenter.

— si $L_C(G//M//T) = \varepsilon$, il apparaît que le modèle n'a pas été exploré. Le cadre expérimental et le modèle ont une compatibilité nulle. La confiance de l'utilisateur de la simulation doit être nulle (sauf volonté de l'utilisateur)

— si $L_{mark}(M) \subset L_C(G//M//T)$, l'ensemble des états marqués a été visité. Si l'exploration n'a pas été minimale, elle a permis d'explorer l'ensemble des états "objectifs". Cela prête à penser que le cadre expérimental est bien construit. La confiance de l'utilisateur de la simulation peut augmenter.

S'il est naturel d'étudier l'exploration du modèle, la source d'information contenue dans le langage de composition du générateur ou du transducteur ne doit pas être négligée.

Soit un générateur G :

— si $L_C(G//M) = L(G)$, alors le générateur est totalement exploré. Le cadre expérimental proposé a une compatibilité totale avec le modèle. Le cadre expérimental est cohérent pour le modèle proposé. L'utilisateur de la simulation peut augmenter sa confiance dans son produit de simulation.

— si $L_C(G//M) = Pref\big(L_{mark}(M)_{/X_M}\big)$, alors le générateur est dit minimal. Cette propriété est équivalente à celle énoncée pour le modèle M.

— si $L_C(G//M) = \varepsilon$, il apparaît que le générateur n'a pas pu stimuler le modèle. Cette propriété est équivalente à celle énoncée pour le modèle M.

Soit un transducteur T :

— si $L_C(M//T) = L(T)$, alors le transducteur est totalement exploré. Le cadre expérimental proposé a une compatibilité totale avec le modèle. Le cadre expérimental est cohérent pour le modèle proposé. L'utilisateur de la simulation peut augmenter sa confiance dans son produit de simulation.

— si $L_C(M//T) = Pref\big(L_{mark}(M)_{/Y_M}\big)$, alors le transducteur est dit minimal. Cette propriété est équivalente à celle énoncée pour le modèle M.

— si $L_C(M//T) = \varepsilon$, il apparaît que le transducteur n'a pas pu être stimuler par le modèle. Cette propriété est équivalente à celle énoncée pour le modèle M.

Cet ensemble de métriques propose une approche générique pour évaluer la compatibilité dynamique entre le modèle et le cadre expérimental. Une étude plus détaillée des ensembles résultants permettra à l'utilisateur de la simulation d'établir les mots menant à l'incompatibilité dynamique entre le modèle et le cadre expérimental.

3.4.4.1 Analyse quantitative de la compatibilité

Il est également possible d'établir des quantifications. Les taux sont un moyen simple de donner à l'utilisateur de la simulation une mesure chiffrée de la compatibilité entre le cadre expérimental et le modèle.

Définition 45. *Le* ***taux d'exploration de l'automate*** *(noté A) par un cadre expérimental donné est défini tel que :*

$$L_{\overline{C}}(A) \times L_C(A) \equiv L(A) \tag{3.10}$$

$$\alpha_{ex} = \frac{|L_C(A)|}{|L(A)|} \tag{3.11}$$

Il semble cependant difficile d'attribuer une confiance forte à un taux d'exploration. Le taux d'exploration est une mesure extrêmement sensible. Une modification minime dans le couple cadre/modèle peut donner un changement du taux d'exploration. Le couplage "1)" (figure 3.12)nous permet d'obtenir une compatibilité totale. Si, suite à une erreur, l'action de sortie "b!" est définie en lieu et place de "a!" (couplage "2)", figure 3.12), le taux d'exploration sera alors nul.

A travers cet exemple extrême, où l'ensemble du modèle est correct à une exception près, apparaissent les problèmes liés à l'utilisation de la mesure d'exploration du modèle. C'est la position des erreurs, plus que leur nombre, ou même leur gravité qui influence le taux d'exploration. Cette mesure reste pourtant un indicateur à ne pas négliger, puisqu'elle reflète une réalité, celle de la simulation.

Pour apporter une solution à ce type de résultat, et toujours dans le but de répondre à notre problématique, il est envisageable de s'intéresser à l'ensemble du produit de composition et non plus à la partie atteignable depuis l'état initial. Cela permet une vision globale de l'opération de composition, pouvant faire apparaître une partie non-atteignable depuis l'état initial, mais possédant une bonne compatibilité modèle/cadre expérimental. Dans notre exemple, il serait intéressant de détecter les n états compatibles par le mot m, même si ces états ne sont pas accessibles depuis l'état initial.

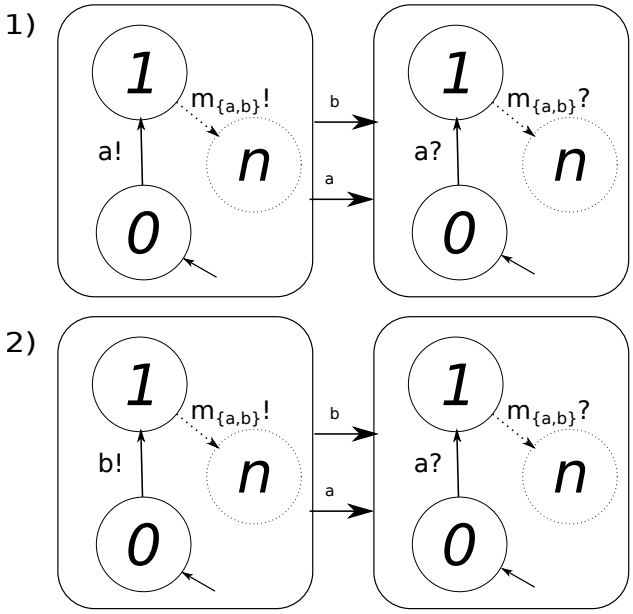


FIGURE 3.12 – Le taux d'exploration est très sensible à l'emplacement de l'erreur.

Dans notre contexte, la comparaison du taux d'exploration du modèle avec celui du cadre expérimental peut également apporter des informations et donc améliorer la confiance de l'utilisateur.

Définition 46. ***Taux d'exploration des états marqués de l'automate :*** *il existe un ensemble d'états marqués dans l'automate respectant une propriété ou un ensemble de propriétés φ défini par l'utilisateur directement ou résultant d'un regroupement d'informations défini par l'utilisateur.*

En utilisant la définition 35 pour exprimer L_{mark_φ}, on a :

$$\alpha_{ex_M} = \frac{|L_{mark}(A) \cap L_C(A)|}{|L_{mark}(A)|} \tag{3.12}$$

Définition 47. ***Taux d'exploration vers les états marqués de l'automate***

$$\alpha_{ex_T} = \frac{|Pref\big(L_{mark}(A)\big) \cap L_C(A)|}{|Pref\big(L_{mark}(A)\big)|} \tag{3.13}$$

Voici un exemple permettant d'illustrer le principe d'exploration, de composition et d'analyse des langages. Un chapitre complet sera dédié à l'illustration de la méthodologie au travers d'un exemple concret, c'est pourquoi nous faisons ici le choix d'un exemple abstrait permettant de se focaliser sur la méthode seulement. La figure 3.13 propose une représentation de la décomposition de deux automates, EF et Model (phase d'exploration). EF représente un générateur minimaliste capable de générer trois évènements a,b ou c (le modèle *EF* et l'arbre de décomposition *EF* sont isomorphes). Le modèle *MODEL*, quant à lui, est capable de recevoir les

évènements a,b,c,d,z.

Sans que cela ne soit clairement abordé, les modèles peuvent également être étiquetés par un ensemble de gardes et d'actions qui sera interprété durant l'exploration de l'automate (dans l'exemple : variable A). Notons également que l'utilisateur de la simulation ajoute une hypothèse statique : l'évènement z ne se produira jamais.

Avant d'effectuer le produit de composition entre *EF* et *MODEL*, il est possible d'étudier la compatibilité du périmètre et d'effectuer l'opération de restriction définie précédemment. L'hypothèse statique proposée annonce que l'évènement z n'arrivera jamais. Cet évènement ne sera donc pas une entrée du modèle *MODEL*. L'opération de restriction définie va alors "bouchonner" le sous-ensemble d'évènements $\{d\}$ qui va devenir un sous-ensemble d'évènements interne (disponibilité immédiate lors de la simulation). Il est alors possible d'effectuer l'opération de composition entre *EF* et *MODEL*.

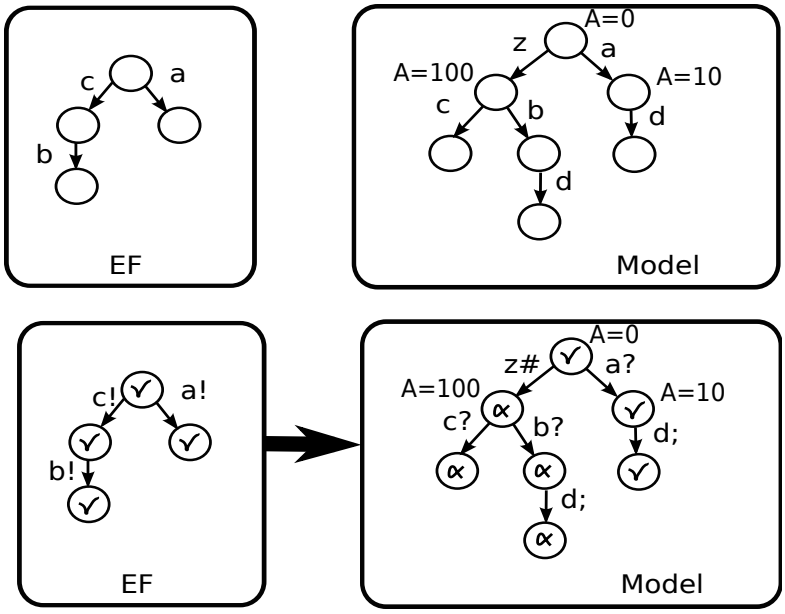


FIGURE 3.13 – Représentation graphique d'un automate et composition

Une étude comparative peut alors être effectuée entre le graphe résultant et les modèles afin d'évaluer la validité du modèle de simulation. L'étude des langages peut nous apporter des informations sur l'exploration du modèle *MODEL*.

— $L(Model) = \varepsilon, z, a, zc, zb, ad, zbd \Rightarrow |L(Model)| = 6$
— $L_C(Model) = \varepsilon, a, ad \Rightarrow |L_C(Model)| = 2$

$$\boxed{\alpha_{ex} = \frac{1}{3}}$$

Une conclusion hâtive annoncerait une mauvaise compatibilité, puisque seulement un tiers de *MODEL* est exploré. Si l'utilisateur de la simulation prend en compte les hypothèses statiques (jamais *z)*, il n'est alors plus nécessaire d'étudier les mots composés de l'élément *z*. Le taux d'exploration est alors de 100%. S'il reste évident qu'il est impossible [11] de conclure sur la validité de la simulation, cette métrique va aider à améliorer la confiance de l'utilisateur. Une étude comparable sur le générateur (modèle *EF*) affichant une compatibilité totale ira également dans ce sens.

Établir une corrélation directe entre la validité d'un modèle de simulation et le taux d'exploration n'est pas envisageable, puisqu'il est possible que l'utilisateur de la simulation s'attende à un taux d'exploration donné (autre que 100%).

— $L(Model)_{\setminus\{z\}} = \varepsilon, a, ad \Rightarrow |L(Model)_{\setminus\{z\}}| = 2$
— $L_C(Model) = \varepsilon, a, ad \Rightarrow |L_C(Model)| = 2$

$$\boxed{\alpha_{ex} = 1}$$

Cet exemple montre que chaque métrique prise indépendamment ne nous permet pas de juger de la compatibilité du modèle avec son cadre expérimental. C'est donc l'analyse conjointe de différentes métriques qui nous permet de juger de la cohésion de l'ensemble afin d'établir un degré de compatibilité entre cadre et modèle. Le regroupement de ces informations est essentiel, il est possible de l'effectuer manuellement, mais pour être efficace, et s'intégrer dans une démarche d'ingénierie système, ce regroupement doit bénéficier d'un outil. Nous avons pu observer les prémisses d'une méthodologie : définir l'objectif de l'expérimentation, définir des hypothèses statiques, développer le cadre expérimental,... Nous reviendrons dans le chapitre suivant sur la nécessité de proposer un outil adapté, associé à une méthodologie. La fin de ce chapitre (voir 3.6.4) consacre également une discussion sur les perspectives de regroupement des différentes métriques.

Malheureusement, cette approche montre ses limites lors du passage à l'échelle, avec une explosion combinatoire inévitable. Cette première étude nous a permis d'établir les bases pour proposer une méthodologie d'évaluation.

3.5 Étude de la compatibilité : Approche automates *DEVS*

Cette section s'intéresse à un formalisme développé dans les années 70 par B.P. Zeigler [Zeigler 1976], Discret Event System Specification (*DEVS*), considéré comme un formalisme dédié à la modélisation, la simulation et l'analyse des systèmes complexes. *DEVS* permet de décrire les systèmes discrets (transition d'état), continus (équation différentielle) et hybrides (continus et discrets). Reconnu pour

11. La taille de l'exemple peut faire paraître cette affirmation totalement absurde. Cette réflexion est générique et s'appuie sur les discussions autour de l'intérêt à porter sur les résultats fournis par une simulation(voir2.5.2)

son expressivité et son universalité, il est utilisé avec succès depuis plusieurs années dans de nombreux projets pour l'étude des systèmes complexes ([Ramat 2003], [Filippi 2004], [Chabrier 2007], [Bisgambiglia 2013]).

Preuve de l'intérêt de ce formalisme, de nombreuses extensions de *DEVS* sont apparues comme *Parallel DEVS* [Chow 1994], *RealTime DEVS* [Cho 1998], *Cell-DEVS* [Wainer 2001] ou encore *Fuzzy-DEVS* [Kwon 1996]. Dans le même temps, sont apparues plusieurs sous-classes de *DEVS* comme *FD-DEVS*, pour "*finite and deterministique DEVS*" ou encore "*Schedule-Preserving DEVS*". La vision modulaire et hiérarchique de *DEVS* en fait un formalisme adéquat pour une approche ingénierie système. FD-DEVS a été développé pour permettre à la fois la vérification formelle et la simulation des systèmes, ce qui fait de ce formalisme un excellent candidat pour l'étude de la compatibilité cadre expérimental/modèle.
Cette seconde approche a été motivée par la prise en compte du temps explicite et non plus ordonné, permettant de s'approcher des principes de simulation classiques tout en gardant l'avantage de l'approche évènementielle.

3.5.1 DEVS

Nous rappelons ici le formalisme DEVS, et plus particulièrement la restriction FD-DEVS telle que définie dans [Zeigler 2000].

FD-DEVS

Définition 48. *Un composant atomique FD-DEVS est un 7-uplet :*

$$A = \langle X, Y, S, s_0, \tau, \delta_x, \delta_y \rangle \text{ où :}$$

- X est un ensemble fini d'entrées, X peut être divisé en deux variables : $X = \{(p, v) |\ p \in Iports,\ v \in X_p\}$;
- Y est un ensemble fini de sorties, Y peut être divisé en deux variables : $Y = \{(p, v) |\ p \in Oports,\ v \in Y_p\}$;
- S est un ensemble fini d'états, $s_0 \in S$ est l'état initial ;
- $\tau : S \to \mathbb{Q}_{[0,\infty]}$ est la *fonction d'avancement du temps*, où $\mathbb{Q}_{[0,\infty]}$ est un ensemble de nombres rationnels non-négatifs plus l'infini. Cette fonction est utilisée pour déterminer la durée de vie de l'état ;
- $\delta_x : S \times X \to S \times \{0, 1\}$ est la *fonction de transition d'état externe*. Elle définit comment un évènement d'entrée change l'état, et si la fonction d'échéance interne du temps est mise à jour ou non. L'échéance interne d'un état $s \in S$ est mise à jour par $\tau(s')$ si $\delta_x(s) = (s', 1)$, sinon (i.e., $\delta_x(s) = (s', 0)$), l'échéance est préservée ;
- $\delta_y : S \to Y^\phi \times S$ est la *fonction de transition interne / sortie*, où $Y^\phi = Y \cup \{\phi\}$ et $\phi \notin Y$ pour le *silent event* ;[12]

12. δ_y peut être divisée en deux fonctions : la fonction de sortie $\lambda : S \to Y$ et la fonction de transition interne $\delta_{int} : S \to S$

Un modèle FD-DEVS a une base de temps explicite, au contraire de DEVS. La *base de temps*, notée $\mathbb{T}$, est un ensemble de nombres réels non-négatifs, i.e. $\mathbb{T} = [0, \infty)$.

$t_e \in \mathbb{T}$ est le *temps écoulé*. Il croît de façon continue et sa valeur reflète le temps passé depuis $t_e = 0$. $t_s \in \mathbb{Q}_{[0,\infty]}$ est une autre variable d'état interne, appelé *lifespan* or *schedule time span*.

$t \in \mathbb{T} \cup \{\infty\}$ est considérée comme la limite haute de t_e.

Le domaine existant de t_e est défini par la fonction $tr : \mathbb{T} \cup \{\infty\} \to 2^{\mathbb{T}}$ s.t. $tr(t) = [0, t]$ if $t < \infty; tr(t) = [0, \infty)$ si $t = \infty$. $Q_p = \{(s, t_s, t_e) |\ s \in S,\ t_s \in \mathbb{Q}_{[0,\infty]},\ t_e \in tr(t_s)\}$ est un ensemble d'états autorisé. $Q_{imp} = \{(imp, \infty, t_e) |\ imp \notin S,\ t_e \in \mathbb{T}\}$ est un ensemble non-autorisé d'états s.t. $Q_p \cap Q_{imp} = \varnothing$. Finalement, l'ensemble état total Q est défini comme $Q = Q_p \cup Q_{imp}$. $Z = X \cup Y^{\phi}$ est l' *ensemble d'évènements totaux* de M.

3.5.1.1 FD-DEVS *Network*

De la même façon que nous avions présenté le couplage d'automate à interface (cf. section 3.4.2), il est possible de coupler les composants FD-DEVS.

Définition 49. *Un composant FD-DEVS network (aussi appelé modèle couplé FD-DEVS) est un 6-uplet :*

$$C = \langle X, Y, D, \{M_1\}, C_x, C_y \rangle \text{ où :}$$

- X est un ensemble fini d'évènements d'entrée ;
- Y est un ensemble fini d'évènements de sortie, tel que $X \cap Y = \varnothing$;
- D est un ensemble fini de noms de sous-composants ;
- $\{M_1\}$ est un ensemble d'index de modèles FD-DEVS, où $i \in D$. M_i peut être un modèle FD-DEVS atomique ou couplé ;
- $C_x \subseteq X \times \bigcup_{i \in D} X_i$ est l'ensemble des entrées de couplage, où X_i est l'ensemble des évènements d'entrée du sous-composant $i \in D$;
- $C_y \subseteq X \times \bigcup_{i \in D} Y_i \times (\bigcup_{j \in D} X_j \cup Y)$, où $i \neq j$ est l'ensemble des sorties du composant couplé, où Y_i est l'ensemble des évènements de sortie du sous-composant $i \in D$.

Si nous voulions le comparer à DEVS, FD-DEVS est restreint par trois hypothèses :

- Le temps passé dans l'état est représenté par un nombre rationnel ou par l'infini ;
- L'échéance interne peut être préservée ou mise à jour par un évènement d'entrée ;
- L'ensemble des évènements et des états est fini.

Ce formalisme n'est pas sans rappeler les "timed automata", proposés par R. Alur [Alur 1990] et utilisés pour la modélisation des systèmes temps réel. Les "timed

automata" sont, de manière générale, plus utilisés que DEVS. N. Giambiasi nous propose dans, *Simulation and verification II : from timed automata to DEVS models* [Giambiasi 2003], une comparaison entre DEVS et timed automata pour expliquer pourquoi les "timed automata" sont plus utilisés que DEVS dans les domaines de la vérification logicielle. L'auteur définit DEVS comme plus adapté à la description des modèles de bas-niveau et donc à l'analyse de systèmes réels, alors que les timed automata sont plus adaptés aux spécifications de haut niveau comme l'exige la spécification logicielle. Toujours selon l'auteur, les timed automata peuvent être utilisés comme un moyen efficace de spécification de haut niveau pour les modèles dédiés à la simulation. Après avoir établi un mapping entre DEVS et timed automata, l'auteur conclut qu'il y a équivalence entre un timed automata déterministe et un modèle DEVS atomique. Ces deux formalismes sont donc proches. C'est sans surprise que DEVS reste le formalisme le plus adapté à la description des modèles de simulation, puisqu'il fut proposé pour cette utilisation.

Malgré ces nombreux succès et l'intérêt qu'il suscite ces dernières années, on constate que le formalisme FD-DEVS, ou même plus généralement DEVS est peu connu. Il est intéressant de présenter ici un exemple que l'on retrouve régulièrement dans la littérature mais qui possède l'avantage d'être très intuitif : l'exemple du Ping-Pong .

L'exemple du Ping-Pong (figure 3.14 [13]) modélise un système autonome de deux joueurs de ping-pong se renvoyant la balle infiniment souvent. Le modèle couplé "$Ping - Pong$" est composé de deux modèles atomiques A et B .

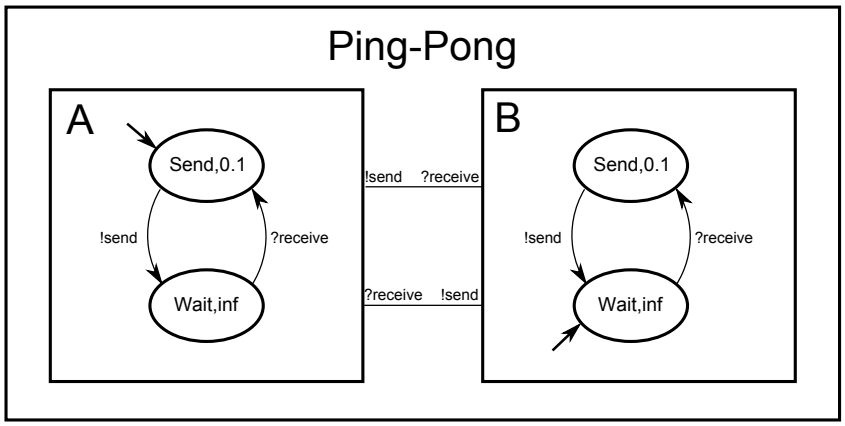


FIGURE 3.14 – L'exemple du Ping-Pong

13. A noter que la syntaxe graphique visible sur la figure 3.14 est utilisée dans la littérature pour illustrer certains exemples simples. Cependant, cette représentation n'a jamais été officiellement adoptée, car semble t-il, trop restrictive pour exprimer toute la sémantique DEVS.

Soit formellement, $A = \langle X, Y, S, s_0, ta, \delta_{ext}, \delta_{int}, \lambda \rangle$ avec
$X = \{?receive\}$;
$Y = \{!send\}$;
$S = \{Send, Wait\}; s_0 = Send$;
$\tau(Send) = 0.1, \tau(Wait) = \infty$;
$\delta_x(Wait, ?receive) = (Send, 1), \delta_x(Send, ?receive) = (Send, 0)$;
$\delta_y(Send) = (!send, Wait), \delta_y(Wait) = (\varnothing, Wait)$.

Le joueur B est modélisé de façon identique, seul l'état initial diffère : $s_0 = Wait$.
Le couplage Ping-Pong est donné par $Ping - Pong = \langle X, Y, D, \{M_i\}, C_{xx}, C_{xy}, C_{yy} \rangle$ avec :
$X = \{\}$;
$Y = \{\}$;
$D = A, B; M_A = A, M_B = B$;
$C_{xx} = \{\}$;
$C_{xy} = \{(A.!send, B.?receive), (B.!send, A.?receive)\}$;
$C_{yy}(A.!send) = \varnothing, C_{yy}(B.!send) = \varnothing$.

Nous retrouvons donc à travers ce formalisme une structure cohérente avec les automates à interface mais proposant une base de temps continue plutôt qu'une base de temps partiellement ordonnée. L'évènement est maintenant un évènement temporel.

3.5.1.2 Évènement temporel et comportement

Pour définir une séquence de changement d'états associée à des évènements, nous avons besoin d'introduire un *évènement temporel* et la séquence associée.

Un *évènement temporel* est une paire, avec un évènement $z \in Z$ est son apparition dans le temps $t \in \mathbb{T}$. Il est noté (z, t).

La *concaténation* de deux évènements (z_1, t_1) et (z_2, t_2) est décrite par $(z_1, t_1)(z_2, t_2)$, et peut être définie si $t_1 \leq t_2$.

L'opération identité de concaténation est l'évènement nul, noté ε. Une séquence d'évènements nuls dans un intervalle de temps $[t_l, t_u] \subseteq \mathbb{T}$ est notée $\varepsilon_{[t_l,t_u]}$.

Soit un ensemble d'évènements Z et un intervalle de temps l $[t_l, t_u] \subseteq \mathbb{T}$, l'ensemble de séquences d'évènement total est noté $\Omega_{[t_l,t_u]}$, et est défini par $\Omega_{[t_l,t_u]} = \{(z,t)^* | z \in z \cup \{\varepsilon\}, t \in [t_l, t_u]\}$ qui est l'ensemble de concaténations des évènements temporels finis ou infinis (plus $\varepsilon_{[t_l,t_u]}$) sur Z et $[t_l, t_u]$).
Soit $\Omega_{[t_l,t_u]}, \omega = (z_1, t_1)(z_2, t_2) \in \Omega_{[t_l,t_u]}$ où $t_l \leq t_1 \leq t_2 \leq t_u$ est équivalent à $\omega = \varepsilon_{[t_l,t_1]}(z_1, t_1)\varepsilon_{[t_1,t_2]}(z_2, t_2)\varepsilon_{[t_2,t_u]}$.

Les auteurs définissent également la fonction de trajectoire d'état C par utilisation de la fonction $\Delta : Q \times \Omega_{[t_l,t_u]} \to Q$. Soit $q = (..., (s_i, t_{si}, t_{ei}), ...) \in Q$, un état total au temps t_l. L'état trajectoire associé à une séquence d'évènements multiples peut être calculé par l'application de séquences de "nul-ou-un-évènement", répétitivement. Basé sur cette fonction de trajectoire d'état, le comportement de C est défini comme étant l'ensemble des possibilités de séquences d'évènements avec lesquelles l'état de C n'entre pas dans un état non autorisé. Formellement, le comportement ou langage de C au travers d'une observation de longueur finie $t \in \mathbb{T}$, notée $L(C, t)$, est

$$L(C, t) = \{\omega \in \Omega_{[0,t]} | \Delta(q_0, \omega) \in Q_p\}. \tag{3.14}$$

L'observation comportementale infinie ou langage de C, noté $L(C)$, est

$$L(C) = \{\omega \in \Omega_{[0,\infty)} | \Delta(q_0, \omega) \in Q_p\}. \tag{3.15}$$

Tout comme nous l'avons définie pour les automates à interface, il est possible de définir l'opération de restriction pour le couplage d'automates DEVS. Cette restriction a été proposée dans [Foures 2012].

Soit, la restriction d'un composant FD-DEVS $A = \langle X, Y, S, s_0, \tau, \delta_x, \delta_{int}, \lambda \rangle$ à un sous-ensemble d'événements $Z' = X' \cup Y'$ tel que $X' \subseteq X, Y' \subseteq Y$ est un composant FD-DEVS $A' = \langle X', Y', S, s_0, \tau, \delta_{x/x'}, \delta_{int}, \lambda_{/y'} \rangle$ avec :

- $\delta_{int} : S \to S$ est la *fonction de transition d'état interne* telle que pour tout δ_{int} définie dans A, elle est aussi définie dans A' ;
- $\delta_{x/x'} : S \times X' \cup \{\varepsilon\} \to S$ est la *fonction de transition d'état externe*, où ε est l'ensemble des évènements non observables, tel que :
  - pour tout $x \in X'$, pour tout $q \in Q, s \in S$, si $(q, x, s) \in \delta_x$ alors $(q, x, s) \in \delta_{x/x'}$
  - pour tout $x \in X' \setminus X$, pour tout $q \in Q, s \in S$, si $(q, x, s) \in \delta_x$ alors $(q, \varepsilon, s) \in \delta_{x/x'}$

  La restriction de la fonction de transition d'état externe masque les évènements qui ne sont pas dans X' avec les évènements non observables ε.
- $\lambda_{/y'} : S \to Y'$ est la *fonction de sortie* telle que $\lambda_{/y'}(s) = \lambda(s)$ pour tout $y \in Y'$.
  La restriction de la fonction de sortie garde la fonction de sortie telle que définie pour un sous-ensemble d'évènements et met de côté la fonction de sortie de tout autre évènement.

3.5.2 Graphe de classes

DEVS a été décrit par T. O'Neill [ONeill 1998] comme "*un formalisme capable de décrire élégamment des changements continus dans une structure de description discrète*". FD-DEVS, tout comme DEVS permet en effet de modéliser des systèmes

continus, si son espace d'états S est fini. La fonction d'avancement du temps autorise le passage d'un état à un autre dans $\mathbb{Q}_{[0,\infty]}$. Une méthode naïve cherchant à explorer chaque état est donc impossible, puisqu'il existe un nombre infini de transitions. En effet, l'exploration de tous les états, appelés "ensemble des états totaux", est définie par $Q = \{(s,t) \mid s \in S \wedge 0 \leqslant t \leqslant t_a(s)\}$. La solution est apportée par M.H. Hwang [Hwang 2009], qui utilise la notion de *time zone*, définie par D. Dill [Dill 1990], pour abstraire le temps continu dans les automates temporisés [Alur 1994]. M.H. Hwang adapte la solution de D. Dill aux modèles FD-DEVS pour construire le graphe d'atteignabilité (*reachability graph*) des modèles FD-DEVS. D'un point de vue comportemental, ce graphe est isomorphe à un modèle couplé FD-DEVS mais a l'avantage de posséder un nombre fini d'états et de transitions. Pour obtenir un ensemble fini à partir de l'espace infini du temps écoulé, l'auteur utilise une conjonction d'inégalité du temps écoulé pour créer des classes d'équivalence. En se basant sur les "time zones", l'auteur propose un algorithme permettant d'obtenir le graphe d'atteignabilité, parfois appelé graphe de classes.
La restriction au sous-ensemble FD-DEVS est conduit par la nécessité d'avoir un ensemble d'états et d'évènements fini et l'expression de la durée de vie de l'état par des nombres rationnels (+ infini).

Définition 50. *Le **graphe d'atteignabilité** d'un FD-DEVS est un 4-uplet :*

$$RG(M) = \langle Z, V, v_0, E \rangle, \text{ où}$$

- M est un FD-DEVS couplé, tel que défini précédemment ;
- $Z = X \cup \bigcup_{i \in D} Y_i^{\phi}$ est l'ensemble des évènements de déclenchement ;
- V est un ensemble de zones. Une zone $v = ((..., (s_i, t_{s_i}), ...), \varphi)$ est une paire, composée d'une échéance de l'état et d'une zone temporelle φ (time zone) ;
- $v_0 = ((..., (s_{0i}, t_i(s_{0i})), ...), \varphi_0) \in V$ est la zone initiale,
  où $\varphi_0 = Successeur(\varphi, R, (s,t)))$, avec comme zone temporelle initiale $\varphi = \bigwedge_{i \in D}[0,0] \wedge \bigwedge_{i,j \in D, i \neq j}[0,0], R = \varnothing, (s,t) = (..., (s_{0i}, t_i(s_{0i})), ...)$;
- $E \subseteq V \times Z \times 2^D \times V$ est la relation de transition. L'arc orienté $(v, z, R, v') \in E$ représente le fait que si un évènement z se produit, il change l'échéance d'état de v vers v' et réinitialise t_{ei} pour tout $i \in R$.

Afin d'illustrer le principe de graphe de classes, nous allons nous intéresser à un exemple introduit par M.H. Hwang et B.P. Zeigler dans [Hwang 2009], l'exemple du "grille-pain". Nous retrouvons sur la figure 3.15 une illustration (a) de l'élément à modéliser. C'est un grille-pain à deux fentes, une fente permet de griller pendant 20 secondes et une autre pendant 40 secondes. Nous retrouvons en (b) un modèle FD-DEVS couplé composé de deux modèles FD-DEVS atomiques représentant le fonctionnement du grille-pain dans une approche systémique. Il s'agit de deux toasters simples (T1 grille pendant 20s et T2 grille pendant 40s) à deux états **I** pour "idle" et **T** pour "toasting". Les deux toasters sont indépendants (absence de communication) et sont tous les deux pilotés de façon autonome. Nous passerons sur la définition formelle de cet exemple qui ne pose aucune difficulté particulière compte tenu des exemples précédents.

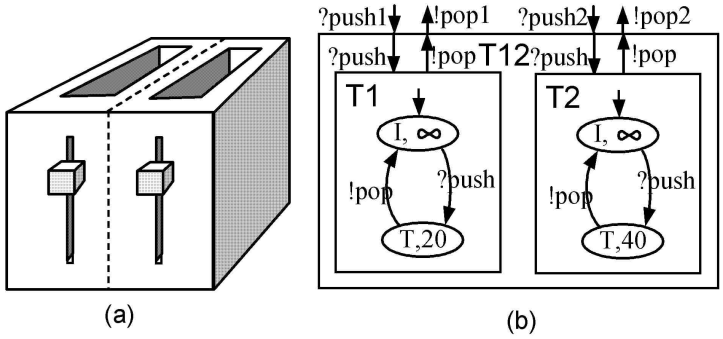


FIGURE 3.15 – L'exemple du toaster [Hwang 2009]

Pour étudier la capabilité du modèle, il est possible d'explorer le comportement du modèle du toaster à l'aide d'une décomposition en arbre, en utilisant le même principe de zone temporelle (temps continu), ou en discrétisant le temps. Cependant, ce modèle en apparence simple va d'ores et déjà mener la décomposition en arbre à sa limite. En effet, telle que nous l'avons présentée, l'approche de décomposition en arbre ne regroupera pas les états "équivalents", la présence de boucles dans le modèle va amener à l'explosion combinatoire. Seule une limite du temps de simulation pourra permettre de stopper cette explosion, ce qui est parfaitement concevable dans le domaine de la simulation.

Le graphe d'atteignabilité va permettre, grâce à une gestion du temps en zones, de proposer un graphe fini de l'ensemble des états qu'il est possible d'atteindre. Il existe cinq fonctions principales qui permettent la création de zones temporelles (*tightening, equality, resetting, sliding et intersection*). M.H. Hwang nous propose, dans [Hwang 2009], l'algorithme permettant obtenir l'ensemble des états totaux d'après un modèle formel FD-DEVS.

La figure 3.16 représente le graphe d'atteignabilité de l'exemple du toaster après application de l'algorithme 1 "*ReachabilityGraph*". Cette figure est extraite de l'article [Hwang 2009] de M.H. Hwang. Il est important de préciser ici que le graphe d'atteignabilité proposé dans la figure 3.15 a fait l'objet de modifications. Notre implémentation de l'algorithme 1 *ReachabilityGraph* , ainsi que des travaux en collaboration sur la transformation des modèles de FD-DEVS vers Fiacre [14], ont permis de constater que l'outil proposé par M.H. Hwang XSY [Hwang 2005] duplique certains états et ajoute certaines transitions dans la version consultée. C'est le cas dans l'exemple du toaster, où nous ne retrouvons pas les transitions $push1,1$ de l'état (8) vers (5) et $push1,1$ de l'état (7) vers (5) présents dans [Hwang 2009]. Ces observations, bien que minimes, ont été fondamentales pour guider notre choix dans le développement de nos propres outils pour l'élaboration des graphes de classes.

En nous intéressant de plus prêt à ces résultats, l'état (1) représente l'état initial du système. Le grille-pain est en attente dans l'état total ((I,∞)(I,∞)), la zone de

14. Fiacre est un langage représentant à la fois les aspects comportementaux et les aspects de synchronisation des systèmes embarqués et distribués pour la vérification et la simulation.

Algorithme 1 ReachabilityGraph($N, \uparrow G$)

$v_0 = ((..., (s_{0i}, t_i(s_{0i})), ...), \varphi_0)$
$V_T := \emptyset$; Add v_0 to V_T
while $V_T \neq \emptyset$ **do**
  $v = ((..., (s_i, t_i(s_i)), ...), \varphi)$ $pop\_Front(V_T)$

  forall the $x \in X$ **do**
    $v_n := copy(v)$
    WhenReceive-Z($N, v, y, R^z := \emptyset, v_n, \uparrow V_T, \uparrow G$)
  end
  forall the $i \in D$ **do**
    if $clock(v).ub(i) = t_{si}$ **then**
      $v_a := copy(v)$
      $TimeZoneAt(i, l_{si}, clock(v_n))$
      $\delta_{yi} = (y, s'_i)$
      $disc(v_n)[i] := (s'_i, \tau_i(s'_i))$
      $B^z := \emptyset$
      Add i to B^z
      WhenReceive-Z($N, v, y, R^z, v_n, \uparrow V_T, \uparrow G$)
    end
  end
end

temps associée est infinie ($0 < t_{e1} < \infty$ et $0 < t_{e2} < \infty$) et le temps passé dans l'état est équivalent pour chaque modèle atomique ($0 < t_{e1} - t_{e2} < 0$). Il est alors possible d'aller dans l'état (2) avec l'évènement $push1$ déclenchant le toaster T1 pour 20s ou dans l'état (3) avec l'évènement $push2$ déclenchant le toaster T2 pour 40s . La notion de zone temporelle apparaît à plusieurs reprises dans cet exemple.

Prenons un évènement $push2$ arrivant à t=0 suivi d'un évènement $push1$ arrivant entre t=0s et t=40s ; Le modèle est alors dans l'état (5), où les deux fentes du toaster grillent. Dès l'instant où l'évènement $push1$ arrive dans l'intervalle $0s < t < 40s$ après déclenchement de $push2$, il n'y a aucune différence comportementale ((T,20)(T,40)), toutes les combinaisons de temps écoulé peuvent être regroupées dans une zone temporelle $0 < t_{e1} < 20$, $0 < t_{e2} < 40$ et $-40 < t_{e1} - t_{e2} < 0$. Il est facile de comprendre la suite du graphe de classes qui répond à la même logique. Nous ne détaillerons pas davantage le principe de graphe de classes, puisqu'il n'est pour nous qu'un outil et non pas le sujet principal de nos travaux.

L'utilisation du graphe de classes va également permettre d'appliquer tous les principes usuels des domaines de la vérification aux modèles FD-DEVS. Nous pensons notamment ici aux propriétés de vivacité, d'atteignabilité, ou encore de sûreté avec l'utilisation de contraintes temporelles (CTL, LTL, etc.). Si cet aspect n'est pas abordé dans cette thèse, il n'en reste pas moins un élément primordial dans l'adop-

Algorithme 2 WhenReceive-Z($(N, v, y, R^z, v_n, \uparrow V_T, \uparrow G)$)

forall the $z, x_i \in C_y$ *or* $z, x_i \in C_x$ **do**
  $\delta_{x,i}(S_i, x_i) = (s'_i, \rho)$
  if $\rho = 1$ **then**
    disc$(v_n)[i]$:=$(s'_i, \tau_i(s'_i))$
    Add i to R^z
  else
    disc$(v_n)[i]$:=$(s'_i, t_{s'_i})$
  end
end
if $R^z \neq \emptyset$ *or* $disc(v) \neq disc(v_n)$ **then**
  for $i \in D$ **do**
    if $t_{s_i} = \infty$ **then**
      Add i to R^∞
    end
    Successor$(clock(v_n), R^z \cup R^\infty, disc(v_n))$
    if $\nexists v' \in G.V$ *s.t.* $v_n = v'$ **then**
      Add v_n to $G.V$ and V_T
    end
    Add$(v, z, R^z \cup R^\infty, v_n)$ to $G.E$
  end
end

tion d'un tel formalisme pour les domaines de l'ingénierie des systèmes critiques.

3.5.3 Proposition de métriques d'évaluation associées aux graphes de classes

Nous proposons, dans cette section, une approche pour la validation des modèles de simulation basée sur l'analyse des graphes de classes, l'analyse arborescente ayant rapidement montré ses limites. L'ensemble des métriques proposé précédemment reste valide, l'évènement étant maintenant un couple temps-évènement. Il existe cependant une perte de capacité sur l'analyse des traces. Il y a en effet perte de l'historique des traces avec l'utilisation des graphes de classes. Cependant, nous allons voir qu'ils permettent une analyse poussée menant à l'amélioration de la crédibilité de la simulation.

De la même façon que précédemment, l'utilisation du produit parallèle nous permet d'obtenir un sous-ensemble du graphe de classes. Il fournit le langage de composition, comprenant l'ensemble des mots reconnus par le triplet GMT noté $L_{C_M}(G//M//T)$.

L'utilisateur de la simulation a toujours pour objectif (dans une certaine mesure) d'explorer seulement l'espace d'états nécessaire et suffisant pour atteindre l'objectif de simulation. La figure 3.17 montre une représentation schématique de l'espace d'états d'un modèle. Le modèle fournit un ensemble d'états atteignables. Il est pos-

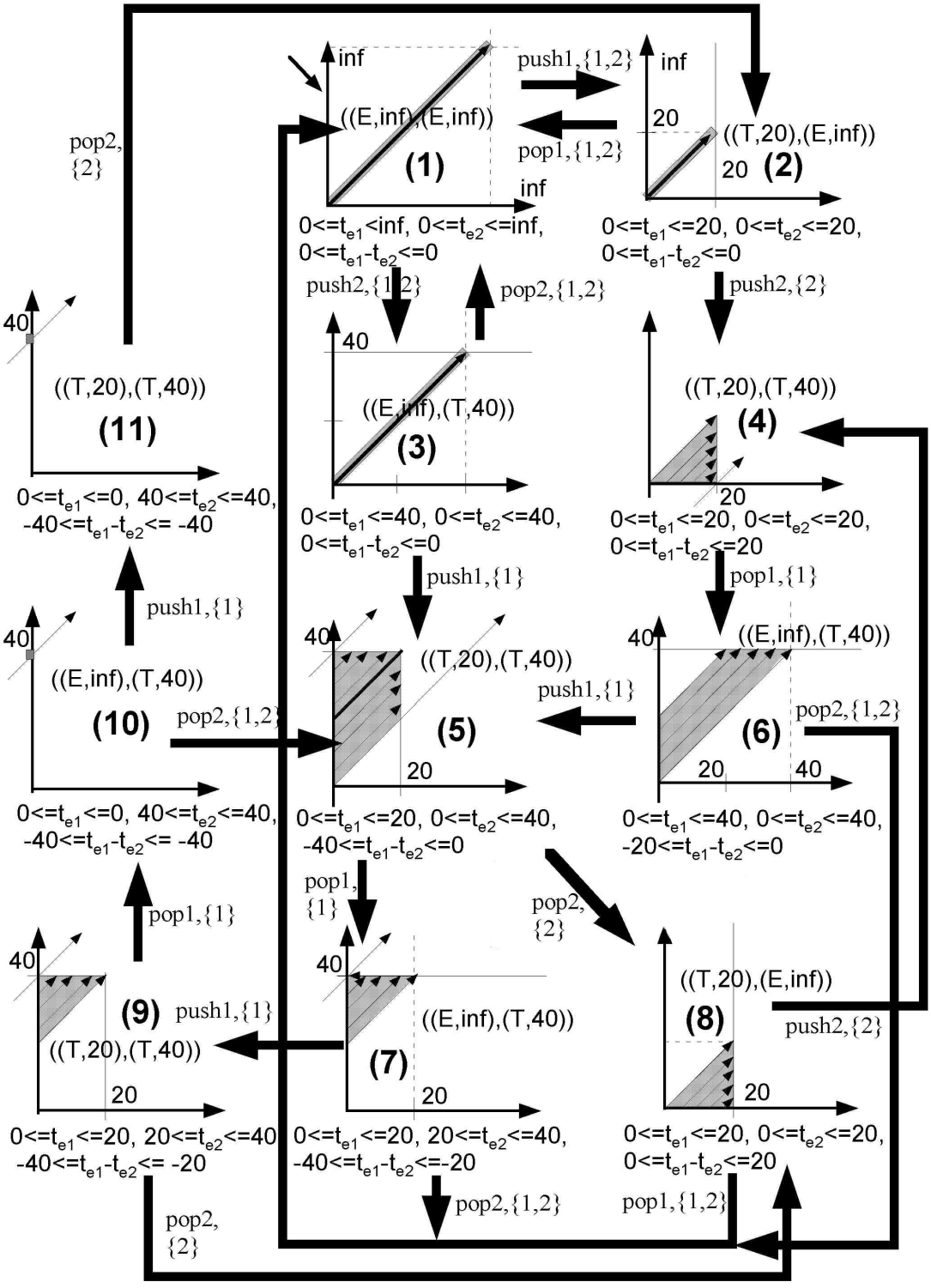


FIGURE 3.16 – Graphe de classes du grille-pain [Hwang 2009]

sible d'ajouter un ensemble d'hypothèses qui fournit une exploration de l'espace d'états suffisant. Ensuite, l'utilisateur peut exprimer les objectifs de simulation et des scénarios.

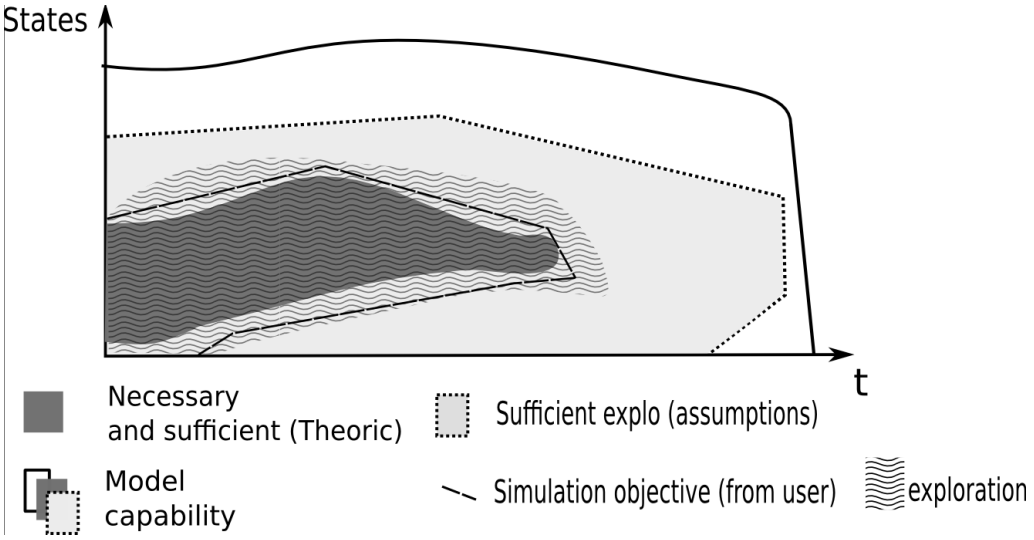


FIGURE 3.17 – Exploration d'état et objectif de simulation

Un état peut être associé jusqu'à cinq ensembles principaux :

— inclus dans le modèle (zone blanche + grise) S_M ;
— inclus dans les hypothèses(zone grise) S_{hyp} ;
— inclus dans les objectifs de simulation (la zone délimitée en pointillés) S_{obj} ;
— inclus dans les états explorés nécessaires et suffisants permettant de satisfaire les objectifs de simulation (gris foncé) $S_{N\&S}$;
— inclus dans les états effectivement explorés (zone en vagues) S_{exp}.

Pour associer à chaque état une ou plusieurs de ces propriétés, il est nécessaire de développer le graphe d'accessibilité du modèle. Lors de l'exploration de chaque état, il est possible de l'associer à un ou plusieurs ensembles cités précédemment. L'étude de ces ensembles aidera à déterminer la cohérence de la simulation.

Définition 51. *Nous définissons S_c, l'ensemble d'états cohérents tel que,*

$$S_c = \{s \mid s \subseteq M \cap S_{hyp}; s \in S_{exp}\}$$

Une simulation peut alors être considérée comme cohérente si $S_{obj} \subseteq S_c$ (par définition $S_{exp} \subseteq S_M$).
L'ensemble théorique $S_{N\&S}$ n'est pas utilisé pour la validation de la simulation. Cet ensemble est en effet une représentation théorique de l'exploration parfaite que l'utilisateur de la simulation cherche à approcher. La connaissance de cet ensemble est impossible par définition. Sans connaissance de cet espace d'états, il devient évident

que le modèle de simulation est impossible à valider de façon directe.
Toutefois, les métriques sont utilisées pour mettre en évidence un ensemble de bonnes propriétés de la simulation, permettant de se rapprocher d'un modèle de simulation valide.

Une étude macroscopique de ces ensembles permet d'établir si la simulation respecte un ensemble de bonnes propriétés, nécessaires à l'évaluation positive de la validation du modèle de simulation.

Définition 52. *La* ***compatibilité des objectifs avec les hypothèses*** *peut être définie par :*

$$S_{obj} \subseteq S_{hyp} \tag{3.16}$$

Définition 53. *L'* ***atteignabilité des objectifs*** *peut être définie par :*

$$S_{obj} \subseteq S_M \tag{3.17}$$

Définition 54. *Le* ***respect des hypothèses*** *peut être défini par :*

$$S_{exp} \subseteq S_{hyp} \tag{3.18}$$

Définition 55. *La* ***couverture des objectifs*** *peut être définie par :*

$$S_{obj} \subseteq S_{exp} \tag{3.19}$$

Cette étude est qualifiée de macroscopique puisqu'elle ne cherche qu'à valider ou invalider l'inclusion de ces ensembles sans étudier de manière détaillée cette inclusion.

L'étude de la différence symétrique (notée Δ), ou encore de la différence ensembliste (notée $\setminus$), va permettre de fournir des informations améliorant la crédibilité de la validation du modèle de simulation.

En s'appuyant sur la figure 3.17, nous pouvons par exemple voir qu'une exploration d'une zone en gris clair en dehors de la zone d'objectif (zone en tirets) représente du temps de simulation dépensé inutilement.

Définition 56. *Simulation non optimale*

$$(S_{exp} \setminus S_{obj}) \setminus S_{hyp} \neq \varnothing$$

Il serait cependant trop restrictif de dire que la simulation n'est pas valide si elle ne respecte pas une telle contrainte. Nous montrerons qu'il est bien plus intéressant d'étudier l'ensemble résultant, que de simplement comparer à l'ensemble vide. Une part inexplorée à l'intérieur de la "zone en tirets" implique une incapacité à atteindre les objectifs de simulation, ce que nous pourrions traduire par :

Définition 57. *Incapacité à atteindre les objectifs de simulation*

$$S_{obj} \setminus S_{exp} \neq \varnothing.$$

L'étude de la cardinalité des ensembles résultants est également une source d'information intéressante.

Par exemple, si le cadre expérimental tente de "mettre" le modèle dans la zone blanche (toujours en référence à la figure 3.17), une incompatibilité entre le cadre expérimental et le modèle est alors détectée.

Définition 58. *Incompatibilité cadre expérimental - modèle*

$$\alpha_{ex_G} = \frac{|L_{C_G}(G//M//T)|}{|L(G)|} < 1$$

Il est possible que le modèle ne fournisse pas le comportement attendu :

Définition 59. *Comportement inattendu du modèle*

$$\alpha_{ex_T} = \frac{|L_{C_T}(G//M//T)|}{|L(T)|} < 1 \ .$$

Alors que la section précédente s'est intéressée pour l'essentiel à l'étude des traces, nous nous sommes focalisés ici sur l'étude des états. La composition parallèle et l'analyse de l'accessibilité sont des concepts clés, permettant d'accroître la validité des modèles de simulation. Le produit parallèle ainsi que les métriques ont besoin d'un ensemble de phases en amont (apparition d'une méthodologie d'élaboration du modèle de simulation) pour être attractifs et efficaces. Une telle méthodologie sera proposée dans le chapitre 4 et illustrée dans le chapitre 5.

Il est important de noter que si la démarche est ici illustrée à l'aide du formalisme DEVS, il reste cependant possible d'appliquer ces métriques à l'ensemble des formalismes permettant la construction de graphes de classes, comme les automates temporisés [Alur 1994] ou les réseaux de Petri temporisés [Zuberek 1991].

3.6 Discussion

3.6.1 Quelle est la valeur ajoutée d'une métrique dans un contexte de validation ?

Il est difficile de répondre à une telle question. Une métrique doit être vue comme un outil. Un outil, suivant le contexte dans lequel il est utilisé, va s'avérer utile voire indispensable, ou parfois totalement inutile. Il en va de même pour une métrique. A noter qu'une métrique seule est souvent d'un intérêt limité. Il est nécessaire de combiner un ensemble de métriques pour élaborer un outil en adéquation avec le contexte de validation. C'est ici que ces travaux manquent de maturité. Il est difficile de prévoir à l'avance l'ensemble de métriques le plus efficace pour une expérimentation donnée, ces métriques n'ayant jamais été confrontées à l'utilisateur dans un

contexte industriel. L'efficacité de ces métriques repose en grande majorité sur la redondance relative d'informations. Cette redondance peut être considérée comme relative, puisqu'elle est cachée à l'utilisateur. Ce dernier fournit des informations en apparence dissociées (hypothèses et propriétés) mais qui doivent devenir cohérentes une fois appliquées au modèle.

3.6.2 Arbre vs graphe de classes : avantages et inconvénients

Durant ce chapitre, aucun choix arrêté n'a été fait sur l'utilisation privilégiée d'arbres ou de graphes de classes pour appliquer nos méthodes de validation de la simulation. Les arbres, tout comme les graphes de classes, présentent des avantages et des inconvénients. Il est difficile de les départager puisque ces deux approches sont en réalité complémentaires. L'arbre permet de récupérer un historique complet des traces d'exécutions. Il est alors possible d'étudier une trace en particulier ou un ensemble de traces. Cependant, cette approche montre rapidement ses limites puisque le nombre d'états à représenter peut être extrêmement important. L'utilisation de l'arbre est donc plébiscitée pour une étude approfondie d'un comportement ciblé, en raffinant les objectifs de simulation ainsi que les hypothèses. Le graphe de classes permet d'obtenir l'ensemble des états du modèles, ce qui permet une étude exhaustive du modèle, mais aussi de fournir un ensemble de métriques objectives à l'utilisateur de la simulation.

Dans une première approche, il semble préférable d'utiliser les graphes de classes, puis de se tourner vers les arbres de décomposition si une étude précise des traces est nécessaire. A noter que, si une trace contre-exemple est nécessaire (à la manière d'un model-checker), alors l'utilisation du graphe de classes est tout à fait adaptée.

3.6.3 Inférence des hypothèses : la création d'un langage d'hypothèses

Nous avons vu dans cette section que la documentation du contexte expérimental est primordial pour améliorer la crédibilité qu'accorde l'utilisateur de la simulation au modèle de simulation. Cependant, la portée de ces travaux est limitée par un manque important : l'absence de formalisme dédié à l'expression des hypothèses et des objectifs de simulation statiques (en opposition à dynamique) qui permettrait d'établir de manière efficace les liens entres les différentes hypothèses, et qui irait même jusqu'à la création d'hypothèses émergentes.

Bien entendu, le cadre expérimental (générateur, transducteur, accepteur) tel que nous le présentons permet d'exprimer les hypothèses et les objectifs de simulation, c'est son rôle. Il est malheureusement peu adapté à l'expression d'hypothèses statiques, qui permettraient d'améliorer la crédibilité de la validation du modèle de simulation, en étudiant la cohérence des informations fournies. L'élaboration d'un formalisme fournissant une bonne expressivité des hypothèses et des objectifs statiques est donc essentielle et doit faire l'objet de travaux plus complets, pour améliorer la prise en compte des hypothèses dans un contexte de simulation.

Certaines hypothèses statiques sont assimilables à des connaissances. Soit un modèle "*eau*" comportant trois états pour définir l'eau (liquide, solide, gazeux). Si l'utilisateur de la simulation annonce que la température de l'eau (pure) ne dépasse pas les 100°C, et que la pression est constante à 1 atm, alors il n'existera pas d'état gazeux pour l'eau, et donc un sous-ensemble du modèle n'a pas besoin d'être exploré. Cela mène à un ensemble d'évènements qui ne doivent jamais arriver : pas de déposition, pas de sublimation, pas de condensation et pas de vaporisation. Cette notion d'inférence des connaissances (des hypothèses) fait rapidement penser à un outil type *Système expert* [Hayes-Roth 1983]. Composé d'une base de faits, de règles et d'un moteur d'inférence, cet outil est capable de produire de nouveaux faits permettant de répondre à une question d'expert. Les avantages d'une telle approche sont multiples pour la simulation. Comme nous l'avons déjà vu, les hypothèses associées à une simulation sont des éléments très importants d'une simulation. La possibilité d'établir s'il y a des incohérences dans les différentes hypothèses statiques, ou encore la possibilité d'impacter les modèles grâce à de nouveaux faits (hypothèses issues du moteur d'inférence), pourrait s'avérer très bénéfique. Les efforts déployés en intelligence artificielle [Mavrovouniotis 2012] avec l'emploi d'ontologies laissent percevoir des possibilités intéressantes d'inférences, capables de générer un ensemble d'hypothèses adapté aux modèles étudiés.

3.6.4 Synthèse des métriques de crédibilité

Durant ce chapitre, nous avons proposé un ensemble de métriques qu'il est possible de combiner ou d'étendre afin d'engendrer un nombre très important de métriques. Ces métriques sont actuellement isolées, seule l'expertise de l'utilisateur de la simulation pourra regrouper les différents résultats, afin d'évaluer le niveau de crédibilité qu'il associe à la simulation. Cette approche est loin d'être idéale. Par exemple, l'obtention de $L_C(G//M//T) = L(M)$ peut être dû à l'utilisation d'un cadre expérimental n'appliquant aucune contrainte sur le modèle. C'est le cas d'un cadre expérimental qui reçoit toutes sorties possibles et qui génère toutes les entrées attendues. $L_C(G//M//T) = L(M)$ n'apporte donc aucune confiance à l'utilisateur de la simulation si elle est utilisé seule. Par contre, si elle est couplé à l'observation de métriques sur le cadre expérimental, comme l'étude de $\alpha_{ex}(G)$ (taux d'exploration du générateur), alors on pourra établir un niveau de confiance.

La proposition d'une méthodologie dans le prochain chapitre va nous permettre de regrouper un certain nombre de métriques, pour aider l'utilisateur, mais sans proposer d'approche générique de groupement des informations.

3.7 Résumé

Le but de ce chapitre était de formaliser un ensemble de métriques permettant d'aider l'utilisateur de la simulation. Après avoir montré que les notions de validité et d'interopérabilité permettaient d'améliorer la crédibilité accordée à un modèle de simulation, et après avoir fait un état de l'art sur les métriques, nous avons proposé

un ensemble de métriques formelles et génériques utilisant la théorie des automates et la théorie des langages.

L'étude a été divisée en deux étapes. Une première étape, utilisant les automates à interface, a permis de s'abstraire des contraintes relatives au temps continu pour s'intéresser à la compatibilité statique, avec la définition d'une opération de restriction. Cette opération nous permet de poursuivre l'exploration du modèle si le générateur s'avère moins capable que le modèle. Un pas a été fait vers l'étude de la compatibilité dynamique en utilisant la décomposition en arbre. Après avoir montré les limites de cette proposition, la seconde étape a consisté à étudier un formalisme à temps continu et à évènement discret, les automates *Discret Event System Specification*. Nous proposons alors un nouvel ensemble de métriques, associé au graphe de classes résultant du produit de composition entre cadre expérimental et modèle, et donc focalisé sur les états. Il reste possible d'appliquer les métriques aux différents formalismes présentés, les limites étant associées à la méthode d'exploration (graphe de classes ou arbre de décomposition).

Dans ce chapitre, nous avons pu voir se dessiner une méthodologie permettant d'obtenir les métriques proposées quelle que soit l'approche associée (automate à interface + arbre de décomposition ou DEVS + graphe de classes). Il est nécessaire de guider l'utilisateur de la simulation pour faciliter l'obtention de ces métriques en associant la méthodologie à un outil ou plutôt à un ensemble d'outils. Cet ensemble d'outils pourrait alors être considéré comme un framework dédié à la simulation.

Chapitre 4

Profil pour la simulation

Sommaire

4.1 Introduction

"The solution lies in a capable tool set that recognizes the influence of modeling objectives and error tolerances, the multidimensional choices of bases for agregation mappings. Formalizing such dimensions of the problem leads to a sophisticated framework that involves concepts such as scope/resolution product, experimental frame, applicability and derivability lattices, and conditions for valid abstractions". B.P. Zeigler [Zeigler 1976]

Cet extrait du livre "Theory of modeling and simulation" a déjà trouvé sa place en introduction de cette thèse puisqu'il a déclenché et motivé l'ensemble de ces travaux. Ce chapitre tente d'apporter une réponse à la réflexion de B.P. Zeigler, en regroupant dans un outil les travaux théoriques sur l'étude de la compatibilité statique et dynamique proposés dans les chapitres précédents.

En conclusion du chapitre 3, nous faisions allusion à l'apparition d'une méthodologie permettant l'amélioration de la crédibilité de la simulation.

De manière standard, toute démarche d'ingénierie système suit une norme définie au travers de processus (voir 1.3.1.3). Ces processus ne vont cependant pas

définir la manière de réaliser le travail. Autrement dit, le processus ne définit pas la méthodologie associée au processus. Il est évidemment possible d'utiliser des méthodologies existantes [1], mais il est parfois nécessaire de développer de nouvelles méthodologies pour mieux respecter la sémantique du processus d'IS ciblé. Notre étude sur la validation des modèles de simulation cherche à améliorer ces processus d'IS dédiés à la V&V. L'étude théorique effectuée dans les chapitres 2 et 3 nous apporte donc une solution au travers d'une méthodologie. R. Guillerm nous rappelle dans [Guillerm 2011] le paradigme *processus-méthodes-outils*, où la méthode est accompagnée d'un outil pour en faciliter l'application. L'étude théorique a fait apparaître une méthodologie qu'il est essentiel d'associer à un outil. C'est pourquoi nous proposons dans ce chapitre l'étude d'un DSL [2] pour la simulation et à l'étude de la validité des modèles de simulation.

La première section de ce chapitre s'intéresse à l'état de l'art sur la création et la transformation des modèles. Nous aborderons ici les solutions proposées dans la littérature pour exprimer un DSL. La seconde section propose une définition de la syntaxe abstraite du langage de modélisation, dédiée à la modélisation des simulations, et à la validation des modèles de simulation. Dès que l'on aborde les outils de modélisation pour la simulation, il est impossible de ne pas s'intéresser à la question des simulateurs, souvent indissociable du langage de modélisation. Une troisième section aborde la question des simulateurs. L'approche IDM [3] nous permet de décorréler le DSL de sa plateforme d'exécution, ce qui nous permet de relâcher les contraintes associées aux détails d'implémentation (variation entre langages de programmation) et rend possible l'utilisation des outils existants par simple transformation. Une dernière section discutera des apports potentiels de la méta-modélisation pour la V&V de la M&S.

4.1.1 Description d'une méthodologie d'évaluation

L'étude théorique des précédents chapitres fait apparaître une démarche méthodique permettant l'obtention de métriques pour améliorer la crédibilité accordée par l'utilisateur de la simulation à son modèle de simulation. Nous allons aborder de manière plus détaillée cette méthodologie.

■ **Phase préliminaire** Étude préliminaire. Cette étape n'est pas détaillée ici mais pourrait être résumée ainsi : l'utilisateur de la simulation (terme pouvant représenter une équipe) prend connaissance du modèle et des objectifs de simulation.

■ **Phase 1** Définition des hypothèses. Cette première phase consiste à définir l'ensemble des hypothèses de modélisation, à l'exception du couple générateur/-

1. Dans [Balci 1998], l'auteur dénombre 77 techniques associées à la V&V&A qui peuvent s'appliquer dans le processus de M&S

2. Un DSL pour Domain Specific Language, ou langage dédié en français, est un langage de programmation dont les spécifications sont dédiées à un domaine d'application précis.[Fowler 2010]

3. Ingénierie Dirigée par les Modèles

transducteur, en s'appuyant sur les exigences et l'expertise. Par étapes successives, il va être possible de formaliser certaines exigences pour arriver à une spécification formelle de l'environnement de simulation et du modèle. L'ensemble des hypothèses va restreindre l'utilisation du modèle et du cadre expérimental. L'expression de ces hypothèses "statiques" va nous permettre d'étudier la cohérence avec les hypothèses "dynamiques" décrites à l'aide du couple générateur/transducteur. Le choix des formalismes permettant de décrire les hypothèses de modélisation n'a pas fait l'objet d'une étude approfondie, les possibilités étant vastes suivant la nature de l'hypothèse. L'utilisation de contraintes OCL [OMG 2004], de propriétés de logique temporelle [Di Giampaolo 2010], de relations mathématiques, de paternes de modélisation [Abid 2011], et de mécanismes d'automatisation des propriétés LTL [Nikora 2009], peut permettre de définir efficacement les hypothèses. Il est également possible d'envisager le "modèle d'hypothèses" sur le modèle triplet tel que l'utilise le web sémantique (RDF)[Berners-Lee 2001], ce qui permet d'utiliser les outils d'inférence actuels [Sintek 2002].

Nous considérons qu'il existe trois types d'hypothèses :

— **Les hypothèses de modélisations** : ces hypothèses sont directement influencées par les objectifs expérimentaux. Elles sont définies par le développeur du modèle et par l'utilisateur de la simulation. Ce sont les hypothèses qui vont demander le plus d'efforts aux modeleurs.

— **Les hypothèses opérationnelles** : ces hypothèses sont directement influencées par l'outil de modélisation et les capacités du simulateur. On y retrouve les hypothèses associées aux restrictions du produit domaine/résolution. La définition de ces hypothèses va demander un effort à l'utilisateur de la simulation qui doit exprimer les limitations associées à la mise en œuvre (les limitations de l'outil), en fonction des objectifs expérimentaux.

— **Les hypothèses rationnelles** : ces hypothèses sont extraites de la physique au sens large. Ce sont les hypothèses logiques associées à un domaine. Suivant le domaine d'application, le nombre d'hypothèses rationnelles peut être très important. L'effort d'élaboration de ces hypothèses doit être considéré sur le long terme puisque pérenne. Il est possible de voir ces hypothèses comme un ensemble de données d'expert, pouvant se rapprocher du système expert.

C'est cet ensemble d'hypothèses (hypothèses de modélisation ∪ hypothèses opérationnelles ∪ hypothèses rationnelles), que nous appellerons *hypothèses statiques*[4] qui va permettre la mise en évidence d'une incohérence avec les hypothèses dynamiques (celle qui sont définies au travers du couple générateur/transducteur).

■ **Phase 2** Modélisation de l'environnement. Durant cette phase, l'utilisateur

4. L'appellation *hypothèses statiques*, regroupe également des hypothèses pouvant être considérées comme dynamiques. Nous pensons notamment aux règles CTL ou LTL qui peuvent être employées pour définir ce type d'hypothèses. Nous conservons ce terme puisque nous cherchons ici à mettre en évidence la différence avec les *hypothèses dynamiques* définies à l'aide du couple générateur / transducteur

de la simulation va décrire les scénarios de simulation : d'un coté, les séquences de stimuli qu'il va mettre en œuvre (*générateur*), de l'autre, le comportement observable supposé du modèle (*transducteur*) en fonction des stimuli d'entrée. L'utilisateur va régulièrement revenir en phase 1 pour exprimer de nouvelles hypothèses statiques. En fonction des scénarios de simulation établis et des possibilités, l'utilisateur de la simulation va jouer sur le triplet domaine-résolution-précision des variables du cadre expérimental. Ce triplet va essentiellement impacter le groupe des hypothèses opérationnelles. Durant cette phase, l'utilisateur de la simulation peut utiliser deux formalismes : les automates à interfaces [Alfaro 2001] ou les automates DEVS [Zeigler 2000].

■ **Phase 3** Compatibilité du périmètre. La compatibilité des entrées/sorties est un pré-requis pour l'utilisation de la simulation en respectant le cadre expérimental. La compatibilité doit porter sur le nom, l'information, le type, la résolution et la précision [Albert 2009]. Les hypothèses statiques pourront également jouer sur cette compatibilité. Si la compatibilité n'est pas totale, chaque incompatibilité doit être montrée à l'utilisateur et les entrées/sorties non compatibles sont "bouchonnées" (remplacées par des évènements internes) après approbation de l'utilisateur de la simulation. Si l'utilisateur de la simulation décide que l'entrée ou la sortie incompatible est d'intérêt, il est nécessaire de revenir à la phase 1 ou 2 pour établir de nouvelles hypothèses. Si l'on observe toujours une incompatibilité, il est alors nécessaire d'en aviser l'équipe de modélisation.

■ **Phase 4** Exploration des modèles. Le modèle (resp. le cadre expérimental) est exploré en totalité pour connaître l'ensemble des possibilités associées au modèle (resp. au carde expérimental) (e.g nombre total d'états, résolution des variables, ...). Cette phase n'est associée à aucune hypothèse, il s'agit d'une exploration totale. C'est l'utilisateur de la simulation qui va choisir la méthode d'exploration : Arbre de décomposition ou Graphe de classes. Le choix de la méthode d'exploration sera fait en fonction des avantages et inconvénients présentés au chapitre 3. Il est possible d'appliquer un sous-ensemble d'hypothèses statiques au modèle afin de limiter l'explosion combinatoire (e.g. temps de simulation).

■ **Phase 5** Composition des automates. Nous effectuons ici un produit parallèle appelé composition parallèle (noté //) en théorie des automates. Le cadre expérimental et l'ensemble des hypothèses sont associés à ce produit afin de restreindre le modèle. Le produit parallèle met en évidence les transitions communes des automates et permet de représenter les évolutions du modèle qui n'appartiennent pas à l'intersection des alphabets.
La phase 3 peut alors sembler superflue. Cependant, elle permet de définir la compatibilité de façon statique. La taille théorique des modèles étant importante, il est primordial d'effectuer un premier test de compatibilité statique avant d'explorer les modèles et ainsi de connaître la compatibilité dynamique (plus coûteuse en temps). Si par exemple deux variables ont un domaine de validité incompatible, il est bon de le détecter de façon statique, plutôt que de le découvrir après l'exploration de

plusieurs milliers d'états. L'utilisateur pourra toujours accepter l'incompatibilité sur la phase 3 pour la comprendre de façon plus détaillée avec l'exploration dynamique durant la phase 5.

■ **Phase 6** Phase d'analyse. Les résultats de la compatibilité sont donnés à l'utilisateur de la simulation. Un ensemble de métriques (détaillées dans le chapitre 3) permet d'aider l'utilisateur de la simulation et d'attribuer une certaine confiance dans la simulation effectuée. Il pourra choisir d'affiner les hypothèses statiques ou dynamiques et de recommencer alors un cycle de simulation.

Ces phases de la méthodologie font apparaître de nombreux retours vers les phases amont. Il est possible de les schématiser comme dans la figure 4.1 qui reprend le nom de chacune des phases en indiquant le nom associé. Cette figure fait également apparaître un bloc modélisation qui doit être considéré comme une phase pouvant être effectuée en amont ou parallélisée jusqu'à la phase 3, où l'utilisateur de la simulation a un besoin explicite du modèle.

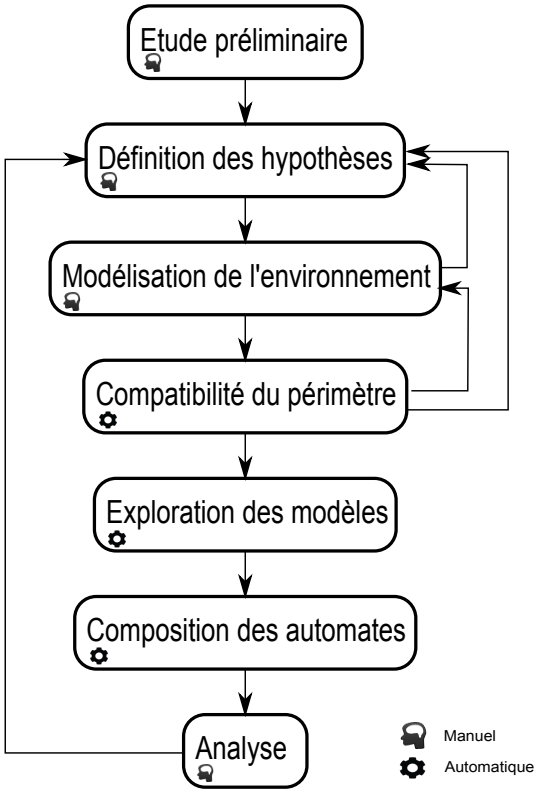


FIGURE 4.1 – Phases de la méthodologie.

Comme précédemment exposé, il est nécessaire de proposer un outil guidant l'utilisateur de la simulation pour appliquer cette méthode. Nous allons voir qu'il existe plusieurs solutions permettant de créer un DSL. Tout d'abord, l'approche Ingénierie Dirigée par les Modèles (IDM) va permettre d'élaborer un DSL de modélisation comportementale des systèmes basé sur le formalisme DEVS, et ainsi de

proposer une modélisation formelle de la phase 3.

Cette approche sera complétée par la création d'un profil SysML pour la capture des hypothèses statiques et dynamiques. Nous considèrerons alors un DSL résultant, combinaison de deux DSL, que nous appellerons SiML.

4.2 État de l'art sur la création et la transformation des modèles

4.2.1 Ingénierie dirigée par les modèles

L'Ingénierie Dirigée par les Modèles (IDM) ou Model-Driven Engineering (MDE) fait partie des domaines de l'informatique et de l'ingénierie système. Elle fournit les outils et concepts nécessaires pour créer ou modifier des langages de modélisation. Ce domaine relativement récent a fait son apparition dans les années 1980, avec un premier outil qui supportait les concepts de l'IDM, "the Computer-Aided Software Engineering" (CASE). La complexité grandissante des systèmes a conduit les développeurs à manipuler des concepts de plus haut niveau. L'Object Management Group s'est intéressé à l'Ingénierie Dirigée par les Modèles et a proposé en 1997 le standard MOF (Meta-Object Facility)[OMG 2006]. On parle également parfois de MDA (Model Driven Architecture), marque déposée de OMG qui reprend et restreint les concepts de l'IDM.
Comme nous le rappelle B. Combemale dans [Combemale 2006b], l'IDM a permis plusieurs améliorations significatives dans le développement des systèmes complexes en proposant une approche plus abstraite. L'idée est de disposer d'autant de langages de description que le nécessite le projet, tout en concevant des liens entre les modèles ainsi créés.

4.2.2 La méta-modélisation

Un méta-modèle est une définition d'un langage de modélisation. C'est le modèle du modèle. A première vue, un modèle ne ressemble à aucun autre modèle. Suivant le système à modéliser, suivant la vision que le concepteur aura du système, suivant sa maîtrise du langage de modélisation, le modèle résultant sera différent. Pourtant tous ces modèles seront regroupés sous une même syntaxe et une même sémantique. Il existe un vocabulaire commun, un formalisme qui peut lui-même être modélisé. C'est le rôle premier de la méta-modélisation. L'OMG a défini MDA (Model Driven Architecture) [Soley 2000] qui préconise une vision sur quatre niveaux de modélisation au travers du MOF [OMG 2006]. La figure 4.2 propose une vue schématique du MOF et de ses quatre niveaux d'abstraction.

Exemple Afin de mieux comprendre la signification de ces quatre niveaux d'abstraction, nous allons prendre l'exemple simple d'une carte topographique figure 4.3.

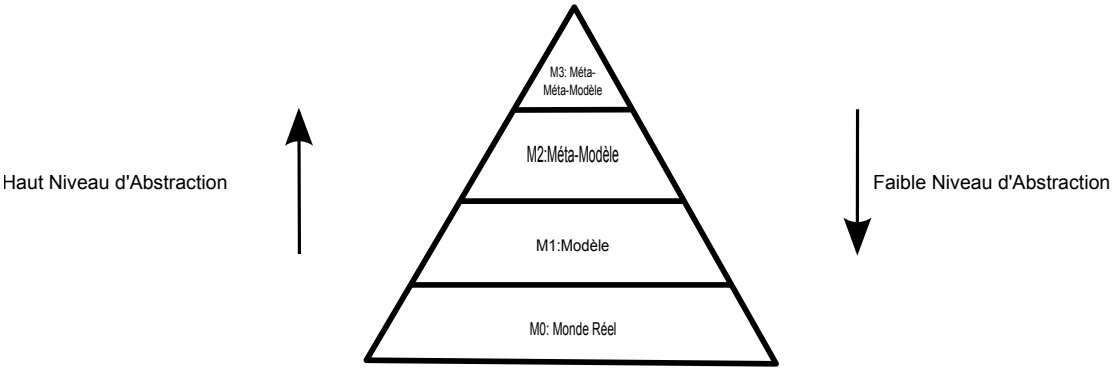


FIGURE 4.2 – Modèle du MOF.

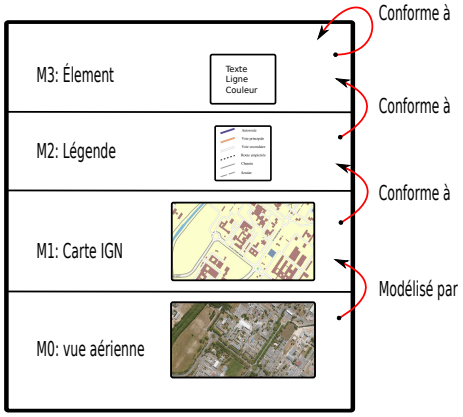


FIGURE 4.3 – Exemple de modélisation

La vue aérienne du LAAS est une vision réelle du laboratoire, c'est le niveau M0. C'est le système à modéliser. Il faut maintenant établir le modèle (M1). Ici, le choix s'est porté sur un modèle de type carte topographique (type IGN top 25). On parle de choix du formalisme. Le modèle aurait pu être tout autre, une carte thermique ou géologique par exemple. C'est donc une des façons de modéliser le laboratoire, de représenter de façon abstraite le système d'intérêt. Cette carte, pour être comprise par tous, se doit d'être conforme à sa légende (M2). C'est ce que l'on appelle le méta-modèle. Par exemple, l'ensemble des cartes IGN sont conformes à une seule et même légende (elles respectent leur méta-modèle). Tous les concepts d'une carte IGN sont présents dans la légende. Si la légende est modifiée, l'ensemble des instances du méta-modèle (l'ensemble des modèles, l'ensemble des cartes IGN) est modifié. Imaginons maintenant une carte thermique du laboratoire. Il est évident que la carte ne sera pas la même, la légende non plus, pourtant les deux légendes auront des points communs. C'est le méta-méta-modèle (M3). C'est lui qui décrit les concepts clefs présents dans les méta-modèles. Pour construire une légende, il faut du texte, des couleurs, etc. Toutes les légendes sont conformes à ce méta-méta-modèle. Il est indispensable que ce niveau M3 soit conforme à lui-même, afin d'éviter

un empilement des niveaux. Ici, du texte permet de définir le méta-méta-modèle. Il est donc conforme à lui même.

Lors de la création d'un méta-modèle, il est important de différencier deux types de syntaxe :

— **Abstraite** : La syntaxe abstraite d'un langage de modélisation (niveau M2) est l'expression structurelle de l'ensemble des concepts et de leurs relations. Il existe de nombreux méta-langages (niveau M3) proposant les concepts et relations de bases permettant l'élaboration de syntaxes abstraites (Ecore [Budinsky 2004] ou encore KM3 [Jouault 2006a]).

— **Concrète** : La syntaxe concrète fournit une représentation textuelle ou graphique permettant de manipuler l'ensemble des concepts de la syntaxe abstraite.

Nos travaux vont s'intéresser en priorité à la syntaxe abstraite du langage de modélisation. La création d'un langage concret, bien que nécessaire, n'est pas la priorité de ces travaux. Un langage concret, efficace et adapté, demande un effort important que nous ne jugeons pas nécessaire dans cette étude. Ainsi, nous nous intéressons en priorité à la faisabilité de l'évaluation de la validité des modèles de simulation, et non à son implémentation pérenne. Nous négligerons par conséquent l'élaboration d'une syntaxe concrète optimale, en nous référant aux possibilités de génération "automatique" de syntaxes concrètes proposées par [Clark 2008] ou encore par [Ledeczi 2001].

4.2.2.1 Avènement des méta-modèles formels

Au-delà de l'aspect syntaxique, il est normal de se poser la question de la sémantique des méta-modèles. Selon la précision de la définition de la sémantique, le langage de modélisation sera qualifié de formel, semi-formel ou informel. Un langage est dit formel si l'ensemble de sa sémantique est défini dans un formalisme mathématique, où l'outil mathématique va assurer une cohérence de la définition. Il n'existe alors qu'une interprétation de la sémantique pour l'utilisateur du langage. C'est le cas de DEVS [Zeigler 1976] ou encore des réseaux de Petri [Peterson 1981].

Au contraire, un langage semi-formel possède une part d'ambiguïté, certains aspects de la sémantique ne sont pas clairement définis et peuvent amener à interpréter différemment le même modèle. C'est le cas de langages comme UML : outre la définition d'une partie du formalisme en langage naturel, sa sémantique est dite *polysémique*[5] [Heon 2010]. C'est cette propriété qui fait qu'un langage semi-formel est plus souple à utiliser qu'un langage formel. Il demande un niveau d'expertise moindre tout en proposant un *guidage représentationnel* [Suthers 2003] que ne propose pas un langage informel. Les avantages des langages semi-formels sont nombreux [Heon 2010] et posent la question de l'utilisation du *formel*.

5. La polysémie fait référence à la propriété d'un symbole de représenter plus d'un objet.

Dans le contexte des systèmes critiques, il est difficile de s'éloigner des langages formels. En effet, la signification exacte d'un modèle est essentielle, comme par exemple, dans le cas des phases de vérifications formelles qui imposent l'élaboration d'un modèle formel. Il est donc essentiel de proposer un langage formel (sémantique unique) et ayant une expressivité suffisante pour offrir à l'utilisateur de la simulation un guidage représentationel efficace.

La communauté de l'IDM s'est rapidement posée la question de la définition de la sémantique des langages abstraits et des moyens associés pour parvenir à un langage de modélisation formel. B. Combemale nous propose une taxonomie des sémantiques dans l'IDM [Combemale 2008]. L'auteur y distingue deux types de sémantiques : la *sémantique statique*, associée aux propriétés indépendantes de l'exécution, et la *sémantique dynamique*, qui permet de décrire le comportement à l'exécution. En reprenant les travaux de G. Winskel sur la sémantique des langages de programmation [Winskel 1993], l'auteur affine sa taxonomie en proposant de hiérarchiser la sémantique d'un langage abstrait en trois types (du plus abstrait au plus concret) : la sémantique axiomatique, la sémantique dénotationnelle et la sémantique opérationnelle.

— La **sémantique axiomatique** est basée sur la logique mathématique et la preuve de propriété. L'OMG préconise l'utilisation de contraintes OCL [OMG 2004]. Le modèle doit respecter ces règles pour être conforme à son méta-modèle. Ces règles sont fondamentales pour s'assurer de la bonne construction du modèle (well-formed rules).

— La **sémantique dénotationnelle** se base sur un formalisme mathématique pour exprimer la sémantique d'exécution. L'IDM propose d'élaborer des règles de traduction vers un langage formel. Il est alors possible d'utiliser les outils de simulation ou de vérification du formalisme de destination. Cette méthode de définition de la sémantique a fait l'objet de travaux préliminaires avec, par exemple, la transformation des diagrammes d'activité SysML vers les réseaux de Petri [Foures 2011].

— La **sémantique opérationnelle** permet de décrire le comportement dynamique de toute "instance" d'un langage de modélisation. B. Combemale propose deux approches permettant de définir cette sémantique. Une méthode utilise le langage de méta-modélisation exécutable Kermeta [Muller 2005], ajoutant un aspect comportemental au méta-langage Ecore. L'autre méthode consiste en un ensemble de transformations *endogènes*[6] mis en œuvre avec le langage de transformation ATL [Jouault 2006b]. Les règles de transformation décrivent le comportement du modèle, chaque transformation correspond à

6. Une transformation est dite *endogène* si le modèle résultant est décrit à partir du même méta-modèle que le modèle source. Au contraire, la transformation sera dite *exogène* si le modèle résultant est conforme à un autre méta-modèle.

une "étape" d'avancement dans l'exécution du modèle.

Aujourd'hui, certains travaux ([Mellor 1999], [Breton 2001], [Combemale 2006b]) tentent d'enrichir les méta-modèles en proposant une sémantique associée. Ce travail ne doit pas être confondu avec d'autres travaux, dans lesquels la sémantique d'exécution est liée à l'outil et non pas directement au langage abstrait (le méta-modèle), de manière indépendante à toute plateforme d'exécution (PIM [7]).

Le lien entre méta-modélisation et exécutabilité du modèle a été étudié par E. Breton dans [Breton 2001]. L'auteur y décrit les mécanismes en jeu lors de la méta-modélisation des concepts abstraits et de leur exécutabilité. Le méta-modèle résultant est composé de trois parties :

— **Structure** : décrit la structure du modèle et des relations entre entités. C'est la partie la plus couramment décrite dans un méta-modèle.
— **Situation** : décrit la situation spécifique du modèle (e.g. marquage initial, état courant). Cette partie est dépendante du temps.
— **Exécution** : décrit les liens qu'il existe entres les différentes situations. La partie "exécution" est dépendante de la partie "situation", qui dépend elle-même de la partie "structure".

Cette découpe fait apparaître une première approche de la description de l'exécutabilité des modèles. Malheureusement, l'auteur ne décrit que la sémantique statique. Même si l'approche s'intéresse à l'aspect comportemental, elle ne fait apparaître que l'aspect structurel de ce comportement. La partie "exécution" définit les liens entre les différentes situations mais n'offre pas la description des mécanismes permettant le passage d'une situation à une autre. Conscient de cette limite, l'auteur propose l'utilisation de règles OCL, permettant d'affiner la sémantique d'exécution et, dans certains cas simples, de décrire la sémantique dynamique. L'auteur conclue qu'une telle approche est malgré tout insuffisante et qu'il est nécessaire de fournir un aspect comportement au méta-langage, permettant la description de l'exécutabilité des modèles. Des conclusions similaires sont observables dans [Mellor 1999]. Pour répondre à ce constat, l'IDM propose d'associer aux méta-langages un aspect comportemental. Le méta-modèle va être composé d'une partie statique, d'une partie dynamique ou d'une partie dédiée à l'interaction [8] [Combemale 2006b].

Les possibilités d'exprimer la sémantique d'exécution des modèles à l'aide de transformations semblent avoir certains avantages vis-à-vis des langages de méta-programmation permettant d'exprimer la sémantique opérationnelle. Elles offrent notamment une séparation totale entre l'aspect structurel et l'aspect comportemental. Cependant, à l'usage, il devient rapidement difficile de maîtriser les règles de transformation lorsque le comportement à définir devient complexe. L'approche basée sur la définition d'opérations proposée dans la nouvelle génération de méta-

7. Platform Independant Model
8. L'aspect interaction provient de la nécessité de représenter les échanges avec l'environnement extérieur au système d'intérêt.

langages (e.g Kermeta [Muller 2005], XOCL [Clark 2004]) semble permettre une description plus aisée des mécanismes complexes, et pourrait donc être une solution plus intéressante.

4.2.3 UML et notion de profil

Lors de l'élaboration d'un langage dédié (DSL) en ingénierie système, il est difficile de passer sous silence UML (Unified Modelling Language) [Uml 2008] et la notion de profil. Ce langage semi-formel, aujourd'hui largement répandu, propose une syntaxe graphique autour de 14 diagrammes permettant de décrire l'aspect structurel, comportemental ou encore dynamique d'un système. Considéré comme un langage de modélisation généraliste (GPML [9]), il offre la possibilité de se spécialiser. Il est alors possible de créer des DSLs dédiés à un ensemble de tâches spécifiques (e.g. TelcoML [Tel 2013], MARTE [Mar 2011], UTP [UTP 2013]).
UML propose un mécanisme d'extension qui permet cette spécification du langage. L'ingénierie système a d'ailleurs proposé un DSL dédié à la modélisation des systèmes avec SysML [SysML 2006] : ce langage apporte une utilisation simplifiée et plus efficace qu'UML en limitant le nombre de concepts et de diagrammes, proposant lui-même la possibilité de se spécialiser. Il est intéressant d'étudier SysML comme base de notre DSL.

Se pose alors la question de l'élaboration du DSL. Vaut-il mieux partir du MOF et élaborer le DSL dans sa totalité, ou au contraire, utiliser les possibilités offertes par un langage existant comme SysML ou tout autre méta-modèle conforme au MOF [10] ?

L'élaboration du langage *UML testing profile* [UTP 2013] a fait l'objet d'une double définition, utilisant d'un côté un profil UML (adopté comme norme par l'OMG) et d'un autre côté l'approche basée MOF (relayée en annexe comme version non-normative basée MOF). Malheureusement, il ne semble pas exister de travaux de conclusion sur ce double développement. La norme de l'OMG elle-même reste vague : "*There is no simple answer for when to create a new metamodel, when to create a new profile and when to create both (one for UML tooling, the other for MOF-based tooling)*"[Uml 2013]. Cette question de l'élaboration d'un profil ou meta-modèle basé MOF n'est pas nouvelle puisque P. Desfray la pose dès 2000 dans [Desfray 2000].

Pour répondre à cette question, J. Bruck nous propose, dans une approche pragmatique, de s'intéresser au chevauchement entre le DSL et le langage source [Bruck 2007]. La technique consiste à évaluer la similitude entre SysML et le DSL.

La littérature ne propose aucune méthode d'évaluation de la superposition des langages permettant de choisir la meilleure approche. Le DSL n'étant encore qu'au

9. General-Purpose Modeling Language
10. Cette possibilité est offerte par UML à l'aide de la relation *reference Metamodel*

stade de projet, l'évaluation méthodique semble en effet difficile, voire impossible. Il est essentiel d'avoir une excellente connaissance du langage source (UML ou SysML), associée à une bonne connaissance du domaine cible et des objectifs du langage, pour une évaluation rapide des similitudes [11]. L'idée est ici d'établir si la superposition est forte (situation de gauche figure 4.4) ou faible (situation de droite figure 4.4).

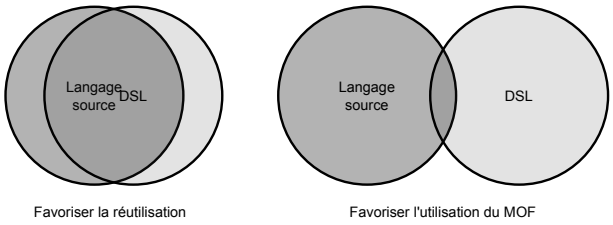


FIGURE 4.4 – Superposition du DSL (Adaptée de [Bruck 2007])

Si l'on ne s'intéresse qu'à la description du comportement dynamique (modélisation formelle du générateur, transducteur et modèle), les formalismes comme DEVS ou les automates à interface ont un nombre limité de concepts et de relations. Le nombre limité d'éléments entraîne de façon mécanique un nombre limité d'éléments en commun avec un langage comme SysML. Ceci engendrerait un nombre de restriction beaucoup trop important.

Cette constatation amène à favoriser l'élaboration d'un DSL construit à partir du MOF. Cependant, notre approche cherche à fournir à terme un outil dédié à l'ingénierie système, où la validation du modèle de simulation n'est qu'une étape dans un processus plus complexe. C'est pourquoi nous avons fait le choix de partir du méta-modèle SysML. Dans un premier temps, l'idée est d'étendre SysML avec la méthodologie proposée en introduction de ce chapitre (4.1.1), et ainsi de faire apparaître le principe de cadre expérimental, ainsi que le principe d'élicitation des hypothèses statiques de simulation. Dans un deuxième temps, nous complèterons le méta-modèle SiML en associant un nouveau formalisme de description comportementale : DEVS.

L'utilisation d'un langage existant comme base de notre DSL amène à s'intéresser à l'extension de langage. Toujours selon [Bruck 2007], il existe quatre types d'extensions de langage :

— ***Featherweight* extension** (extension poids plume) :
Cette extension consiste à utiliser la technique d'*Interface* des méta-types de langage source. Ce type d'extension sert à adapter le vocabulaire utilisé à un domaine sans aucun changement conceptuel sur le langage. Aucune restriction du langage n'est possible.

11. Notons que cette double expertise est un des freins pour l'adoption des DSLs basés profil. En plus de bonnes compétences dans le développement des langages, il est nécessaire d'avoir une expertise dans le domaine cible du DSL, ce qui est rare.

— ***Lightweight* extension** (extension légère) :
Cette extension est plus connue sous le nom de *profil* tel que défini par UML/-SysML. Elle offre seulement la possibilité de contraindre/étendre les concepts du langage. Il est, par contre, impossible de supprimer ou d'ajouter un élément au langage.

— ***Middleweight* extension** (extension poids moyen) :
Cette extension, plus permissive que l'extension *lightweight*, ajoute la possibilité de relâcher des contraintes associées au langage grâce à la spécialisation des méta-types UML/SysML. Il est alors possible d'ajouter des concepts au langage. Il reste cependant impossible d'en retirer. Cette méthode est à éviter puisqu'elle crée une dépendance à la version du méta-modèle UML/SysML utilisé, ce qui pose des problèmes de maintenabilité en cas d'évolution du méta-modèle UML. Il y a perte d'interopérabilité avec les outils existants.

— ***Heavyweight* extension** (extension lourde) :
Cette extension est très permissive, elle permet d'ajouter, de supprimer, d'étendre, ou de restreindre les concepts du langage source. C'est une approche voisine de celle basée sur le MOF. Elle permet cependant la réutilisation des concepts existants. Il est également difficile de maintenir ce type d'extension. Il y a de même perte d'interopérabilité avec les outils existants.

Plus l'extension choisie sera lourde, plus le coup de développement sera équivalent à celui d'une approche basée MOF. La même réflexion peut être faite sur la maintenabilité du langage. Dans notre contexte une approche de type *Heavyweight* semble être la plus adaptée. Très permissive, elle nous permet l'ajout, la suppression, la restriction et l'extension de concepts nécessaires pour apporter une solution adaptée à notre méthodologie et au bon guidage de l'utilisateur de la simulation. Il est également possible de voir notre approche en deux temps : une première partie, respectant les principes de profils UML (*lightweight* extension), dédiée à la création du cadre de modélisation (cadre expérimental, hypothèses), et une deuxième partie qui va associer une approche basée MOF dédiée à la description du méta-modèle DEVS formel.

4.3 SiML : Langage de modélisation dédié à la simulation

Comme nous l'avons écrit précédemment, il est possible de voir la construction du DSL SiML comme l'addition de deux sous-DSLs. Un premier DSL utilise le principe de profil SysML pour élaborer un cadre méthodologique pour l'utilisateur de la simulation. Ce premier DSL fait l'objet d'une section dédiée (voir 4.3.1). Un deuxième DSL est développé pour proposer une solution de modélisation comportementale formelle basée sur le MOF. Nous verrons dans la section 4.3.2 comment nous avons élaboré ce méta-modèle et quelles ont été nos motivations.

4.3.1 Développement utilisant le principe de profil UML

Cette section s'intéresse à l'élaboration de la structure du langage de simulation SiML. L'élaboration d'un profil ne s'improvise pas. La norme proposée par l'OMG [Uml 2013] décrit la spécification du *Profile*. Le premier profil UML standardisé par l'OMG est le profil CORBA [CORP 2002]. Les auteurs y proposent une méthode en quatre étapes pour élaborer un profil :

— **Identifier** un sous-ensemble de concepts du langage source nécessaire pour le profil cible.
— **Établir** les règles de cohérence en langage naturel ou à l'aide d'un langage de contraintes (e.g. OCL)
— **Définir** les nouveaux concepts (*"ajouts standards"*) rattachés au sous-ensemble sélectionné à l'aide des extensions proposées par UML (SysML dans notre cas) : *Stereotypes, Constraintes* et *Tagged Value*
— **Spécifier** la sémantique des nouveaux concepts en langage naturel.

C'est à partir de cette étude que le projet Accord [Accord 2002a] propose un comparatif des différents profils créés, pour extraire les règles émergentes de spécification issues des différents profils déjà disponibles et de leurs structures [Accord 2002b]. Accord était un projet destiné à proposer différents profils développés par différents acteurs, industriels et académiques (CNAM, EDF R&D, ENST, ENSTBretagne, France Telecom R&D, INRIA, LIFL et Softeam). Il leur était nécessaire de proposer une convention commune permettant d'élaborer des profils respectant un modèle générique de document de spécification. Les acteurs du projet Accord proposent de structurer le document en quatre étapes :

— **Description du domaine du méta-modèle.** Dans notre cas, nous nous référons aux premiers chapitres de cette thèse qui présentent le domaine de la M&S.
— **Définition technique du profil.** La suite de cette section va proposer une vue partielle du profil d'un point de vue technique.
— **Élaboration de la partie opérationnelle du profil** [12] (e.g. règles de transformation, règles de génération automatique de code). L'élaboration d'une partie opérationnelle, respectant les spécifications de l'OMG, n'est possible qu'en utilisant le principe de transformation (sémantique dénotationnelle). Comme indiqué précédemment, notre choix s'est porté sur une approche basée méta-langages comportementaux pour l'élaboration de la partie opérationnelle. Cette étape fait référence à la section suivante 4.3.2 dédiée à la méta-modélisation formelle.
— **Illustration du profil.** Cette dernière étape sera détaillée dans le prochain chapitre (cf chapitre 5) intitulé "Modèle et application de la méthodologie", qui illustrera l'utilisation du profil.

12. Cette partie n'apparaît pas dans la définition de profil UML.

Le but étant ici de montrer la faisabilité d'une telle description, et d'en apercevoir les bénéfices, nous ne proposons pas une description conventionnelle du profil SiML mais une vue diluée tout au long de cette thèse. Cette section va apporter une vue d'ensemble d'éléments techniques du profil, avec quelques éléments définis de façon détaillée (e.g. contraintes OCL).

SiML peut être considéré comme un profil SysML, qui est lui-même un profil UML (figure 4.5). Nous verrons par la suite que SiML utilise également un langage externe à UML lui aussi basé sur le MOF (section 4.3.2).

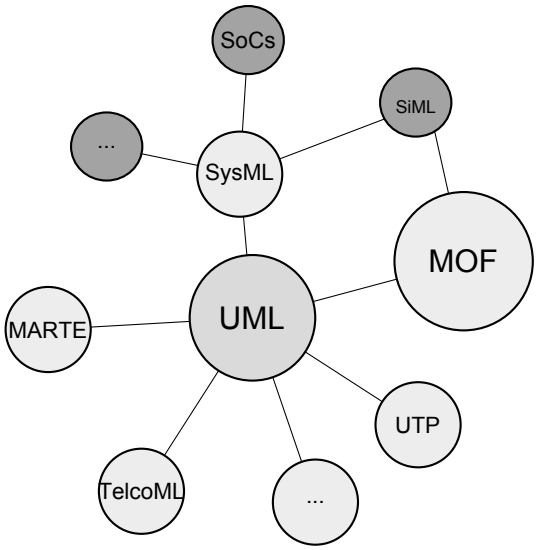


FIGURE 4.5 – Ecosystème en lien avec SiML

Le profil SiML est composé d'un seul package. Il est complété avec le package DEVS, regroupant l'ensemble des éléments nécessaires à la déclaration d'un comportement formel basé sur le formalisme DEVS (voir figure 4.6).

La figure 4.7 présente une vision haut-niveau du profil SiML. Nous retrouvons sur cette figure les différentes entités mentionnées durant ces travaux. Entre autres, envisageons le quadruplet modèle/générateur/transducteur/accepteur, qui peut être associé soit à une modélisation formelle *FormalSpecification* (modèle DEVS ou automates à interface), soit aux différents diagrammes comportementaux de SysML. Une capture des hypothèses s'associe à ces comportements. Elle peut être de trois types : opérationnelle, de modélisation ou rationnelle. Notons que ces hypothèses doivent interagir sur le modèle comportemental ou directement sur les différents paramètres de la simulation. Il existe également une notion non implémentée, associée au niveau hiérarchique du modèle, en fonction de l'objectif de simulation, reprenant l'idée de capacité du modèle (morphisme et dérivabilité). Pour rappel, un modèle a une capacité donnée qui est fonction de l'objectif de simulation, et sera plus ou moins capable selon l'objectif.

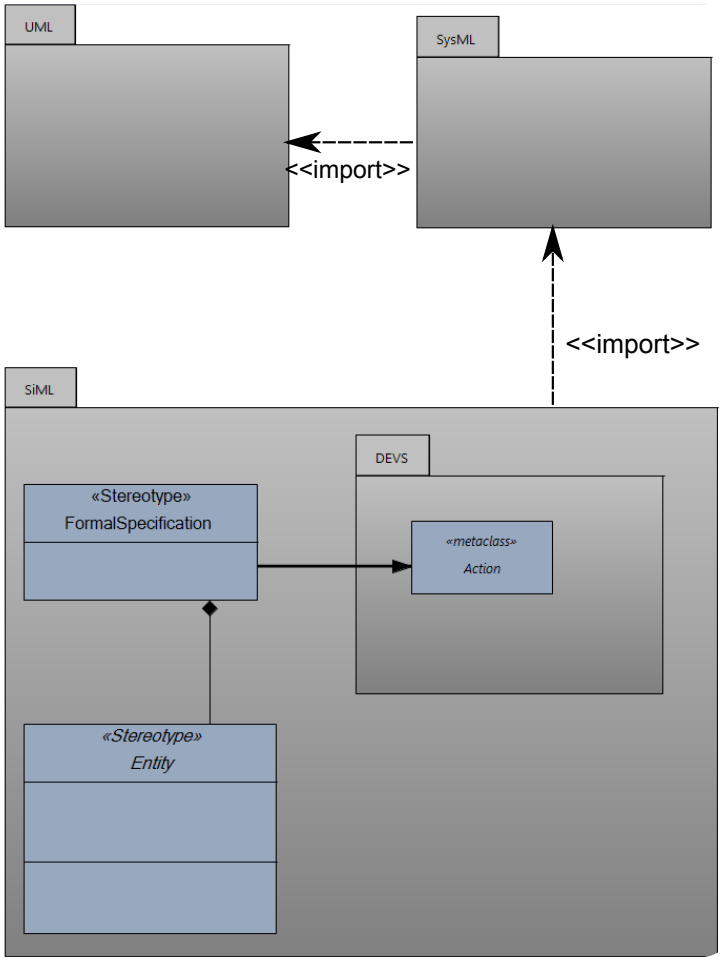


FIGURE 4.6 – Architecture SiML

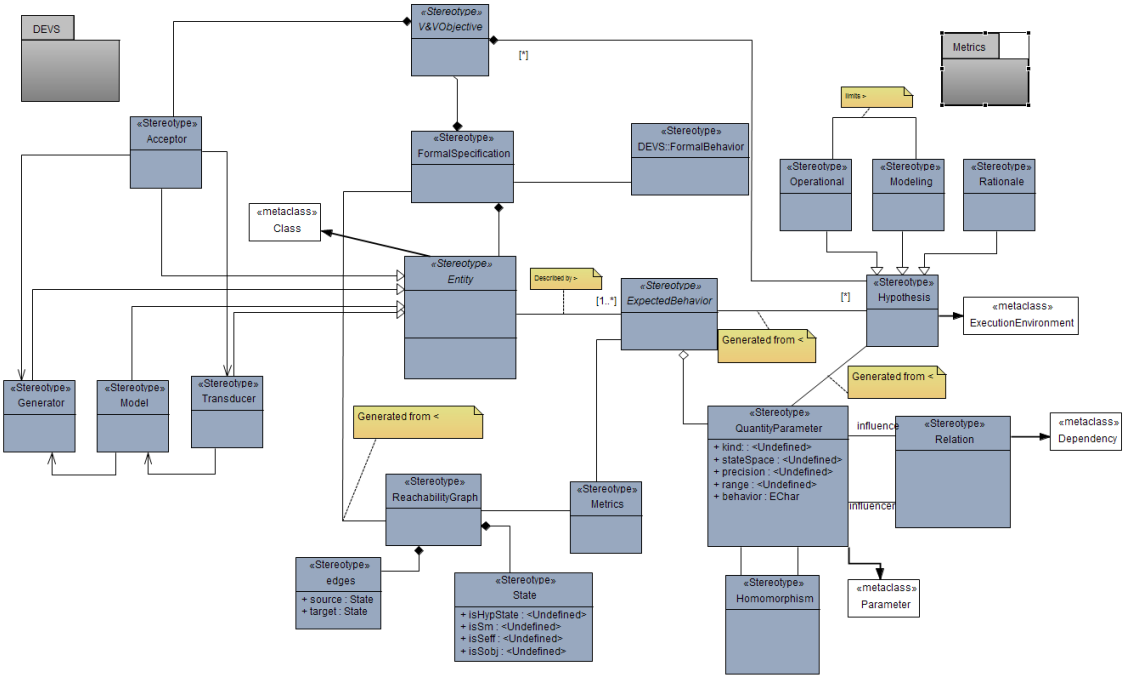


FIGURE 4.7 – Vue simplifiée du profil SiML.

Il apparaît dans la figure 4.7 un usage des liens d'association entre stéréotypes. Habituellement, l'usage d'associations est limité, voire proscrit dans l'élaboration d'un profil. Il est même possible de retrouver dans la littérature des passages stipulant que "*[...] des stéréotypes ne peuvent participer à des relations d'association.*"(Extrait de [Accord 2002b]). Il est intéressant d'observer que certains profils respectent cette particularité. L'Unified Testing Profil (UTP), déjà présenté, nous offre la possibilité de comparer l'approche basée MOF, avec l'approche basée Profile. Dans l'approche basée MOF de l'UTP, apparaissent clairement des liens d'association entres concepts, alors que l'approche Profile ne fait apparaître aucun lien d'association entre stéréotypes (nouveaux concepts). Cependant, afin de respecter une sémantique équivalente à l'approche basée MOF, les liens entres concepts sont décrits en langage naturel dans la sémantique des stéréotypes. Les évolutions du standard UML (neuf versions depuis les travaux du projet Accord), font maintenant apparaître clairement la possibilité d'utiliser les liens d'association entre stéréotypes : *"Stereotypes can participate in binary associations. The opposite class can be another stereotype, a non-stereotype class that is owned by a profile, or a metaclass of the reference metamodel."* Extrait de [Uml 2013]. Ceci est confirmé par le mapping proposé entre CMOF [13] et Profile (toujours dans la spécification UML).

Afin de mieux guider l'utilisateur, il est possible d'ajouter des WFR (Well Formed Rules), autrement appelées règles de bonne formation. L'idée est ici de préciser la sémantique du méta-modèle, afin de garantir la cohérence de la construction si celle-ci n'a pas encore été capturée. Comme nous l'avons vu dans la section 4.2.2.1, l'OMG préconise l'utilisation de contraintes OCL [OMG 2004] pour étendre la sémantique axiomatique d'un langage. Par exemple, un objectif de simulation doit toujours être associé à au minimum un modèle.

```
1 context v&vobjectives inv:
2 self.Entity.Model -> size() >= 1
```

Listing 4.1 – Définition de WFR, contrainte OCL

Cette approche nous a permis de définir un langage semi-formel, guidant l'utilisateur de la simulation dans l'élaboration de son cadre expérimental. A ce jour, l'utilisation du profil est limitée. L'absence d'une implémentation complète et d'une syntaxe concrète adaptée rend le profil peu convaincant. L'effort nécessaire à l'élaboration d'un tel profil a été sous-estimé et demandera des travaux complémentaires concernant l'ajout de règles de bonnes formations et un travail plus important sur la gestion des hypothèses [14]. La nécessité d'élaborer une description formelle du comportement du modèle et du cadre expérimental a rendu difficile l'étude détaillée du profil SiML. La section suivante présente l'élaboration d'un sous-ensemble formel dédié à la description comportementale de la simulation.

13. Pour Complete MOF, en opposition à EMOF pour Essential MOF.

14. Nous retrouvons ici un problème lié à l'absence de langage d'hypothèses permettant les inférences (vu dans la section 3.6.3)

4.3.2 Développement utilisant le principe de méta-modèle formel : Approche avec le formalisme DEVS

Cette section est issue de la collaboration mise en place durant cette thèse avec le laboratoire "*Arizona Center for Integrative Modeling & Simulation (ACIMS), Arizona State University*" et le professeur Hessam Sarjoughian, co-directeur de l'ACIMS.

Pour modéliser le comportement d'un modèle DEVS, de nombreux procédés peuvent être utilisés. Un modèle peut, par exemple, être décrit en utilisant la notation de la théorie des ensembles, des diagrammes UML, et encore du pseudo code. Chacun de ces modèles a un but et a ensuite besoin d'être mis en correspondance avec un code exécutable. Un modèle mathématique est utile pour définir la structure et le comportement, indépendamment des outils de simulation (PIM [15]). Compte tenu d'un simulateur cible, les diagrammes de classe et/ou les machines à états sont utiles pour la conception de modèles de simulation complexes. De toute évidence, chacune de ces méthodes a ses forces et faiblesses et aucune ne peut être considérée comme contenant toutes les connaissances nécessaires pour un code de simulation exécutable.

L'idée est ici de proposer un formalisme :

— permettant une description indépendante des outils de simulations (PIM),
— accordant une facilité de description des systèmes complexes,
— proposant une vérification automatique de la bonne construction des modèles (respect du formalisme),
— proposant une validation automatique du comportement des modèles élaborés.

L'IDM apporte ici une réponse claire car chaque modèle est bien construit (conforme au méta-modèle). Une première approche a été proposée par H. Sarjoughian dans [Sarjoughian 2012]. Il utilise une approche basée IDM avec l'utilisation de l'outil Eclipse EMF [Steinberg 2008], permettant la vérification automatique des propriétés structurelles des modèles DEVS par consistance avec le méta-modèle. Les limites de EMF n'autorisent pas le développement d'une solution permettant la validation automatique de l'aspect comportemental du modèle. Cet outil implémente EMOF [16] qui ne permet pas la description comportementale du méta-modèle. Cela limite donc dans la description du méta-modèle à sa sémantique statique (résultat équivalent à [Breton 2001]).

Lors de l'état de l'art sur la méta-modélisation, une solution permettant de modéliser l'aspect dynamique du méta-modèle est apparue. L'utilisation d'un méta-langage comportemental permet de s'intéresser à la conformité du modèle avec le méta-modèle, étape essentielle à la validation du modèle de simulation.

15. Plateform Independant Model
16. rappel : Essentiel MOF.

Un méta-langage comportemental est composé d'une partie comportementale (*meta-action language*) et d'une partie structurelle (*meta-structural language*). Il permet l'élaboration d'un méta-modèle à deux aspects : méta-modèle structurel et meta-modèle comportemental. Ce langage permet alors l'élaboration de modèles exécutables de manière indépendante de l'outil de simulation. Ce principe est illustré par la figure 4.8

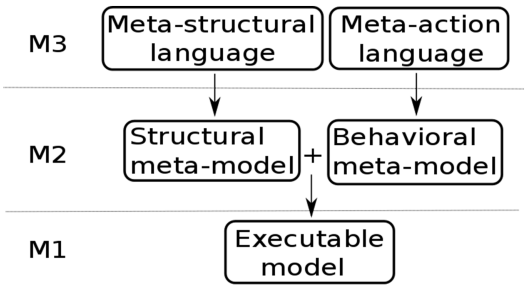


FIGURE 4.8 – Méta-modélisation utilisant un méta-langage

La création d'un méta-langage comportemental utilise le principe de promotion. La figure 4.9 illustre le processus de fabrication. Le MOF décrit un méta-modèle comportemental (Action Meta-model) et le MOF lui-même. Cet ensemble peut alors être combiné et promu au rang de méta-langage, la seule contrainte liée à la promotion étant de pouvoir s'auto-définir. Il est alors possible d'utiliser ExecutableMOF pour décrire l'aspect structurel et comportemental d'un méta-modèle (voir figure 4.8).

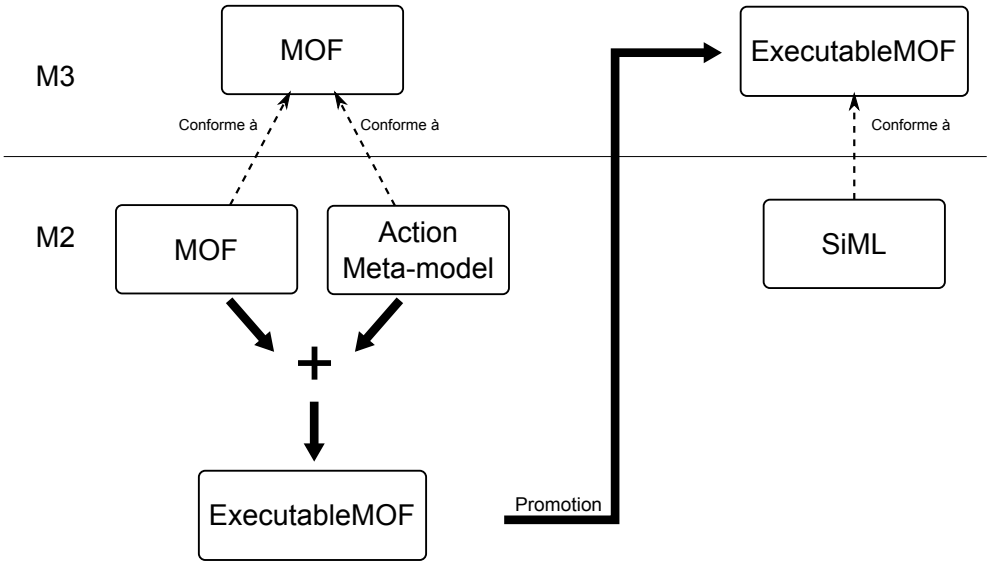


FIGURE 4.9 – Principe de création d'un méta-langage comportemental (adapté de [IRISA 2015])

C'est sur ce principe que se base Kermeta, en proposant une extension basée sur l'EMOF permettant la description comportementale au niveau méta-modèle. Il est considéré comme un langage de méta-modélisation exécutable, ou encore de méta-programmation objet [Combemale 2006a].

Les travaux proposés dans cette section sont basés sur la version 2 de Kermeta (K2) [17] [18] . Kermeta est composé de quatre langages fournissant les outils nécessaires à la méta-modélisation (voir figure 4.10).

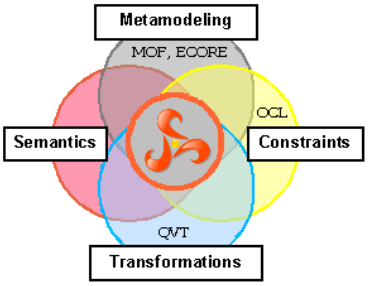


FIGURE 4.10 – Position de Kermeta (extrait de [IRISA 2015])

Kermeta est composé de quatre ensembles :

— ECORE : Méta-langage de description structurelle ;
— Semantics : Méta-langage de description comportementale ;
— QVT : Langage de transformation *model2model* et *model2text* ;
— OCL : Langage de contrainte.

Nous allons nous intéresser ici à la description comportementale du cadre expérimental et du modèle utilisant le formalisme DEVS, et plus particulièrement FD-DEVS (pour les raisons évoquées au chapitre 3). Toute démarche de modélisation commence avec l'élaboration de la structure du méta-modèle.

Le nombre de méta-modèles FD-DEVS proposés dans la littérature est important ([Pasqua 2012], [Mittal 2007], [Yonglin 2009], etc.). Ainsi, il devient nécessaire de fournir une base de réflexion commune, avec l'élaboration d'un méta-modèle DEVS cohérent avec les travaux déjà présent dans la communauté. Confrontés à des choix de conception, les développeurs proposent des méta-modèles différents. Une description unique du méta-modèle DEVS devient nécessaire : elle permettra à chacun de travailler à partir d'un standard, et offrira une possibilité d'interaction entre les différents outils de simulation basés sur le formalisme DEVS. Dans cette

17. A l'heure où ces lignes sont écrites, l'équipe DiverSE (ex-équipe Triskell) propose une nouvelle version de Kermeta, K3 [Degueule 2014], basée sur Xtend[18] et gommant certains problèmes associés aux précédents choix technologiques qui ne permettaient pas le respect de la totalité des principes de l'IDM. Sur l'aspect technologique, c'est une totale refonte qui est proposée dans cette version, mais les concepts associés sont comparables.

18. Xtend peut être considéré comme un dialecte Java. [Efftinge 2012]

optique, le réseau DEVS (RED), créé en Septembre 2014, soumet l'idée de l'élaboration d'un méta-modèle DEVS unique. D'autres initiatives vont dans ce sens, comme [Wainer 2010] ou encore [Sarjoughian 2011], en proposant les bases pour l'élaboration d'une norme DEVS.

L'absence de norme ou de standard pour le méta-modèle DEVS nous amène à présenter une nouvelle syntaxe abstraite des modèles DEVS, incluant des choix arbitraires de conception. La figure 4.11 présente la structure d'un modèle atomique FD-DEVS utilisant la syntaxe concrète des diagrammes de classe UML, extraits de la représentation Ecore classique, mais dont la lecture est moins aisée (voir figure 4.12). Nous retrouvons dans cette figure l'ensemble des éléments évoqués dans la présentation mathématique du formalisme au chapitre 3. Comme expliqué précédemment, cette structure seule est insuffisante pour évoquer toute la sémantique du DSL. Il est alors nécessaire d'ajouter un ensemble structurel supplémentaire, décrit comme un phénomène d'expansion nécessaire par E. Breton [Breton 2001].

Pour définir la sémantique opérationnelle d'un modèle atomique, nous avons du ajouter de nouveaux concepts au méta-modèle : t_L pour représenter le temps du dernier évènement, t_N pour représenter le temps du prochain évènement et $currentTime$ permettant une mise à jour du temps. Kermeta est un langage orienté aspect (POA [19]), ce qui permet de bien séparer les différents points de vue sur le méta-modèle (la structure, le comportement et l'interaction). Cette extension de la sémantique inclut également un ensemble d'opérations. Un des points forts du formalisme DEVS est de proposer une séparation forte entre la description des modèles et la description du simulateur abstrait. L'ensemble d'opérations nécessaire à la description du comportement des modèles est déjà proposé par B.P. Zeigler dans [Zeigler 2000]. L'exécution de ces opérations permet l'évolution du modèle DEVS d'un état à l'autre. L'opérateur $run()$ permet de lancer le simulateur et les opérateurs associés. $step()$ va alors permettre de calculer la date du prochain évènement dans le modèle atomique. Nous retrouvons ici le fonctionnement du simulateur abstrait tel que défini dans [Zeigler 1976].

19. Programmation Orientée Aspect

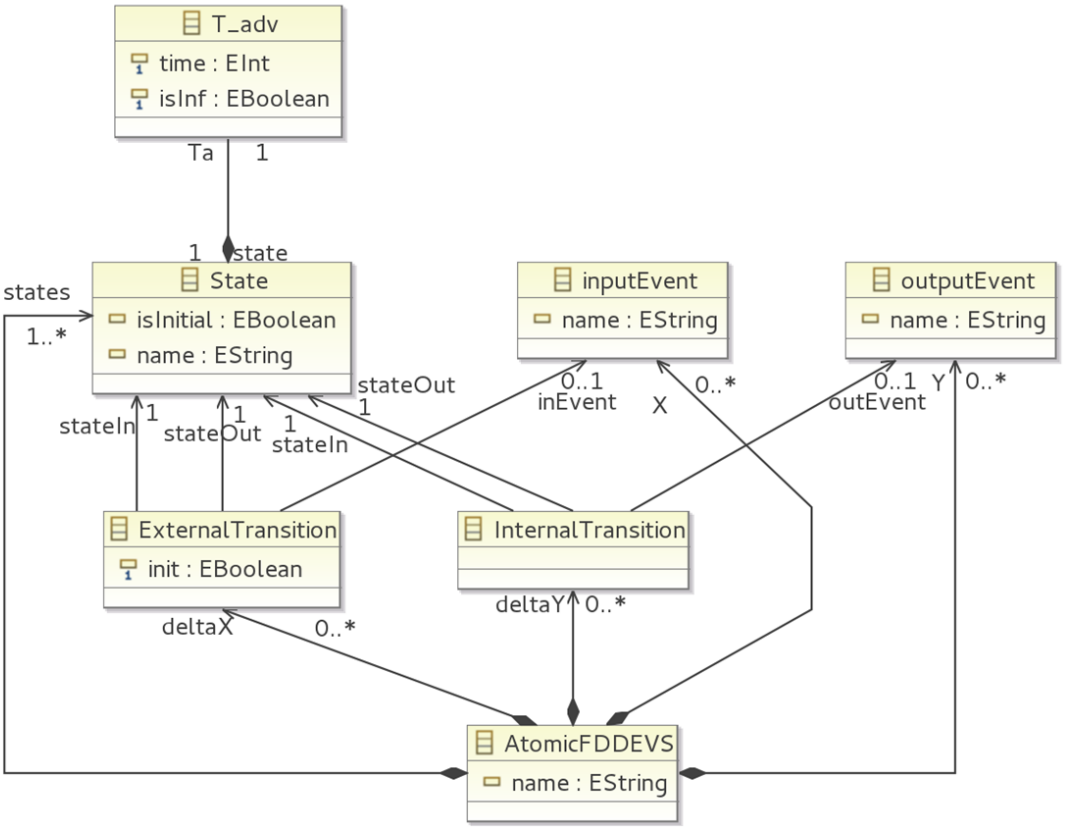


FIGURE 4.11 – Meta-modèle Atomic FD-DEVS

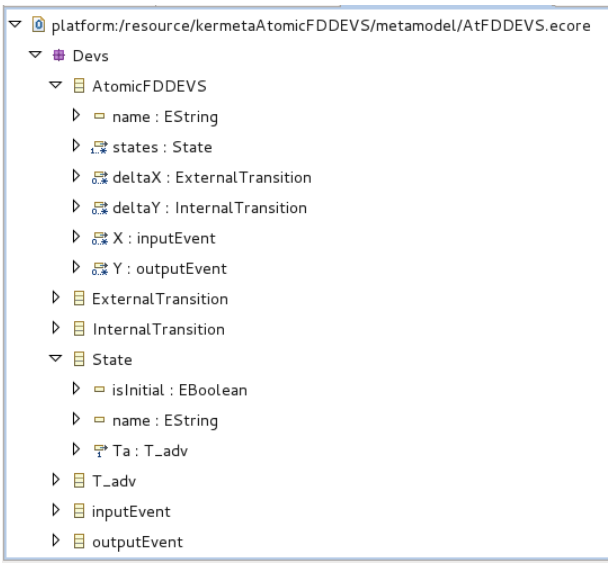


FIGURE 4.12 – Représentation du méta-modèle FD-DEVS en utilisant la syntaxe concrète Ecore (Équivalence stricte avec la figure 4.11)

Le listing 4.2 propose un fragment de code implémentant le protocole de simulation DEVS. A travers l'ajout d'un aspect à la classe *AtomicFDDEVS*, il est possible de définir le comportement légitime d'un modèle DEVS (e.g. transition, génération de sortie). Des fonctions d'interaction avec l'utilisateur apparaissent également (l7), elles ne devront pas être prises en compte dans le méta-modèle, mais elle facilitent le développement.

```
1  aspect class AtomicFDDEVS
2  {
3     ...
4     operation step(time: Integer)
5     ...
6     dy:=self.deltaY.detect{s|s.stateIn==currentState}
7     stdio.writeln("output: "+dy.outEvent.name)
8     self.currentState:=State.clone(dy.stateOut)
9     tL:=tN
10    tN:=addTime(tL,currentState.Ta.time)
11    end
12 }
```

Listing 4.2 – Protocole d'exécution

Il est alors possible d'élaborer des modèles conformes au méta-modèle utilisant le langage concret d'Ecore. La figure 4.13 reprend l'exemple du toaster proposé au chapitre 3 et illustré dans la figure 3.15. Nous retrouvons deux états (Idle et Toast), avec le temps Ta associé pour griller. Le modèle proposé ici peut, par contre, être directement vérifié. Le comportement du simulateur n'est plus décrit au niveau de la plateforme de simulation, mais bien dans le méta-modèle, ce qui permet une validation automatique du comportement du modèle.

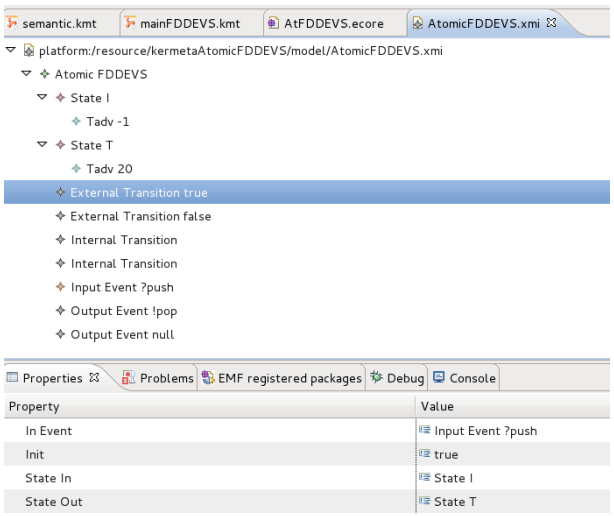


FIGURE 4.13 – Exemple du toaster

Kermeta propose une plateforme de simulation intégrée à l'outil, donnant la possibilité de simuler le modèle (voir exemple toaster figure 4.14). L'ajout d'un aspect "interaction" est nécessaire pour pouvoir agir avec le modèle (génération de push et observation de pop). Un modèle autonome, lui, n'aura pas besoin d'interactions (eg. cadre expérimental + modèle)

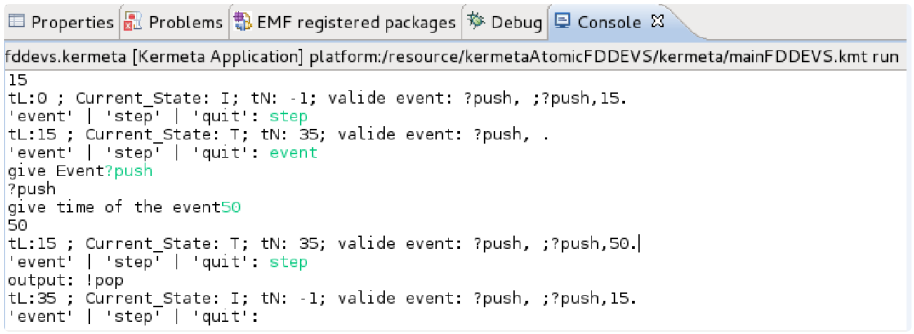


FIGURE 4.14 – Exécution du modèle toaster conforme à la spécification

Cet exemple montre la faisabilité d'une telle approche. Cependant, d'un point de vue pragmatique, il est fastidieux de définir un modèle par cette approche. Au-delà de la syntaxe concrète Ecore, peu adaptée à la définition de modèles, l'absence d'un langage d'action propre à DEVS limite l'utilisation du langage. Si l'on observe les différents simulateurs DEVS implémentés, tous utilisent le formalisme DEVS et son approche systémique mais ajoutent une part non définie dans le formalisme DEVS. Comme l'illustre la figure 4.15, lors du passage vers le simulateur, le formalisme DEVS est accompagné d'actions. Le simulateur abstrait DEVS ne décrit pas comment doivent être gérées ces actions. Ceci se traduit dans le protocole de simulation par une *boîte noire* "action", appelée durant l'exécution de la simulation. Le simulateur ne contrôle généralement pas ce qui se passe à l'intérieur de cette boîte noire. Suivant le simulateur, une action dans un modèle atomique peut : modifier des variables externes, générer de nouveaux évènements, ou encore changer le temps passé dans l'état. La plupart des simulateurs DEVS actuels ([Quesnel 2007], [Capocchi 2011], [Franceschini 2014]) ont été développés dans un cadre où le niveau de validation exigé est faible. Ce type de comportement est donc tout à fait acceptable. Dans un contexte de développement de systèmes complexes critiques, où le niveau d'exigence de la validation est fort, ceci n'est plus acceptable. Il devient nécessaire de s'assurer que le comportement associé aux actions est en accord avec le formalisme, ceci afin d'assurer une validation automatique du comportement du modèle. En d'autres termes, il faut s'intéresser à cette "boîte noire" pour vérifier que l'utilisateur respecte le formalisme durant les actions.

Pour contrôler le comportement associé à cette boîte noire, il est possible d'ajouter au formalisme DEVS un langage d'action. L'approche IDM nous a permis d'importer le langage d'action Logo [Feurzeig 1969] (voir figure 4.16).

Défini en utilisant Kermeta par l'équipe Triskell, KMLogo [Jézéquel 2011] nous

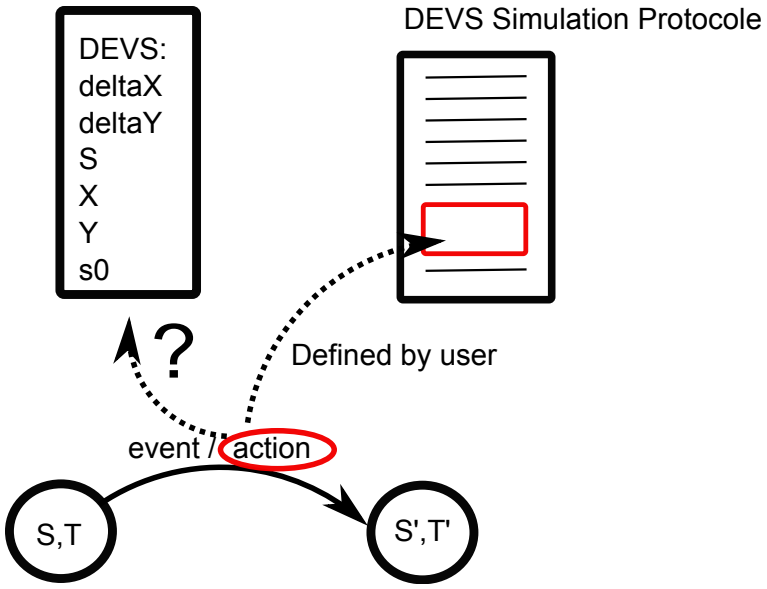


FIGURE 4.15 – Ajout d'un comportement "Action" dans le formalisme DEVS

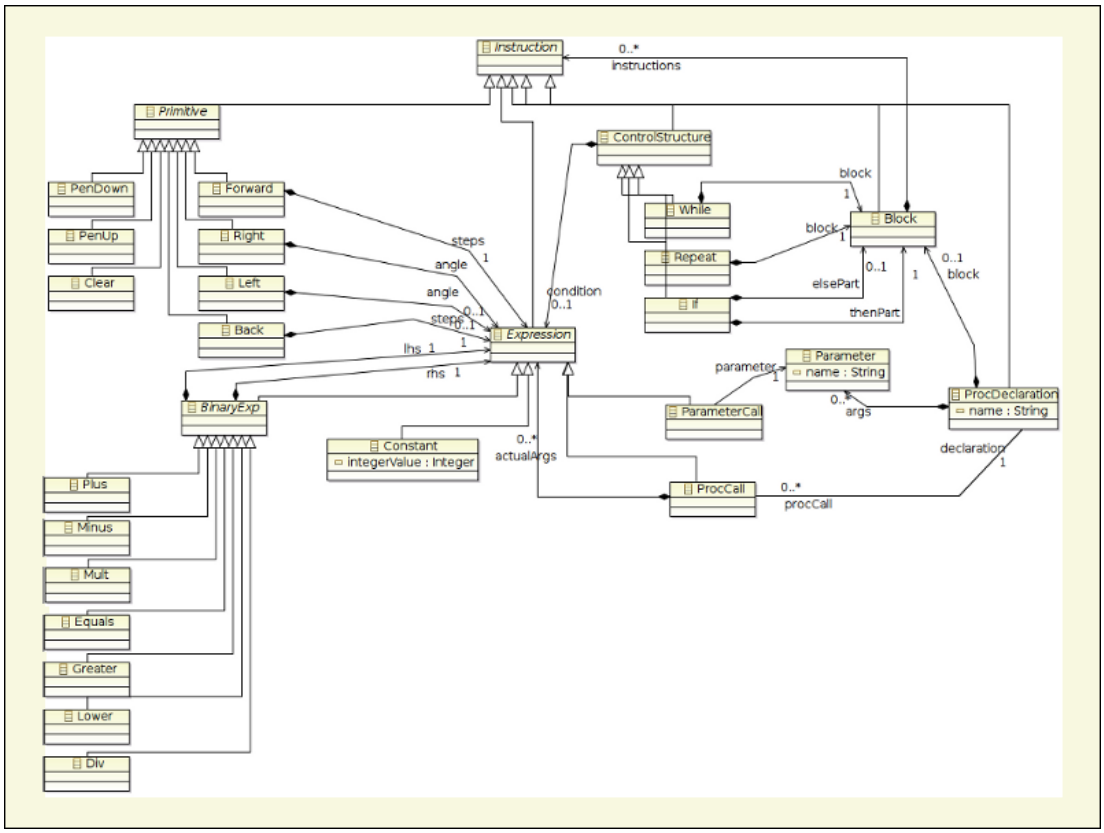


FIGURE 4.16 – Méta-modèle du langage KmLogo (Extrait de [IRISA 2013])

permet d'apporter une solution rapide à cette problématique, mise en lumière lors de notre collaboration avec le laboratoire ACIMS. Logo est un langage de programmation orienté objet réflexif permettant (malgré son image enfantine [20]) de décrire les actions les plus courantes. La figure 4.17 propose une vision Ecore où apparait le langage d'action associé au formalisme DEVS. L'ajout de contraintes dans le méta-modèle Logo nous permet de limiter et donc de contrôler les possibilités offertes à l'utilisateur lors des actions.

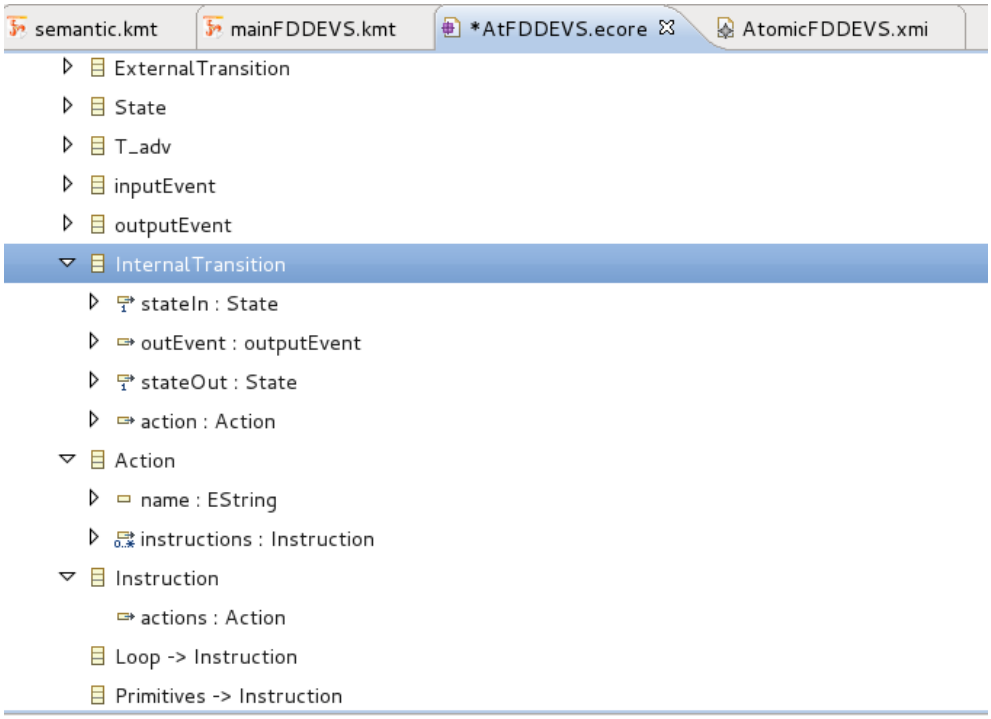


FIGURE 4.17 – Méta-modèle DEVS+KmLogo

Comme nous l'avons indiqué précédemment, Kermeta est défini en utilisant le langage Ecore, mais il a la particularité de comporter en son sein Ecore lui-même. Il est donc possible de rapatrier la totalité du langage SiML (défini en utilisant Ecore) au travers d'une description en Kermeta. Cela permet d'obtenir un langage SiML cohérent, au sein d'un formalisme unique. La figure 4.18 propose un résumé de l'approche IDM pour l'élaboration du langage SiML, à partir du méta-langage Ecore.

20. Logo est souvent associé à l'utilisation d'une tortue permettant aux débutants de visualiser le résultat d'un programme, d'où une image faussement puérile.

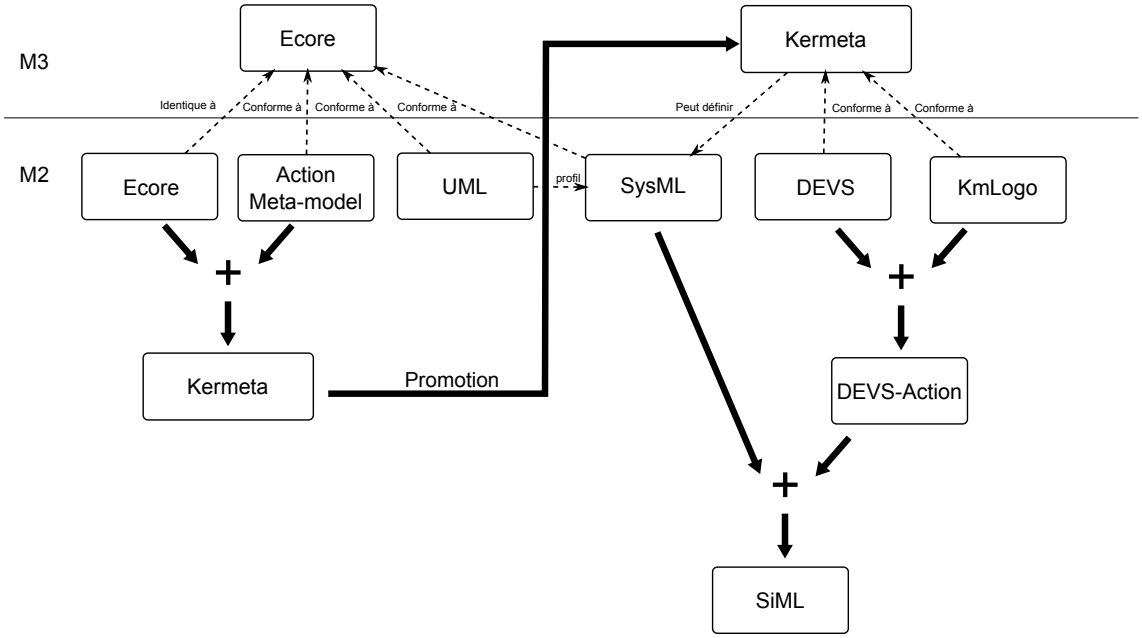


FIGURE 4.18 – Élaboration du langage SiML dans l'environnement Eclipse Ecore

De récents travaux sur la vérification formelle des modèles ont étudié les avantages de l'approche IDM. Ils se sont intéressés aux méta-langages et plus particulièrement à Kermeta.

M. Amrani propose dans [Amrani 2012] une preuve mathématique affirmant que la sémantique de Kermeta est formelle. Toujours selon M. Amrani, les méta-modèles et les modèles résultants conformes au méta-langage sont eux aussi formels. C'est un résultat logique, puisque l'intérêt d'un méta-langage comme Kermeta est d'éliminer l'utilisation du langage naturel, sujet à interprétation, dans la description de la sémantique.

Cette section a montré que des solutions existaient pour définir la sémantique d'exécution au niveau méta-modèle. Afin de s'adapter aux différentes situations, on assiste à la multiplication des simulateurs DEVS, fournissant chacun une interprétation des simulateurs abstraits proposés par B.P. Zeigler. Il devient nécessaire [RED 2015] de valider le comportement des différents simulateurs. Le réseau RED propose la mise en place d'une certification [21] des simulateurs. A terme, l'utilisation d'un méta-modèle formel, unique et complet, peut permettre, par simple transformation, de s'adapter à toute plateforme, assurant ainsi un comportement unique au modèle, quelle que soit la plateforme d'exécution choisie.

21. Un simulateur est dit certifié s'il fournit des résultats cohérents, soit en comparaison à des données expérimentales fournies, soit en comparaison à d'autres simulateurs certifiés. Cette méthode doit s'appuyer sur un nombre important de modèles pour être efficace. Une mauvaise interprétation du modèle à définir peut également être un frein vers la certification.

4.4 Outils de vérification et de simulation à évènement discret

Les possibilités de transformation offertes par l'IDM donnent accès aux différents outils de simulation et de vérification existants, moyennant un effort contrôlé.

L'IDM parle de passage d'un modèle indépendant de la plateforme à un modèle dépendant (PIM vers PSM[22]), au travers de mécanismes de transformation. La figure 4.19 illustre le processus de développement en Y de l'IDM. Partant d'une description haut-niveau des exigences (CIM[23]), il est ensuite développé une solution indépendante de toute considération technique (PIM). Un ensemble de règles de transformation, associé au modèle de description de la plateforme, va permettre de générer un modèle dédié à une plateforme spécifique (PSM).

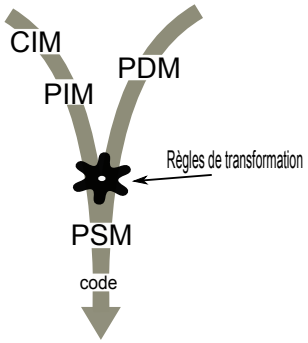


FIGURE 4.19 – Processus en Y de l'IDM

Dans cette section, nous proposons de comparer différents outils basés évènement, fournissant pour la plupart les briques élémentaires nécessaires à l'élaboration de nos métriques, à savoir : la génération et l'accessibilité du RG, la gestion des *guards*, variables et autres actions, la gestion des réels, la connaissance du temps écoulé dans l'état, la discrétisation du temps passé dans l'état (t_e), la possibilité de modification du t_a.

Les outils proposant l'élaboration du graphe de classes à partir du formalisme DEVS sont peu nombreux. Il est cependant possible d'utiliser le mécanisme de transformation proposé par l'IDM pour s'orienter vers d'autres formalismes. Un modèle DEVS peut, par exemple, être transformé en automate temporisé si le modèle DEVS comporte un nombre fini d'états discrets [Dacharry 2005]. L'utilisation de FD-DEVS est donc compatible avec les possibilités de transformation proposées, ce qui nous permet de regarder également les outils basés sur les automates temporisés.

22. PSM : Platform Specific Model
23. CIM : Computation Independant Model.

Propriété \ Outil	XSY	UPPAAL	FIACRE*	ProDEVS
Accessibilité du RG	✓	✗	✓	✓
Gestion Guard	✓	✓	✓	✓
Gestion Variable	✓	✓	✓	✓
Gestion Action	✓	✓	✓	✓
Gestion Réel	✓	✗	✗	✓
Temps écoulé dans l'état (t_e)	✓	✗	✓*	✓
t_e Discrétisé	✓	✓	✓	✓
t_a dynamique	✗	✓	✓*	✓
Passage de paramètre	✓	✓	✓	✓
Orienté composant	✓	✓	✓	✓
Model-checking	✓	✓	✓	(✓)

TABLE 4.1 – Sélection d'outils de vérification et de simulation basés évènement

L'outil XSY, développé par H.M.Hwang [Hwang 2009], est une implémentation des algorithmes de génération de graphe de classes. Il propose la majorité des éléments nécessaires à l'élaboration des métriques. Cependant, comme nous l'avons vu dans la section 3.5.2, l'outil génère des times-zones supplémentaires et des transitions entre zones non-existantes, ce qui limite la crédibilité de l'outil dans l'état actuel des choses[24]. L'absence de la gestion dynamique de t_a n'empêche pas l'élaboration des métriques, mais limite les possibilités de modélisation. Nous pensons notamment ici aux méthodes numériques d'intégration basées QSS (Quantized State System)[Cellier 2006][Kofman 2001], qui nécessitent une gestion dynamique de la fonction t_a et qui permettent une modélisation efficace des systèmes hybrides [Bergero 2011].

L'outil UPPAAL [Bengtsson 1996] est en fait un ensemble d'outils dédié à la modélisation, la validation et la vérification des systèmes temps réel. Il utilise les automates temporisés et typés pour la modélisation des systèmes. L'outil est développé en collaboration entre le ministère des technologies de l'information à l'Université d'Uppsala, en Suède, et le Département d'informatique de l'université d'Aalborg au Danemark. L'absence de gestion des réels dans l'outil n'est pas un problème, puisqu'une *mise à l'échelle* à partir des entiers permettra de contourner ce type de problèmes. Le défaut majeur de cet outil (pour notre utilisation) vient de l'impossibilité d'obtenir le graphe d'atteignabilité dans son ensemble, ce qui est, de notre point de vue, très réducteur.

Le troisième outil étudié est Fiacre [Berthomieu 2008], un langage formellement défini, qui permet de représenter l'aspect comportemental des systèmes embarqués et distribués, dans un but de vérification formelle et de simulation. La vérification

24. Notons qu'il s'agit d'un outil académique en cours de développement.

formelle en Fiacre s'effectue via une transformation du modèle Fiacre vers un modèle TINA [25]. Ce langage n'offrait initialement pas la possibilité de gérer dynamiquement la fonction t_a. L'équipe [26] en charge du développement de l'outil TINA a su apporter une réponse rapide à ce manque, en proposant une extension des réseaux de Petri temporels, appelée Dynamic Time Petri Nets (DTPN) [Berthomieu 2014]. Cette extension permet une gestion dynamique de la date de tir d'une transition dans le réseau, ce qui fait de TINA un outil intéressant pour l'élaboration des métriques. Les modifications apportés par cette extension sont visibles par l'ajout de "✓*" dans le tableau 4.1.

Le dernier outil, nommé ProDEVS, a été développé par L.H. Vu en interne [27]. Initialement, ProDEVS était une réponse au manque d'outils permettant la description graphique du comportement des systèmes hybrides, à l'aide de composants DEVS couplés, mais aussi atomiques. Il est considéré dans cette thèse comme une réponse possible au manque de syntaxe concrète graphique pour le package DEVS du langage abstrait SiML. ProDEVS fournit ainsi un support concret à l'élaboration des métriques. A ce jour, une transformation vers les TPNs (DTPNs) nous permet d'obtenir le graphe de classes, ou la décomposition en arbre, à l'aide de la boîte à outils TINA basés sur les réseaux de Petri [Berthomieu 2004]. A l'instant où nous écrivons ces lignes la génération automatique de modèle TINA à partir de modèle ProDEVS n'est pas pleinement opérationnelle, c'est pourquoi le tableau comporte "(✓)" dans la propriété *Model-checking*

Nous avons participé au développement de l'outil ProDEVS, et plus particulièrement au choix de la syntaxe graphique permettant la description comportementale des composants atomiques. La syntaxe est basée sur les *state diagrams* (diagrammes État-Transitions). Ce choix est motivé par l'envie de proposer aux concepteurs des systèmes (modélisateur ou utilisateur de la simulation) un langage de spécification de haut niveau basé sur une représentation graphique. La syntaxe concrète, associée au couplage des composants, est basée sur le principe "boîte-flèche", permettant une description simultanée du flux d'informations et de la hiérarchie du système. Elle reprend la syntaxe graphique proposée par la plupart des outils de modélisation DEVS, comme [Kofman 2003], [Capocchi 2011], [Seo 2013] ou encore [Kim 2009]. La figure 4.20 présente l'exemple du ping-pong, vu en section 3.5.1.1, en utilisant la syntaxe concrète ProDEVS permettant de définir le couplage de composants. On peut voir sur cette figure deux composants atomiques (joueur A et joueur B) connectés à l'aide des ports *send* et *receive*.

25. TINA pour TIme petri Net Analyser, est un outil qui permet la construction et l'analyse de graphe de classes à partir d'une description en réseaux de Petri.
26. Équipe VerTICS du LAAS-CNRS.
27. Au sein du groupe Ingénierie Système et Intégration (ISI) du LAAS-CNRS

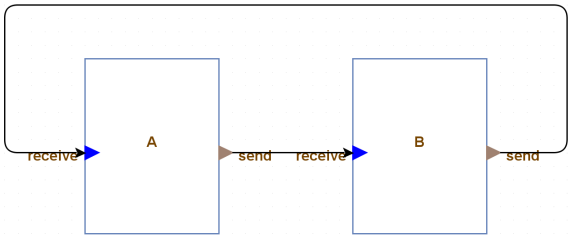


FIGURE 4.20 – Modèle Ping-Pong (vue "couplage")

La navigation hiérarchique de ProDEVS permet d'aller du composant couplé de plus haut niveau vers les composants atomiques, comme l'illustre la figure 4.21. Toujours en correspondance avec l'exemple du ping-pong de la section 3.5.1.1, nous retrouvons deux phases, *Wait* et *Send*, liées par des transitions orientées (!send et ?receive, images des fonctions δ_{int} et δ_{ext}).

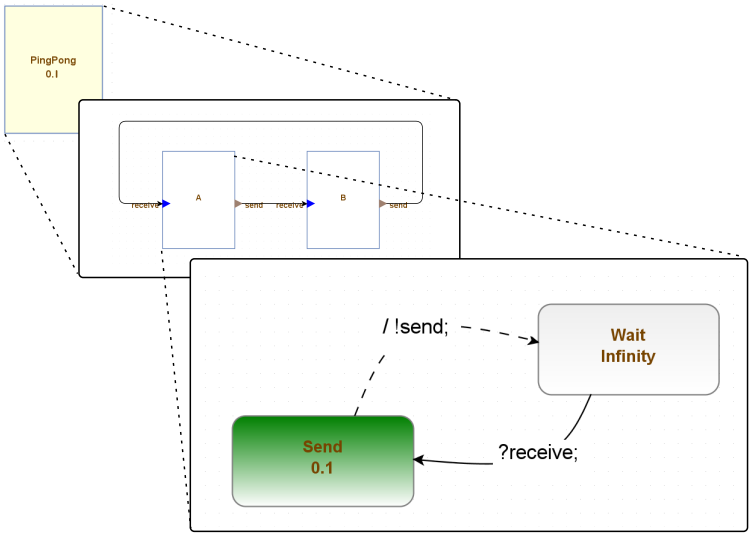


FIGURE 4.21 – Modèle Joueur A (vue atomique)

Une transition peut être associée à :

— un *trigger*, qui spécifie une entrée, celle-ci pouvant déclencher une transition externe, en cas d'évènement externe ou une sortie en cas d'événement interne (le temps autorisé passé dans une phase est atteint) ;

— un *garde*, qui fournit un contrôle sur le tir de la transition en utilisant des expressions booléennes. Un garde est évalué lorsque l'occurrence d'un évènement, interne ou externe, intervient. Si le garde est vrai à cet instant, la transition est autorisée, sinon elle est refusée ;

— une *action*, qui spécifie une mission particulière à effectuer (e.g. modification d'une variable) lors du tir de la transition ;

— une *source*, qui désigne la phase de provenance ;

— une *cible*, qui désigne la phase cible atteinte.

Le principe de *garde* et d'*action* n'est pas initialement proposé par le formalisme DEVS, mais répond à l'absence de langage "action" du formalisme présenté précédemment. Cette syntaxe graphique reprend en majorité les concepts énoncés dans UML [Uml 2013], et plus particulièrement le package BehaviorStateMachine. Cette approche de modélisation hybride (textuel + graphique) peut être considérée comme de la modélisation textuelle embarquée dans de la modélisation graphique [Scheidgen 2008]. Cette approche permet de proposer une interface graphique de modélisation multi-niveaux (couplés et atomiques) conviviale, ne nécessitant pas de compétences particulière en programmation.

4.5 Discussion : divergence implémentatoire, l'interprétation de la sémantique est-elle toujours décisive ?

Le terme *divergence implémentatoire* a semble-t-il été introduit pour la première fois par F. Michel dans [Michel 2004]. L'auteur utilise ce terme dans l'étude des simulations de systèmes multi-agents. Selon lui, le manque de spécification des modèles multi-agents n'autorise pas une implémentation non ambiguë du simulateur, et oblige à interpréter certains comportements du modèle. En d'autres termes, un modèle exécuté par deux simulateurs différents, peut donner des résultats différents selon l'interprétation qui en est faite. C'est ce que l'auteur appelle le *phénomène de divergence implémentatoire*.

Durant ces travaux, notre choix s'est tourné vers le formalisme DEVS, qui possède l'avantage de fournir une description formelle du modèle et de son comportement (au travers de la description du simulateur abstrait). En théorie, il n'y a donc aucune interprétation à faire pour implémenter un modèle DEVS. Cependant, en pratique, des choix existent. La *divergence implémentatoire* peut venir de plusieurs éléments, comme le type de variables utilisé (e.g. entier, flottant à simple précision ou à double précision étendue), ou la méthode de comparaison des flottants [Goldberg 1991].

Prenons un exemple plus détaillé. Les communications dans le formalisme DEVS sont faiblement synchrones : si le destinataire d'un évènement n'est pas prêt à recevoir le message à l'instant de son émission, le message est perdu. Le formalisme PDEVS [28] autorise l'exécution simultanée de composants, ce qui fait qu'un modèle atomique peut recevoir plusieurs évènements simultanément. Se pose alors la question de la gestion des évènements. Selon la définition formelle, l'arbitrage doit être effectué par la fonction *confluent()*(voir [Zeigler 1976]). Cependant, le modélisateur peut parfois omettre certaines déclarations de fonctions *confluent()*(e.g. il suppose que deux évènements ne peuvent pas être simultanés). En théorie, le simulateur doit alors arrêter la simulation. En pratique, l'implémentation du simulateur asso-

28. PDEVS pour Parallel DEVS

cie un comportement par défaut lorsque la fonction *confluent()* n'est pas spécifiée par l'utilisateur. Si le modèle reçoit un *bag* d'évènements, le simulateur peut réagir de différentes manières : choisir un évènement parmi n (aléatoirement ou décidé durant l'exécution par l'utilisateur de la simulation), ou faire un fork dans la simulation pour appliquer chacune des possibilités. La figure 4.22 illustre ces choix. En pratique, c'est souvent la solution *choix aléatoire* qui est implémentée. La seconde solution se rapproche davantage d'une méthode acceptable, mais qui peut devenir extrêmement coûteuse pour la simulation.

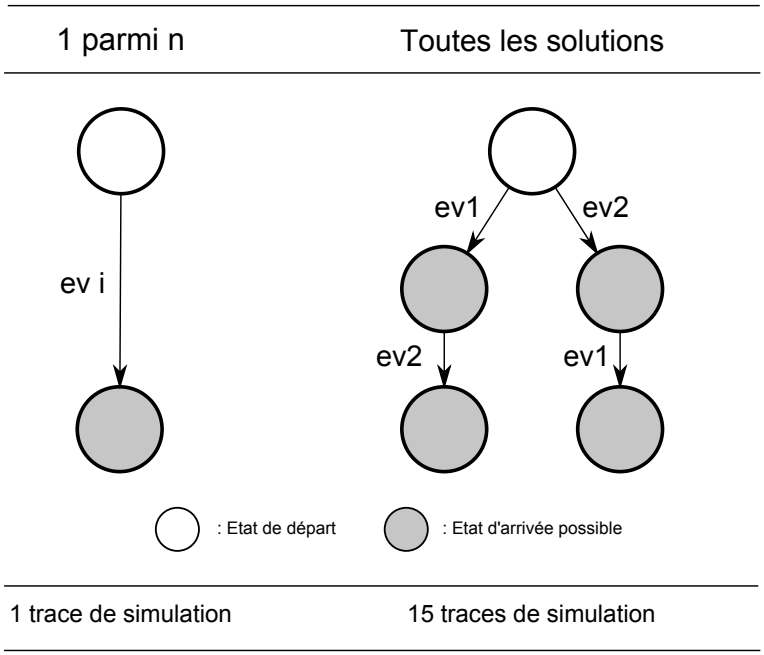


FIGURE 4.22 – Solutions pour la gestion d'un *bag* d'évènements

L'application d'un cadre expérimental va ainsi parfois amener à interpréter le comportement du modèle. Le simulateur perd son principe de neutralité. Il est alors légitime de se poser la question de l'impact des choix du simulateur sur les résultats, et donc de la crédibilité des résultats.

Il est possible d'imaginer que certains choix faits par le simulateur, ou certaines mauvaises interprétations de la sémantique du modèle par le modeleur, n'ont aucun impact sur les résultats de la simulation. Cette propriété découle directement de la relation de morphisme, décrite par B.P. Zeigler et abordée dans la section 2.4. Deux modèles distincts peuvent fournir des résultats de simulation "similaires" au regard des objectifs de simulation. Reprenons les évènements de la figure 4.22 et appliquons-les au modèle de la figure 4.23. Dans cet exemple, l'interprétation de la sémantique est inutile dans le cas où la seule variable d'intérêt est B. *A contrario*, la gestion des évènements sera décisive si l'on s'intéresse à la variable A.

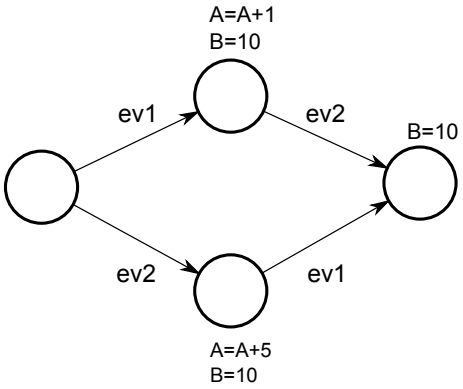


FIGURE 4.23 – Exemple d'application des évènements ev1 et ev2

Une définition formelle des objectifs de simulation et des hypothèses, comme cela a été proposé dans ces travaux, peut également permettre à l'utilisateur de la simulation, au développeur du modèle, ou même au développeur du simulateur, d'éviter certains choix sémantiques qui semblent à première vue indispensables, mais qui s'avèrent sans impact lorsque l'objectif de simulation est pris en compte. Cette idée peut donc permettre l'arbitrage de certains choix délicats ainsi que la génération automatique des modèles "abstraits".

4.6 Résumé

Ce chapitre a permis d'apporter les bases d'une réponse à la problématique posée par B.P. Zeigler, en proposant un outil et un langage dédié à la M& S.

L'idée de ce chapitre a été d'utiliser une approche IDM pour l'élaboration du langage de modélisation.

Une première partie a été dédiée à la création d'un langage haut-niveau. Ce langage permet d'aider l'utilisateur à appréhender la simulation et l'application du concept de cadre expérimental. Une seconde partie a proposé une description formelle d'un langage de modélisation comportementale basé sur le formalisme DEVS. Ce langage utilise le langage Kermeta, ce qui permet l'extraction de métriques pour l'étude de la validation des modèles de simulation.

Une dernière section a présenté une approche concrète pour l'élaboration des métriques, au travers d'outils de modélisation et de simulation. Ces outils sont capables de générer des graphes d'atteignabilité, et donc les métriques présentées au chapitre précédent.

Afin de compléter ce travail, nous allons illustrer les principes de cadre expérimental, de métriques de crédibilité, ou d'utilisation des outils de M&S. Cette présentation se fera au travers d'un cas pratique étudié dans le chapitre suivant.

Chapitre 5

Modèle et application de la méthodologie

Sommaire

5.1 Introduction

Pour illustrer les travaux de cette thèse, nous présentons une étude de cas dans ce dernier chapitre. Le système à étudier est une batterie. Il inspiré d'un exemple proposé par Y. Iwasaki et AY. Levy dans [Iwasaki 1994]. Cet exemple va permettre d'illustrer la méthodologie pour mieux comprendre l'utilité de la démarche proposée. Nous présenterons dans un premier temps le système à étudier avec sa spécification à différents niveaux d'abstraction. Nous tiendrons, à ce moment-là, le rôle de modélisateur du système d'intérêt. Dans un second temps, nous tiendrons le rôle d'utilisateur de la simulation, où nous complèterons le profil SiML présenté au chapitre 4, avec l'élaboration du cadre expérimental (définition des hypothèses de simulation). Nous pourrons alors étudier la compatibilité entre cadre expérimental et modèle telle qu'elle a été présentée au chapitre 2. Il sera alors possible d'étudier les métriques présentées au chapitre 3 dans la section Analyse. Pour ce faire, nous introduirons volontairement des erreurs de couplage ou de modélisation afin de détecter des incohérences dans la simulation, et donc de mettre en évidence l'utilité des métriques.

5.2 Cas pratique : Présentation du système

Le cas étudié ici est une simple batterie rechargeable, accompagnée de son panneau solaire (voir figure 5.1). Le système d'intérêt sera restreint à la batterie seule.

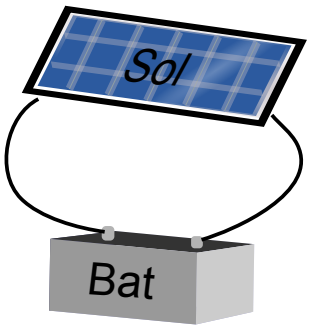


FIGURE 5.1 – Système batterie (Bat) + panneau solaire (Sol)

Une batterie ou batterie d'accumulateurs est un réservoir électrique qui a deux fonctions principales : emmagasiner de l'énergie sous forme chimique et restituer cette énergie sous la forme d'un courant continu adapté à la charge électrique connectée. En apparence simple, l'étude d'une batterie peut devenir complexe et fait encore l'objet de nombreux programmes de recherche pour améliorer ses performances et sa gestion ([Tremblay 2007], [Badey 2012], [Raël 2014]). Les batteries jouant souvent le rôle de réservoir dans les systèmes embarqués, leur étude reste aujourd'hui un enjeu important (e.g. l'autonomie des véhicules électriques).

La modélisation et la simulation de ces systèmes électrochimiques peut, par exemple, être utilisée pour établir une sélection optimale du composant "batterie" dans un système plus complexe (e.g. véhicule électrique [Badey 2012], [Tremblay 2007]). Comme nous l'avons déjà vu (c.f. chapitre 2), un modèle doit être suffisant pour répondre à la question que l'opérateur se pose à propos du système et capable de répondre de manière identique au système réel. Dans la littérature, un nombre important de modèles de batteries est proposé, du modèle idéal capable de fournir une tension constante sur une durée infinie, au modèle multiphysique prenant en compte de nombreux phénomènes, comme le vieillissement de l'électrolyte ou les changements de température de l'environnement de la batterie [Raël 2014].

Y. Iwasaki et AY. Levy proposent dans, *Reasoning with multiple abstraction models.* [Iwasaki 1994], un ensemble de modèles "batterie", organisé selon différents niveaux d'abstraction, répondant à différents objectifs de simulation. Les modèles sont divisés en trois classes d'hypothèses, regroupant les modèles ayant une influence sur les mêmes paramètres :

— **Battery-voltage-ac** : Regroupe l'ensemble des modèles ayant une influence directe sur la tension de sortie de la batterie.

— **Battery-charge-level-ac** : Regroupe l'ensemble des modèles ayant une influence directe sur le niveau de charge de la batterie .
— **Battery-damage-ac** : Regroupe l'ensemble des modèles ayant une influence directe sur l'état de santé de la batterie.

Toujours selon la classification proposée par Y. Iwasaki et AY. Levy, chacune de ces classes organise ses modèles selon un treillis d'abstraction [1]. Les modèles inclus dans ce treillis représentant le même phénomène physique, il suffit alors de sélectionner un modèle dans le treillis en fonction du niveau d'abstraction désiré.

5.2.1 Treillis d'abstraction des classes d'hypothèses

Avant de présenter les différents modèles et leurs treillis d'abstraction, il est nécessaire d'introduire la notation utilisée dans cet exemple :

— CL : Niveau de charge.
— C : Capacité.
— V : Tension de sortie de la batterie.
— TEMP : Température ambiante.
— I : Courant.
— DOD : Profondeur de décharge (Depth Of Discharge)
— TSLC : Temps depuis la dernière remise en état (Time Since Last Conditioning)
— Cond : Remise en état (Conditioning)
— TPOG : Temps d'utilisation prévu (Time Period Of Goal)
— Rel() : Définit les paramètres pertinents (Relevante) (nég : $\sim Rel()$)

5.2.1.1 Classe "Battery-voltage-ac"

Cette classe possède cinq modèles ayant une influence directe sur la tension de sortie de la batterie (voir tableau 5.1).

Modèle	**Comportement**	**Éléments d'influence**
Constant-voltage	$V = c_0$	Damaged(BAT)
Binary-voltage	$V = \begin{cases} 0 \ si \ CL < c_0 \\ v_1 \ si \ c_0 \leq CL \end{cases}$	Damaged(BAT)
Normal-degrading	$V = f(TIME)$	Damaged(BAT)
Charge-sensitive	$V = f(CL)$	Damaged(BAT), Rechargeable(BAT)
Temperature-sensitive	$V = f(CL, TEMP)$	Damaged(BAT), Rechargeable(BAT)

TABLE 5.1 – Modèle de la classe d'hypothèses "Battery-voltage-ac"

1. Un treillis (T) (ou lattice en anglais), est un ensemble ordonné tel que deux éléments quelconques x et y de T ont : un infimum (= borne inférieure) noté $x \bigwedge y$ et un supremum (= borne supérieure) noté $x \bigvee y$

Nous considérons ici qu'il existe cinq façons différentes de décrire le comportement en tension (V) aux bornes d'une batterie. Le comportement le plus simple consiste à fournir de façon permanente une tension constante (Constant-voltage). Il est possible de prendre en considération le niveau de charge (CL), en fournissant deux niveaux de tension suivant le niveau de charge (Binary-volatage). Une autre possibilité est de prendre en considération la décharge naturelle de la batterie à travers le temps (Normal-degrading). Il est possible de combiner ces deux modèles (Charge-sensitive), ou encore d'y ajouter une sensibilité à la température (Temperature-sensitive).

Ceci permet de définir un treillis d'abstraction des différents modèles proposés dans la classe "Battery-voltage" (voir figure 5.2).

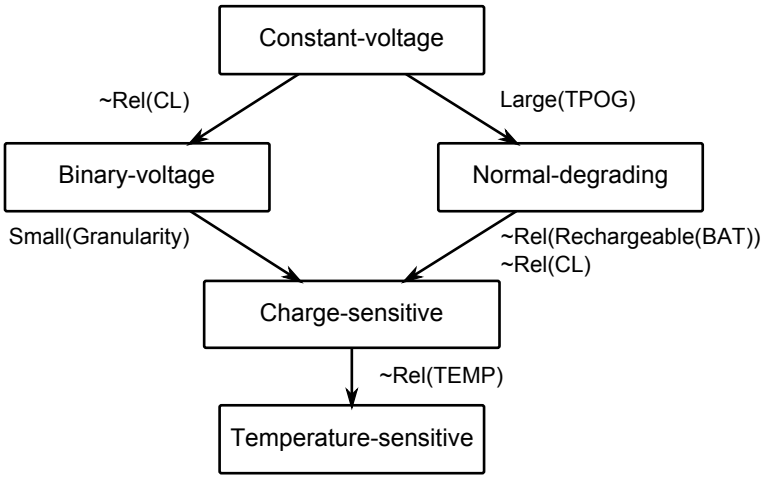


FIGURE 5.2 – Treillis d'abstraction de la classe "Battery-voltage"

5.2.1.2 Classe "Battery-charge-level-ac"

Cette classe possède trois modèles ayant une influence directe sur le niveau de charge de la batterie (voir tableau 5.2).

Modèle	Comportement	Éléments d'influence
Constant-charge-level	$CL = c_1$	Damaged(BAT)
Normal-accumulation	$CL = f(I, TIME)$	Damaged(BAT), Rechargeable(BAT)
Accumulation-with-aging	$CL = f(I, TIME, TSLC)$	Damaged(BAT), Rechargeable(BAT)

TABLE 5.2 – Modèle de la classe d'hypothèses "Battery-charge-level-ac"

Nous considérons ici trois modèles permettant de décrire le niveau de charge (CL) d'une batterie. Le modèle le plus simple consiste à proposer un niveau de charge constant (Constant-charge-level). Un modèle de batterie rechargeable devra prendre en considération un courant de recharge (Normal-accumulation). Un dernier modèle prend en compte l'âge de la batterie et le temps depuis la dernière remise

en état (Accumulation-with-aging).

Ceci permet de définir un treillis d'abstraction des différents modèles proposés dans la classe "Battery-charge-level" (voir figure 5.3).

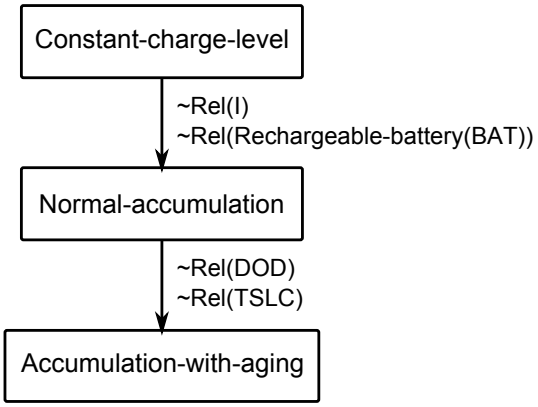


FIGURE 5.3 – Treillis d'abstraction de la classe "Battery-charge-level"

5.2.1.3 Classe "Battery-domage-ac"

Cette classe possède un unique modèle ayant une influence directe sur l'état de santé de la batterie.

Modèle	**Comportement**	**Éléments d'influence**
Battery-domage	$Damage = f(TIME, DOD, TEMP, CL)$	Damaged(BAT),Rechargeable(BAT)

TABLE 5.3 – Modèle de la classe d'hypothèses "Battery-domage-ac"

Nous ne considérons ici qu'un seul modèle pour décrire l'état de santé de la batterie. Selon le niveau de charge, l'âge de la batterie, le nombre de cycles et la température d'utilisation, le modèle renvoie de manière binaire l'état de la batterie.

Cette classe ne possédant qu'un seul modèle, elle n'a donc pas de treillis d'abstraction.

5.2.2 Vers la modélisation comportementale

Nous avons maintenant une hiérarchisation des modèles intéressante pour l'étude de notre batterie. A ce stade, les modèles ne possèdent qu'une définition de haut niveau, n'offrant qu'une vue sommaire de leurs comportements (e.g. V=f(CL,TEMP)).

Pour la suite de l'étude, il sera nécessaire de proposer des modèles dont la structure et le comportement ont été définis formellement, tant du point de vue du modélisateur que de celui de l'utilisateur de la simulation, tel que cela a été défini dans la méthodologie détaillée dans la section 4.1.1.

5.3 Complétion du modèle SiML

Selon l'objectif de simulation, le modélisateur propose un modèle et un ensemble d'hypothèses à l'aide du modèle SiML. L'utilisateur de la simulation, de son côté, va définir un cadre expérimental associé aux objectifs de simulation.
Une grande partie du cycle de développement du système a été occultée ici pour se focaliser sur la M&S, et plus particulièrement sur la contribution de cette thèse. Le degré d'abstraction des modèles, qui vont être proposés par la suite, fait penser que nous sommes ici dans une phase amont du projet, telle que la phase de conceptualisation (voir chapitre 1), où la M&S va aider à la comparaison de solutions envisageables. [2]

Pour cette étude, nous allons considérer deux objectifs de simulation différents, présentés dans le tableau 5.4.

Objectif	Description	Environnement associé
Objectif 1	L'objectif de la simulation est de s'assurer que la batterie est capable de proposer une tension acceptable 12V +/-0.5V après une période de stockage d'un mois.	Stockage normal : La batterie est stockée dans une plage de température comprise entre 0°C et 35°C. Un remplacement de la batterie est effectué tous les ans.
Objectif 2	L'objectif de la simulation est de s'assurer que la batterie n'est pas détruite après un mois d'utilisation en conditions normales. Une charge qui consomme 2,5W est associée en dehors des périodes de recharge. La batterie est rechargée 1 heure par jour à courant constant avec un courant égal à 10% de sa capacité (C/10 soit 5A).	Utilisation conditions normales : La plage de température d'utilisation de la batterie est respectée, température comprise entre 0°C et +35°C. Le niveau de charge de la batterie peut descendre à 40%.

Table 5.4 – Objectifs de simulation

5.3.1 Objectif de simulation n°1 : Point de vue "Modélisateur"

Le modélisateur va proposer à l'utilisateur de la simulation des modèles de batteries capables de satisfaire les objectifs de simulation. Pour ce faire, il va compléter une partie du modèle SiML. Il va alors établir un modèle formel de la batterie capable de répondre à l'objectif de simulation et y associer un ensemble d'hypothèses statiques d'utilisation. La classification proposée précédemment peut être vue comme une représentation concrète de l'expertise métier du modélisateur dans le domaine des batteries rechargeables. Appliquer cette expertise revient ici à choisir un modèle dans chaque classe d'hypothèses, de façon à construire un modèle capable de respecter l'objectif ciblé.

2. Nous avons cherché à ce que les modèles et résultats de simulation proposés dans ce chapitre respectent les mécanismes et ordres de grandeurs des systèmes des batteries d'accumulateurs. Cependant, certains raccourcis ont été pris lors de l'élaboration des modèles afin de garder un niveau de modélisation compréhensible à la première lecture, le but étant ici d'illustrer la méthodologie établie durant cette thèse. Ces raccourcis rendent donc les résultats de simulations inexploitables.

Rappel de l'objectif n°1 *: S'assurer que la batterie est capable de proposer une tension acceptable 12V +/-0.5V après une période de stockage d'un mois.*

Pour le premier objectif, le modélisateur propose la sélection de modèles présentée dans le tableau 5.5.

Classe d'hypothèse	Modèle sélectionné	Note
Battery-voltage-ac	Normal-degrading	La batterie est en période de stockage, ce qui suppose une absence de prise en compte de charge ; la plage de température de stockage ne nécessite pas de prise en compte des phénomènes de température.
Battery-charge-level	NA	NA
Battery-domage-ac	NA	NA

TABLE 5.5 – Proposition du modélisateur pour l'objectif 1.

Nous considérons le tableau 5.5 comme une réflexion préliminaire du modélisateur, d'où une justification informelle du choix du modèle ("Note"). Le modélisateur peut alors définir un premier ensemble d'hypothèses statiques. Cet ensemble est amené à évoluer durant la phase de modélisation formelle.
Comme nous l'avons déjà évoqué dans les chapitres 3 et 4, les hypothèses statiques nécessitent un langage (voir discussions 3.6.3).

Nous proposons ici d'exprimer les hypothèses dans la notation de l'inégalité mathématique ($<, >, \leq, \geq, \neq$), associée à celle de la logique combinatoire ($\&, ||, \neg$) . L'idée est de pouvoir établir si un état dans le graphe de classes respecte ou non cette hypothèse. Les hypothèses proposées dans cet exemple ont un niveau de complexité faible [3]. A terme, l'utilisation d'une logique temporelle est envisagée pour une meilleure spécification des hypothèses statiques.

Chaque hypothèse statique est associée à un ensemble : hypothèses de modélisation, hypothèses opérationnelles ou hypothèses rationnelles (voir 4.1.1). Voici un sous-ensemble d'hypothèses proposé par le modélisateur :

— $\phi_1 := 10 < V < 12$
— $\phi_2 := \neg temp$
— $\phi_3 := \neg cl$
— ...

Ce qui de manière textuelle peut se traduire par : *Le modèle a été fait pour une*

3. Durant l'élaboration de cette méthodologie, nous avons également utilisé la notation de la logique temporelle et plus particulièrement de la logique CTL* [Emerson 1986], afin de spécifier nos hypothèses. Issue de l'informatique théorique, CTL* est un sur-ensemble de CTL [Clarke 1982] et LTL [Pnueli 1977], deux autres logiques temporelles. L'utilisation de cette logique a été motivée par notre première approche associée au dépliage des graphes en arbre, que l'on retrouve dans la notion de dépliage des structures de Kripke [Kripke 1963] de la logique CTL. L'avantage de CTL* est de permettre la qualification de propriétés sur les chemins mais aussi sur les états. Les travaux présentés ici manquent de maturité pour tirer parti de l'ensemble des avantages associés à l'utilisation de CTL*, et en particulier des structures de Kripke associées à la définition des formules CTL*. Les structures de Kripke sont une sortie de graphe de classes (graphe d'états), où les transitions ne sont pas étiquetées.

tension comprise entre 10 volt et 12 volt, sans changement de température et sans prise en compte du niveau de charge, etc.

Le modélisateur peut alors proposer un modèle formel (ici un modèle ProDEVS), permettant de satisfaire l'objectif de simulation n°1 (voir figure 5.4).

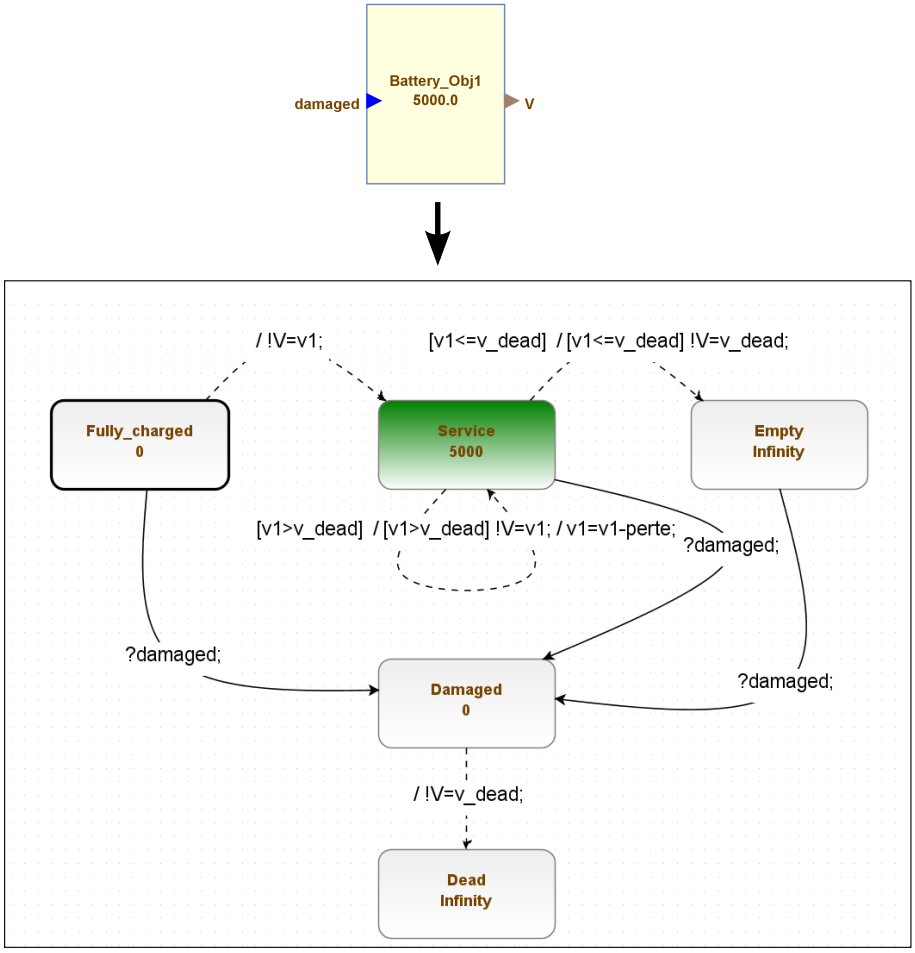


FIGURE 5.4 – Modèle ProDEVS "Battery-obj1"

Ce premier modèle (figure 5.4), proposé par l'utilisateur de la simulation, est composé de cinq phases. Trois phases représentent les états de la batterie : Fully-charged (batterie pleine charge), Service (batterie en service) et Empty (batterie vide). La batterie est supposée être initialement à pleine charge, avec une tension de 12V. On considère que la batterie possède un courant de fuite qui induit une perte en tension de 0.0002 volt par heure. Le modélisateur établit qu'une résolution de 0,001 volt est suffisante pour modéliser la tension aux bornes de la batterie. Cela se traduit par un "déclenchement" de perte toutes les 5000 ut (sachant que 1h $\equiv$ 1000 ut). Le domaine associé à la tension est : $[10; 12]$ pour une précision de 1×10^{-15} . Les deux phases restantes sont dédiées à la gestion de la défaillance de la

batterie : lorsque la batterie passe en mode défaillance, elle propose instantanément une tension défaillance de 10V.

La figure 5.5 propose une vue de l'environnement de modélisation ProDEVS dans lequel est créé le modèle "Battery-obj1". On retrouve ici les principaux éléments des environnements de développement (EDI) actuels (e.g Eclipse), à savoir : onglet de navigation, vue générale du projet, barre d'outils ou encore menu contextuel. Un onglet propriétés permet de spécifier un ensemble d'informations associé à chaque phase et à chaque transition du modèle :

— **Pour les phases** :

- t_a : définit le temps associé à la phase.
- *Gestion de conflit de transition* : définit la manière de gérer les conflits de transition pour les transitions à niveaux de priorité équivalents (manuel ou aléatoire).
- *External transition priority* : définit l'ordre de priorité des différentes transitions externes.
- *Internal transition priority* : définit l'ordre de priorité des différentes transitions internes.

— **Pour les transitions internes** :

- *Guard* : définit la (ou les) condition(s) nécessaire(s) pour autoriser le franchissement de la transition.
- *Outs* : définit la fonction de sortie de la transition.
- *Guard de sortie* : définit la (ou les) condition(s) nécessaire(s) pour autoriser la fonction de sortie.
- *Effects* : définit l'action associée à la transition.
- *Guard des actions* : définit la (ou les) condition(s) nécessaire(s) pour autoriser l'action associée à la transition.

— **Pour les transitions externes** :

- *Guard* : définit la (ou les) condition(s) nécessaire(s) pour autoriser le franchissement de la transition.
- *Trigger* : définit l'entrée qui sensibilise la transition.
- *Effects* : définit l'action associée à la transition.
- *Guard des actions* : définit la (ou les) condition(s) nécessaire(s) pour autoriser l'action associée à la transition.

L'outil ProDEVS intègre un outil de vérification composé d'un ensemble de règles permettant de s'assurer de la bonne construction du modèle, étape indispensable vers la génération du graphe de classes. L'outil offre également la possibilité d'effectuer différents scénarios de simulation et d'en observer les résultats. La figure 5.6 illustre ici la réponse en tension du modèle "Battery-voltage-Normal-degrading" après un stockage de 720 heures et la dégradation de la batterie après 600 heures de stockage.

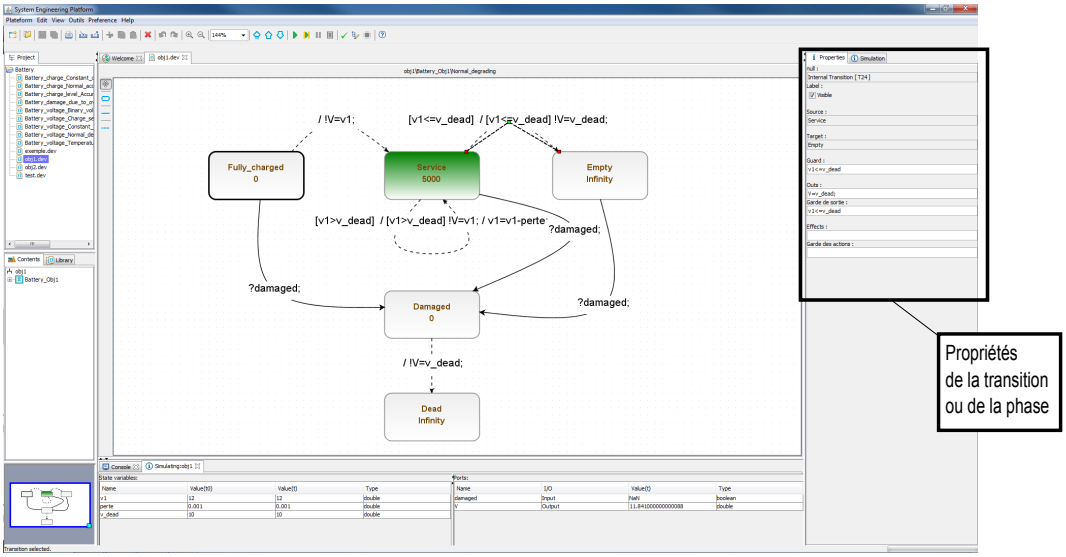


FIGURE 5.5 – Elaboration du modèle "Battery-obj1" dans l'environnement ProDEVS

Habituellement, après l'étude d'un ensemble de scénarios de simulation tel que celui-ci, le modèle est validé soit par un expert, soit par comparaison aux données d'expérimentation. Comme nous l'avons vu dans le chapitre 2, les domaines critiques demandent un niveau de validité plus important. L'idée est donc de poursuivre l'étude du modèle, pour améliorer son niveau de validité. En d'autres termes, il faut s'assurer que le modèle est capable de répondre de façon identique au système réel pour les objectifs de simulation fixés (contraintes de précision, comportement, etc.). C'est ici qu'intervient l'utilisateur de la simulation.

Le modèle est confié à l'utilisateur de la simulation. De manière classique, seuls le modèle et une spécification sont fournis à l'utilisateur de la simulation. Dans notre approche, le modèle fourni par le modeleur devient un modèle formel accompagné d'un ensemble de contraintes statiques. Le modèle va alors subir une première transformation, permettant d'obtenir le graphe de classes du modèle associé de manière transparente pour l'utilisateur.

A l'heure où ces lignes sont écrites la transformation directe de ProDEVS vers TPN et l'outil Tina [Berthomieu 2004] pour la génération de graphes de classes n'est pas totalement opérationnelle. Nous utiliserons donc dans cet exemple une transformation vers le langage Fiacre [Berthomieu 2008](présenté dans la section 4.4). L'approche compositionnelle de Fiacre offre des similitudes structurelles avec ProDEVS, permettant un passage plus aisé du formalisme DEVS, vers le formalisme RdP [4] pour une transformation manuelle.

4. Les règles de transformation d'un modèle ProDEVS vers un modèle Fiacre ne seront pas présentées dans cette thèse. Les outils de construction de graphe de classes issus de modèles DEVS

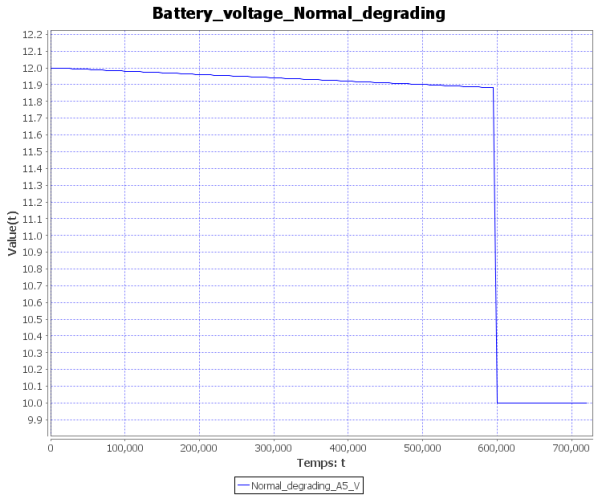


FIGURE 5.6 – Réponse en tension du modèle "Battery-voltage-Normal-degrading" pour un stockage de 720 heures, dégradation à t=600 heures.

```
1  type tension is 0..12000
2
3  process normal_degrading(&Damaged: bool, &V : tension) is
4      states fully_charged,service,s_empty,s_damaged
5
6  var v1 : tension := 12000, v_damaged : bool := false
7
8    from fully_charged
9      select
10       on(v_damaged <> Damaged); wait[0,0];
11         Damaged:=false; V:=10000;
12       to s_damaged
13   []  on(v_damaged = Damaged);wait[0,0];
14         V:=v1;
15       to service
16     end
17
18   from service
19   [...]
20
21   from s_empty
22   [...]
23     end
```

sont en phase de développement et devraient permettre à terme de fournir une solution directe aux problèmes de génération du graphe de classes [Hwang 2009].

```
24
25 /* Main component */
26
27 component Battery-obj1 is
28      var V : tension :=10000, Damaged: bool :=false
29       par * in
30         normal_degrading(&Damaged,&V)
31 end
32
33 /* Entry point */
34 Battery-obj1
```

Listing 5.1 – Équivalent du modèle "Battery-voltage-Normal-degrading" DEVS dans le langage pivot Fiacre

Le listing 5.1 représente une partie du code Fiacre équivalent au modèle Pro-DEVS "Battery-voltage-Normal-degrading". On retrouve ligne 3 la déclaration du *process*, équivalent d'un composant atomique. La déclaration des états en ligne 4 est une représentation directe des phases. Les lignes 8 à 24 représentent les règles de transitions d'un état à l'autre, et les contraintes temporelles associées. A la ligne 25, on trouve la déclaration de "Battery-obj1", équivalent au composant couplé DEVS "Battery-obj1". La dernière ligne déclare le point d'entrée du modèle, le composant de plus haut niveau. Notons que la première ligne permet de définir le domaine de la variable tension, afin de limiter l'explosion combinatoire lors de la génération de graphes. Fiacre opérant avec des entiers, une simple mise à l'échelle a du être effectuée, notre résolution étant à l'origine de 0,001V.

Le modèle Fiacre est alors transformé afin d'être analysé par l'outil TINA [Berthomieu 2004], permettant la génération du graphe de classes. Le listing 5.2 est un extrait du graphe de classes généré par l'outil.

```
1  REACHABILITY ANALYSIS ------------------------------------------
2  bounded
3  8004 classe(s), 10004 transition(s)
4
5  CLASSES:
6  class 0
7       marking
8    normal__degrading_1_sfully__charged
9      {normal__degrading_1_vv1=12000,
10     Battery_obj1_1_vDamaged=false}
11      domain
12   0 <= normal__degrading_1_t0 <= 0
13
```

```
14 [...]
15
16 class 7447
17     marking
18   normal__degrading_1_sservice
19    {normal__degrading_1_vv1=10139,
20    Battery_obj1_1_vDamaged=false}
21     domain
22   5000 <= normal__degrading_1_t2 <= 5000
23
24 [...]
```

Listing 5.2 – Graphe de classes généré à partir d'un modèle Fiacre par l'outil Tina

Nous considérons ce graphe de classes comme la capabilité du modèle (voir figure 3.17). Il représente l'ensemble des possibilités du modèle ($L(M)$ et S_M), chaque possibilité étant équivalente à une trace d'exécution (c'est le principe d'exploration exhaustive du model-checking). Le graphe de classes proposé (voir listing 5.2) est composé de 8004 classes, reliées par 10004 transitions. Les deux classes proposées ici sont représentatives du graphe. La classe 0 est équivalente au marquage initial, la batterie est totalement chargée (phase : fully-charged et v1=12000 ; la batterie n'est pas dégradée (v-Damage=false). La classe 7447 représente un des états durant la décharge de la batterie, sans avoir encore subi de dégât.

Mettons de côté le modèle de la batterie et la vision "modélisateur" pour nous intéresser maintenant au cadre expérimental et à la vision "utilisateur de la simulation".

5.3.2 Objectif de simulation n°1 : Point de vue "Utilisateur de la simulation"

De son côté, l'utilisateur de la simulation doit mettre en place un ensemble de scénarios de simulations dans le but de vérifier des propriétés sur le modèle. Pour ce faire, il va à son tour compléter le modèle SiML, en fournissant un modèle formel du cadre expérimental, ainsi qu'un ensemble d'hypothèses statiques associé.

Rappel de l'objectif n°1 *: S'assurer que la batterie est capable de proposer une tension acceptable 12V +/-0.5V après une période de stockage d'un mois.*
L'utilisateur de la simulation définit un ensemble d'hypothèses statiques :

- $\phi_1 := \neg damaged$
- $\phi_2 := 11 < V < 12.5$
- $\phi_3 := 0 < t_{simu} < 744000$
- ...

Ce qui de manière textuelle peut se traduire par : *Le scénario se déroulera pour un modèle respectant l'état "not damaged", pour une tension comprise entre 11 volts*

et 12,5 volts, et un temps de simulation de 744000 unités de temps, etc.

Il peut alors décrire le cadre expérimental et les modèles formels associés au générateur et au transducteur.

Au vu des objectifs de modélisation, l'utilisateur de la simulation décide qu'un générateur n'est pas nécessaire. Le générateur n'aura donc aucun rôle ici. Nous passons ainsi directement à l'étude du transducteur [5].

La figure 5.7 représente l'idée que se fait l'utilisateur de la simulation du comportement supposé du modèle. Les hypothèses associées à une telle modélisation sont nombreuses. En effet, l'utilisateur de la simulation suppose que le modèle est capable de fournir à $t = 0$ une tension de 12,5V. Ensuite, le transducteur suppose le modèle de la batterie en phase de stockage pendant 774000 ut, soit 1 mois ($1000ut \equiv 1$ heure). Si la tension descend en dessous de 11,5V, la batterie sera considérée comme sous-dimensionnée.

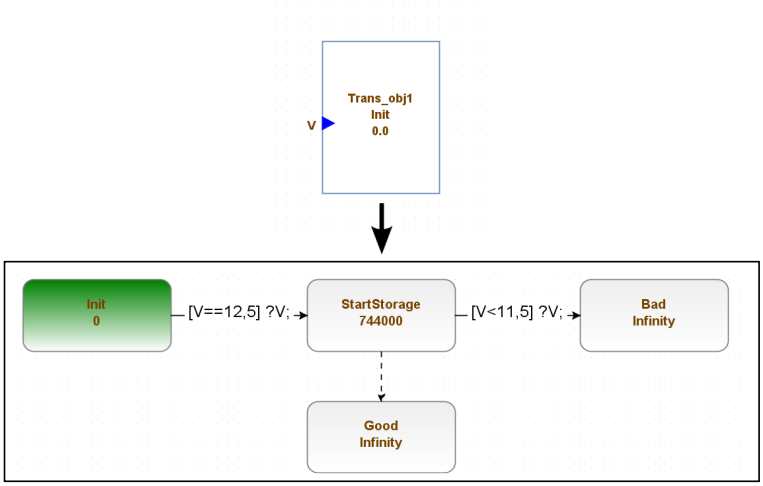


FIGURE 5.7 – Modèle ProDEVS du transducteur pour l'objectif n°1

L'objectif du cadre expérimental (représenté dans notre cas par le transducteur seul) est d'aider l'utilisateur de la simulation à valider le modèle. Comme nous l'avons déjà dit, notre rôle est d'améliorer la crédibilité du modèle de simulation, c'est-à-dire d'étudier le couple modèle-cadre expérimental (ici, le couple modèle-transducteur) pour s'assurer de la validité du modèle de simulation, et donc permettre à l'utilisateur de la simulation de conclure dans de bonnes conditions si le modèle est valide ou pas.

Comme pour le modèle "Battery-obj1", le cadre expérimental est transformé en modèle Fiacre afin d'en obtenir le graphe de classes.

5. Le transducteur peut être vu comme une image du vecteur de sortie, tel que le conçoit l'utilisateur de la simulation au vu des objectifs de simulation.

Toujours selon le même principe, il est possible de sélectionner et d'appliquer les hypothèses statiques au graphe de classes, afin de marquer les états respectant les hypothèses définies précédemment et associées au transducteur.

Nous sommes maintenant en possession du graphe de classes du cadre expérimental (S_T et $L(T)$)et du modèle (S_M et $L(M)$). Il est donc possible d'aller en phase d'analyse afin de s'assurer de la cohérence cadre expérimental-modèle.

5.3.3 Objectif de simulation n°1 : Analyse

La phase d'analyse se fait en deux temps. Une première analyse vise à s'assurer de la compatibilité de périmètre. Il faut s'assurer que les entrées du modèle sont compatibles avec le cadre expérimental (voir 2.4.2). La seconde étape de l'analyse consiste à étudier la compatibilité dynamique après l'opération de restriction. Pour cela, il faut établir le produit parallèle entre les deux graphes de classes (modèle et transducteur) tel qu'il a été défini dans la section 3.4.2.1 présentant les différents produits de composition. Ceci nous permet d'obtenir les états effectivement atteints par le modèle et par le transducteur lors du couplage ($L_C(M//T)$ et S_{exp})(voir zone en vagues de la figure 3.17), ce qui permet de les comparer avec les états qui étaient initialement atteignables $S(M)$ et $S(T)$ (voir *model capability* du la figure 3.17).

Le couple à l'étude est présenté dans la figure 5.8.

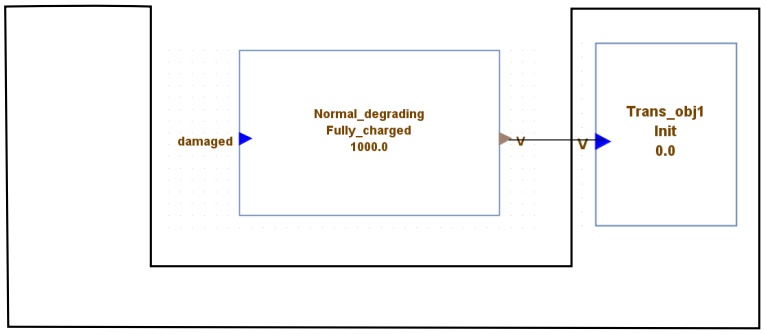


FIGURE 5.8 – Couple cadre expérimental-modèle pour l'objectif n°1

Le premier élément notable est l'entrée non utilisée *damaged*, qui suggère une incompatibilité de périmètre. L'utilisateur de la simulation doit alors choisir s'il bouchonne l'entrée *damaged* ou s'il revoit son cadre expérimental. Au vu de l'objectif de simulation, la notion de dommage de la batterie n'apparaît pas ; une opération de restriction (voir section 3.4.2.1) semble donc être une solution acceptable.

Nous considérons que l'unité, la résolution, la précision, ainsi que le domaine de l'élément V (représentant la tension) respectent les critères de compatibilité de périmètre.

Il est maintenant nécessaire d'étudier la compatibilité dynamique et donc la composition "cadre expérimental-modèle" à l'aide des métriques.

La figure 5.9 donne un graphe récapitulatif de nombre d'états du modèle en fonction de différentes hypothèses.

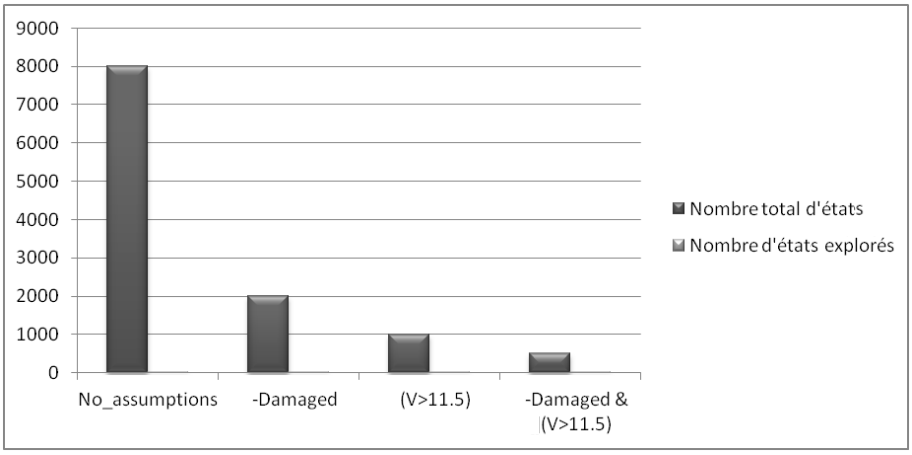


FIGURE 5.9 – Nombre d'états existants et explorés du modèle "Battery-voltage-Normal-degrading" en fonction d'une sélection d'hypothèses.

Comme nous pouvons le constater facilement, le niveau d'exploration du modèle est extrêmement limité. Le modèle sans hypothèse a un graphe de classes de plus de 8000 états. Le niveau de compatibilité avec le transducteur semble très faible voir nul. Seuls deux états du graphe de classes sont accessibles. On retrouve le même phénomène avec l'étude des états du graphe de classes respectant certaines hypothèses. Près de 2000 états respectent l'hypothèse $\neg Damaged$, alors qu'un seul état est accessible à l'application du transducteur. S'il n'est pas possible de conclure avant d'avoir étudié le graphe de classes du transducteur, il semble qu'il y ait incompatibilité dynamique. Une étude similaire du transducteur est nécessaire.

L'utilisateur de la simulation observe que le taux de couverture du transducteur est de 2%, alors que le taux de couverture de certaines hypothèses statiques est autour de 80%, ce qui amène le taux relatif d'exploration selon les hypothèses statiques autour de 1% (niveau d'exploration de l'hypothèse après application du modèle). Il semble clairement que nous soyons dans le cas d'un modèle de simulation non valide.

Deux solutions s'offrent alors à l'utilisateur de la simulation. Il est possible de revoir manuellement son transducteur ou de partir sur une étude fine des traces d'exécution (décomposition en arbre) pour détecter les limites du transducteur (présentée en section 3.4.3). La simplicité du modèle et le niveau très faible d'exploration invitent à préférer la première solution. La première contrainte n'est pas respectée puisque l'utilisateur de la simulation attend une tension nominale à 12,5V alors que le modèle fournit une tension à 12V. Après une mise en avant du problème auprès

du modélisateur, la solution proposée est de modifier la contrainte de tension. La mesure du taux d'utilisation du domaine de la variable V prévu par le modèle par rapport au domaine effectif de la variable aurait également permis la détection de cette incohérence.

Le nouveau transducteur proposé par l'utilisateur de la simulation est donné en figure 5.10.

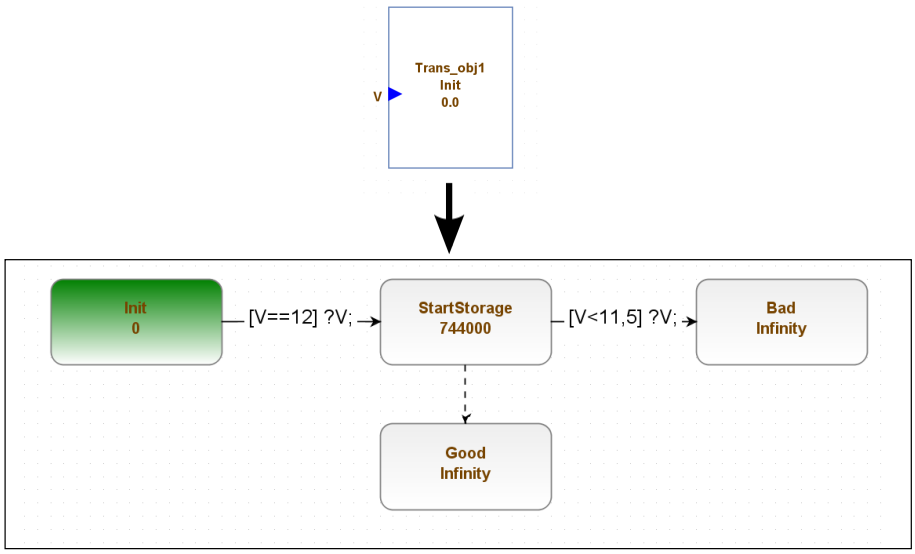


FIGURE 5.10 – Nouvelle proposition de transducteur pour l'objectif n°1

La figure 5.11 nous donne les nouveaux résultats de compatibilité pour le modèle "Battery-voltage-Normal-degrading"

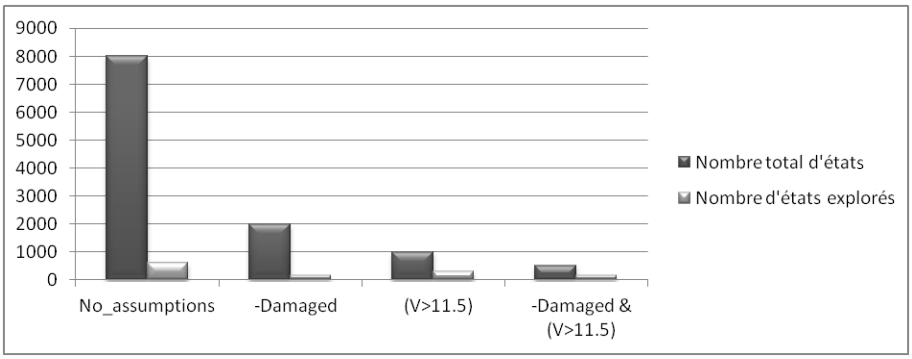


FIGURE 5.11 – Nombre d'états existants et explorés du modèle "Battery-voltage-Normal-degrading" en fonction d'une sélection d'hypothèses après application d'une modification sur le transducteur.

La représentation de la figure 5.10 s'intéresse directement au nombre d'états du modèle. Nous admettons que le taux d'exploration représente mieux le niveau de

compatibilité et permet de mieux détecter les incohérences. Un nombre peu élevé n'est pas synonyme d'incohérence, et à l'inverse, un grand nombre d'états explorés ne signifie pas qu'il y a cohérence.

Reprenons les données de la figure 5.10 pour les exprimer grâce au taux d'exploration (voir 3.6.4).

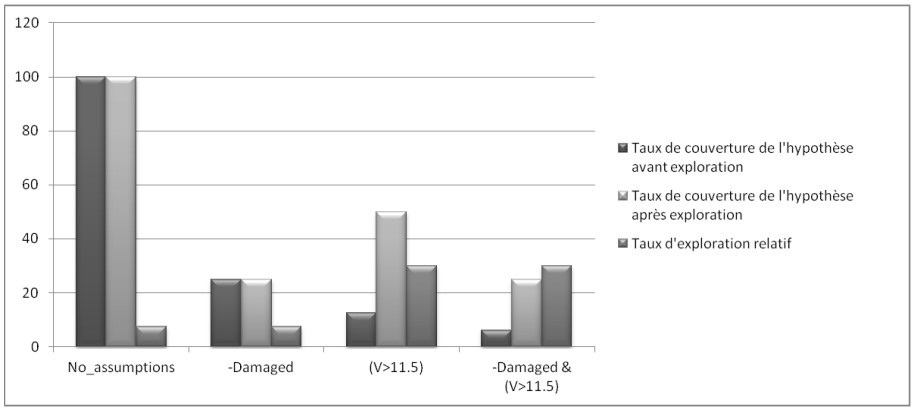


FIGURE 5.12 – Récapitulatif des taux d'exploration du modèle "Battery-voltage-Normal-degrading" après application du transducteur modifié.

La figure 5.12 nous propose une étude du taux d'exploration selon différentes hypothèses. Pour rappel, le taux de couverture de l'hypothèse avant exploration représente l'ensemble des états accessibles du modèle (S_{hyp}) lorsqu'une hypothèse est appliquée. Par définition le taux de couverture avant exploration de "no-assumption" est toujours de 100% (S_M). Le taux de couverture de l'hypothèse après exploration est une image du nombre d'états respectant les hypothèses par rapport au nombre total d'états explorés. Finalement, le taux d'exploration relatif représente le pourcentage d'états respectant l'hypothèse avant l'exploration par rapport à ceux respectant l'hypothèse après simulation.

Si l'on observe le taux d'exploration relatif des différentes hypothèses (toujours figure 5.12), il est difficile de conclure. On observe un taux d'exploration relatif des différentes hypothèses bas (<50%). Ceci traduit un modèle qui semble plus capable que nécessaire. A noter que l'étude d'hypothèses supplémentaires pourrait prouver le contraire.

Une étude équivalente sur le transducteur doit permettre d'améliorer la crédibilité. Comme cela a été indiqué tout au long de ce manuscrit, c'est l'étude comparative des taux d'exploration du générateur (ici absent), du modèle et du transducteur qui peut permettre d'améliorer la crédibilité de la simulation.

La figure 5.13 reprend les résultats d'exploration pour le transducteur modifié.

L'observation de ces taux de couverture est une représentation des niveaux d'inclusion des différents ensembles tels que définis dans la section 3.5.3. On observe

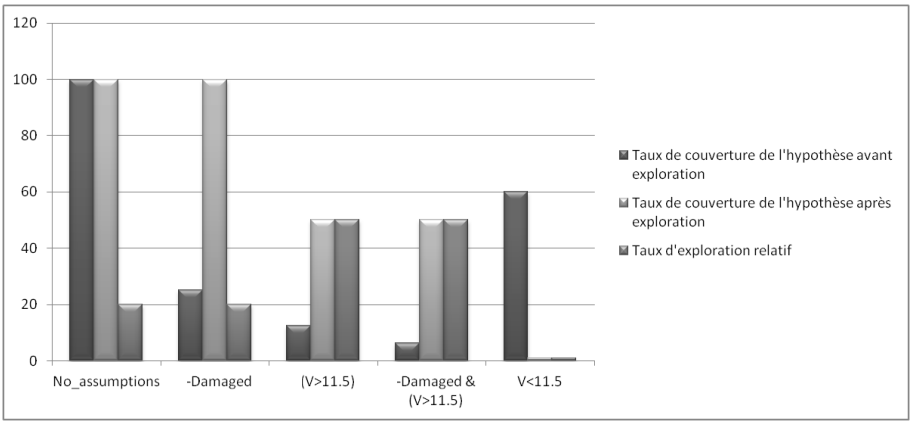


FIGURE 5.13 – Récapitulatif des taux d'exploration du transducteur modifié pour l'objectif n°1

que seulement 20% du transducteur est exploré. Deux indices nous confortent dans la crédibilité à associer aux résultats de simulation. Le premier indice est un respect de l'hypothèse $\neg Damaged$. Lors de l'exploration, l'ensemble des états restants respectent cette hypothèse. Le second indice est l'absence totale d'état exploré avec une tension inférieure à 11,5V.

Si l'utilisateur de la simulation décide que les taux d'exploration sont cohérents, il peut alors s'exprimer sur la validité du modèle. Au contraire, si une métrique ou un ensemble de métriques est incohérent, il peut décider de mettre en place de nouvelles métriques, associées à de nouvelles hypothèses. Nous considérons pour cette exemple que le niveau de cohérence est suffisant. Nous pouvons maintenant effectuer un travail similaire pour l'objectif n°2.

5.3.4 Objectif de simulation n°2 : Point de vue "Modélisateur"

Rappel de l'objectif n°2 :*S'assurer que la batterie n'est pas détruite après un mois d'utilisation en conditions normales. Une charge qui consomme 2,5W est associée en dehors des périodes de recharge. La batterie est rechargée 1 heure par jour à courant constant avec un courant égal à 10% de sa capacité (C/10 soit 5A).*

L'étude de cet objectif sera limitée à l'expression d'hypothèses statiques, à la modélisation en DEVS et à l'analyse des métriques que l'utilisateur final sera amené à utiliser.

Pour le second objectif, le modélisateur propose la sélection présentée dans le tableau 5.6.

Le modélisateur propose un ensemble d'hypothèses statiques associé à cette mo-

Classe d'hypothèse	Modèle sélectionné	Note
Battery-voltage-ac	Temperature-sensitive	Prise en compte de la température et du niveau de charge.
Battery-charge-level	Accumulation-with-aging	Prise en compte des cycles de charge/décharge, du DOD, du courant de charge.
Battery-damage-ac	Battery-damage	Étude de la destruction de la batterie ; prise en compte des décharges et de la température.

TABLE 5.6 – Proposition du modélisateur pour l'objectif 2.

délisation :

— $\phi_1 := -0.2 < I < 5$
— $\phi_2 := I = -0.2$
— $\phi_3 := I = 5$
— $\phi_4 := I < 5 \ \& \ I > -0.2$
— $\phi_5 := \neg Cond$
— $\phi_6 := -20 < Temp < 55$
— ...

A l'opposé de l'objectif n°1, le second objectif nécessite un modèle plus complexe, composé de trois éléments principaux (voir figure 5.14). On retrouve dans cette illustration les différents choix du modélisateur présentés dans le tableau 5.6 (Temperature-sensitive, Accumulation-with-aging et Battery-Damage), ainsi que les couplages entre les différents composants.

La modélisation de la batterie en elle-même n'a que peu d'intérêt. Cependant, nous allons nous attarder sur la présentation des différents sous-composants afin de mieux percevoir l'impact futur du cadre expérimental sur les différents éléments du modèle [6].

Le composant atomique "Temperature-sentitive" fournit une nouvelle tension de sortie à chaque évènement d'entrée : niveau de charge (CL) ou température (Temp). Lorsque la température descend en dessous de -20°C, la batterie propose une tension de secours de 11,5V constants. A tout instant la batterie peut être détruite, et la tension aux bornes de la batterie est alors nulle (0V).

Le composant couplé "Accumulation-with-aging" évalue le niveau de charge de la batterie en fonction du courant traversant la borne positive de la batterie (voir figure 5.16). Un composant atomique "Aging" s'occupe du vieillissement de la batterie, en prenant en compte les reconditionnements ayant pour effet d'améliorer le niveau de charge maximum permis pour la batterie.

Le second composant atomique, appelé "CL-estimation", évalue le niveau de

6. L'utilisateur de la simulation n'a normalement pas à connaître la structure interne du modèle. L'utilisateur de la simulation a une vision boîte noire du modèle.

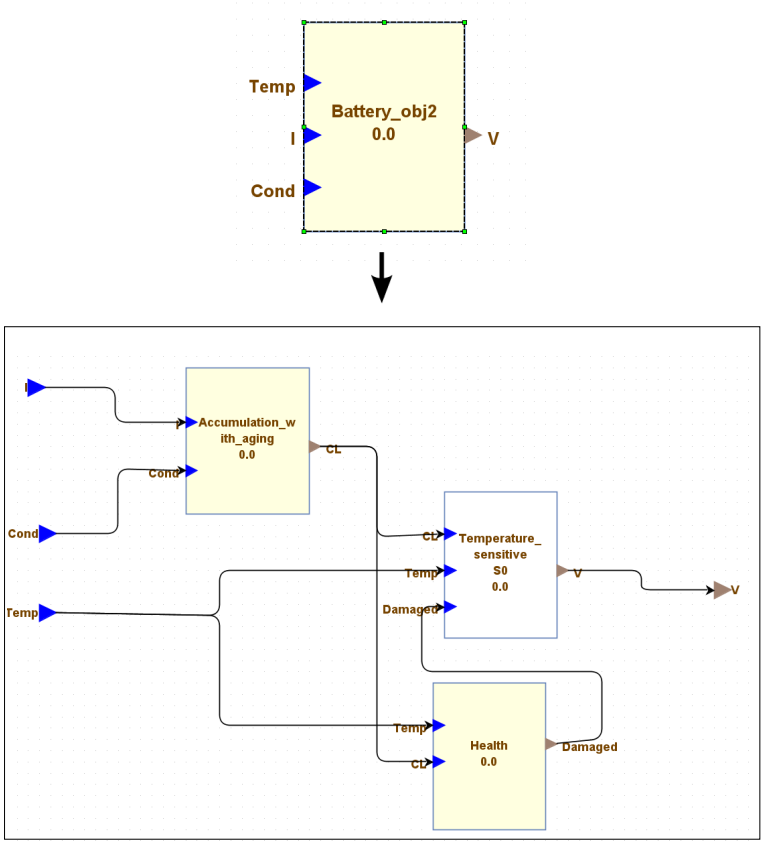


FIGURE 5.14 – Modèle ProDEVS "Battery-obj2".

charge de la batterie en fonction du courant de charge I. Le modèle est découpé en trois "secteurs" [7] distincts réunis autour d'une phase centrale de stockage. Initialement en phase de stockage, la batterie peut soit subir un rajeunissement (Cond), soit recevoir un courant $I > 0$ (la batterie est alors en mode charge), soit recevoir un courant $I < 0$ (la batterie est alors en mode décharge). Ces trois "secteurs" ne sont reliés que par l'état stockage. Il est donc nécessaire de retourner dans l'état de stockage ($I = 0$) pour passer d'une "section" à l'autre. Un schéma illustre ce principe (voir figure 5.17). Le modèle prend en compte la surcharge (resp. décharge profonde) dans le secteur charge (resp. décharge). Une mauvaise gestion de la charge ou de la décharge (e.g. dépassement du temps de charge, décharge profonde) mène à la destruction de la batterie.

Le dernier composant sélectionné par le modélisateur est le modèle "Battery-damage". Ce dernier est composé de deux modèles atomiques : Life et Battery-healt. Le composant "Life" définit une durée de vie maximum de 43 000 000 ut soit en-

7. Le terme "secteur" n'a aucune représentation formelle et sert simplement à diviser le modèle en trois parties distinctes afin de clarifier le comportement du modèle. A ne pas confondre avec les concepts d'état, de phase, ou encore de zone.

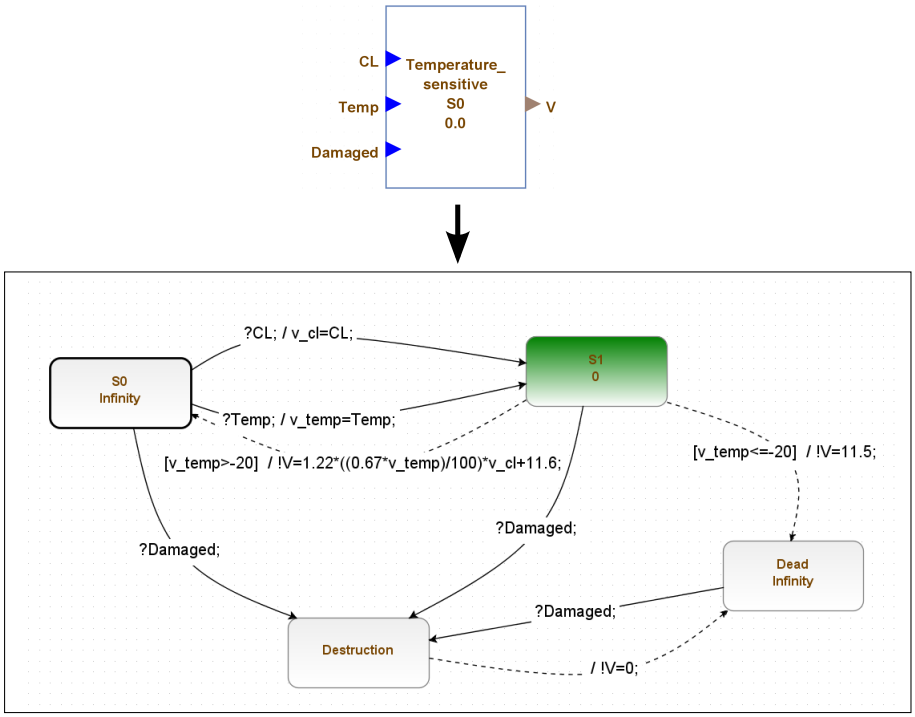


FIGURE 5.15 – Modèle ProDEVS "Battery-voltage-ac-Temperature-sensitive".

viron 5 ans d'utilisation. Le composant "Battery-health va établir si les données constructeur sont respectées (un niveau de charge supérieur à 40%, et une température d'utilisation comprise entre -20 et +65°C).

Tout comme pour l'objectif n°1, le modélisateur doit s'assurer de la bonne construction de son modèle. Il peut également visualiser la réponse de son modèle à certains scénarios (débogage) avant de proposer son modèle pour la validation. Le modèle peut alors être transformé en modèle Fiacre pour générer le graphe de classes associé, image de la capabilité du modèle. A ce stade, nous allons laisser le modèle "Battery-obj2" de côté pour nous intéresser à la modélisation du cadre expérimental, et donc au rôle de l'utilisateur de la simulation.

5.3.5 Objectif de simulation n°2 : Point de vue "Utilisateur de la simulation"

Rappel de l'objectif n°2 : *S'assurer que la batterie n'est pas détruite après un mois d'utilisation en conditions normales. Une charge qui consomme 2,5W est associée en dehors des périodes de recharge. La batterie est rechargée 1 heure par jour à courant constant avec un courant égal à 10% de sa capacité (C/10 soit 5A).*

Toujours selon la même approche, l'utilisateur de la simulation propose un ensemble d'hypothèses statiques :

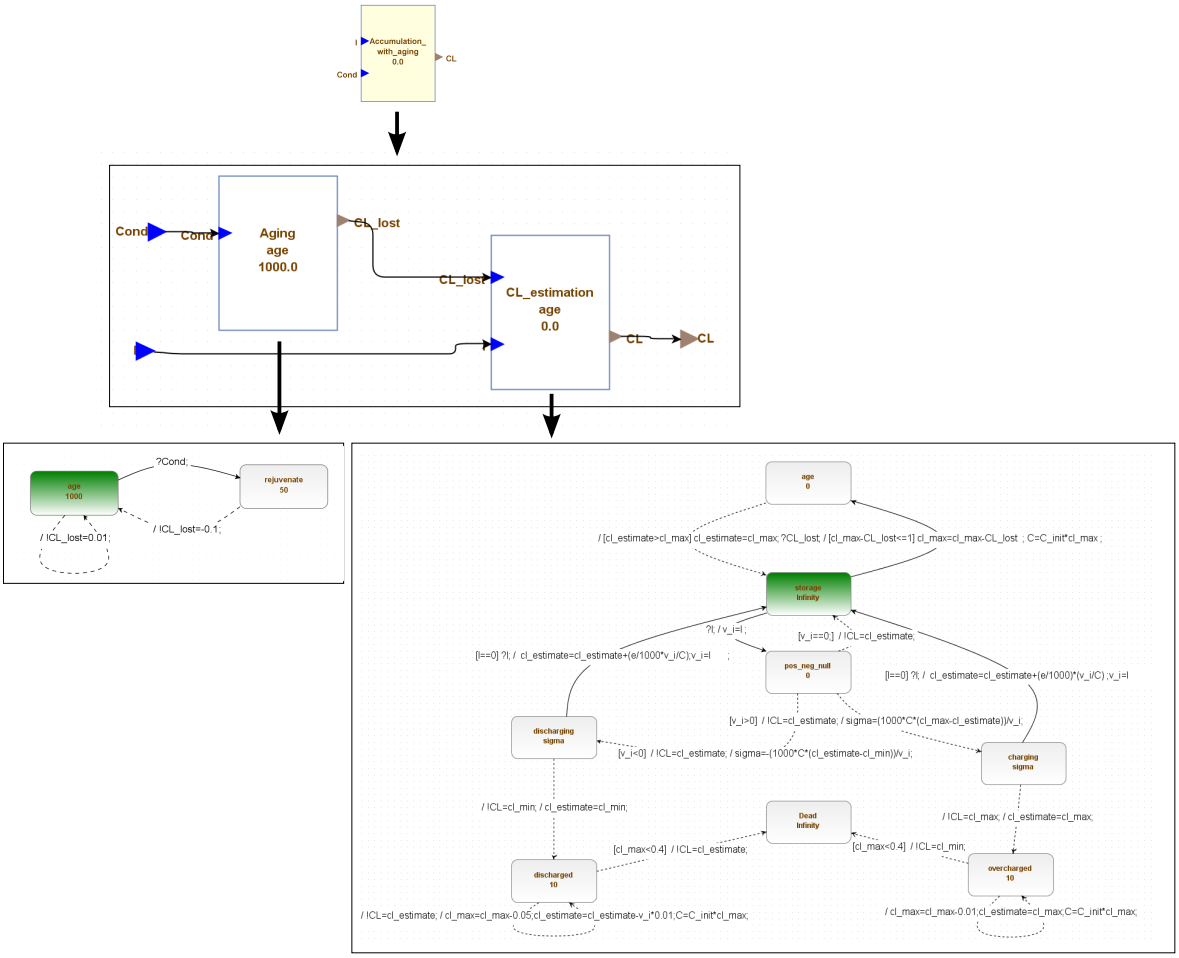


FIGURE 5.16 – Modèle ProDEVS "Battery-charge-level-Accumulation-with-aging".

— $\phi_1 := \neg Cond$
— $\phi_2 := 0 < Temp < 35$
— $\phi_3 := 10 < V < 12$
— $\phi_4 := Temp < 35$
— $\phi_5 := \neg damaged$
— $\phi_6 := \neg(V \leq 10)$
— ...

L'utilisateur de la simulation peut maintenant proposer une modélisation formelle du cadre expérimental. Nous rappelons que les hypothèses statiques peuvent être complétées durant la phase de modélisation.

Le rôle de l'utilisateur de la simulation est de proposer un ensemble de stimuli destinés à exciter le modèle en fonction des objectifs de simulation (élaboration du générateur) et d'imaginer la réaction associée du modèle en sortie (élaboration du transducteur).

La figure 5.19 donne une représentation schématique du cadre expérimental pro-

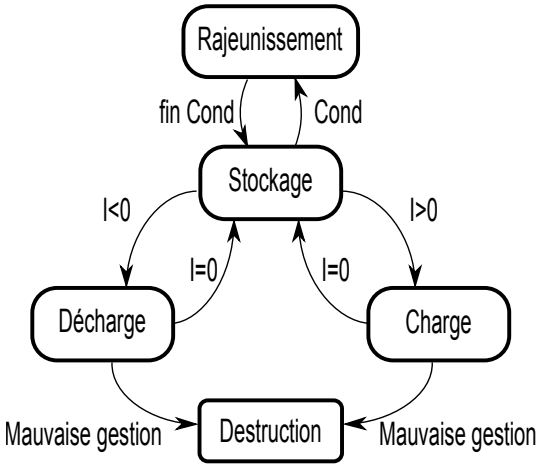


FIGURE 5.17 – Vue schématique des transitions entre secteurs dans le composant atomique "CL-estimation"

posé par l'utilisateur de la simulation pour l'objectif n°2. Le générateur est un composant couplé regroupant deux modèles atomiques. "Gen-Temp" est un générateur aléatoire de température, proposant un domaine de température en °C de $[1; 2]$ pour une résolution de 1 (voir figure 5.20). "Gen-I" propose une phase de charge d'une 1 heure à 5A, suivie d'une décharge de 23h à 0.2, correspondant approximativement à une charge qui consomme 2,5W aux bornes de notre batterie (voir figure 5.21).

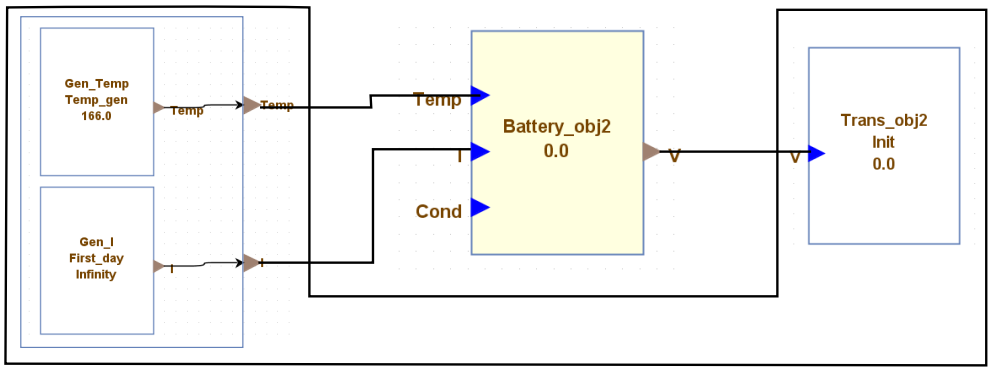


FIGURE 5.19 – Couple cadre expérimental-modèle pour l'objectif n°2.

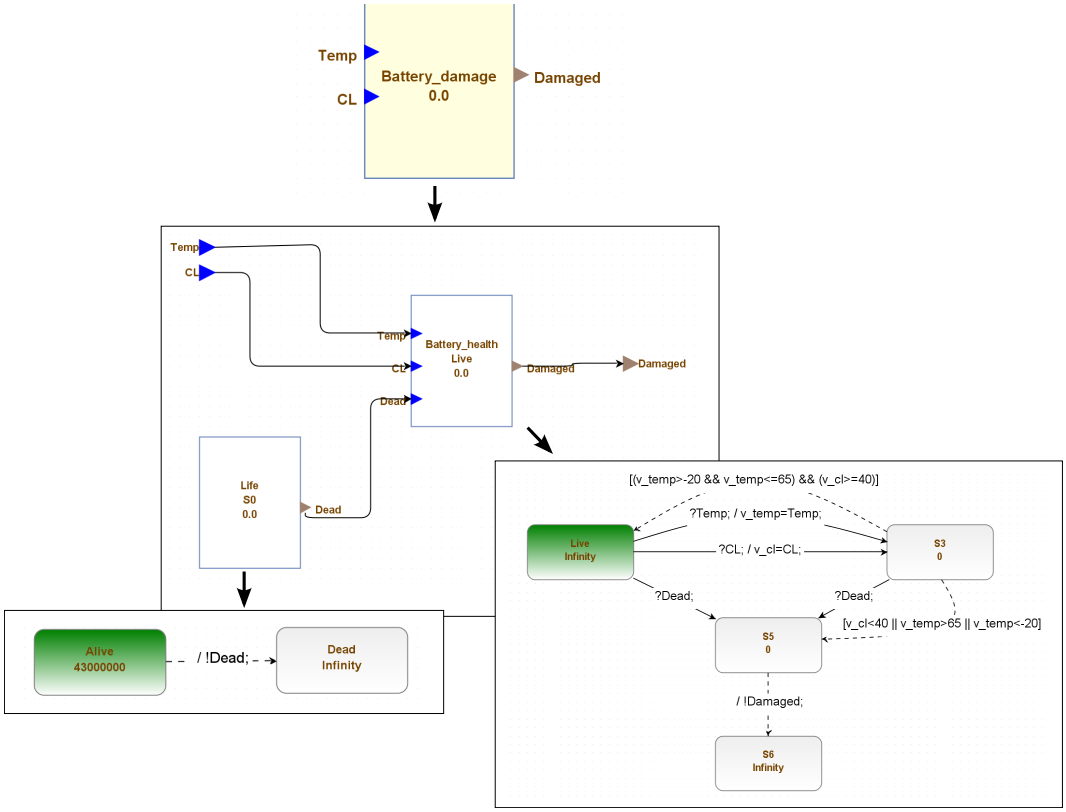


FIGURE 5.18 – Modèle ProDEVS "Battery-damage".

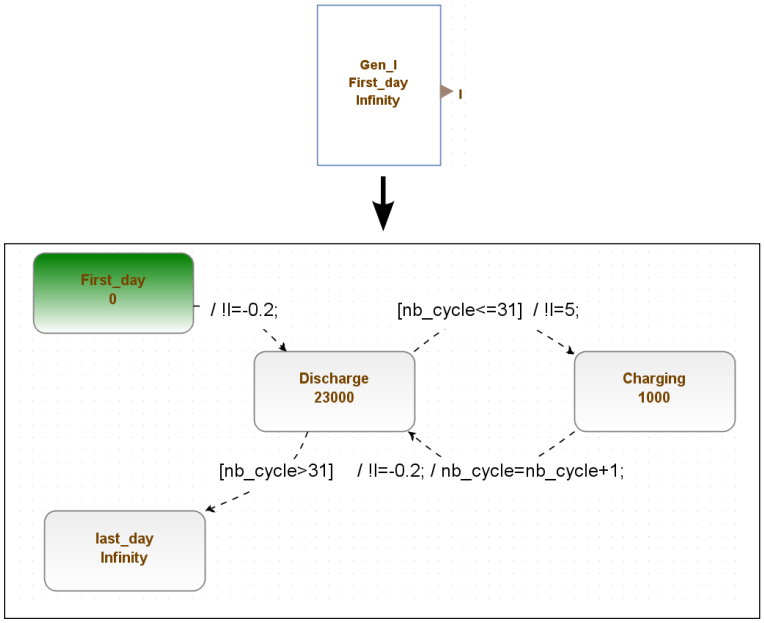


FIGURE 5.21 – Générateur Charge/Décharge.

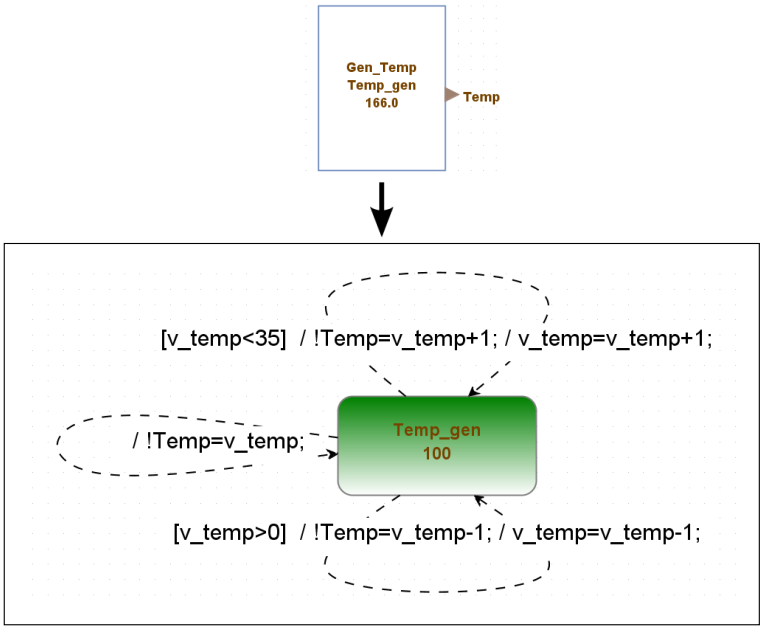


FIGURE 5.20 – Générateur température

La présentation de ce générateur montre déjà l'absence de reconditionnement de la batterie (Cond). Le modèle est donc plus capable que nécessaire. L'utilisateur décide de bouchonner cette entrée (opération de restriction).

Le composant "Trans-obj2" est présenté en figure 5.22. Relativement similaire à "Trans-obj1", le transducteur attend en entrée une tension inférieure ou égale à 12V. Si la tension ne descend pas en dessous de 10V, la batterie n'est alors pas considérée comme détruite.

Comme pour l'objectif n°1, les modèles sont transformés pour générer les graphes de classes associés. Pour rappel, à ce stade le graphe de classes du modèle (resp. du cadre expérimental) est considéré comme une image de ce dont est capable le modèle (resp. le cadre expérimental). Il est possible d'appliquer un sous-ensemble d'hypothèses au modèle pour en réduire préalablement les capacités (e.g. limiter le domaine d'une variable, interdire certaines transitions internes).

5.3.6 Objectif de simulation n°2 : Analyse

Les différents graphes de classes sont générés. A titre d'exemple, le générateur de température "Gen-Temp" représenté par la figure 5.20 génère pas moins de 116259 classes et 227983 transitions. Tout comme pour l'objectif n°1, nous appliquons une sélection d'hypothèses aux graphes de classes et obtenons un nouveau graphe que nous appelons "graphe de classes marqué". Afin de réduire au maximum l'analyse de ce second objectif, nous considèrerons que la compatibilité de périmètre a été menée. Nous pouvons maintenant étudier la compatibilité dynamique du cadre ex-

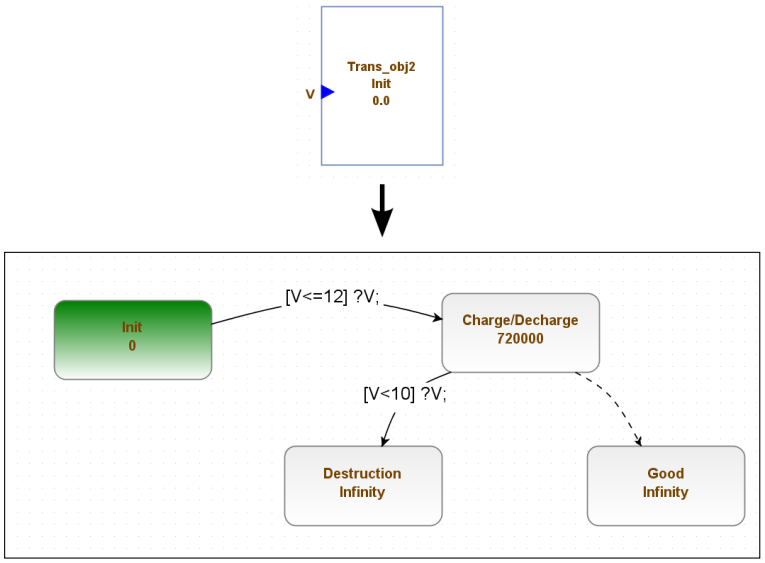


FIGURE 5.22 – Transducteur pour l'objectif n°2.

périmental avec le modèle.

Après avoir sélectionné un ensemble de métriques d'intérêt, l'utilisateur de la simulation obtient les résultats suivants concernant le générateur. (figure 5.23).

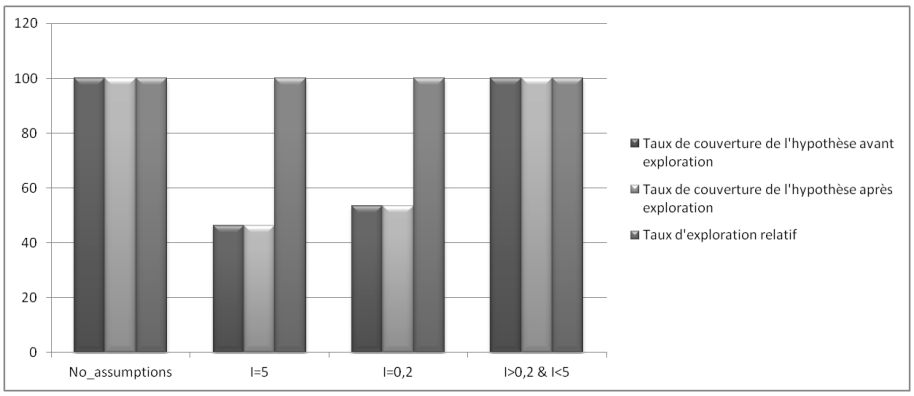


FIGURE 5.23 – Récapitulatif des taux d'exploration du générateur pour l'objectif n°2

Aux vu des résultats de la figure 5.23, l'utilisateur de la simulation semble avoir un indice de confiance important. En effet, le générateur semble avoir une compatibilité "parfaite" avec le modèle.

Intéressons-nous maintenant aux résultats fournis par le graphe de classes du modèle[8], présenté dans la figure 5.24.

8. Afin de limiter l'explosion combinatoire du modèle, certaines hypothèses de restriction ont été mises en place dans le modèle (e.g, non prise en compte de Cond). Ceci améliore artificiellement

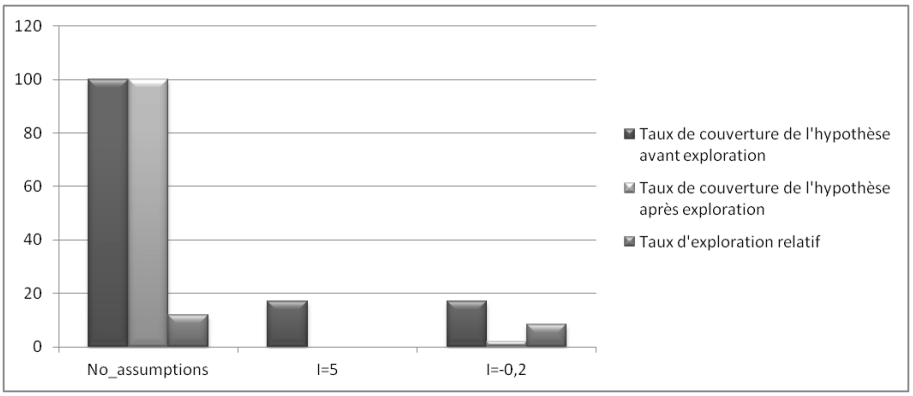


FIGURE 5.24 – Récapitulatif des taux d'exploration du modèle pour l'objectif n°2

Il devient difficile, voire impossible d'expliquer les taux de couverture. Pourtant, il apparaît clairement qu'aucun état ne prend la valeur I=5. Ce qui, à première vue, peut paraître étrange. En effet, nous avons vu que le générateur, lui, a pu fournir l'ensemble des valeurs au modèle et a été totalement parcouru.

Ce résultat s'explique par l'erreur introduite dans le générateur. Normalement, le modèle attend de passer par une étape I=0 avant de partir vers une charge ou une décharge, ce qui cause l'incompatibilité. Le principe de gestion de charge et de décharge est illustré dans le schéma de la figure 5.17. Le générateur fournit bien une tension à l'instant prévu (il y a avancement du générateur) mais le modèle ne reçoit pas la bonne valeur (non respect du guard), il reste dans le même état, ce qui explique l'avancée du générateur (bon taux de couverture) mais pas du modèle (mauvais taux de couverture). Si la localisation de l'erreur est simple pour nous, il n'en est pas de même pour l'utilisateur de la simulation. L'objectif de cette thèse n'est pas de cibler l'erreur, mais une étude approfondie de la trace d'exécution permettrait cependant de détecter cette erreur, ceci en utilisant la première proposition de cette thèse (voir section 3.4).

Nous passerons sous silence l'étude de la compatibilité dynamique du modèle avec le transducteur qui n'apporte aucun élément particulier par rapport à l'étude de l'objectif n°1.

5.4 Résumé

Ce chapitre avait pour objet d'illustrer les propositions faites dans ce manuscrit. La méthodologie présentée au chapitre 4 a été appliquée dans un contexte d'ingénierie système tel que présenté aux chapitres 1 et 2. Ceci a permis de mettre en évidence l'utilité de l'approche proposée dans le chapitre 3.

le taux d'exploration. De plus, nous avons été obligés de limiter artificiellement la résolution et le domaine des variables.

Cet exemple propose une vision à deux niveaux. Grâce à la simplicité de l'objectif n°1 nous avons pu revenir sur chacune des étapes de la méthodologie, même si certaines sont automatisées et donc transparentes pour l'utilisateur. L'objectif n°2, plus complet, a montré la faisabilité d'une telle méthodologie pour un système plus complexe. Nous avons pu mettre en évidence l'importance des hypothèses statiques et la nécessité de définir un langage dédié, comme nous l'avions déjà noté dans les discussions du chapitre 3. Cet exemple met en avant certains manques dans les rouages de la méthodologie, comme l'existence d'un générateur de graphe de classes à partir d'un modèle DEVS, permettant de limiter les erreurs de transformation vers un autre formalisme.

Pour finir, cet exemple a permis de mettre en action les métriques génériques proposées dans le chapitre 3. Cet exemple reste limité et ne peut réellement mettre en évidence la pleine utilité de la méthode proposée. Une grande partie de la démarche d'ingénierie système est passée sous silence, l'utilisation d'hypothèses rationnelles pour s'assurer que nous respectons certaines contraintes fondamentales n'a pas été introduite. On pense par exemple à l'évolution de la variable de température. Il est tout à fait envisageable de s'assurer qu'elle respecte une progression cohérente (pas de saut brusque de température).

L'utilisation de notre méthodologie apporte à l'utilisateur de la simulation un outil supplémentaire pour s'assurer de la cohérence de la simulation qu'il mène. Considérant que la modélisation est incontournable quelle que soit la méthode, une chaîne automatisée depuis le modèle SiML est parfaitement envisageable, ce qui n'engendre pour l'utilisateur de la simulation qu'un coût limité, celui de l'expression des hypothèses statiques.

Conclusion générale

Ce travail de thèse s'est intéressé à la validité des modèles de simulation dans le cadre du développement des systèmes complexes et critiques. Une analyse de l'approche d'ingénierie système, et plus particulièrement de l'aspect modélisation et simulation, a permis de constater qu'il est difficile d'établir de façon directe la validité du modèle de simulation. De nombreux points sont à l'origine de cette impossibilité, comme une mauvaise formulation ou compréhension des objectifs de simulation, une incohérence implémentatoire, une restriction de l'outil de modélisation mal maîtrisé, les limites du moteur de simulation, etc. La validité d'un modèle de simulation étant définie pour un objectif de simulation, il est apparu important de proposer une approche globale de la M&S, associant un ensemble d'outils capable de détecter des incohérences entre les objectifs de simulation et les modèles du système sous test. Ces outils, à destination de l'utilisateur de la simulation, permettent l'amélioration du niveau de confiance dans le modèle de simulation et donc dans les résultats de simulation.

Notre étude se base sur la théorie de la M&S telle que proposée par B.P. Zeigler [Zeigler 1976]. En considérant le concept de cadre expérimental qui y est introduit, nous avons pu proposer un cadre méthodologique capable d'exprimer les objectifs de simulation de manière claire. Ce cadre méthodologique nous permet d'étudier les problématiques d'application et d'accommodation de la M&S, que nous regroupons sous la problématique de compatibilité.

Ainsi, notre premier objectif a été de proposer une approche capable de mesurer l'incohérence entre les objectifs de simulation et le modèle du système. En s'appuyant sur les méthodes formelles et la théorie des automates, nous avons établi un ensemble de métriques capable de mesurer le degré de compatibilité dynamique entre cadre expérimental et modèle du système d'intérêt. Pour cela, nous étudions en premier lieu la compatibilité dynamique entre automates à interface en utilisant la décomposition en arbre. Montrant les avantages, mais aussi les inconvénients d'une telle approche, nous sommes passés à l'étude de la compatibilité entre modèles DEVS en utilisant la génération de graphes de classes, autrement appelés graphes d'atteignabilité. Cette étude formelle de la compatibilité nous permet de proposer un ensemble de bonnes propriétés de la simulation.

Ensuite, nous nous sommes concentrés sur la définition d'une méthodologie qui permet de guider l'utilisateur de la simulation dans l'élaboration de métriques permettant de mesurer ce niveau de compatibilité. S'appuyant sur les concepts de l'ingénierie dirigée par les modèles, nous proposons un langage abstrait dédié à la simulation, capable de capturer les informations nécessaires à l'évaluation de la validité des modèles de simulation, tout en guidant l'utilisateur de la simulation dans la

démarche d'élaboration des mesures de compatibilité. L'usage d'un méta-langage d'action pour l'élaboration de ce DSL nous permet de proposer une description structurelle et comportementale du formalisme DEVS afin de réduire les divergences implémentatoires. Durant cette seconde étape, nous proposons un langage concret de modélisation comportementale, hiérarchique et modulaire du système, basé sur le formalisme DEVS, que nous avons intégré à l'outil ProDEVS.

Pour finir, nous avons illustré notre proposition sur l'exemple de batterie de Y. Iwasaki et AY. Levi. En appliquant la méthodologie proposée dans ce manuscrit, nous avons montré qu'il était possible d'obtenir un ensemble de mesures de compatibilité entre un cadre expérimental et un modèle, permettant d'évaluer la crédibilité associée au modèle de simulation et donc aux résultats de la simulation. L'exemple a été modélisé à partir du langage ProDEVS, ce qui a permis d'illustrer la capacité d'expression de ce langage graphique.

Perspectives

Les perspectives pour améliorer et continuer les travaux présentés dans cette thèse sont nombreuses. Deux axes de recherche se détachent : méthodologie d'élaboration des métriques et définition d'un langage dédié à la M&S .

La méthodologie d'élaboration des métriques

La méthodologie d'élaboration des métriques proposée dans cette thèse demande à être améliorée. Comme nous l'avons vu dans le chapitre 4, mais aussi avec l'exemple d'application du chapitre 5, la chaîne d'élaboration des métriques comporte de nombreuses étapes. Plusieurs améliorations sont à envisager. L'absence d'outils performants pour la génération des graphes de classes à partir d'un modèle DEVS oblige aujourd'hui à effectuer plusieurs transformations, sources potentielles d'erreurs. Évoquée dans cette thèse, l'exploration d'autres possibilités d'application des hypothèses statiques est à envisager. Les possibilités offertes par l'utilisation de propositions de la logique temporelle CTL* pourraient permettre l'application d'hypothèses sur les chemins mais aussi sur les états, augmentant considérablement l'expressivité des hypothèses et donc l'efficacité des métriques. Une autre piste à envisager est l'application de la théorie des fonctions de croyances, plus connue sous l'appellation théorie de Dempster-Shafer, qui propose d'utiliser des fonctions de croyance et de raisonnement plausibles pour calculer la probabilité d'apparition d'un évènement. Sans qu'aucune étude de faisabilité n'ait été faite ici, il semble intéressant d'étudier la théorie pour apporter un ensemble de métriques probabilistes sur l'état de croyance de la validité du modèle de simulation.

La définition d'un langage dédié à la M& S

La proposition faite d'un langage dédié à la M& S dans cette thèse n'est qu'une base de réflexion permettant de regrouper paramètres, objectifs de simulation, modèles et hypothèses sous un même formalisme. Une extension de ce langage abstrait permettrait une plus grande efficacité des métriques. Nous avons vu dans le chapitre 4 la nécessité d'y associer un langage d'hypothèses. Le cadre expérimental n'est alors plus limité à un triplet : générateur, transducteur et accepteur, mais à un ensemble plus complexe dont certaines interactions sont encore à décrire. Nous rejoignons ici la réflexion émise par B.P. Zeigler sur la nécessité d'un langage dédié à la M&S capable de gérer ces interactions. La proposition de langage concret faite dans ProDEVS n'est qu'une ébauche. Il sera nécessaire d'étendre le langage abstrait, et d'élaborer une version concrète du langage offrent une syntaxe adaptée aux différents acteurs de la modélisation et de la simulation.

On pourra reprocher à la proposition faite dans cette thèse de ne pas avoir été confrontée à un cas réel. A ce stade, il est peu envisageable d'étudier des cas complexes. En effet, l'utilisation des méthodes de model-checking nous confronte à l'explosion combinatoire et limite aujourd'hui la complexité des modèles étudiés. Seule l'élaboration d'un langage d'hypothèses autorisant la restriction des modèles peut permettre l'utilisation de cette méthodologie avec un cas complexe. En termes M& S, il s'agit de s'intéresser aux relations de dérivabilité et de morphisme définies par B.P. Zeigler.

L'idée est de générer un modèle plus abstrait, à partir d'un ensemble d'hypothèses et de l'objectif de simulation. Si l'application d'hypothèses statiques est une réponse acceptable et permet une réduction significative du modèle, une solution performante semble reposer sur la génération d'hypothèses plus complexes, issues d'une étude conjointe du comportement du modèle et d'un ensemble d'hypothèses statiques, en fonction de l'objectif de simulation pour générer une abstraction du modèle.

Bibliographie

[Abid 2011] N Abid, S Dal Zilio et D Le Botlan. *Verification of Real-Time Specification Patterns on Time Transition Systems*. Rapport technique n ° 11365, 2011. (Cité en page 109.)

[Accord 2002a] Projet Accord. *Assemblage de composants par contrats en environnement ouvert et réparti*. RNRT 06, 2002. (Cité en page 120.)

[Accord 2002b] Projet Accord et France Telecom R&D Bretagne. *Conventions communes aux profils UML*. Rapport technique, Technical report, RNTL, 2002. (Cité en pages 120 et 123.)

[AFIS 2014] AFIS. *Association Francaise d'Ingenierie Systeme (AFIS)*, http ://www.afis.fr 2014. (Cité en pages 8, 12 et 20.)

[Akiyama 1971] Fumio Akiyama. *An Example of Software System Debugging*. In IFIP Congress (1), volume 71, pages 353–359, 1971. (Cité en page 70.)

[Albert 2009] Vincent Albert. *Evaluation de la validité de la simulation dans le cadre du développement des systèmes embarqués*. PhD thesis, Université Paul Sabatier-Toulouse III, 2009. (Cité en pages vii, 2, 40, 48, 53, 56 et 110.)

[Albert 2012] Vincent Albert. Support de cours en : Modélisation et simulation des systèmes complexes. Université Paul Sabatier, Toulouse, France, September 2012. (Cité en pages vii et 38.)

[Alfaro 2001] Luca Alfaro et Thomas A. Henzinger. *Interface automata*. In Proceedings of the Ninth Annual Symposium on Foundations of Software Engineering (FSE), ACM, pages 109–120. Press, 2001. (Cité en page 110.)

[Alur 1990] Rajeev Alur et David Dill. *Automata for modeling real-time systems*. In Automata, languages and programming, pages 322–335. Springer, 1990. (Cité en page 91.)

[Alur 1994] Rajeev Alur et David L Dill. *A theory of timed automata*. Theoretical computer science, vol. 126, no. 2, pages 183–235, 1994. (Cité en pages 95 et 102.)

[Amrani 2012] Moussa Amrani. *A formal semantics of Kermeta*. 2012. (Cité en page 133.)

[Apfelbaum 1997] Larry Apfelbaum et John Doyle. *Model based testing*. In Software Quality Week Conference, pages 296–300, 1997. (Cité en page 30.)

[Applegate 2001] David Applegate, Robert Bixby, Vašek Chvátal et William Cook. *TSP cuts which do not conform to the template paradigm*. In Computational Combinatorial Optimization, pages 261–303. Springer, 2001. (Cité en page 33.)

[Badey 2012] Quentin Badey. *Étude des mécanismes et modélisation du vieillissement des batteries lithium-ion dans le cadre d'un usage automobile*. PhD thesis, Université Paris Sud-Paris XI, 2012. (Cité en page 142.)

[Balci 1998] Osman Balci. *Verification, validation, and accreditation.* In Proceedings of the 30th conference on Winter simulation, pages 41–4. IEEE Computer Society Press, 1998. (Cité en page 108.)

[Beizer 1990] Boris Beizer. *Software testing techniques.* New York, ISBN : 0-442-20672-0, 1990. (Cité en pages 25 et 27.)

[Bengtsson 1996] Johan Bengtsson, Kim Larsen, Fredrik Larsson, Paul Pettersson et Wang Yi. Uppaal a tool suite for automatic verification of real-time systems. Springer, 1996. (Cité en page 135.)

[Berger 2013] Christian Berger, Delf Block, Christian Hons, Stefan Kühnel, André Leschke, Bernhard Rumpe et Torsten Strutz. *Meta-metrics for simulations in software engineering on the example of integral safety systems.* In Proceedings des, volume 14, pages 136–148, 2013. (Cité en page 72.)

[Bergero 2011] Federico Bergero et Ernesto Kofman. *PowerDEVS : a tool for hybrid system modeling and real-time simulation.* Simulation, vol. 87, no. 1-2, pages 113–132, 2011. (Cité en page 135.)

[Bernard 2008] HOMÈS Bernard. Les tests logiciels. Lavoisier, 2008. (Cité en page 30.)

[Berners-Lee 2001] Tim Berners-Lee, James Hendler, Ora Lassila*et al. The semantic web.* Scientific american, vol. 284, no. 5, pages 28–37, 2001. (Cité en page 109.)

[Berthomieu 2004] Bernard Berthomieu, P-O Ribet et François Vernadat. *The tool TINA–construction of abstract state spaces for Petri nets and time Petri nets.* International Journal of Production Research, vol. 42, no. 14, pages 2741–2756, 2004. (Cité en pages 136, 150 et 152.)

[Berthomieu 2008] Bernard Berthomieu, Jean-Paul Bodeveix, Patrick Farail, Mamoun Filali, Hubert Garavel, Pierre Gaufillet, Frederic Lang et François Vernadat. *Fiacre : an intermediate language for model verification in the TOPCASED environment.* In ERTS 2008, 2008. (Cité en pages 135 et 150.)

[Berthomieu 2014] Bernard Berthomieu, Silvano Dal Zilio, Łukasz Fronc et François Vernadat. *Time petri nets with dynamic firing dates : Semantics and applications.* In Formal Modeling and Analysis of Timed Systems, pages 85–99. Springer, 2014. (Cité en page 136.)

[Bézivin 2001] Jean Bézivin et Olivier Gerbé. *Towards a precise definition of the OMG/MDA framework.* In Automated Software Engineering, 2001.(ASE 2001). Proceedings. 16th Annual International Conference on, pages 273–280. IEEE, 2001. (Cité en page 34.)

[Bhattacharya 2005] Sutirtha Bhattacharya et Dewayne E Perry. *Contextual reusability metrics for event-based architectures.* In Empirical Software Engineering, 2005. 2005 International Symposium on, pages 10–pp. IEEE, 2005. (Cité en page 70.)

[Bisgambiglia 2013] Paul-Antoine Bisgambiglia, Romain Franceschini, François-Joseph Chatelon, Jean-Louis Rossi et Paul Antoine Bisgambiglia. *Discrete event formalism to calculate acceptable safety distance*. In Proceedings of the 2013 Winter Simulation Conference : Simulation : Making Decisions in a Complex World, pages 217–228. IEEE Press, 2013. (Cité en page 90.)

[Boehm 1986] Barry Boehm. *A spiral model of software development and enhancement*. ACM SIGSOFT Software Engineering Notes, vol. 11, no. 4, pages 14–24, 1986. (Cité en page 15.)

[Boehm 1987] Barry W Boehm. *Improving software productivity*. In Computer. Citeseer, 1987. (Cité en pages 25 et 28.)

[Boehm 1988] Barry W. Boehm. *A spiral model of software development and enhancement*. Computer, vol. 21, no. 5, pages 61–72, 1988. (Cité en page 12.)

[Breton 2001] Erwan Breton et Jean Bézivin. *Towards an understanding of model executability*. In Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001, pages 70–80. ACM, 2001. (Cité en pages 116, 124 et 127.)

[Brito e Abreu 1996] Fernando Brito e Abreu et Walcelio Melo. *Evaluating the impact of object-oriented design on software quality*. In Software Metrics Symposium, 1996., Proceedings of the 3rd International, pages 90–99. IEEE, 1996. (Cité en page 70.)

[Brooks Jr 1995] Frederick P Brooks Jr. The mythical man-month, anniversary edition : Essays on software engineering. Pearson Education, 1995. (Cité en page 25.)

[Bruck 2007] James Bruck et Kenn Hussey. *Customizing UML : Which technique is right for you*. White paper, Eclipse UML Project, page 47, 2007. (Cité en pages viii, 117 et 118.)

[Budinsky 2004] F Budinsky, D Steinberg, R Ellersick et TJ Grose. *Book, Chapter 5 Ecore Modeling Concepts.*, 2004. (Cité en page 114.)

[Cai 2000] Xia Cai, Michael R Lyu, Kam-Fai Wong et Roy Ko. *Component-based software engineering : technologies, development frameworks, and quality assurance schemes*. In Software Engineering Conference, 2000. APSEC 2000. Proceedings. Seventh Asia-Pacific, pages 372–379. IEEE, 2000. (Cité en page 70.)

[Calimet 2009] Stéphane Calimet, Philippe Fimin et Eric Leleu. *Les Test : L'état de l'Art*, 2009. (Cité en page 30.)

[Capocchi 2011] Laurent Capocchi, Jean-François Santucci, Bastien Poggi et Celine Nicolai. *DEVSimPy : A collaborative python software for modeling and simulation of DEVS systems*. In 2nd International Track on Collaborative Modeling & Simulation-CoMetS'11, pages 6–pages. IEEE, 2011. (Cité en pages 130 et 136.)

[Cellier 2006] François E Cellier et Ernesto Kofman. Continuous system simulation. Springer Science & Business Media, 2006. (Cité en page 135.)

[Chabrier 2007] P Chabrier, F Garcia, R Martin-Clouaire, G Quesnel et H Raynal. *Toward a simulation modeling platform for studying cropping systems management : the Record project.* In International Congress on Modelling and Simulation, International Society for Computer Simulation, pages 10–13, 2007. (Cité en page 90.)

[Chidamber 1991] Shyam R Chidamber et Chris F Kemerer. Towards a metrics suite for object oriented design, volume 26. ACM, 1991. (Cité en page 70.)

[Cho 1998] Seong Myun Cho et Tag Gon Kim. *Real-time DEVS simulation : Concurrent, time-selective execution of combined RT-DEVS model and interactive environment.* In Proceeding of 1998 Summer Simulation Conference, Reno, Nevada, 1998. (Cité en page 90.)

[Chow 1994] Alex Chung Hen Chow et Bernard P Zeigler. *Parallel DEVS : a parallel, hierarchical, modular, modeling formalism.* In Proceedings of the 26th conference on Winter simulation, pages 716–722. Society for Computer Simulation International, 1994. (Cité en page 90.)

[Clapaud 2012] A. Clapaud. *PSA : deux fois moins de tests physiques pour concevoir la Peugeot 208.* 2012. (Cité en page 35.)

[Clark 2004] Tony Clark, Paul Sammut et James Willans. *Applied metamodelling.* book to be published, 2004. (Cité en page 117.)

[Clark 2008] Tony Clark, Paul Sammut et James Willans. *Applied metamodelling : a foundation for language driven development.* 2008. (Cité en page 114.)

[Clarke 1982] Edmund M Clarke et E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. Springer, 1982. (Cité en page 147.)

[Combemale 2006a] B Combemale, S Rougemaille, X Cregut, F Migeon, M Pantel et C Maurel. *Experiences pour decrire la semantique en ingenierie des modeles.* 2eme journee IDM, 2006. (Cité en page 126.)

[Combemale 2006b] Benoit Combemale, Sylvain Rougemaille, Xavier Cregut, Frederic Migeon, Marc Pantel et Christine Maurel. *Experiences pour d'ecrire la semantique en ingenierie des modeles.* IDM06, pages 18–19, 2006. (Cité en pages 112 et 116.)

[Combemale 2008] Benoit Combemale. *Approche de metamodelisation pour la simulation et la verification de modele–Application a l'ingenierie des procedes.* PhD thesis, Institut National Polytechnique de Toulouse-INPT, 2008. (Cité en page 115.)

[CORP 2002] OMG CORP. *UML Profile For CORBA, OMG Formally Released Versions Of CORP.* OMG, Needham, 2002. (Cité en page 120.)

[Crothaus 2011] M. Crothaus. *iPad 2 would have bested 1990s-era supercomputers.* 2011. (Cité en page 35.)

[Cusumano 1995] M Cusumano et Richard W Selby. *Microsoft Secrets.* New York : The FreePress, 1995. (Cité en page 25.)

[Dacharry 2005] Hernan Dacharry et Norbert Giambiasi. *From timed automata to devs models : Formal verification*. Proceedings of SpringSim, vol. 5, 2005. (Cité en page 134.)

[De Alfaro 2001] Luca De Alfaro et Thomas A Henzinger. *Interface automata*. ACM SIGSOFT Software Engineering Notes, vol. 26, no. 5, pages 109–120, 2001. (Cité en page 73.)

[Degueule 2014] Thomas Degueule, Olivier Barais, Arnaud Blouin et Benoit Combemale. *The K3 Model-Based Language Workbench*. 2014. (Cité en page 126.)

[Desfray 2000] Philippe Desfray. *UML Profiles versus Metamodel extensions : An ongoing debate*. In OMG s UML Workshops : UML in the. com Enterprise : Modeling CORBA, Components, XML/XMI and Metadata Workshop, pages 6–9, 2000. (Cité en page 117.)

[Di Giampaolo 2010] B. Di Giampaolo, G Geeraerts, J.F. Raskin et N. Sznajder. *Safraless procedures for timed specifications*. In 8th International Conference on Formal Modelling and Analysis of Timed Systems, 2010. (Cité en page 109.)

[Dill 1990] David L Dill. *Timing assumptions and verification of finite-state concurrent systems*. In Automatic verification methods for finite state systems, pages 197–212. Springer, 1990. (Cité en page 95.)

[DMSO 2000] DMSO. *RPG Reference Document*. Defense Modeling and Simulation Office (DMSO), 2000. (Cité en page 42.)

[DoD 1994] DoD. *5000.59*. DoD Modeling and Simulation (M&S) Management, vol. 4, 1994. (Cité en page 41.)

[DoD 2000] DoD. *Validation and Accreditation (VV&A) Recommended Practices Guide*. Office of the Director, Defense Research and Engineering, Defense Modeling and Simulation Office, 2000. (Cité en pages 64 et 65.)

[Efftinge 2012] Sven Efftinge, Moritz Eysholdt, Jan Köhnlein, Sebastian Zarnekow, Robert von Massow, Wilhelm Hasselbring et Michael Hanus. *Xbase : implementing domain-specific languages for Java*. In ACM SIGPLAN Notices, volume 48, pages 112–121. ACM, 2012. (Cité en page 126.)

[EIA 2003] EIA. *latest version of Processes for Engineering a System : EIA-632 (1999)*. EIA Electronics Industries Alliance, September 2003. (Cité en pages 9, 17 et 27.)

[Eickhoff 2009] Jens Eickhoff et Hans-Peter Roeser. Simulating spacecraft systems. Springer, 2009. (Cité en page 36.)

[Emerson 1986] E Allen Emerson et Joseph Y Halpern. *Sometimes and not never revisited : on branching versus linear time temporal logic*. Journal of the ACM (JACM), vol. 33, no. 1, pages 151–178, 1986. (Cité en page 147.)

[Evrot 2008] Dominique Evrot. *Contribution à la vérification d'exigences de sécurité : application au domaine de la machine industrielle*. PhD thesis, Université Henri Poincaré-Nancy I, 2008. (Cité en page 16.)

[Falkenhainer 1991] Brian Falkenhainer et Kenneth D. Forbus. *Compositional modeling : finding the right model for the job.* Artif. Intell., vol. 51, no. 1-3, pages 95–143, 1991. (Cité en page 48.)

[Felix 2009] Patrick Felix. *Cours : Test et Validation du Logiciel.* 2009. (Cité en page 30.)

[Feurzeig 1969] Wallace Feurzeig*et al.* *Programming-Languages as a Conceptual Framework for Teaching Mathematics. Final Report on the First Fifteen Months of the LOGO Project.* 1969. (Cité en page 130.)

[Filippi 2004] Jean-Batiste Filippi et Paul Bisgambiglia. *JDEVS : an implementation of DEVS based formal framework for environmental modellinf.* Environmental Modelling and Software, 2004. (Cité en page 90.)

[Forsberg 2005] Kevin Forsberg, Hal Mooz et Howard Cotterman. Visualizing project management : Models and frameworks for mastering complex systems. Wiley. com, 2005. (Cité en page 12.)

[Foures 2011] Damien Foures, Albert Vincent et Jean-Claude Pascal. *ActivityDiagram2PetriNet : Transformation-Based Model in Accordance with the OMG SysML Specification.* In The 2011 European Simulation and Modelling Conference, 2011. (Cité en page 115.)

[Foures 2012] D Foures, V Albert et A Nketsa. *Formal compatibility of experimental frame concept and finite and deterministic DEVS model.* In International Conference on Modeling Optimization SIMulation (MOSIM 2012), pages – 10, Jun 2012. (Cité en page 94.)

[Fowler 2010] Martin Fowler. Domain-specific languages. Pearson Education, 2010. (Cité en page 108.)

[Franceschini 2014] Romain Franceschini, Paul-Antoine Bisgambiglia, Paul Bisgambiglia et David Hill. *DEVS-ruby : a domain specific language for DEVS modeling and simulation (WIP).* In Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative, page 15. Society for Computer Simulation International, 2014. (Cité en page 130.)

[Frantz 1995] Frederick K. Frantz. *A taxonomy of model abstraction techniques.* In WSC '95 : Proceedings of the 27th conference on Winter simulation, pages 1413–1420, Washington, DC, USA, 1995. IEEE Computer Society. (Cité en page 48.)

[Genero 2000] Marcela Genero, Mario Piattini et Coral Calero. *Early measures for UML class diagrams.* L'Objet, vol. 6, no. 4, pages 489–515, 2000. (Cité en page 70.)

[Giambiasi 2003] Norbert Giambiasi, Jean-Luc Paillet et Frédéric Châne. *Simulation and verification II : from timed automata to DEVS models.* In Proceedings of the 35th conference on Winter simulation : driving innovation, pages 923–931. Winter Simulation Conference, 2003. (Cité en page 92.)

[Girard 2005] Antoine Girard et George J Pappas. *Approximation metrics for discrete and continuous systems.* 2005. (Cité en page 71.)

[Girard 2008] Antoine Girard, A Agung Julius et George J Pappas. *Approximate simulation relations for hybrid systems.* Discrete Event Dynamic Systems, vol. 18, no. 2, pages 163–179, 2008. (Cité en page 71.)

[Godfrey-Smith 2009] Peter Godfrey-Smith. Theory and reality : An introduction to the philosophy of science. University of Chicago Press, 2009. (Cité en page 40.)

[Goldberg 1991] David Goldberg. *What every computer scientist should know about floating-point arithmetic.* ACM Computing Surveys (CSUR), vol. 23, no. 1, pages 5–48, 1991. (Cité en page 138.)

[Gronback 2004] Richard Gronback. *Model validation : Applying audits and metrics to uml models.* In Borland Developer Conference, 2004. (Cité en page 70.)

[Gross 1999] DC Gross. *Report from the fidelity definition and metrics implementation study group (FDM-ISG).* In Proceedings of the 1999 Spring Simulation Interoperability Workshop, 1999. (Cité en page 65.)

[Gui 2009] Gui Gui et Paul D Scott. *Measuring software component reusability by coupling and cohesion metrics.* Journal of computers, vol. 4, no. 9, pages 797–805, 2009. (Cité en page 70.)

[Guillerm 2011] R. Guillerm. *Intégration de la sûreté de fonctionnement dans les processus d'ingénierie système.* PhD thesis, Université Paul Sabatier-Toulouse III, 2011. (Cité en pages 9 et 108.)

[Haskins 2011] Cecilia Haskins et Kevin Forsberg. *Systems Engineering Handbook : A Guide for System Life Cycle Processes and Activities ; INCOSE-TP-2003-002-03.2. 1.* Incose, 2011. (Cité en pages 9 et 10.)

[Hayes-Roth 1983] Frederick Hayes-Roth, Donald A Waterman et Douglas B Lenat. *Building expert system.* 1983. (Cité en page 104.)

[Heon 2010] Michel Heon, Josianne Basque, Gilbert Paquette*et al.* *Validation de la semantique d'un modele semi-formel de connaissances avec OntoCASE.* Acte des 21 emes 21es Journees Francophones d'Ingenierie des Connaissances, 2010. (Cité en page 114.)

[Hetzel 1988] William C Hetzel et Bill Hetzel. The complete guide to software testing. QED Information Sciences Wellesley, MA, 1988. (Cité en page 30.)

[Hwang 2005] HM Hwang. *An introduction to input/output automata.* 2005. (Cité en page 96.)

[Hwang 2009] Moon Ho Hwang et Bernard P Zeigler. *Reachability graph of finite and deterministic devs networks.* IEEE Transactions on Automation Science and Engineering, vol. 6, no. 3, page 468, 2009. (Cité en pages viii, 95, 96, 99, 135 et 151.)

[IEEE 2005] IEEE. *IEEE Standard for Application and Management of the Systems Engineering Process.* IEEE Std 1220-2005 (Revision of IEEE Std 1220-1998), 2005. (Cité en pages 9 et 17.)

[INCOSE 2004] INCOSE. *Systems Engineering Handbook : INCOSE 2. 1.* INCOSE, 2004. (Cité en page 8.)

[INCOSE 2013] INCOSE. *Brief History of Systems Engineering*, 2013. (Cité en page 10.)

[IRISA 2013] IRISA. *Kermeta*, 2013. (Cité en pages viii et 131.)

[IRISA 2015] IRISA. *Kermeta*, 2015. (Cité en pages viii, 125 et 126.)

[ISO/IEC 2008] ISO/IEC. *Systems and software engineering System life cycle processes.* ISO/IEC 15288 :2008IEEE Std 15288-2008 (Revision of IEEE Std 15288-2004), pages 1–84, 2008. (Cité en page 17.)

[IVVQ 2014] groupe de travail IVVQ. *Association Francaise d'Ingenierie Systeme (AFIS), groupe de travail Integration, Validation, Verification et Qualitification*, http ://www.afis.fr 2014. (Cité en pages vii et 28.)

[Iwasaki 1993] Yumi Iwasaki. *Reasoning with multiple abstraction models.* Recent advances in qualitative physics, pages 67–82, 1993. (Cité en page 48.)

[Iwasaki 1994] Yumi Iwasaki et Alon Y Levy. *Automated model selection for simulation.* In AAAI, pages 1183–1190. Citeseer, 1994. (Cité en pages 3, 141 et 142.)

[Jézéquel 2011] Jean-Marc Jézéquel, Olivier Barais et Franck Fleurey. *Model driven language engineering with kermeta.* In Generative and Transformational Techniques in Software Engineering III, pages 201–221. Springer, 2011. (Cité en page 130.)

[Jouault 2006a] Frederic Jouault et Jean Bezivin. *KM3 : a DSL for Metamodel Specification.* In Formal Methods for Open Object Based Distributed Systems, pages 171–185. Springer, 2006. (Cité en page 114.)

[Jouault 2006b] Frederic Jouault et Ivan Kurtev. *Transforming models with ATL.* In Satellite Events at the MoDELS 2005 Conference, pages 128–138. Springer, 2006. (Cité en page 115.)

[Karlsson 2008] Daniel Karlsson, Petru Eles et Zebo Peng. *Model validation for embedded systems using formal method-aided simulation.* Computers & Digital Techniques, IET, vol. 2, no. 6, pages 413–433, 2008. (Cité en page 71.)

[Kim 2009] Sungung Kim, Hessam S Sarjoughian et Vignesh Elamvazhuthi. *DEVS-suite : a simulator supporting visual experimentation design and behavior monitoring.* In Proceedings of the 2009 Spring Simulation Multiconference, page 161. Society for Computer Simulation International, 2009. (Cité en page 136.)

[Kofman 2001] Ernesto Kofman et Sergio Junco. *Quantized-state systems : a DEVS Approach for continuous system simulation.* Transactions of the Society for Modeling and Simulation International, vol. 18, no. 3, pages 123–132, 2001. (Cité en page 135.)

[Kofman 2003] Ernesto Kofman, Marcelo Lapadula et Esteban Pagliero. *PowerDEVS : A DEVS–based environment for hybrid system modeling and simulation.* School of Electronic Engineering, Universidad Nacional de Rosario, Tech. Rep. LSD0306, 2003. (Cité en page 136.)

[Kripke 1963] Saul A Kripke. *Semantical considerations on modal logic.* 1963. (Cité en page 147.)

[Kwon 1996] Yi Wan Kwon, H Park, S Jung et Tag Gon Kim. *Fuzzy-DEVS formalism : concepts, realization and applications.* In Proceedings Of The 1996 Conference On AI, Simulation and Planning In High Autonomy Systems, pages 227–234. Citeseer, 1996. (Cité en page 90.)

[Le Moigne 1990] Jean-Louis Le Moigne. La modélisation des systèmes complexes, volume 2. Dunod Paris, 1990. (Cité en page 9.)

[Ledeczi 2001] Akos Ledeczi, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle et Peter Volgyesi. *The generic modeling environment.* In Workshop on Intelligent Signal Processing, Budapest, Hungary, volume 17, 2001. (Cité en page 114.)

[Liao 2014] Xiangke Liao, Liquan Xiao, Canqun Yang et Yutong Lu. *Milkyway-2 supercomputer : system and application.* Frontiers of Computer Science, vol. 8, no. 3, pages 345–356, 2014. (Cité en page 33.)

[Lin 2013] Qi Lin et Zhi Li. *Credibility Evaluation of Simulation Models.* Advanced Materials Research, vol. 765, pages 713–716, 2013. (Cité en page 64.)

[Liu 2005] Fei Liu, Ming Yang et Zicai Wang. *Study on simulation credibility metrics.* In Proceedings of the 37th conference on Winter simulation, pages 2554–2560. Winter Simulation Conference, 2005. (Cité en pages 64 et 65.)

[Lothaire 1983] M Lothaire. *Combinatorics on Words, volume 17 of Encyclopedia of Mathematics and its Applications*, 1983. (Cité en page 72.)

[Lothaire 1997] M_ Lothaire. Combinatorics on words. Cambridge University Press, 1997. (Cité en page 72.)

[Lynch 1988] Nancy A Lynch et Mark R Tuttle. *An introduction to input/output automata.* 1988. (Cité en page 73.)

[Mar 2011] *UML Profile For MARTE : Modeling And Analysis Of Real-Time Embedded Systems*, 2011. Version 1.1, OMG. (Cité en page 117.)

[Mavrovouniotis 2012] Michael Mavrovouniotis. Artificial intelligence in process engineering. Elsevier, 2012. (Cité en page 104.)

[Meinadier 1998] Jean-Pierre Meinadier. Ingénierie et intégration des systèmes. Hermes, 1998. (Cité en page 8.)

[Mellor 1999] Stephen J Mellor, Steve Tockey, Rodolphe Arthaud et Philippe LeBlanc. *Softwareplatform-independent, precise action specifications for UML.* The Unified Modeling Language, UML, pages 281–286, 1999. (Cité en page 116.)

[Meyer 1997] Marc H Meyer, Peter Tertzakian et James M Utterback. *Metrics for managing research and development in the context of the product family.* Management Science, vol. 43, no. 1, pages 88–111, 1997. (Cité en page 72.)

[Michel 2004] Fabien Michel. *Formalisme, outils et éléments méthodologiques pour la modélisation et la simulation multi-agents.* These de doctorat, Université Montpellier II, vol. 129, 2004. (Cité en page 138.)

[Minsky 1968] Marvin Lee Minsky, Marvin Minsky et Marvin Minsky. Semantic information processing, volume 142. MIT press Cambridge, MA, 1968. (Cité en page 34.)

[Mittal 2007] Saurabh Mittal, José Luis Risco-Martín et Bernard P Zeigler. *DEVSML : automating DEVS execution over SOA towards transparent simulators.* In Proceedings of the 2007 spring simulation multiconference-Volume 2, pages 287–295. Society for Computer Simulation International, 2007. (Cité en page 126.)

[Montalk 1993] JP Potocki De Montalk. *Computer software in civil aircraft.* Microprocessors and Microsystems, vol. 17, no. 1, pages 17 – 23, 1993. (Cité en pages 10 et 11.)

[Muller 2005] Pierre-Alain Muller, Franck Fleurey et Jean-Marc Jézéquel. *Weaving executability into object-oriented meta-languages.* In Model Driven Engineering Languages and Systems, pages 264–278. Springer, 2005. (Cité en pages 115 et 117.)

[Narasimhan 2004] V Lakshmi Narasimhan et Bayu Hendradjaya. *A new suite of metrics for the integration of software components.* In Proceedings of the The First International Workshop on Object Systems and Software Architectures (WOSSA'2004). Citeseer, 2004. (Cité en page 70.)

[Narasimhan 2009] V Lakshmi Narasimhan, PT Parthasarathy et M Das. *Evaluation of a suite of metrics for component based software engineering (cbse).* Issues in informing science and information technology, vol. 6, no. 5/6, pages 731–740, 2009. (Cité en page 70.)

[Nikora 2009] Allen P. Nikora et Galen Balcom. *Automated identification of LTL patterns in natural language requirements.* ISSRE'09, 2009. (Cité en page 109.)

[OMG 2003] MDA OMG. *Guide Version 1.0. 1.* Object Management Group, vol. 62, 2003. (Cité en page 34.)

[OMG 2004] OMG. *UML 2.0 OCL Specification.* Object Management Group, 2004. disponible sur http ://www.omg.org/docs/ptc/03-10-14.pdf. (Cité en pages 109, 115 et 123.)

[OMG 2006] OMG. *Meta Object Facility (MOF) 2.0 Core Specification*, 2006. Version 2. (Cité en page 112.)

[ONeill 1998] Tyrone ONeill et Norman Foo. *Reasoning about continuous change.* In PRICAI98 : Topics in Artificial Intelligence, pages 294–304. Springer, 1998. (Cité en page 94.)

[Pace 1999] Dale K Pace. *Development and documentation of a simulation conceptual model.* In Proceedings of the 1999 Fall Simulation Interoperability Workshop, 1999. (Cité en page 42.)

[Pasqua 2012] Roberto Pasqua, Damien Foures, Albert Vincent et Alexandre Nketsa. *From Sequence Diagrams UML 2. x To FD-DEVS By Model Transformation.* In European Simulation and Modelling Conference, pages pp–37, 2012. (Cité en page 126.)

[Paulish 2001] Daniel J Paulish et Len Bass. Architecture-centric software project management : A practical guide. Addison-Wesley Longman Publishing Co., Inc., 2001. (Cité en page 25.)

[Peterson 1981] James L Peterson. *Petri net theory and the modeling of systems.* 1981. (Cité en page 114.)

[Pnueli 1977] Amir Pnueli. *The temporal logic of programs.* In Foundations of Computer Science, 1977., 18th Annual Symposium on, pages 46–57. IEEE, 1977. (Cité en page 147.)

[Pradat-Peyre 2009] JF Pradat-Peyre et J Printz. *Pratique des tests logiciel*, 2009. (Cité en page 24.)

[Printz 1998] Jacques Printz. Puissance et limites des systèmes informatisés. Hermès, 1998. (Cité en page 22.)

[Printz 2012] Jacques Printz. Architecture logicielle-3e édition : Concevoir des applications simples, sûres et adaptables. Dunod, 2012. (Cité en page 25.)

[Printz 2013] Jacques Printz, Bernard Mesdon et Nicolas Treves. Estimation des projets de l'entreprise numérique : Approche systémique, coûts, qualité et délais. Lavoisier, 2013. (Cité en page 25.)

[Quesnel 2007] Gauthier Quesnel, Raphaël Duboz, Éric Ramat et Mamadou K Traoré. *VLE : a multimodeling and simulation environment.* In Proceedings of the 2007 summer computer simulation conference, pages 367–374. Society for Computer Simulation International, 2007. (Cité en page 130.)

[Raël 2014] Stéphane Raël, Matthieu Urbain et Hugues Renaudineau. *Modèle multiphysique de batterie lithium-ion implanté dans un logiciel de simulation des systèmes électriques.* In Symposium de Génie Électrique, 2014. (Cité en page 142.)

[Ramat 2003] E Ramat et Philippe Preux. *Virtual laboratory environment(VLE) : a software environment oriented agent and object for modeling and simulation of complex systems.* Simulation Modelling Practice and Theory, vol. 11, no. 1, pages 45–55, 2003. (Cité en page 90.)

[Ramat 2006] Eric Ramat, Vincent Ginot, Hervé Monod, Frédéric Amblard, Pierre Bommel, Juliette Rouchier, Denis Phan, Franck Varenne, Nigel Gilbert, Robert Axtell*et al. Chapitre 2. Introduction à la simulation : principaux concepts..... 37.* 2006. (Cité en page 34.)

[RED 2015] RED. *DEVS-Network*. http ://devs-network.org/stage2015-sujet1, 2015. (Cité en page 133.)

[Robertson 1991] Neil Robertson et Paul D Seymour. *Graph minors. X. Obstructions to tree-decomposition*. Journal of Combinatorial Theory, Series B, vol. 52, no. 2, pages 153–190, 1991. (Cité en page 82.)

[Royce 1970] Winston W Royce. *Managing the development of large software systems*. In proceedings of IEEE WESCON. Los Angeles, 1970. (Cité en page 12.)

[Rumbaugh 1999] James Rumbaugh, Ivar Jacobson et Grady Booch, editeurs. The unified modeling language reference manual. Addison-Wesley Longman Ltd., Essex, UK, UK, 1999. (Cité en page 70.)

[Sage 2000] A.P. Sage et J.E. Armstrong. Introduction to systems engineering. Wiley series in systems engineering. Wiley, 2000. (Cité en page 9.)

[Sakarovitch 2003] Jacques Sakarovitch. Eléments de théorie des automates. Vuibert, 2003. (Cité en page 81.)

[Sargent 2000] Robert G Sargent. *Verification, validation, and accreditation : verification, validation, and accreditation of simulation models*. In Proceedings of the 32nd conference on Winter simulation, pages 50–59. Society for Computer Simulation International, 2000. (Cité en page 65.)

[Sargent 2005] Robert G Sargent. *Verification and validation of simulation models*. In Proceedings of the 37th conference on Winter simulation, pages 130–143. Winter Simulation Conference, 2005. (Cité en pages 40 et 42.)

[Sarjoughian 2011] Hessam S Sarjoughian et Yu Chen. *Standardizing devs models : an endogenous standpoint*. In Proceedings of the 2011 Symposium on Theory of Modeling & Simulation : DEVS Integrative M&S Symposium, pages 266–273. Society for Computer Simulation International, 2011. (Cité en page 127.)

[Sarjoughian 2012] Hessam S Sarjoughian et Abbas Mahmoodi Markid. *EMF-DEVS modeling*. In Proceedings of the 2012 Symposium on Theory of Modeling and Simulation-DEVS Integrative M&S Symposium, page 19. Society for Computer Simulation International, 2012. (Cité en page 124.)

[Scheidgen 2008] Markus Scheidgen. *Textual modelling embedded into graphical modelling*. In Model Driven Architecture–Foundations and Applications, pages 153–168. Springer, 2008. (Cité en page 138.)

[Seidewitz 2003] Edwin Seidewitz. *What models mean*. IEEE software, vol. 20, no. 5, pages 26–32, 2003. (Cité en page 34.)

[Seo 2013] Chungman Seo, Bernard P Zeigler, Robert Coop et Doohwan Kim. *DEVS modeling and simulation methodology with ms4me software*. In Theory of Modeling & Simulation Symposium, SpringSim Multi-Conference, San Diego, CA, 2013. (Cité en page 136.)

[Sewell 2006] Christopher Sewell, Dan Morris, Nikolas H Blevins, Sumit Agrawal, Sanjeev Dutta, Federico Barbagli et Kenneth Salisbury. *Validating metrics*

for a mastoidectomy simulator. Studies in health technology and informatics, vol. 125, page 421, 2006. (Cité en page 71.)

[Shmoys 1985] DB Shmoys, JK Lenstra, AHG Rinnooy Kan et EL Lawler. The traveling salesman problem, volume 12. Wiley, 1985. (Cité en page 32.)

[Sintek 2002] Michael Sintek et Stefan Decker. *TRIPLE A query, inference, and transformation language for the semantic web*. In The Semantic Web ISWC 2002, pages 364–378. Springer, 2002. (Cité en page 109.)

[Soley 2000] Richard Soley*et al.* *Model driven architecture*. OMG white paper, vol. 308, page 308, 2000. (Cité en page 112.)

[Steinberg 2008] Dave Steinberg, Frank Budinsky, Ed Merks et Marcelo Paternostro. Emf : eclipse modeling framework. Pearson Education, 2008. (Cité en page 124.)

[Suthers 2003] Daniel D Suthers. *Representational guidance for collaborative inquiry*. In Arguing to Learn, pages 27–46. Springer, 2003. (Cité en page 114.)

[SysML 2006] OMG SysML. *OMG Systems Modeling Language*, 2006. (Cité en page 117.)

[Tel 2013] *UML Profile For Advanced And Integrated Telecommunication Services (TelcoML)*, 2013. Version 1.0, OMG. (Cité en page 117.)

[Tremblay 2007] Olivier Tremblay, L-A Dessaint et A-I Dekkiche. *A generic battery model for the dynamic simulation of hybrid electric vehicles*. In Vehicle power and propulsion conference, 2007. VPPC 2007. IEEE, pages 284–289. Ieee, 2007. (Cité en page 142.)

[Uml 2008] OMG Uml. *UML 1.4 with Action Semantics*. OMG, Needham, 2008. (Cité en page 117.)

[Uml 2013] OMG Uml. *2.4 OMG Unified Modeling Language*. OMG, Needham, 2013. (Cité en pages 117, 120, 123 et 138.)

[UTP 2013] *UML Testing Profile (UTP)*, 2013. Version 1.2, OMG. (Cité en page 117.)

[Van Veenendaal 2008] Erik Van Veenendaal. Foundations of software testing : Istqb certification. Cengage Learning EMEA, 2008. (Cité en page 30.)

[Vanderhaegen 2003] Frédéric Vanderhaegen. Analyse et contrôle de l'erreur humaine. Hermes science publ., 2003. (Cité en page 24.)

[Verries 2010] Jean Verries. *Approche pour la conception de systèmes aéronautiques innovants en vue d'optimiser l'architecture. Application au système portes passager*. PhD thesis, Université Paul Sabatier-Toulouse III, 2010. (Cité en page 12.)

[Vienne 2014] Marc Vienne. *Methode spirale*, http ://www.marcvienne.fr/methode-spirale/3592516 2014. (Cité en pages vii et 15.)

[Von Bertalanffy 1956] Ludwig Von Bertalanffy. *General system theory*. General systems, vol. 1, no. 1, pages 11–17, 1956. (Cité en pages 8, 10 et 43.)

[Wainer 2001] Gabriel Wainer et Norbert Giambiasi. *Timed Cell-DEVS : modeling and simulation of cell spaces.* In Discrete event modeling and simulation technologies, pages 187–214. Springer, 2001. (Cité en page 90.)

[Wainer 2010] Gabriel A Wainer, Khaldoon Al-Zoubi, Saurabh Mittal, JL Risco Martín, Hessam Sarjoughian et Bernard P Zeigler. *An Introduction to DEVS standardization.* Discrete-Event Modeling and Simulation : Theory and Applications. London : Taylor and Francis, 2010. (Cité en page 127.)

[Washizaki 2003] Hironori Washizaki, Hirokazu Yamamoto et Yoshiaki Fukazawa. *A metrics suite for measuring reusability of software components.* In Software Metrics Symposium, 2003. Proceedings. Ninth International, pages 211–223. IEEE, 2003. (Cité en page 70.)

[Weld 1992] Daniel S. Weld. *Reasoning about model accuracy.* Artif. Intell., vol. 56, no. 2-3, pages 255–300, 1992. (Cité en page 48.)

[Weng 1999] Gezhi Weng, Upinder S Bhalla et Ravi Iyengar. *Complexity in biological signaling systems.* Science, vol. 284, pages 92–96, 1999. (Cité en page 10.)

[Winskel 1993] Glynn Winskel. The formal semantics of programming languages : an introduction. The MIT Press, 1993. (Cité en page 115.)

[Wu 2008] Ji Wu, Chun Wang, Xiao-xia Jia et Chao Liu. *Mining Open Source Component Behavior for Reuse Evaluation.* In High Confidence Software Reuse in Large Systems, pages 112–115. Springer, 2008. (Cité en page 70.)

[Yonglin 2009] Lei Yonglin, Wang Weiping, Li Qun et Zhu Yifan. *A transformation model from DEVS to SMP2 based on MDA*. Simulation Modelling Practice and Theory, vol. 17, no. 10, pages 1690–1709, 2009. (Cité en page 126.)

[Zeigler 1976] BP Zeigler. *Theory of Modelling and Simulation.*, 1976. (Cité en pages vii, 45, 46, 47, 55, 64, 76, 89, 107, 114, 127, 138 et 171.)

[Zeigler 2000] Bernard P Zeigler, Herbert Praehofer et Tag Gon Kim. Theory of modeling and simulation : integrating discrete event and continuous complex dynamic systems. Academic press, 2000. (Cité en pages 2, 43, 45, 48, 90, 110 et 127.)

[Zuberek 1991] WM Zuberek. *Timed Petri nets definitions, properties, and applications.* Microelectronics Reliability, vol. 31, no. 4, pages 627–644, 1991. (Cité en page 102.)

Auteur : Damien FOURES
Titre : Validation des modèles de simulation
Directeurs de thèse : Vincent ALBERT et Alexandre NKETSA

Résumé : Ce travail de thèse s'est intéressé à la validité des modèles de simulation dans le cadre du développement des systèmes complexes et critiques. Une analyse de l'approche d'ingénierie système, et plus particulièrement de l'aspect modélisation et simulation, a permis de constater qu'il est impossible d'établir de façon directe la validité du modèle de simulation. De nombreux points sont à l'origine de cette impossibilité, comme une mauvaise formulation des objectifs de simulation, une incohérence implémentatoire, les limites du moteur de simulation, etc. La validité d'un modèle de simulation étant définie pour un objectif de simulation, il est apparu important de proposer une approche globale de la M&S, associant un ensemble d'outils capables de détecter des incohérences entre les objectifs de simulation et les modèles du système d'intérêt. Ces outils, à destination de l'utilisateur de la simulation, permettent l'amélioration du niveau de confiance dans le modèle de simulation et donc dans les résultats de simulation. Notre étude se base sur la théorie de la M&S telle que proposée par B.P. Zeigler. En considérant le concept de cadre expérimental qui y est introduit, nous avons pu proposer un cadre méthodologique capable d'exprimer les objectifs de simulation de manière claire. Ce cadre méthodologique nous permet d'étudier les problématiques d'application et d'accommodation de la M& S que nous regroupons sous la problématique de compatibilité.

Ainsi, notre premier objectif a été de proposer une approche capable de mesurer l'incohérence entre les objectifs de simulation et le modèle du système. En s'appuyant sur les méthodes formelles et la théorie des automates, nous avons établi un ensemble de métriques capables de mesurer le degré de compatibilité dynamique entre cadre expérimental et modèle du système d'intérêt. Pour cela, nous étudions en premier lieu la compatibilité dynamique entre automates à interface en utilisant la décomposition en arbre. Montrant les limites d'une telle approche, nous sommes passé à l'étude de la compatibilité entre modèles DEVS en utilisant la génération de graphes de classe, autrement appelés graphes d'atteignabilité. Cette étude formelle de la compatibilité nous permet de proposer un ensemble de bonnes propriétés de la simulation.

Nous proposons finalement une méthodologie qui permet de guider l'utilisateur de la simulation dans l'élaboration de métriques permettant de mesurer ce niveau de compatibilité. S'appuyant sur les concepts de l'ingénierie dirigée par les modèles, nous proposons un langage dédié à la simulation permettant de guider l'utilisateur de la simulation dans l'évaluation de la validité des modèles de simulation.
Mots clés : Modélisation, Simulation, Validité, Cadre expérimental, DEVS
Discipline administrative : Systèmes Informatiques

Validation of simulation models

Abstract : This work is focused on the validity of simulation models during development of complex and critical systems. The analysis of the system engineering approach and, especially the modeling and simulation aspect, showed that it was impossible to directly determine simulation models validity. Many aspects can cause this unattainability, such as bad formulation of simulation objectives, implementation inconsistency, limits of the simulation engine, etc. The validity of a simulation model being defined for a specific simulation goal, it seemed important to provide a global M&S approach, combining a set of tools to detect inconsistencies between objectives and models of the system under test. These tools, dedicated to the simulation user, allow to improve confidence level of the simulation model and thus in simulation results. Our study is based on the M& S theory as proposed by B.P. Zeigler. Using the concept of experimental frame, we are able to propose a methodological framework to express simulation objectives clearly. This allows us to study applicability and accommodation, witch we grouped under compatibility issue.

Thus, our first objective was to propose an approach able to measure inconsistencies between experimental frame and model of the system. Based on formal methods and automata theory, we propose a set of metrics that measure the degree of dynamic compatibility between experimental frame and model system of interest. For this, we firstly study the dynamic compatibility between interface automata using tree decomposition. Showing limits of this approach, we studied compatibility between DEVS models using reachability graphs analysis. This formal study of the compatibility help us to propose a set of good properties of the simulation.

Finally, we propose a methodology to guide the simulation user in metrics development to measure the compatibility level. Based on model-driven engineering approach, we propose a simulation dedicated language, to help users to asses the validity of simulation models.

Keywords : DEVS, Validation, Simulation, Formal Matching, Model, Experimental frame