Un cadre formel pour l'intégration de connaissances du domaine dans la conception des systèmes : application au formalisme Event-B

Souad Kherroubi

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Un cadre formel pour l'intégration de connaissances du domaine dans la conception des systèmes : Application au formalisme Event-B

THÈSE

présentée et soutenue publiquement le 21 Décembre 2018

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Souad KHERROUBI

Composition du jury

Rapporteurs :	Pr. Ladjel BELLATRECHE	LIAS, ENSMA
	Dr (HDR). Brahim HAMID	IRIT, Université Jean Jaurès
Examinateurs :	Dr(MdC). Marie DUFLOT-KREMER	LORIA, Université de Lorraine
	Pr. Yamine AÏT AMEUR	IRIT, ENSEIHT
	Pr. Régine LALEAU	Université Paris-Est Créteil
Directeur de Thèse :	Pr. Dominique MÉRY	LORIA, Université de Lorraine
Invité :	Pr. Laurent VIGNERON	LORIA, Université de Lorraine

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Mis en page avec la classe thesul.

Remerciements

Je tiens à remercier mon directeur de thèse M. Dominique MÉRY pour le sujet si passionnant qu'il m'a proposé, pour m'avoir laissé libre choix de mes orientations ce qui me donne toujours confiance en moi. Je le remercie pour le temps qu'il m'a accordé durant ces années de travail et ce malgré son emploi du temps si chargé, pour toutes les corrections qu'il a apportées pour ma thèse, pour m'avoir appris la rigueur dans les définitions et surtout pour sa patience durant les moments difficiles.

Je tiens à remercier l'équipe qui m'a acueillie durant ces années de travail. Merci à :

— Stéphan, pour sa gentillesse, son aide et ses orientations durant mes débuts de thèse ;
— Pascal, pour toutes les discussions que j'ai eu la chance d'avoir avec lui et qui m'ont été d'une grande source d'inspiration et d'encouragement pour mes travaux de thèse, pour sa modestie et son soutien durant la rédaction de ma thèse ;
— Martin, pour toutes les discussions que j'ai eu avec lui autour des logiques et des prouveurs, mais aussi pour sa sympathie ;
— Rémi, pour toutes les pauses passées avec lui durant lesquelles j'ai appris la philosophie et le sens de l'analyse des choses, et autour desquelles j'ai eu le plaisir de débattre certains points de vue pour finalement affirmer mes choix et mes orientations. Je le remercie pour tout le temps qu'il m'a consacré à m'écouter et m'encourager ;
— Marie, pour la personne qu'elle est, pour son soutien durant les moments de détresse, mais aussi pour avoir pris le temps de m'écouter pour les répétitions de ma soutenance. Je la remercie pour les agréables moments passés avec elle ;
— Bruno, pour tout le temps et les discussions très utiles pour mes travaux, mais aussi pour ses encouragements, son écoute et sa modestie durant les débuts de ma thèse. Je le remercie d'avoir répondu à toutes mes nombreuses questions même après son départ ;
— Denis, pour tout le temps conséquent qu'il m'a accordé à m'écouter, pour toute son aide autour de LaTeX, mais aussi pour tous les soirs durant lesquels on partageait des moments à rigoler pour me faire oublier le stress du travail, pour tous les moments de détente et les séances de tennis qu'on a joué ensemble ;

Je remercie M. Ladjel BELLATRECHE et M. Brahim HAMID d'avoir accepté de rapporter ma thèse. Je les remercie d'avoir pris soin d'analyser mon travail et pour toutes les remarques faites sur le manuscrit. Un merci particulier à M. Ladjel BELLATRECHE pour ses conseils et sa bonne humeur durant les conférences MEDI à Barcelone. Merci aux examinateurs : Mme. Régine LALEAU, M. Yamine AÏT AMEU et Mme. Marie DUFLOT-KREMER d'avoir accepté de faire partie de mon jury de thèse.

Un merci particulier à M. Laurent VOISIN pour avoir pris le temps pour répondre soigneusement à mes questions autour de Rodin et les prouveurs automatiques. Je le remercie pour toutes ses explications et ses justifications qui m'ont été d'une grande aide pour mes travaux.

Je tiens à remercier M. Laurent VIGNERON, et M. Pierrick GAUDRY et Mme Chantal CHRETIEN pour le soutien et l'aide psychologiques qu'ils m'ont apportés durant les derniers mois, la dernière ligne droite. Je les remercie pour leur écoute et leurs encouragements. Merci du fond du cœur.

Je n'oublie pas de remercier mes nouveaux collègues dans mon nouveau bureau, Irène et Sergueï, pour leur sympathie qui rend toujours agréable le travail et détend l'atmosphère. Je remercie également Nazim pour toutes les discussions autour de l'intelligence artificielle.

Merci à Kamel pour sa sympathie et pour toutes les discussions qu'on a eu, pour son soutien et les agréables moments passés ensemble.

Merci à Mme. Virginie BLAVIER pour m'avoir fait venir tous les bouquins et tous les documents demandés en un temps record. Je n'oublie pas de remercier Mme. Véronique CORTIER et M. Yannick TOUSSAINT d'avoir répondu à mes questions.

Merci à la recherche, à tous les auteurs cités dans ce manuscrit, et à bien d'autres que je n'ai pas eu le temps de citer et qui, par leurs raisonnements, justifications et explications, m'ont permis d'accomplir ce modeste travail. Merci à tous mes enseignants depuis le primaire qui m'ont fait aimer les mathématiques et le fondamental.

Merci à mes deux meilleurs et fidèles amis, Hmimi et Yasmine. Merci à Hmimi pour toute son aide, et tous ses encouragements durant les années universitaires passées à Tizi-Ouzou. Je le remercie aussi pour son aide et ses conseils durant mes débuts en France, mais aussi d'avoir toujours répondu présent à mes sollicitations. Je remercie celle avec qui je partage mon chemin depuis l'université, Yasmine, ma meilleure amie. Je la remercie pour toutes ces années passées ensemble, pour tous les trajets et les moments qu'on a partagés à 4h du matin pour aller travailler sur la capitale. Ces moments resteront gravés en ma mémoire. Je lui suis très reconnaissante pour son soutien sans faille de l'autre côté de la Méditerranée durant mes débuts en France.

Je remercie ma famille, mon frère, mes sœurs, et mon fiancé pour leur soutien, mais aussi leur compréhension pour les choix qui sont les miens.

Enfin, je tiens à exprimer ma reconnaissance à ceux sans qui je ne serais pas arrivée à ce stade, mes très chers parents, ma très chère mère et mon très père, ceux à qui je dois tout. Une pensée particulière à mon très cher père à qui je dis, je t'aime.

À ma source d'inspiration
À celui à qui je ressemble
À mon très cher père
À ta mémoire
et tout ce que tu représentes pour moi
J'espère que tu te reposes en paix là où tu es

« Premier principe :
ne jamais se laisser abattre par des personnes ou par des événements. »
- Marie Curie

« Dans la vie, rien n'est à craindre,
tout est à comprendre. »
- Marie Curie

« Ce n'est qu'en essayant continuellement que l'on finit par réussir.
Autrement dit : plus ça rate, plus on a de chances que ça marche »
- Jacques Rouxel

« L'erreur est humaine, persévérer est diabolique. »
- Proverbe italien

« Ne laissez jamais les perturbations du moment limiter la portée de vos perspectives d'avenir. »
- Mouctar Keïta

Table des matières

Table des figures **xi**

Introduction générale

Chapitre 1
Ingénierie des connaissances et génie logiciel

1.1 Introduction . . . 12
1.2 Notions de base . . . 14
1.3 Ingénierie des connaissances . . . 15
1.3.1 Ontologies . . . 16
1.3.2 Quelles propriétés ? . . . 17
1.3.3 Conceptualisation, peuplement et granularité des ontologies . . . 18
1.4 Ingénierie système et génie logiciel . . . 20
1.4.1 Modélisation . . . 21
1.4.2 Différents formalismes pour la spécification des systèmes . . . 22
1.4.3 Quelles propriétés ? . . . 24
1.4.4 Vérification et validation à base de modèles . . . 24
1.5 Formalismes logiques . . . 26
1.5.1 Logiques avec contraintes du domaine pour la représentation des connaissances . . . 26
1.5.2 Logiques pour la spécification en vérification et validation . . . 32
1.5.3 Synthèse . . . 35
1.6 Ontologies pour la conception de systèmes . . . 37
1.7 Vérification et validation pour les ontologies . . . 40
1.8 Synthèse . . . 40
1.8.1 Lien entre domaine, conception ou modélisation d'un système . . . 41
1.9 Contextualisation et dépendance . . . 44
1.9.1 Pourquoi le contexte ? . . . 44
1.9.2 Définition du contexte . . . 45
1.9.3 Contexte et logique . . . 47

1.9.4 Le contexte comme théorie : une sémantique basée sur les situations 49
1.10 Synthèse 51

Chapitre 2
Le formalisme Event-B

2.1 Introduction 56
2.2 Représentation et modélisation des connaissances en Event-B 57
2.2.1 Description d'un contexte Event-B 58
2.2.2 Description d'une machine Event-B 59
2.2.3 Raffinement de modèles Event-B 62
2.2.4 Sémantique implicite des modèles Event-B : Obligations de preuves 63
2.3 Les différentes approches de structuration des modèles Event-B 65
2.3.1 Composition/décomposition de modèles 66
2.3.2 Patrons de conception 67
2.4 Extension Temporelle pour le formalisme Event-B 68
2.4.1 Extension TLA pour Event-B 68
2.5 Outils Event-B utilisés 73
2.6 Synthèse 73

Chapitre 3
Contextualisation et intégration de contraintes du domaine en Event-B

3.1 Introduction 76
3.2 Modélisation du contexte dans Event-B 78
3.3 Des modèles Event-B dépendants 81
3.3.1 Étude de cas : système de gestion ERP 86
3.3.2 Modèle Event-B équivalent aux modèles dépendants 91
3.4 Stratégies d'intégration de contraintes du domaine pour la conception de systèmes 93
3.4.1 Exemple de motivation 93
3.4.2 Logique du premier ordre avec les contraintes du domaine pour Event-B : BL-DC 95
3.4.3 Contextualisation versus. raffinement 99
3.4.4 Application à l'étude de cas 102
3.4.5 Analyse et discussions 114
3.5 Travaux en annexe 115
3.6 Synthèse 117

Chapitre 4
Patrons de conception par dépendance-Application aux systèmes de vote

4.1 Introduction . . . 122
4.2 Conception des systèmes de vote . . . 124
4.2.1 Caractéristiques des systèmes de vote . . . 125
4.2.2 Exigences et propriétés d'un système de vote électronique . . . 127
4.2.3 Analyse des systèmes et des protocoles de vote électronique . . . 130
4.3 Patrons de développement pour les systèmes de vote . . . 131
4.3.1 Phase du vote . . . 134
4.4 Relation de dépendance entre les phases du vote . . . 159
4.4.1 Contexte dépendant pour la phase de dépouillement . . . 160
4.4.2 Phase de dépouillement . . . 166
4.5 Extensions possibles des patrons des protocoles de vote . . . 173
4.5.1 Composition des modèles des protocoles de vote et le modèle de l'attaquant Dolev-Yao . . . 177
4.6 Exploitation des connaissances du domaine pour concevoir des systèmes de vote . . . 179
4.6.1 Extraction pour la vérification . . . 179
4.6.2 Extraction pour la validation . . . 182
4.7 Conclusion . . . 183

Chapitre 5
Conclusion et perspectives

5.1 Bilan des travaux réalisés . . . 185
5.2 Perspectives . . . 187
5.2.1 Sur le plan des dépendances . . . 187
5.2.2 Sur le plan de l'extraction par intégration . . . 188

Annexes

Annexe A
Les logiques de description et la logique du premier ordre

A.1 Relations des logiques de description avec la logique du premier ordre . . . 195
A.1.1 Logique de description et logique du premier ordre FOL . . . 195

Annexe B
Correspondances entre la logique de description et le formalisme Event-B

Bibliographie **201**

Table des figures

1.1 Exemple de T-Box . . . 31
1.2 Exemple de A-Box . . . 31
1.3 Bilan comparatif entre ontologies et modélisation pour la V&V . . . 42
1.4 Définition d'un système selon l'hypothèse sémiotique . . . 43

2.1 Les notations ensemblistes du formalisme Event-B . . . 57
2.2 Structure d'un modèle Event-B . . . 58
2.3 Les gardes et les prédicats avant-après . . . 61
2.4 Les obligations de preuve de la chaîne de raffinement de modèles Event-B . 64
2.5 Utilisation des patrons de conception dans Event-B [149] . . . 67
2.6 La méthodologie du paradigme service-as-event . . . 70

3.1 Dépendance entre modèle Event-B . . . 83
3.2 Modèles Event-B décomposés par dépendance . . . 84
3.3 Système de gestion ERP . . . 91
3.4 Modèles Event-B recomposés par dépendance . . . 92
3.5 Structure des services de réservation (avions et voitures) . . . 95

4.1 Structure générale de développement des patrons pour les systèmes de vote 132
4.2 Stratégies de raffinement pour la phase du vote . . . 160
4.3 Composition des patrons de vote avec le modèle de l'attaquant . . . 178

5.1 Structure ACL pour Event-B et les ontologies . . . 193

B.1 Récapitulatif des transformations entre les logiques de description (Plages de données, propriétés, individus et valeurs de données) et Event-B . . . 197
B.2 Récapitulatif des transformations entre les logiques de description (Axiomes et Faits) et Event-B . . . 198
B.3 Récapitulatif des transformations entre Event-B et logique de description . 199

Résumé

Les travaux présentés dans cette thèse relèvent de l'utilisation des techniques de l'ingénierie des connaissances issues de l'intelligence artificielle, dans la modélisation pour la vérification et la validation des systèmes. Cette thèse vise à définir des techniques, et des méthodes pour mieux exploiter les connaissances issues du domaine dans l'objectif de rendre compte de la réalité de systèmes qualifiés de complexes et critiques. La conception d'un système suit une approche constructiviste qui prescrit des solutions à des problèmes. La modélisation est une étape indispensable pour effectuer des vérifications et exprimer des propriétés qu'un système doit satisfaire selon des besoins établis dans les cahiers de charge. La modélisation est une représentation, certes simplificatrice, mais réductionniste de la réalité d'un système. Or, un système complexe ne peut se réduire à un modèle. Un modèle qui représente un système doit toujours s'intégrer dans sa théorie observationnelle pour rendre compte des anomalies qu'il peut y contenir. Notre étude montre clairement que le contexte est la première problématique à traiter car principale source de conflits dans le processus de conception d'un système. Le raisonnement sur la correction d'un système repose donc sur une relation ternaire entre les besoins, le système et le contexte.

L'approche retenue dans cette thèse est celle d'intégrer des connaissances du domaine en associant le système à concevoir à des formalismes déclaratifs qualifiés de descriptifs qu'on appelle des ontologies de domaine. L'objectif est de définir des stratégies qui peuvent servir pour intégrer des connaissances pour aider à faire des vérifications et des validations à base de modèles. Une attention particulière est portée au formalisme Event-B dont l'approche correct-par-construction appelée raffinement est le principal mécanisme au coeur de ce formalisme, qui permet de faire des preuves sur des représentations abstraites de systèmes pour exprimer et vérifier des propriétés de sûreté et d'invariance. Nos travaux répondent aux problématiques qui suivent : Le premier problème traité est celui de la représentation et la modélisation des connaissances du contexte en vérification et en validation de modèles. Pour trouver des réponses à ces questions, une première étude a concerné les différentes sources de conflits, suite à laquelle nous avons établi de nouvelles définitions et des règles pour une extraction de connaissances du domaine par raffinement pour la vérification et la validation en Event-B. Une étude des formalismes de représentation et d'interprétation logiques du contexte qui se fondent sur les logiques initiées par McCarthy, a permis de définir un nouveau mécanisme pour mieux structurer les modèles en Event-B. Une deuxième étude concerne l'apport que peuvent avoir les connaissances du domaine pour la vérification et la validation des modèles. De l'ensemble de l'étude menée, nous définissons une logique pour le formalisme Event-B avec contraintes du domaine fondées sur les logiques de description, établissons des règles à exploiter pour l'intégration des connaissances du domaine à des fins de vérification et de validation de modèles. L'évaluation des propositions faites dans cette thèse portent sur des études de cas assez complexes telles que les systèmes de vote dont des patrons de conception sont également développés dans cette thèse.

Notre étude soulève des problématiques fondamentales sur la complémentarité que peut avoir l'intégration des connaissances du domaine à des modèles Event-B par raffinement, en particulier, pour l'utilisation des raisonnements ontologiques. Les perspectives de nos

travaux se situent alors à comment il est possible de mécaniser l'extraction des connaissances du domaine (à l'aide d'ontologies) par raffinement sur les deux plans, à savoir la vérification et la validation ? Notre choix s'est porté sur la définition de nouvelles structures pour une extraction partiellement automatisée.

Mots-clés: Méthodes formelles, preuve, modélisation, représentation des connaissances, raisonnement ontologique, formalismes logiques, Event-B, raffinement.

Abstract

The work presented in this thesis is based on the use of artificial intelligence engineering techniques, in modeling for the verification and validation of systems. This thesis aims at defining techniques, and methods to better exploit the knowledge provided from the domain in order to account for the reality of systems described as complex and critical. The design of a system follows a constructivist approach that prescribes solutions to problems. Modeling is an essential step in performing verifications and expressing properties that a system must satisfy according to the needs and requirements established in the specifications. Modeling is a representation that simplifies the reality of a system. However, a complex system can not be reduced to a model. A model that represents a system must always fit into its observational theory to account for any anomalies that it may contain. Our study clearly shows that the context is the first issue to deal with as the main source of conflict in the design process of a system. The reasoning on the correction of a system is therefore based on a ternary relationship between needs, system and context.

The approach adopted in this thesis is that of integrating knowledge of the domain by associating the system to design with declarative formalisms qualified of descriptive ones that we call domain ontologies. The goal is to define strategies that can be used to integrate knowledge to help perform model-based verifications and validations. A particular attention is given to the Event-B formalism, whose correct-by-construction approach called refinement is the main mechanism at the heart of this formalism, which makes it possible to make proofs on abstract representations of systems for expressing and verifying properties of safety and invariance. Our work addresses the following issues : The first problem treated is the representation and modeling of contextual knowledge in verification and validation of models. To find answers to these questions, a first study looked at the different sources of conflict, following which we established new definitions and rules for a refinement domain knowledge extraction for Event-B verification and validation. A study of logical formalisms that represent and interpret the context, which are based on the logic initiated by McCarthy, allowed us to define a new mechanism for better structuring Event-B models. A second study concerns the contribution that domain knowledge can make to the verification and validation of models. From the entire conducted study, we define a logic for the Event-B formalism with domain constraints based on the description logic, we define rules to integrate domain knowledge for model verification and validation. The evaluation of the proposals made in this thesis deal with very complex case studies such as voting systems whose design patterns are also developed in this thesis.

Our study raises fundamental questions about the complementarity that the integration of domain knowledge can bring to Event-B models by refinement, in particular, for the use of ontological reasoning. The perspectives of our work are then situated at how it is possible to mechanize the extraction of domain knowledge (using ontologies) by refinement on both levels, namely the verification and validation ? Our choice focused on the definition of new structures for a partially automated extraction.

Keywords: Formal Methods, Proof, Systems Design, Knowledge Representation, Ontological Reasoning, Logical Formalisms, Event-B, Refinement.

Introduction générale

**« Au travail, le plus difficile c'est d'allumer la petite lampe du cerveau.
Après, ça brule tout seul. »**
- Jules Renard

**« Toute gloire ne s'achète pas au prix du sang et des ruines ;
il en est de paisible, comme celle des lettres, des sciences, des arts,
qui, avec moins d'éclat, sont plus dignes d'une éternelle admiration.
Cette gloire ne s'arrache point,
c'est le fruit mûr d'un long et persévérant travail,
qui, cueilli avec patience, doit se goûter avec modestie. »**
- Louis-Auguste Martin

Problématique et champs de recherche concernés

Cette thèse se situe au carrefour des disciplines de l'intelligence artificielle et du génie logiciel. Les techniques de l'intelligence artificielle utilisées dans cette thèse sont appelées la représentation des connaissances faisant l'usage des ontologies. Ces techniques sont utilisées pour la modélisation, la vérification et la validation des systèmes informatiques qui privilégient les méthodes formelles dans le génie logiciel.

Nos travaux s'insèrent dans le cadre du projet IMPEX (*IMPlicit and Explicit semantics integration in proof based developments of discrete systems*). Ce projet vise à rendre explicite la modélisation des contextes et des environnements associés à des domaines d'application. En analyse des besoins, les deux termes *implicite*, et *explicite* sont utilisés pour distinguer l'expression *déclarative* (descriptive) et *opérationnelle* (prescriptive) des besoins [254]. Le projet vise à traiter formellement les termes *implicite*, et *explicite* dans le processus d'ingénierie du logiciel. La composition et l'interopérabilité de systèmes hétérogènes peuvent être source de problèmes liés à l'absence de sémantique explicite dans le contexte de chaque composant, d'où la difficulté pour les comprendre. Cette difficulté est liée, d'une part, aux exigences croissantes qui s'attachent à la technologie et aux performances ainsi qu'à leur criticité, mais aussi aux connaissances sur les méthodes pour résoudre les problèmes de leur conception. Ceci est principalement dû à l'absence de systèmes qui facilitent l'accès aux connaissances pertinentes à considérer dans le processus de conception. Citons à titre indicatif les systèmes de vote étudiés dans cette thèse, où les principaux facteurs qui rendent complexe et difficile la compréhension des phénomènes de ces systèmes sont liés, d'une part, à la complexité liée aux données manipulées dans

ce domaine à laquelle s'ajoutent la complexité conceptuelle, la complexité ontologique et sémantique, et d'autre part, la complexité technique liée aux primitives de chiffrement devant être intégrées comme exigence pour la conception de ces systèmes.

Par ailleurs, le recours aux méthodes formelles permet le développement de systèmes sûrs avec la possibilité de prouver des propriétés de nature différentes comme les propriétés de sûreté et de sécurité, mais également des propriétés statiques et dynamiques d'un système. L'utilisation de telles méthodes devient de plus en plus justifiée, notamment dans les systèmes où le niveau de criticité est très élevé. Ces méthodes se distinguent par un langage formel, un ensemble de règles qui régissent la manipulation des expressions du langage et un système de preuve. Le langage formel, défini par une syntaxe et une sémantique précises, représente le premier maillon de la chaîne de développement formel.

L'adoption des méthodes formelles constitue une avancée considérable dans la mise en œuvre des techniques de vérification et de validation de systèmes critiques. En particulier, ces techniques possèdent un potentiel de raisonnement rigoureux fondé sur la logique mathématique pour spécifier et vérifier des systèmes informatiques. Leur utilisation augmente la compréhension des systèmes en révélant des incohérences, des ambiguïtés qui, autrement n'auraient pas été détectées au début du cycle de développement. Ces méthodes permettent la démonstration de la satisfaction d'un système donné vis-à-vis de ses exigences décrites dans une spécification. Le processus de vérification consiste alors à vérifier que l'implémentation du système répond aux exigences exprimées dans la spécification. Les systèmes de preuves utilisés à cet effet sont la preuve de théorèmes et la vérification de modèles.

Un modèle est une abstraction de la réalité selon une certaine conceptualisation [143]. Bien que notre connaissance de la façon d'appliquer ces abstractions ne cesse d'augmenter, nous avons toujours de la difficulté à transposer correctement les informations pertinentes d'un contexte réaliste à un contexte dans lequel les analyses peuvent ensuite se poursuivre [41]. Une telle abstraction élimine souvent les informations indispensables pour leur compréhension. Les informations ou les connaissances omises peuvent concerner la sûreté d'un système de manière non évidente. Sans ces connaissances, l'analyse est erronée. C'est d'autant plus vrai dans le cadre des systèmes de vote en Europe qui sont face à deux défis importants dans leur fonctionnement. D'une part, il faut assurer une interopérabilité entre les systèmes hétérogènes de chaque pays européen qui sont développés, en général, indépendamment du système européen. Les propriétés qu'un système de vote d'un pays en Europe doit vérifier ne doivent pas être en contradiction ou en conflit avec les recommandations établies en Europe. D'autre part, l'environnement de ces systèmes est dynamique ce qui entraîne des changements selon les besoins et les lois juridiques et politiques instaurées dans chaque pays européen. Il s'ensuit que chaque pays peut avoir ses propres contraintes en terme de propriétés fonctionnelles. La conception de ces systèmes de vote, comme pour tout système complexe, est fondée sur des compromis et des arbitrages [129].

Un système se réduit donc à une vision *constructiviste* où les détails inutiles et sans intérêt pour le concepteur sont abstraits. La conception suit des idées, des hypothèses selon différentes vues qui constituent chacune un contexte. Or, un système complexe est irréductible à un modèle car il représente des phénomènes complexes [19]. Dans l'idéal, il est indispensable pour la conception de ces systèmes d'avoir une normalisation bien construite des données devant être compréhensibles pour décider de la manière dont les problèmes d'interopérabilité doivent être traités. Par ailleurs, au fur et à mesure que ces systèmes

continuent à se complexifier, la normalisation des données pour prendre en charge ces systèmes devient elle aussi plus complexe. De nos jours, le respect des exigences et la prise en compte des besoins dans ces systèmes nécessitent, non seulement leur évaluation, mais aussi la modification de ces normes. La création de consensus qui permettrait d'automatiser ces mises-à-jour et leur maintien deviendrait presque impossible. C'est d'autant plus vrai pour la tâche de vérification et de validation de ces systèmes.

L'avènement des ontologies a contribué de façon significative à résoudre ces conflits sémantiques, puisqu'elles fournissent une représentation explicite de la signification des concepts du monde réel. Leur capacité à raisonner permet la dérivation de faits moyennant des langages tels que les logiques de description [24]. Ces facultés de raisonnement permettent la détection des incohérences et des ambiguïtés sémantiques. Ces formalismes décrivent donc une *réalité* comme étant quelque chose.

Contributions et apports de la thèse

Si l'on considère que les perspectives sont celles d'établir des cadres formels de la justification des conceptions des systèmes complexes, cette thèse présente une étude des approches de conception de systèmes en association avec les approches fondées sur la justification et basées sur les connaissances sémantiques d'un domaine. L'objectif est d'établir des approches de base ayant suffisamment de connaissances pertinentes pour améliorer l'explication des solutions de conception. Le cadre proposé dans cette thèse constitue une base pour associer les formalismes orientés connaissances qu'on appelle des ontologies de domaine, et les formalismes de conception formels fondés sur la logique. Nous nous intéressons plus particulièrement au formalisme Event-B qui est basé sur la logique des prédicats et la théorie des ensembles. Ce formalisme est basé sur une approche de modélisation *correcte par construction* qu'on appelle le *raffinement*. Il s'agit de montrer comment les connaissances dans la conceptualisation d'un domaine peuvent être exploitées pour montrer leur efficacité en terme de facilité d'utilisation par *intégration* dans la conception d'un système.

Les approches de conception sont toujours orientées vers des objectifs définis par les exigences à remplir par le futur système, et les contraintes liées à leur conception. Pour capturer les justifications en lien avec leur sémantique pour une explication de la manière de conception de ces systèmes, ces derniers doivent entretenir des liens avec l'ensemble de leur théorie observationnelle, qu'on appelle *domaine d'application*. Car lorsque une anomalie surgit ne permettant pas de rendre compte de l'expérience, il est indispensable d'intégrer ces systèmes dans leur ensemble, dans leur structure globale qu'est leur domaine qui définit leur dimension sémantique comme *étant quelque chose*, puisque ces anomalies n'ont de sens que dans leur domaine d'application. Ce principe qu'on appelle *holisme ontologique* est défini par Quine [219]. Le raisonnement sur la correction d'un système repose donc sur une relation ternaire entre les besoins, le système et le contexte [46, 154]. Un système est donc défini par trois dimensions, à savoir, la dimension syntaxique, la dimension sémantique et la dimension pragmatique.

Dans ce cadre, il faut savoir exploiter les connaissances du domaine et du contexte dans le processus de conception, de vérification et de validation des systèmes. Nous cherchons à répondre aux questions suivantes :

Qu'est ce qu'un contexte en vérification et en validation de modèles ? Comment est re-

présentée et comment est modélisée la connaissance du contexte ?
Pour trouver des réponses à ces questions, une première étude a concerné les différentes sources de conflits, suite à laquelle nous avons établi de nouvelles définitions et des règles pour une extraction de connaissances du domaine par raffinement pour la vérification et la validation à base de modèles. Une étude des formalismes de représentation et d'interprétation logiques du contexte nous a permis de définir un nouveau mécanisme pour mieux structurer les modèles en Event-B.

Quels apports pourront avoir les connaissances du domaine pour la vérification et la validation des modèles ?
Il s'agit de définir des règles à exploiter pour la vérification et la validation.

Comment alors peut-on mécaniser cette extraction ?
Il s'agit de définir de nouvelles structures pour une automatisation partielle de cette extraction.

Les contributions scientifiques proposées dans cette thèse incluent :

— la définition d'un nouveau mécanisme de preuve basé sur les logiques qui interprétent le contexte par des situations. Ce mécanisme appelé *dépendance* de modèles Event-B, est défini sous la condition ou la contrainte temporelle de stabilité, définie elle aussi sur les traces d'exécution des modèles définies par une extension temporelle à la manière de TLA [174] pour le formalisme Event-B. Cette contribution est fondée sur le principe de McCarthy défini dans sa logique appelée calcul des situations, et interprétée dans la théorie des situations. Les dépendances sont définies par un paramétrage des contraintes statiques par des situations sous la condition de stabilité ;

— une approche d'association entre modèles formels pour la conception, la vérification et la validation au moyen du formalisme Event-B, et les ontologies de domaine. Cette association est obtenue en identifiant les différents points en commun ou de différence entre les deux paradigmes que sont la représentation des connaissances, plus particulièrement les ontologies de domaine, et le génie logiciel dont une application particulière est dédiée au formalisme Event-B. Cette identification est basée sur les points clés situés à des niveaux de représentations dans chaque paradigme pour trouver des réponses aux problèmes traités dans cette thèse. Nous nous basons sur les architectures de chaque formalisme, la granularité des connaissances ontologiques d'un côté, et le raffinement et les abstractions des modèles Event-B de l'autre côté, pour montrer ensuite comment le raisonnement ontologique peut être exploité pour rendre explicite l'implicite dans la preuve en modélisation au moyen du formalisme Event-B. Le formalisme logique utilisé pour les ontologies est la logique de description. Nous montrons également comment les propriétés ontologiques peuvent être exploitées pour structurer les états dans la partie dynamique des modèles Event-B, à savoir les machines. Les liens entre les exigences et les contraintes de conception nous ont permis de définir des stratégies d'intégration de connaissances issues du domaine par induction et basées sur les ontologies, et par déduction selon le contexte pour la vérification, la validation de systèmes ;

— Les approches définies dans cette thèse sont ensuite appliquées à des cas réel de systèmes. Citons à titre indicatif, les systèmes de vote, où des contraintes réelles dans la conception de ce type de système imposent, non seulement des critères tels que la fiabilité, la sécurité et la disponibilité des services, mais aussi des normes et des impératifs de traçabilité que les spécifications indiquées dans les cahiers de charges doivent respecter. De plus, l'utilisation des technologies dans le processus de vote

démocratique ne devrait pas diminuer les garanties d'une élection et les conditions de leur légitimité qui se traduisent par la non-discrimination et le respect des élections démocratiques et qui sont des contraintes obligatoires dans tout processus de vote.

Organisation du document

Le premier chapitre dresse un état de l'art sur les deux disciplines que sont la représentation des connaissances, où un intérêt particulier est consacré aux ontologies de domaine, et le génie logiciel, et plus précisément, la vérification et la validation à base de modèles formels. Nous étudions les objectifs de chaque paradigme et les différentes propriétés qui sont étudiées et établies par chacun de ces paradigmes. Une attention particulière sera portée aux différents formalismes logiques utilisés dans ces paradigmes. Une justification sera alors donnée pour l'utilisation des ontologies pour la vérification et la validation de systèmes. Une deuxième étude sera consacrée au contexte. Cette étude concerne, d'une part, la prise en compte de cette notion et des différents points de vue dans le cycle de développement logiciel, et d'autre part, les différentes représentations et interprétations qui lui sont données. On s'intéresse plus particulièrement aux formalismes logiques ainsi qu'à leur interprétation du contexte. Nous présentons un bilan comparatif entre ces approches pour en tirer profit des solutions que nous proposons dans cette thèse.

Le deuxième chapitre sera consacré au formalisme de modélisation Event-B, où seront exposées les principales notions utilisées dans ce formalisme, à savoir, la représentation des connaissances, les propriétés exprimées dans ce formalisme, le raffinement, les obligatoires de preuve, les différentes approches pour structurer les développements de modèles ainsi que l'extension temporelle basée sur le formalisme TLA.

Le troisième chapitre présente les contributions théoriques de cette thèse. Nous proposons une nouvelle définition du contexte en preuve, et une classification de cette notion en vérification et validation des modèles en Event-B. Sur cette base, nous définissons ensuite un nouveau mécanisme appelé *dépendance* de preuve fondé sur une interprétation de contraintes statiques qu'on appelle axiomes par des situations représentées par des états en Event-B. Nous montrons ensuite comment le raffinement permet de contextualiser les modèles et ainsi fournir des précisions pour lever certaines ambiguïtés. Nous présentons ensuite de nouvelles stratégies pour exploiter les connaissances du domaine par le biais d'une ontologie pour définir de nouvelles structures dédiées pour des extractions futures. Une extraction sera alors possible pour rendre compte des connaissances du domaine de manière inductive, et en fonction du contexte de manière déductive. Nous étudions les répercussions sur l'ensemble des obligations de preuve dans ce formalisme.

Le quatrième chapitre présente une application de nos contributions théoriques aux systèmes de vote. Nous montrons les principales propriétés et exigences dans un premier temps, ensuite, nous définissons des patrons de conception pour ces systèmes. Nous montrons comment il est possible d'exprimer des propriétés telles que la stabilité, la vérifiabilité, ainsi que d'autres propriétés qui caractérisent ces systèmes. Nous montrons comment étendre nos patrons pour prendre en compte les spécificités de chaque système, les primitives de chiffrement ainsi que d'autres contraintes liées à la certification de ces

systèmes. Nous montrons enfin comment il est possible d'intégrer les connaissances du domaine en appliquant nos stratégies pour avoir des preuves justifiées, et par conséquent, certifier ces modèles.

Le cinquième chapitre présente une synthèse de nos contributions de thèse, ouvre de nouvelles perspectives en continuité de nos travaux. Nous expliquons en particulier comment à partir des stratégies définies dans notre thèse, il serait possible d'appliquer des techniques d'extraction de connaissances pour une intégration partiellement automatisée.

Chapitre 1

Ingénierie des connaissances et génie logiciel

Sommaire

1.1 Introduction 12
1.2 Notions de base 14
1.3 Ingénierie des connaissances 15
1.3.1 Ontologies 16
1.3.2 Quelles propriétés ? 17
1.3.3 Conceptualisation, peuplement et granularité des ontologies . . 18
1.4 Ingénierie système et génie logiciel 20
1.4.1 Modélisation 21
1.4.2 Différents formalismes pour la spécification des systèmes 22
1.4.3 Quelles propriétés ? 24
1.4.4 Vérification et validation à base de modèles 24
1.5 Formalismes logiques 26
1.5.1 Logiques avec contraintes du domaine pour la représentation des connaissances 26
1.5.2 Logiques pour la spécification en vérification et validation . . . 32
1.5.3 Synthèse 35
1.6 Ontologies pour la conception de systèmes 37
1.7 Vérification et validation pour les ontologies 40
1.8 Synthèse 40
1.8.1 Lien entre domaine, conception ou modélisation d'un système . 41
1.9 Contextualisation et dépendance 44
1.9.1 Pourquoi le contexte ? 44
1.9.2 Définition du contexte 45
1.9.3 Contexte et logique 47
1.9.4 Le contexte comme théorie : une sémantique basée sur les situations 49
1.10 Synthèse 51

« C'est avec la logique que nous prouvons
et avec l'intuition que nous trouvons. »
- Henri Poincaré

« Il existe des conceptions vulgaires tout à fait
suffisantes pour la vie pratique ;
elles doivent même être la nourriture des hommes.
Elles ne suffisent cependant pas à l'intelligence. »
- Averroès

1.1 Introduction

Les méthodes formelles sont réputées pour développer des systèmes sûrs, dont la fiabilité et la robustesse sont des caractéristiques indispensables pour la conception de logiciels et ou de systèmes. Ces méthodes permettent de vérifier et de prouver la correction d'un logiciel ou d'un système par rapport à sa spécification. De nombreuses techniques et de nombreux formalismes sont développés dans la littérature selon le type du système à concevoir et les besoins exprimés. Cette diversité des techniques et des systèmes a rendu complexe et difficile la tâche de conception de système. Par définition, un système [19] *est une représentation d'un phénomène perçu complexe qui est donc irréductible à un modèle pouvant déterminer la prévision certaine de ses comportements*. On parle alors de systèmes complexes, citons à titre indicatif les systèmes de vote étudiés dans cette thèse. Les systèmes complexes sont des systèmes dont le comportement, la structure et les fonctions sont difficiles à comprendre. Cette difficulté est souvent due d'une part, aux formalismes qu'utilisent les méthodes formelles durant les phases de développement d'un système, et d'autre part, au caractère complexe de ces systèmes à concevoir.

Le caractère irréductible d'un système à un modèle se justifie par les hypothèses des abstractions qui sont certes simplificatrices, mais réductionnistes de la réalité. Elles ne tiennent compte des connaissances sur le système qu'en partie. Cette partialité revient à masquer des détails sans intérêt relatifs aux phénomènes étudiés. Avoir une vision complète d'un système nécessite donc d'avoir des vues plus ou moins détaillées selon les acteurs impliqués dans le système. Comprendre le système revient à raisonner et à bien chercher les faits, les idées et les hypothèses de chaque vue définie pour d'une part, mieux communiquer entre acteurs et d'autre part, pour avoir une vision complète du système. Ces vues définissent le contexte de conception qui inclut les intentions des concepteurs définies par les détails du processus de modélisation.

Par ailleurs, pour concevoir un système, il faut concevoir une *machine* en la décrivant. *Un système est donc une machine introduite dans le monde pour être affectée dans ce même monde [154].* Les parties du monde qui affectent cette machine et qui seront influencées par cette machine sont appelées *domaine d'application*. Ce que fait le système doit être recherché dans le domaine d'application puisque le problème se situe dans le domaine d'application, alors que le système est la solution. L'ensemble des connaissances axiomatiques dont dispose ce domaine définit donc l'ingénierie système supposées toujours vérifiées par le modèle. Prêter attention au domaine de l'application signifie qu'il faut écrire rigoureusement une description explicite et précise. Les besoins des utilisateurs

exprimés sous forme de propriétés formulées des concepts issus du domaine enrichissent les connaissances axiomatiques acquises du domaine d'application. Une spécification à travers des modèles décrit le système (As-is) ou souhaité (To be). Les modèles décrivent la structure, les comportements ainsi que les fonctions tout en vérifiant les besoins conformes aux connaissances axiomatiques. Ces connaissances définissent la conceptualisation du domaine d'application.

Habituellement, les approches et les formalismes de conception sont associés pour se compléter ou pour réduire les écarts qui peuvent exister entre les utilisateurs et la communauté des développeurs. Citons par exemple, l'association des formalismes graphiques tels que l'UML à des formalismes tels que les réseaux de Petri pour pouvoir exploiter les aspects graphiques simplistes des diagrammes que présente le formalisme UML dans les formalismes des réseaux de Petri [56]. Des approches systématiques doivent être développées afin d'exploiter les connaissances issues du domaine dans l'objectif d'améliorer la qualité des produits à concevoir. Dans cette thèse, nous nous intéressons à de nouvelles associations qui peuvent permettre l'utilisation de formalismes issus de l'intelligence artificielle et basés sur la représentation des connaissances. Il s'agit des ontologies de domaine. Ces formalismes connus par leur caractère formel, consensuel, sont dotés d'un raisonnement systématique pour raisonner sur des connaissances et en déduire de nouvelles afin d'expliciter les connaissances implicites, source principale d'ambiguïtés. La question qui peut alors se poser est comment exploiter ces connaissances et comment la conceptualisation d'un domaine peut aider à concevoir un système ou un modèle ?

L'objectif de ce chapitre est de fournir une étude précise afin d'identifier chacun des paradigmes utilisés pour la conceptualisation et la conception des systèmes. Nous montrons les avantages de décrire le domaine et d'expliciter les connaissances issues de ce dernier comme une étape préalable afin d'aider à développer des systèmes sûrs. Pour cela nous allons mener une analyse des paradigmes que sont l'ingénierie des connaissances et le génie logiciel. Nous nous intéressons en particulier aux ontologies de domaine pour ce qui concerne le premier paradigme, et aux techniques de vérification et de validation pour ce qui concerne le génie logiciel. Nous comparons ici ces deux paradigmes. La comparaison sera axée sur les points suivants : l'objectif de chaque paradigme i.e., description vs. prescription, la représentation des connaissances, la modélisation dans les deux paradigmes, les propriétés vérifiées, et l'accent sera mis sur les logiques utilisées dans chaque paradigme. Nous étudions les conséquences des ontologies sur l'activité de vérification et de validation. Nous nous intéressons ensuite à la notion de contexte, où une étude plus approfondie des travaux réalisés dans la littérature est effctuée. Cette étude sera aussi centrée sur les formalismes logiques utilisés pour représenter et modéliser le contexte, mais aussi les différentes interprétations de ces dernières. Nous montrons l'intérêt de cette démarche pour en tirer les justifications et les raisons pour lesquelles nous proposons les solutions données dans nos contributions.

La section 1.2 introduit les notions de base qui reviendront de manière récurrente tout au long de cette thèse. La section 1.3 introduit la première discipline qu'est l'ingénierie des connaissances dans laquelle nous nous orientons vers les ontologies, leur propriétés, leurs préoccupations et leurs vérifications. La section qui suit 1.4 aborde le génie logiciel, ses techniques utilisées pour la conception de systèmes, les propriétés visées, ainsi que la vérification et la validation dans la conception de systèmes. La section 1.5 discutent les différentes logiques utilisées dans les deux disciplines. La section 1.6 aborde les travaux qui se basent sur les ontologies pour la conception des systèmes. La section 1.7 aborde les travaux en vérification et validation qui peuvent être utilisés pour les ontologies et plus gé-

néralement dans la représentation des connaissances. La section 1.8 présente une synthèse des deux sections précédentes. La section 1.9 discutent des travaux sur le contexte réalisés dans la littérature. Enfin, la section 1.10 présente une synthèse des lectures exposées dans le présent chapitre.

1.2 Notions de base

Avant de rentrer dans le vif du sujet, et pour mieux comprendre la terminologie utilisée dans cette thèse, nous donnons ici quelques définitions relatives aux principales notions utilisées dans ce mémoire. Les notions qui suivent sont définies de différentes manières et dans plusieurs thèmes : Par exemple, les termes : connaissance, information, donnée. Le terme connaissance, par exemple, a été défini en philosophie et en informatique. Les définitions des trois termes sont tirées de [163, 232].

Définition 1.2.1 (donnée) *Les données sont le résultat d'observations [163].*

Définition 1.2.2 (information) *Les informations sont le résultat d'interprétations de ces données [163].*

Définition 1.2.3 (connaissance) *Une connaissance est un ensemble de données et d'informations mises en œuvre pour assumer une tâche ou produire une nouvelle information [232]. La connaissance est souvent associée à deux aspects : la* ***finalité*** *car la connaissance est mise en œuvre pour atteindre un objectif, et sa* ***capacité générative*** *puisque la connaissance permet de produire de nouvelles informations.*

La définition de connaissance met en exergue deux aspects : *1)* le premier est lié à la **finalité** de la connaissance pour atteindre un objectif précis ; *2)* le second est relatif à la **capacité de raisonner** pour produire de nouvelles informations, de nouvelles propriétés, de nouveaux faits. La connaissance regroupe des faits, des propositions, des suppositions, des croyances, des hypothèses, des idées, des savoirs, ainsi que des techniques, des heuristiques, etc.

La connaissance est donc définie dans l'objectif de raisonner et d'inférer de nouvelles connaissances. Il faut distinguer *raisonnement* et *inférence*. Le raisonnement est inclus dans l'inférence. L'inférence est l'élément de base de toute description, et est guidée par la connaissance. Les connaissances sont de différentes natures, on peut citer les connaissances de définition, les connaissances évolutives, les connaissances typiques, etc.

En clair et plus formellement [232, 261] :
— Données = signes + syntaxe ;
— Information = données + sens (sémantique) ;
— Connaissance = information (syntaxe et sémantique)+ une capacité d'utiliser l'information.

Puisque, suite aux ambiguïtés, une donnée ne peut être comprise ni par les machines, ni par les humains, l'information lui associe un sens ou une sémantique. La connaissance vient donner la capacité à l'information d'être utilisée pour effectuer des raisonnements.

Définition 1.2.4 (domaine) *Un domaine est un univers de discours, petit ou grand, une structure (i) d'entités, c'est-à-dire de « choses », d'individus, dont certains sont désignés comme des composants d'état ; (ii) des fonctions, sur des entités, qui, lorsqu'elles sont appliquées, deviennent éventuellement des actions du domaine qui changent d'état ;*

(iii) des événements, impliquant éventuellement des entités, se produisant dans le temps et exprimables sous forme de prédicats sur un ou plusieurs états (avant/après) ; et (iv) des comportements, des ensembles de séquences d'actions et d'événements éventuellement interdépendants [42].

Il existe une définition plus générale, mais qui explique le lien avec un système [233] :

Définition 1.2.5 (domaine) *Un domaine est un monde reél, hypothétique, ou abstrait habité par un ensemble distinct d'objets qui se comportent selon des règles et des politiques caractéristiques ou propres à ce domaine.*

Il existe différents types de domaine [233] : des domaines d'application, des domaines de services, des domaines architecturaux et des domaines d'implémentation. Les domaines d'application sont ceux auxquels nous nous intéressons, puisque par définition, un *domaine d'application* est l'objet du système du point de vue de l'utilisateur final du système. C'est un matériau auquel on pense normalement dans le contexte de l'analyse des exigences : de quoi l'utilisateur a besoin pour faire ce système ?

Dans cette thèse, nous ferons référence à l'une ou l'autre de ces deux définitions.

Définition 1.2.6 (prédicat) *Un prédicat est une propriété qui exprime une contrainte vraie qui caractérise les objets du domaine considéré et est exprimée dans le langage en question. C'est une fonction à n arguments sur l'univers du discours, qui retourne une valeur de vérité.*

Définition 1.2.7 (système) *Un système est un ensemble composite de personnels, de matériels et de logiciels organisés pour que leur fonctionnement permette, dans un environnement donné, de remplir les missions pour lesquelles il a été conçu [186].*

En modélisation, la définition de système est souvent liée à celles données en introduction du présent chapitre qui concernent la vision de la théorie des systèmes indiquée dans ses références.

Définition 1.2.8 (concept) *Un concept est une abstraction pertinente d'un fragment du monde réel en fonction d'un domaine d'application. Ce terme fait référence à la représentation d'un objet, d'une notion ou d'une idée dans l'ontologie.*

Un concept se définit selon plusieurs dimensions : le niveau d'abstraction (concret ou abstrait), l'atomicité (élémentaire ou composé) et le niveau de réalité (réel ou fictif) [134]. Un concept définit une unité de connaissance, et son contenu fait référence au triangle sémiotique.

1.3 Ingénierie des connaissances

L'ingénierie des connaissances se retrouve au cœur de demandes fortes au sein de la société et des entreprises en matière de gestion des connaissances, de veilles technologiques, de gestion documentaire et de recherche d'information. Son évolution a débuté la fin des années 1970. Les systèmes de connaissances sont la progéniture industrielle et commerciale la plus importante de la discipline appelée intelligence artificielle. Cette ingénierie traite de connaissances dans la mesure où les systèmes informatiques à concevoir doivent assister les utilisateurs dans des tâches mal définies, d'une complexité élevée faisant appel à des savoir-faire. Les connaissances en ingénierie sont des connaissances fonction du

domaine et des tâches. La connaissance dépend beaucoup du contexte [232]. Cette observation concernant la dépendance contextuelle des connaissances se retrouve, selon une terminologie différente, dans différents champs d'étude de la connaissance [23].

L'origine des difficultés des spécifications d'un système est liée d'une part, à la résolution informatique s'appuyant sur des algorithmes non connus ; d'autre part, pour la réalisation de tâches dans un ensemble de contextes, on ne cherche à en automatiser le traitement qu'en parties. L'expertise humaine dont devait rendre compte la base de connaissances de systèmes experts était la première question à laquelle cette communauté s'est intéressée. Vient ensuite le problème de rendre explicite des savoir-faire, et de produire des formules logiques. Son évolution est orientée vers l'organisation structurée dans des modèles conceptuels de différents types de connaissances nécessaires à la résolution de problèmes. La reconstruction de manière artificielle des représentations de connaissances choisies en fonction d'un objectif opérationnel était réalisée par des modèles. Les modèles formels sont conçus afin de décrire une théorie du domaine, ou dans l'objectif de rendre compte des connaissances du domaine. Ce choix n'a pas été toujours formulé clairement. Des formalismes ont vu le jour pour à la fois représenter, mais aussi pour raisonner sur les connaissances, ce qui a donné des systèmes déductifs. Ces systèmes formels qu'on appelle des *logiques* cherchent à modéliser certaines connaissances et certains comportements.

Le problème de l'ingénierie des connaissances a été formulé comme un problème de modélisation, où la question de mettre en place des modèles adéquats à partir de traces de connaissances et les rôles que jouent ces modèles conceptuels dans la modélisation étaient le centre d'intérêt de nombreux travaux. Alors que les connaissances statiques sont modélisées au moyen d'ontologies, les méthodes de résolution de problèmes spécifient des mécanismes de raisonnement génériques.

1.3.1 Ontologies

Les ontologies ont été utilisées dans de nombreux domaines de recherche tels que le traitement automatique du langage naturel, la fouille de données, l'éducation, la bioinformatique, le commerce électronique, le web sémantique, l'ingénierie des systèmes d'information ... Cette notion reflète le besoin essentiel et récurrent en modèles définissant des connaissances structurées. Le raisonnement ontologique cherche à rendre *explicite* l'*implicite*. L'objectif était de définir des approches d'acquisition des connaissances et un environnement de structuration de ces connaissances avant de les représenter sous forme de règles de production.

Les ontologies sont de différents types. Elle peuvent être existentielles, formelles, linguistiques, ... Elles sont définies de différentes manières selon l'intérêt porté par les concepteurs. Selon Gruber [139], une ontologie en informatique (représentation par une machine) est la spécification explicite d'une *conceptualisation*. Emprunté à la philosophie, ce terme était défini comme une étude systématique de l'*existence*. En philosophie, la question était de savoir quelle est l'essence des choses à travers les changements. Les différents modes de l'être pour établir un système de catégories : substance, qualité, action, relation... était la principale préoccupation pour classifier toute chose qui peut être fondée (affirmée) [18]. En informatique tout ce qui existe est tout ce qui peut être représenté par la machine. Par conséquent, une ontologie est la représentation dans un formalisme *déclaratif* des *connaissances d'un domaine*.

C'est à partir de standardisations de la syntaxe et des primitives de représentation et de raisonnement que l'on parvient à partager et réutiliser la conceptualisation et la représentation formelle des connaissances d'un domaine. Les travaux menés dans le domaine de la représentation des connaissances ont permis de modéliser explicitement les aspects structurels et descriptifs des concepts d'un domaine à travers des modèles qui font *consensus*, conduisant ainsi à une représentation explicite de la sémantique des données, en termes de *classes* et de *propriétés*.

Les concepts et les relations, qui les lient, constituent les connaissances d'un domaine dans les ontologies. À chaque entité de l'univers du discours est associé un symbole du formalisme de représentation, qui reflète les relations entre concepts par des *axiomes* qui portent sur les termes de ce même formalisme. Une ontologie définit une *théorie logique* déclarée et représentée. Les axiomes contraignent l'interprétation des formules logiques, et représentent les connaissances compatibles avec les connaissances que l'on dispose a priori sur le domaine. Les ontologies sont aussi vues comme une base de connaissances définies par une partie axiomatique terminologique qu'on appelle T-Box, et d'une partie assertionnelle appelée A-Box. La subsomption est au cœur des ontologies qui permet de raisonner notée par $\sqsubseteq$, ou *is-a* dans le cas du raisonnement a priori, ou *is-case-of* dans le cas du raisonnement a posteriori. Nous détaillons ces notions au niveau sémantique basée sur les logiques de description en sous-section 1.5.1.1.

Différents formalismes ou *modèles d'ontologies* permettent de définir des ontologies. Ceux-ci se distinguent par : leur mode de raisonnement : raisonnement a priori vs. raisonnement a posteriori ; raisonnement dans un monde fermé vs. monde ouvert ; ainsi que les modèles formels sur lesquels leur sémantique est basée. De nombreux formalismes de représentation pour les ontologies ou *modèles d'ontologies* ont vu le jour. Habituellement, les modèles de connaissances font l'usage de constructeurs issus de la logique, d'où leur qualification ou caractérisation *formelle*. Ces modèles sont centrés sur les capacités d'inférence possibles.

Il existe deux modèles stables utilisés dans le domaine du web sémantique tels que OWL et les domaines techniques tels que PLIB. Le premier cité adopte un raisonnement a priori qui est basé sur les logiques de description [24], et raisonne dans un monde ouvert ; mais le second adopte un raisonnement a posteriori dans un monde fermé et est basé sur les logiques des frames [166]. Il existe un autre modèle ontologique appelé DFT [92, 28]. Celui-ci adopte un raisonnement dans un monde fermé, et la logique utilisée est la logique intuitionniste constructive et le calcul des constructions étendu [181] (que l'auteur a étendu pour prendre en compte les constantes et le sous-typage), et la théorie des situations [100], ainsi que la théorie des types dépendants [82, 212].

1.3.2 Quelles propriétés ?

Les propriétés ontologiques sont de différents types [140, 142, 141, 227]. Elles permettent d'organiser, de structurer et de raisonner sur une ontologie, et conduisent à des types de subsomption différents. Parmi ces propriétés [1] on peut citer : les propriétés d'identité, d'unité, d'essence, de rigidité et de dépendance, les propriétés nécessaires, les propriétés suffisantes, et les propriétés nécessaires et suffisantes (définissantes).

1. Il existe d'autres classifications comme celles de D. Bjørner [45] qui définit des entités endurantes et perdurantes que nous ne détaillons pas dans ce document car l'auteur développe des terminologies basées sur les méréologies (ontologies formelles) qui se préoccupent des parties-tout et tout-partie.

Les propriétés d'*identification* donnent une identité unique aux objets. Les propriétés ou les rôles qui peuvent être définis pour caractériser les objets ou les individus par *essence* sont des propriétés *intrinsèques*. Par contre les propriétés *extrinsèques* sont affectées par les utilisateurs en fonction des besoins par des actions, et nous pouvons classer les propriétés de dépendance dans cette catégorie. Les propriétés de rigidité sont des propriétés essentielles pour toutes les instances du même concept.

Par exemple, le concept vehicle est défini ou existe si les propriétés suivantes sont données : avoir des roues, avoir un moteur, avoir la capacité de se déplacer ... Ces propriétés définissent donc ce qu'est un véhicule. Les objets de cette classe sont *identifiés* par les immatriculations attribuées à chaque engin suite à sa construction. En revanche, des propriétés comme « avoir un propriétaire du véhicule », « position du véhicule », etc, peuvent être ajoutées à des fins d'utilisation par exemple, pour définir les actions dans un système telles que : envoyer des factures au propriétaire, l'assurance des véhicules, et bien d'autres besoins. Le changement de ces propriétés n'entraîne pas de changement de l'objet en soi. Ce sont des propriétés qui peuvent changer à l'initiative des utilisateurs. En conséquence, le domaine peut être augmenté de ces propriétés extrinsèques. Les propriétés intrinsèques, quant à elles, changent suite au changement des objets.

Le raisonnement sur les propriétés nécessaires ne permet pas de classer les individus, mais permet la vérification du résultat du raisonnement par l'inclusion dans la description d'un concept. On peut donner dans l'exemple des vehicle, les camions (truck) qui sont aussi des véhicules, ils héritent donc les propriétés d'un véhicule (voir l'exemple section 1.5.1.4). Ces propriétés sont déduites. Les propriétés suffisantes permettent la classification des individus dans leur classe sémantique (concept). Et nous pouvons citer l'inférence des domaines et des images d'une propriété (rôle) ontologique qui restreignent sa définition. Les propriétés nécessaires et suffisantes sont des propriétés d'équivalence. Nous détaillons un peu ces inférences au niveau du modèle formel de OWL que sont les logiques de description (cf. sous-section 1.5.1.1). Les travaux dans la littérature ont montré que la définition des concepts ontologiques requiert l'explicitation de leurs propriétés définissantes (nécessaires et suffisantes) [227].

Nous montrons que les propriétés d'identification définies intrinsèquement permettent de typer les éléments statiques (définis dans les contextes Event-B), que les propriétés extrinsèques permettent de structurer les états dans les machines Event-B, que les propriétés suffisantes permettent de faire les preuves inductivement selon le domaine, et que les propriétés nécessaires sont déduites en fonction du contexte (cf. chapitre 3).

1.3.3 Conceptualisation, peuplement et granularité des ontologies

Le rôle des ontologies est l'étude de l'existant par le biais d'une représentation formelle d'un ensemble de concepts dans un domaine et les relations entre ces concepts [266]. Leur objectif est de fournir un cadre intégré pour que l'information puisse être bien organisée, largement publiée, largement partagée, facilement récupérée et simplement intégrée. La représentation se situe au niveau de la sémantique, et indépendamment de la structure et de la mise en oeuvre des données. En outre, les ontologies permettent l'intégration de données hétérogènes et l'interopérabilité entre des systèmes disparates [266].

La conceptualisation [240] fait référence à un modèle abstrait d'un phénomène dans le monde en identifiant les concepts pertinents de ce phénomène. Le partage reflète la notion

qu'une ontologie capture la connaissance consensuelle, c'est-à-dire acceptée par un groupe. Les problèmes qu'aborde l'ingénierie des ontologies pour leur modélisation sont liés à leur conceptualisation, leur peuplement, mais aussi aux difficultés liées à leur partage car la conceptualisation d'une ontologie doit être partagée. Le problème du partage requiert l'utilisation des techniques appelées médiation, alignement et fusion (*merging* en anglais).

Le problème de la conceptualisation consiste à trouver une organisation hiérarchique des concepts et des propriétés, mais aussi de définir des domaines et images des propriétés (rôles). La subsomption est le principal mécanisme utilisé pour raisonner sur ces différents éléments. Les raisonnements utilisés sont de deux types : la déduction de nouvelles connaissances et la vérification de la cohérence des connaissances modélisées. Ce raisonnement porte sur deux niveaux : le niveau terminologique, et le niveau assertionnel. Le premier infère de nouveaux axiomes terminologiques et permet de vérifier la cohérence des axiomes ; le second permet de classer les individus (trouver les classes sémantiques des individus), et de vérifier la cohérence des assertions faites sur les individus avec leur axiomes terminologiques (cf. sous-section 1.5.1.1).

La conceptualisation d'ontologies est subjective car elle est dépendante de la tâche et l'objectif visés. Alors que la conceptualisation aborde la question de distinguer les concepts des individus et permet de définir la structure qui concerne la T-Box d'une ontologie, le peuplement quant à lui vise à fournir les instances/individus des concepts ou à alimenter la A-Box. Il existe de nombreux travaux qui discutent ces deux points pour plus de détails consulter [227]. Nous expliquons dans ce qui suit très brièvement les points importants pour ce qui nous concerne. La conceptualisation joue un rôle central dans la conception d'un système par l'identification et la structuration des états comme nous le verrons dans le chapitre 3. Le peuplement permet en revanche de déterminer des configurations valides dans le cas de la validation, et d'étendre les mécanismes de vérification par des déductions qui peuvent être appliquées pour des justifications des preuves en Event-B.

À titre d'exemple (exemple extrait de [227]), *twingo* est un terme qui selon le choix de conception, peut être considéré comme une instance du concept Vehicles, ou du sous-concept Renault-Vehicles. Il peut être considéré comme un sous-concept du concept Renault-Vehicles, qui peut avoir l'individu *tiwngo-AA-114-AA*. Notre vision est de considérer l'individu *tiwngo-AA-114-AA* qui possède l'immatriculation *AA-114-AA*, comme une entité du monde unique qui correspond aux éléments donnés dans les ensembles en Event-B. Cela est dû au fait que les individus dans le formalisme Event-B sont définis essentiellement dans les ensembles porteurs et que chaque entité du monde appartient à ces ensembles pour des fins de typage. Les individus dans notre cas dénotent des entités du monde réel et doivent donc avoir la propriété d'unicité de sorte à pouvoir les classer dans des ensembles porteurs en Event-B. Les travaux des auteurs [210] distinguent entre concepts de haut niveau et les concepts de niveau intermédiaire. Les concepts de niveau intermédiaire sont identifiés dans les corpus à partir de leur forme de surface, alors que les concepts de haut niveau sont composés de concepts de niveau intermédiaire. Les concepts de haut niveau [210] dans notre cas définissent les ensembles porteurs qui caractérisent explicitement ou implicitement par leur propriétés (rôles) les individus au moyen de constantes ou de variables (selon que la propriété est statique ou dynamique).

Mais, la classification des concepts et individus d'une ontologie dépend fortement de sa granularité. La granularité définit le niveau de profondeur hiérarchique d'une ontologie. Elle reflète le niveau de la spécialisation des concepts ontologiques. Là encore, il existe des travaux qui discutent ce point selon leurs objectifs (pour plus de détails consulter [227]).

Pour notre part, la construction d'une ontologie doit nous permettre de distinguer chaque niveau de raffinement dans la modélisation en Event-B. En effet, c'est en fonction du niveau de profondeur que chaque niveau de raffinement introduit les éléments, à savoir, les constantes et les variables définies dans les machines Event-B [2]. Notre point de vue rejoint la vision de l'auteur dans ses travaux [227]. Ceci dit, les propriétés nécessaires et suffisantes sont les propriétés qui doivent être explicitées selon le niveau de raffinement dans lequel nous nous situons dans la modélisation en Event-B. Cependant, ces propriétés sont déterminées de sorte à ce que les preuves des invariants et des propriétés de sûreté soient réussies.

1.4 Ingénierie système et génie logiciel

L'ingénierie système [186, 145] cherche à définir des approches de modélisation, de vérification et de simulation dans l'objectif de valider les exigences de conception d'un système. Cette discipline vise à développer des méthodes de conception, de modélisation et d'analyse de systèmes. La traçabilité des exigences, la justification des choix ainsi que la vérification et la validation des résultats inhérents à une démarche sont les principales activités fournies par l'ingénierie système. L'ensemble des phases de conception d'un système et l'enchaînement de leurs activités définissent le cycle de vie de ce système. Ces phases incluent la conceptualisation, la conception, la réalisation, l'intégration, l'exploitation et le retrait des services. Plusieurs méthodes de conception ont été définies dans la littérature, on peut citer le cycle en V, méthode la plus utilisée, la méthode de développement en cascade [225], et la méthode en spirale [51].

Le génie logiciel a pour vocation la conception de systèmes dans des domaines où les enjeux en terme de fiabilité et de sécurité sont importants. Les défis que soulève le génie logiciel pour garantir une exigence qualité à « zéro défaut » sont la mise au point de méthodes formelles [262] qui adoptent des techniques de vérification et de validation. La vérification et la validation sont des techniques qui appliquent des actions à tous les produits issus du cycle de développement pour assurer la conformité de ces produits par rapport à un ensemble de dispositions pré-établies. La vérification permet de s'assurer que le système, à différents niveaux d'abstraction, présente les propriétés voulues et est donc conforme à ses spécifications. La validation permet de vérifier que le système fonctionne comme attendu par le client dans son environnement opérationnel [50]. Alors que la vérification répond à la question : le système est-il correctement construit ?, la validation tente de répondre à la question : est ce que nous construisons le bon système ?

La recherche, l'ordonnancement, la caractérisation, la hiérarchisation et la valorisation des fonctions attendues d'un système sont les principales tâches pour exprimer des exigences. Spécifier des exigences définit l'étape initiale de tout processus de développement de logiciel. Une exigence (cf. définition 1.4.1) se définit comme l'expression d'une caractéristique à laquelle un système doit répondre. Pour capturer les incohérences et les ambiguïtés qui peuvent être détectées dans le cycle de développement d'un système, ces méthodes doivent démontrer la satisfaction du système vis-à-vis de ses exigences décrites par une spécification. Ces activités qu'on appelle ingénierie des exigences, cherchent, étant

2. L'introduction des ensembles porteurs qui, dans notre cas correspondent aux concepts de plus haut niveau de la hiérarchie, importe peu : on peut les introduire au niveau du contexte Event-B le plus général, tout comme il est possible de les introduire selon le besoin (graduellement) dans les machines Event-B.

donné une spécification d'un système exprimée dans un modèle ayant une sémantique formelle, à vérifier et valider des propriétés comportementales issues de cette spécification dans le but de détecter des erreurs éventuelles.

Définition 1.4.1 (exigence) *Une exigence est une caractéristique d'un système dont un utilisateur ou un client a besoin pour résoudre un problème ou atteindre un objectif dans un contexte donné [206].*

Ce sont les clients qui fournissent les documents qui définissent les exigences du système à concevoir exprimées sous forme de besoins. Ces besoins sont les résultats de perception qu'à l'utilisateur du système ou du logiciel. Les exigences incluent, non seulement les besoins des clients, mais aussi des normes et des référentiels métier. L'acquisition et l'extraction de connaissances issues d'exigences sont nécessaires pour établir des listes de recommandations complètes, consistantes et sans ambiguïtés. Il est toujours difficile de détecter les erreurs tôt dans le processus de conception, d'où l'intérêt d'utiliser les méthodes formelles dans le but d'obtenir un système validé par construction. Ces techniques sont caractérisées par des raisonnements rigoureux qui se fondent sur la logique mathématique.

Les techniques utilisées en méthodes formelles sont généralement fournies par des langages de spécification, pour définir les activités de développement logiciel, vérifiées à des fins d'automatisation [262]. Les fondements mathématiques sur lesquels se basent les méthodes formelles définissent des notions caractérisées par leur cohérence et leur exhaustivité, et en particulier, des spécifications et des mises en oeuvre dites correctes. La preuve de programmes corrects était la première problématique à laquelle s'intéressent les méthodes formelles, où une spécification correcte est supposée disponible, et le but était de montrer qu'un programme dans un langage de programmation concret satisfait cette spécification. Ces méthodes fournissent des mécanismes pour prouver qu'une spécification donnée d'un système était réalisable, correctement implémentée, et que les propriétés du système sont prouvées sans nécessairement exécuter le système pour en déterminer son comportement.

Les spécifications peuvent servir de contrat utilisé comme moyen de communication entre les parties prenantes : les clients, les spécifieurs, les développeurs, etc. Des outils ont été développés pour automatiser le processus d'analyse de modèles pourvu que la sémantique du langage soit suffisamment restreinte pour que l'analyse de ces modèles soit rigoureuse et mécanisée. La complétude et la cohérence sont les critères attendus pour assurer la qualité d'une spécification. Une spécification *complète* garantit la présence de toute l'information nécessaire à la représentation d'une vue du système considérée, alors que la *cohérence* ou la *consistance* d'une spécification impose l'absence de contradiction dans la représentation de ses différents modèles.

1.4.1 Modélisation

La modélisation est l'activité de construire un modèle pour représenter le système en cours de développement qu'on appelle *système d'intérêt*, et l'environnement dans lequel le système opère [221]. Cette construction s'accomagne toujours d'une phase de récolte d'informations à propos du système avant toute activité de conception, de développement ou de test du système.

Les modèles sont utilisés comme moyen pour décrire des systèmes à différents niveaux

dans le processus de développement, en commençant par la phase d'expression des besoins, jusqu'à arriver au code final, en passant par la description de modules et en raffinant chacun de ces modules. Les modèles sont des descriptions formelles du cahier des charges au moyen de langages formels ou semi-formels. Les modèles servent à analyser et comprendre les systèmes, et sont donc un moyen pour la conception des systèmes. Les modèles sont des approximations, des vues plus ou moins abstraites de la réalité [123]. Les modèles sont définis dans l'objectif de simplifier cette réalité selon un point de vue établi pour atteindre un objectif donné. La connaissance du système étudié n'est représentée qu'en partie par un modèle. Des techniques qu'on appelle des *abstractions* sont utilisées pour ne représenter que les parties pertinentes d'un système dans son environnement d'utilisation, et masquer les détails inutiles.

Frantz [123] a montré que s'il existe des hypothèses liées à la perception et la connaissance qu'un développeur de modèle a de la réalité, alors il existe aussi des hypothèses permettant d'abstraire ce modèle pour faciliter sa compréhension et réduire les coûts de son implémentation selon l'objectif d'utilisation et cela de manière intentionnelle. Il montre par ailleurs, qu'une abstraction n'est valide qu'à condition qu'elle préserve la validité des résultats de la simulation. Selon l'auteur [123] le processus de construction d'une simulation est décrit : par le passage du monde réel vers un modèle conceptuel par un mécanisme d'abstraction ; puis, par le passage du modèle conceptuel vers un modèle de simulation par des mécanismes d'implémentation.

De nombreux modèles doivent être construits en fonction des usages et des objectifs des différents acteurs d'un projet. De plus, le choix de notations appropriées pour décrire un système n'est pas trivial, puisque il faut faire des compromis entre expressivité des langages de spécification et le niveau d'abstraction que ces langages prennent en charge [57, 262]. L'*expressivité* s'associée à la capacité d'une notation textuelle ou graphique à spécifier les systèmes de façon plus ou moins riche. Les spécifications peuvent concerner plusieurs aspects d'un système. Le niveau de détail accordé à une spécification d'un système définit son *abstraction*. L'attachement aux caractéristiques et aux propriétés les plus importantes dans un système conduisent à des spécifications d'un haut niveau d'abstraction. Ceci permet d'avoir des spécification concises et plus faciles à valider. Dans cette catégorie de langages nous pouvons citer les logiques d'ordre supérieur : par exemple, HOL [136], PVS [208], Coq [99], les algèbres de processus : CSP [65, 152], CCS [192].

1.4.2 Différents formalismes pour la spécification des systèmes

Les spécifications d'un système comportent les composantes statique et dynamique d'un système. La partie statique correspond à la structure et aux données qui caractérisent le système. Les aspects dynamiques définissent le comportement et les interactions avec l'environnement du système. Les formalismes sont donc classés selon ces orientations statiques et dynamiques [3] :

— Les formalismes qui permettent de spécifier les aspects statiques d'un système sont orientés données : on peut citer les approches algébriques *orientées propriétés* [263, 114], ou les approches *orientées états* (VDM [159], B [2], Z [179]). La première catégorie de formalismes expriment les propriétés des systèmes par la définition de domaines des objets manipulés, des opérations autorisées sur ces objets et d'une axiomatisation des propriétés des opérations. Les structures de données

3. La classification donnée ici est celle de l'auteur dans ses travaux [56].

sont décrites de manière abstraite. Sur la base de cette description des équations sont définies en utilisant la logique des prédicats du premier ordre. Parmi les spécifications les plus connues, on peut citer : Pluss [127], Raise [138] et Casl [113]. Dans cette même catégorie, on trouve aussi les logiques d'ordre supérieur telles que PVS [208], Coq [99] ou Isabelle [203]. Dans la seconde catégorie les spécifications du système sont modélisées par des opérations, quant aux données, elles sont représentées par des ensembles. Les opérations sont appliquées pour manipuler les ensembles et consulter les états du système. Les opérations sont soit : sous forme d'algorithme abstrait tel que c'est le cas de VDM sous sa forme explicite, soit sous forme de substitutions généralisées tel que c'est le cas dans B, soit sous forme de pré et post-conditions tel que c'est le cas dans Z.

— La dynamique d'un système se traduit par des caractéristiques liées au comportement du système. Les comportements sont les enchaînements des opérations ou des événements du système. Parmi ces caractéristiques on peut citer : le comportement qui relève des opérations définies dans le système ; les enchaînements ou la chronologie des opérations à effectuer dans le système considéré. Cette caractéristique peut relever par exemple au non-déterminisme des opérations ; la communication que le système entretient avec d'autres composants ou avec son environnement (système fermé ou ouvert). Dans cette catégorie, on peut citer : les logiques temporelles telles que LTL (Linear Temporal Logic en anglais) [173, 211], les logiques arborescentes telles que CTL (Computation Tree Logic) [77] ; les approches basées sur les systèmes de transitions telles que [20] ; ou les algèbres de processus telles que CSP [65, 152] etc. Les approches basées sur les pré- et post-condition telles que B, etc.

De toute cette diversité, il en découle que la construction ou la conception de ces systèmes devient très complexe. Cette complexité est due en grande partie à l'absence de sémantique explicite dans les spécifications. Cette absence conduit à des besoins mal exprimés, mal formulés ou mal perçus, où l'imprécision ou l'incomplétude est présente, d'où des solutions non justifiées/validées, ... Les besoins de compréhension et de maîtrise de ces systèmes sont indispensables.

Les formalismes peuvent être associés pour se compléter et exprimer plus de propriétés ou combler les lacunes de l'un par l'autre des formalismes associés. Les associations entre formalismes sont classées en [56] : *association par dérivation*, ou *association par intégration*.

— L'association par intégration implique l'intégration de l'aspect statique avec les aspects dynamiques. La combinaison se base sur l'utilisation complémentaire des caractéristiques du système et sur la nature des formalismes. Dans cette classe, on peut citer l'association des spécifications algébriques et Statecharts de Harel (langage Casl et des Statecharts) [228]. Ce type d'association répond à des besoins d'expressivité et peut induire à des difficultés lors de la validation. Ces approches peuvent donner naissance à de nouveaux formalismes ;

— L'association par dérivation : Le couplage dans ce type d'association conduit souvent à des traductions dans un langage cible afin de profiter des possibilités de vérification qui peuvent être permises par l'outil visé. On peut citer parmi ces outils la méthode B avec les prouveurs de théorèmes basés sur les logiques d'ordre supérieur telles que HOL ou Coq. Parmi ces associations on trouve : l'association de B dévivé dans Coq et PVS [49]. D'autres associations sont définies pour traduire l'algèbre de processus CSP en spécifications B [67]. Cette traduction permet de décrire les activités séquentielles en B car la méthode B donne la possibilité de représenter en terme d'événements des activités distribuées, mais ne permet pas de représenter les

activités séquentielles telle que c'est le cas dans CSP. Ces approches assurent en général des possibilités non négligeables en terme de vérification et de génération de code.

Dans cette thèse, nous nous focalisons sur de nouvelles associations basées sur le langage formel Event-B et les formalismes orientés connaissances que sont les ontologies de domaine. Ces approches sont nouvellement étudiées dans la littérature, car à la base, les ontologies sont développées à des fins de représentation des connaissances telles que indiqué par la section précédente (cf. section 1.3). Ces points seront développés en section 1.6.

1.4.3 Quelles propriétés ?

Le développement des systèmes fait intervenir différents aspects : données, comportements, contrôle, temps, etc. Les modèles sont des *prescriptions* [243] qui décrivent ce qu'un système *doit satisfaire* pour être *conforme aux besoins énoncés* dans le cahier de charges en vérifiant des propriétés (cf. définition 1.4.2).

Définition 1.4.2 (propriété) *Une propriété est une caractéristique intrinsèque (comportementale, fonctionnelle, structurelle ou organique, dépendante ou non du temps) d'une entité (un système, un modèle, une entité de modélisation, un phénomène etc.) [74].*

Les propriétés représentent un grain de connaissance devant être décrite par l'acteur et vérifiée sur un modèle du système afin de prouver la cohérence de la représentation demandée par l'utilisateur. La question qui se pose est : quels sont les types de propriétés à spécifier et à vérifier ? Lors de la conception de systèmes, les propriétés sont classées en propriétés *statiques* ou *dynamiques*, selon que la propriété assure la cohérence du système, ou traite de l'occurence ou du séquencement des événements [74]. Les propriétés d'*invariance*, de *sûreté* et de *vivacité* sont classées dans ce deuxième type de propriétés.

— La sûreté se définit par : « quelque chose de mauvais ne doit jamais arriver ». Par exemple, « le solde d'une carte ne doit jamais être négatif » est une propriété de sûreté.
— La vivacité se définit par : « quelque chose d'attendu arrivera nécessairement ». Par exemple, « si un utilisateur demande un service de retrait d'argent par carte bancaire, alors le service lui sera fourni fatalement ».

D'autres propriétés sont aussi données dans la littérature telles que les propriétés d'accessibilité (atteignabilité) et les propriétés de précédence [39]. Les propriétés liées au temps ne peuvent être exprimées qu'au moyen de logique temporelle. Les invariants permettent de restreindre l'espace des états issus des comportements du système étudié. Nous abordons ces propriétés en détail dans le chapitre consacré au formalisme Event-B (cf. chapitre 2), et nous décrivons brièvement le formalisme TLA dans la section qui suit (cf. sous-section 1.5.2.1), puisque c'est en se basant sur ce formalisme que nous exprimons la vivacité dans le formalisme Event-B.

1.4.4 Vérification et validation à base de modèles

En méthodes formelles il est question de vérifier et de valider des spécifications et des conceptions au moyen de formalismes par rapport à des propriétés devant être satisfaites par le système attendu. Au niveau de la modélisation [59], ces deux activités répondent respectivement aux questions suivantes : Vérification : Ai-je bien construit le modèle

correctement ?, validation : Ai-je bien construit le bon modèle ? La réalisation de ces deux tâches peut être faite par des preuves de théorème tel que [114, 174], de la vérification de modèle (model checking) tel que PVS [209] ou des tests d'équivalence [176].

C'est en se basant sur des modèles que l'activité de vérification parvient à exprimer des propriétés et à construire des preuves. Les ensembles des états auxquels le système peut accéder sont capturés et exprimés par des formules logiques qu'on appelle *assertions*. Les techniques utilisées à cette fin sont la preuve de théorèmes et la vérification de modèles. Une première étape de modélisation définit un modèle de comportement du système. Cette étape est suivie d'une phase de génération de formules qu'on appelle *obligations de preuve*. Vient ensuite l'étape de vérification que ces obligations de preuve sont bien déchargées. L'introduction des outils prouveurs de théorèmes et des vérificateurs de modèles (model-checkers en anglais) a rendu possible la construction de preuves de cohérence bien fondée vérifiables [57].

Il existe deux grandes familles de vérification pour résoudre ce problème :

— les techniques basées sur le *model-checking* : elles sont développées initialement par [78, 218]. Ces techniques explorent l'ensemble de toutes les configurations possibles d'un système afin de s'assurer qu'aucune d'elles ne présente de dysfonctionnement. Ces techniques présentent l'avantage d'être automatiques, où l'utilisateur n'intervient pas pour avoir le résultat de la vérification. Ces méthodes peuvent faire face au problème lié au nombre infini des configurations possibles ;
— les techniques de preuves : historiquement c'est avec Floyd et Hoare [120, 151] que ces méthodes ont vu le jour. Elles traduisent la conformité d'un programme vis-à-vis du modèle en un ensemble de formules, que sont les *obligations de preuve*. La preuve que ces formules sont vraies vis-à-vis d'une théorie (validité de ces formules) garantit que le système est en conformité avec son modèle ou est conforme à une propriété donnée. Ces techniques utilisent des logiciels qu'on appelle prouveurs, pour faire les preuves de ces formules, automatiquement ou interactivement. Ces techniques font l'hypothèse que le modèle représente bien le cahier des charges, d'où l'intérêt de les combiner avec les techniques de validation.

Théoriquement, il est possible de vérifier complètement un modèle formel, mais dans la pratique, des spécifications imprécises, la complexité du modèle ou le manque de connaissance du modèle empêchent une vérification exhaustive. L'utilisation d'un modèle ou des résultats de simulation d'un modèle s'attend à ce qu'il soit adapté à l'objectif visé, dans les limites acceptables [59]. La validation des modèles ou des résultats de simulation est effectuée par rapport à un système réel et toujours en fonction de l'objectif prévu de l'utilisation du modèle. L'auteur dans ses travaux [59] explique que la satisfaction des besoins est liée à sa "suitability", décomposée en "capability", "fidelity" et "accuracy". Selon le même auteur la validation se définit comme le processus de démontrer que le modèle et son comportement sont une représentation adéquate ("suitable") du système réel et de son comportement selon un objectif d'utilisation de la simulation. La principale différence par rapport à la vérification est que le modèle est vérifié en tant que solution de remplacement du système réel par rapport à son utilisation prévue (en supposant qu'il est correct). Cette activité est réalisée dans un environnement opérationnel du système. Elle peut également être effectuée par simulation de cet environnement opérationnel pour assurer que les exigences sont bonnes menant ainsi le processus de développement à une conclusion satisfaisant l'utilisateur. Les méthodes utilisées en validation sont aussi basées sur le text, l'analyse, l'inspection ainsi que la démonstration et la simulation.

1.5 Formalismes logiques

Les logiques sont aussi vues comme des méthodes d'argumentation, des sciences de la démonstration, des disciplines dont la norme est la vérité, une composition formelle des concepts, des recherches théoriques de l'informatique ... L'évolution de la logique à travers des siècles donne des réponses qui représentent son histoire. L'objectif était d'établir les bases du raisonnement ou construire un fondement théorique des mathématiques. Des logiques qu'on appelle *logiques classiques* ont été développées à cette fin. Ces logiques comportent une contribution historique, puisqu'elles sont adoptées dans de nombreux travaux dans le génie logiciel pour la démonstration de propriétés que les systèmes conçus (ou doivent être conçus) doivent satisfaire. Les logiques utilisées en intelligence artificielle sont des *logiques non-classiques* utilisées pour modéliser des situations réelles où la connaissance est incomplète et imprécise, puisqu'il faut faire des hypothèses dont la plausibilité est avérée, quelque fois réfutables a posteriori, par l'acquisition de nouvelles connaissances. Cette distinction constitue le premier point de différence entre le génie logiciel et l'ingénierie des connaissances.

La logique classique ciblée ici est la logique du premier ordre puisque le langage utilisé dans nos travaux qu'est le formalisme Event-B (cf. chapitre 2) est basé sur cette logique. La logique du premier ordre, également appelée logique des prédicats, exprime des connaissances sur les objets du monde pouvant être très nombreux. Ces connaissances sont exprimées en termes d'un ensemble de propriétés de chaque objet et des relations qui existent entre eux. La possibilité de faire des déclarations générales sur plusieurs objets dans le monde est rendue réalisable grâce à l'utilisation de variables logiques et de quantificateurs, ce qui fait d'elle un formalisme très expressif, plus approprié pour l'expression des connaissances structurées.

1.5.1 Logiques avec contraintes du domaine pour la représentation des connaissances

Si la logique du premier ordre est très expressive, ceci est au détriment de sa décidabilité. Premièrement, la vérification de la validité et la satisfiabilité d'une proposition (*sentence*) est indécidable en logique du premier ordre complète, mais est rendue décidable dans les logiques du premier ordre avec les contraintes du domaine [25, 98]. Deuxièmement, il peut être souhaitable d'interpréter les théories logiques de premier ordre en fonction des théories propositionnelles qu'elles représentent, lorsque toutes les formules sont fondées. Une telle théorie propositionnelle correspondante existe toujours, mais elle n'est finie que lorsque les quantificateurs s'étendent sur des domaines finis. De plus, le maintien d'un lien étroit avec la logique propositionnelle rend possible de capitaliser sur les avancées algorithmiques qui ont eu lieu au cours des dernières décennies dans le monde du raisonnement propositionnel automatisé comme la résolution SAT [156] qui est une grande réussite en informatique. Des progrès similaires ont été réalisés pour le problème de dénombrement de modèles [133]. L'utilisation des contraintes des domaines est aussi motivée dans le monde du raisonnement probabiliste, où de nombreux systèmes réduisent les problèmes du premier ordre à des problèmes de propositions pour répondre aux requêtes dans l'apprentissage qu'on appelle la propositionnalisation ou la construction d'un modèle fondé sur la connaissance (pour plus de détails consulter [98, 226]). Parmi les logiques qui prennent en considération les contraintes du domaine, on peut citer les extensions : la

logique du premier ordre avec contraintes du domaine FOL-DC (pour First Order Logic with Domain Contraints en anglais) [98] et les logiques de description [25].

1.5.1.1 Les logiques de description

Les logiques de description DLs [24] sont une famille de formalismes pour la représentation des connaissances avec des propriétés formelles bien comprises. Elles sont introduites par Brachman [58] dans les années 1979, et sont issues de la logique des prédicats, destinées à la représentation des connaissances. Une base de connaissances Σ exprimée dans une logique de description est constituée de deux composants, traditionnellement appelés T-Box et A-Box (à l'origine de "Terminological Box" et Assertional Box" respectivement) :

— La T-Box, appelée aussi *connaissances terminologiques*, stocke un ensemble d'assertions quantifiées, indiquant les propriétés générales des concepts et des rôles. Par exemple, une assertion de ce genre est celle qui stipule qu'un certain concept, Parent par exemple, est défini comme une expression donnée qui utilise d'autres concepts et rôles, par exemple "Personne avec au moins un enfant".

— La A-Box, connue sous le nom de *connaissances factuelles*, comprend des assertions sur les objets individuels, appelées assertion d'instances. Typiquement une assertion dans la A-Box est celle affirmant qu'un *individu* est une instance d'un certain concept. Par exemple, on peut affirmer que Bill est une instance de "Personne avec au moins un enfant". Ces assertions définissent des relations d'instanciation entre les individus et les concepts, entre les paires d'individus et les rôles.

Trois principales entités de base définissent respectivement les *objets*, les *classes* ainsi que les *attributs* du modèle objet. Le concept est la principale unité de connaissances crée par une unique combinaison de propriétés.

— les *individus* par qui la notion d'identité est reflétée ;
— les *concepts* dénotant des ensembles d'individus, l'existence d'un jeu de constructeurs qui permettent de combiner des concepts pour en former d'autres les distingue des classes. Leur définition fournit à la fois des conditions *nécessaires* et *suffisantes* ;
— les *rôles* qui dénotent des relations binaires entre les individus ;

Un modèle du domaine en logique de description est intégré dans la Base de Connaissances Σ définie par la T-Box et la A-Box.

Plusieurs tâches de raisonnement peuvent être effectuées sur une base de connaissances Σ. La forme la plus simple de raisonnement consiste à calculer la relation de *subsomption* entre deux expressions conceptuelles, c'est-à-dire, vérifier si une expression désigne toujours un sous-ensemble des objets désignés par une autre expression. Ces connaissances établissent des relations de spécialisation/généralisation entre les concepts. La subsomption permet de capturer différents types de mécanismes de sous-classification ; D'autres types de relations peuvent être modélisés, tels que le regroupement, la matérialisation et l'agrégation partielle.

Les principales caractéristiques de la logique de description résident dans les concepts permettant d'établir des relations entre les concepts. Les plus basiques sont celles de *restrictions de valeur*. Par exemple, une restriction de valeur, écrite $\forall R.C$, exige que tous les individus qui sont dans la relation R avec le concept décrit appartiennent au concept C (techniquement, ce sont tous les individus qui sont dans la relation R avec un individu décrit par le concept en question qui sont eux-même descriptibles comme C). La sémantique

sous-jacente est que les concepts sont interprétés comme un ensemble d'individus et les rôles sont interprétés comme des ensembles de paires d'individus. Les concepts atomiques ou primitifs sont donc interprétés comme des sous-ensemble du domaine d'interprétation, alors que la sémantique des autres constructeurs est ensuite spécifiée en définissant l'ensemble des individus désignés par chaque construction.

Dans ce qui suit, nous adoptons les notations de [118].

1.5.1.2 Syntaxe des concepts et des rôles

Définition 1.5.1 (signature) *Un langage d'*expressions conceptuelles *est caractérisé par une signature* $S = (O, C_a, R_a, Cstr)$ *où :*

- O *est un ensemble d'objets (notés* o_i*) ;*
- C_a *est un ensemble de concepts atomiques (notés* A_i*) ;*
- R_a *est un ensemble de rôles atomiques (notés* r_i*) ;*
- $Cstr$ *est un ensemble de constructeurs permettant de former des concepts (notés* C_i*) et des rôles (notés* R_i*) ;*

Les constructeurs qui permettent de former toute une famille de logiques de description sont nombreux. La logique la plus basique est notée $\mathcal{AL}$, pour "*A*ttributive concept *L*anguage" et regroupe les constructeurs suivants :

$$\begin{array}{rcl} \top & : & \text{top (concept le plus général)} \\ \bot & : & \text{bottom (concept le plus spécifique)} \\ C_1 \sqcap C_2 & : & \text{conjonction de deux concepts} \\ \neg A & : & \text{négation d'un concept atomique} \\ \forall r.C & : & \text{quantification universelle sur un rôle} \\ \exists r & : & \text{quantification existentielle sur un rôle} \end{array}$$

L'exemple qu'on peut prendre est le suivant :

$$Ensemble \ \sqcap \ ((\forall membre.Personne) \ \sqcap \ (\exists chef))$$

cette expression dénote les ensembles qui ont un chef et dont tous les membres sont des personnes.

Les restrictions sont imposées sur le co-domaine du rôle dans le cas de la quantification universelle, alors que la quantification existentielle impose que le co-domaine ne soit pas vide. D'autres constructeurs étendent le langage $\mathcal{AL}$:

$$\begin{array}{rcl} \neg C & : & \text{négation de concepts non-atomiques) } (\mathcal{C}) \\ C_1 \sqcup C_2 & : & \text{disjonction de concepts } (\mathcal{U}) \\ \exists r.C & : & \text{quantification existentielle qualifiée} (\mathcal{E}) \\ \geq n \ r & : & \text{cardinalité minimale d'un rôle} (\mathcal{N}) \\ \leq n \ r & : & \text{cardinalité maximale d'un rôle} (\mathcal{N}) \\ r_1 \sqcap r_2 & : & \text{conjonction de rôles atomiques } (\mathcal{R}) \end{array}$$

Les extensions $\mathcal{U}$ et $\mathcal{E}$ sont incluses dans l'extension $\mathcal{C}$ car on a l'égalité $C_1 \sqcup C_2 = \neg(\neg C_1 \sqcap \neg C_2)$ et $\exists r.C = \neg\forall r.\neg C$.

D'autres constructeurs peuvent être ajoutés. Par exemple, $\mathcal{H}$ qui correspond à la Hierarchie des rôles ou sous-rôles ($\mathcal{H}$ierarchies), $\mathcal{I}$ représente les rôles inverses ($\mathcal{I}$nverse), $\mathcal{O}$

représente un nominal (n$\mathcal{O}$minals), et $\mathcal{Q}$ représente les restrictions sur les nombres qualifiées ($\mathcal{Q}$ualified). Un constructeur nominal donne la possibilité d'utiliser les noms pour la description des concepts : si a est un nom d'individu, alors $\{a\}$ est un concept, appelé nominal, interprété par un ensemble singleton.

Les logiques de description sont un fragment décidable de la logique du premier ordre [55, 25]. qui considèrent les rôles comme des relations binaires et les concepts comme des relations unaires. Ces logiques sont les modèles formels des langages ontologiques OWL. OWL 1 DL est équivalent à l'extension de $\mathcal{SHOIN}(\mathcal{D})$ avec seulement le concept le plus général $\top$ comme étant le concept autorisé dans les restrictions des nombres qualifiés avec des nominaux, alors que $\mathcal{SHOIQ}(\mathcal{D})$ est considéré comme un modèle formel du langage d'ontologie OWL 2 DL, ayant une capacité de raisonnement qui est complet où toutes les conclusions sont calculables, et décidable c-à-d : les conclusions sont calculables en un temps fini. Par conséquent, cette variante est mieux adaptée pour son utilisation à la fois avec le formalisme Event-B et pour l'extraction de connaissances issues du domaine.

1.5.1.3 Sémantique des concepts et des rôles

La sémantique des logiques de description bien fondée leur assure des mécanismes d'inférence bien fondés pour lesquels on peut démontrer des résultats de cohérence et de complétude. Les notions d'*interprétation* et de *modèle* sont à la base de cette sémantique. Il faut noter que les atomes des logiques de description sont interprétés comme des ensembles d'individus, et non comme des valeurs de vérité tel que c'est le cas pour la logique classique. Un ensemble d'individus $\Delta_\mathcal{I}$ est introduit en guise de domaine d'interprétation, et une fonction d'interprétation $.^\mathcal{I}$ qui associe à chaque expression conceptuelle un sous-ensemble du domaine d'interprétation.

Définition 1.5.2 (interprétation) *Etant donnée une signature $S = (O, C_a, R_a, Cstr)$. Une interprétation $\mathcal{I} = (\Delta_\mathcal{I}, .^\mathcal{I})$ de S est la donnée d'un ensemble $\Delta_\mathcal{I}$ appelé domaine de l'interprétation et d'une fonction d'interprétation $.^\mathcal{I}$ qui fait correspondre à tout objet o_i un individu $o_i^\mathcal{I} \in \Delta_\mathcal{I}$, à tout concept atomique A_i un sous-ensemble $A_i^\mathcal{I} \subseteq \Delta_\mathcal{I}$, et à tout rôle atomique r_i, un sous-ensemble $r_i^\mathcal{I} \subseteq \Delta_\mathcal{I} \times \Delta_\mathcal{I}$.*

La sémantique des constructeurs donnés dans ce qui prècède est définie comme suit :

$$\begin{aligned}
\top^\mathcal{I} &= \Delta_\mathcal{I} \\
\bot^\mathcal{I} &= \varnothing \\
(\neg C)^\mathcal{I} &= \Delta_\mathcal{I} \setminus C^\mathcal{I} \\
(C_1 \sqcup C_2)^\mathcal{I} &= C_1^\mathcal{I} \cup C_2^\mathcal{I} \\
(C_1 \sqcap C_2)^\mathcal{I} &= C_1^\mathcal{I} \cap C_2^\mathcal{I} \\
(\forall r.C)^\mathcal{I} &= \{x \in \Delta_\mathcal{I} \mid \forall y : (x,y) \in r^\mathcal{I} \Rightarrow y \in C^\mathcal{I}\} \\
(\exists r.C)^\mathcal{I} &= \{x \in \Delta_\mathcal{I} \mid \exists y : (x,y) \in r^\mathcal{I} \wedge y \in C^\mathcal{I}\} \\
(\geq n\ r)^\mathcal{I} &= \{x \in \Delta_\mathcal{I} \mid card\{y \mid (x,y) \in r^\mathcal{I}\} \geq n\} \\
(\leq n\ r)^\mathcal{I} &= \{x \in \Delta_\mathcal{I} \mid card\{y \mid (x,y) \in r^\mathcal{I}\} \leq n\}
\end{aligned}$$

Définition 1.5.3 (subsomption et consistance) *Soient deux concepts C et D. On dit que*

- *C est subsumé pat D, et qu'on note par $C \sqsubseteq D$, si pour toute interprétation $\mathcal{I}$ on a $C^\mathcal{I} \subseteq D^\mathcal{I}$;*
- *C est équivalent à D, noté par $C \equiv D$, si pour toute interprétation $\mathcal{I}$ on a $C^\mathcal{I} = D^\mathcal{I}$;*

— *C et D sont incompatibles ou disjoints si pour toute interprétation $\mathcal{I}$ on a $C^{\mathcal{I}} \cap D^{\mathcal{I}} = \varnothing$;*
— *C est satisfiable ou consistant s'il existe une interprétation $\mathcal{I}$ pour laquelle $C^{\mathcal{I}} \neq \varnothing$.*

Ces notions constituent la part du raisonnement terminologique qui permet la *classification* des concepts selon la relation de *subsomption*, qui définit une relation de spécialisation/généralisation vu qu'elle se traduit par l'inclusion des ensembles d'individus dénotés par les concepts. En fait, les quatre notions précédentes peuvent être réduites au test de subsomption.

Propriété 1.5.1 *Pour tous concepts C, D, on a les relations suivantes :*

1. *$C \equiv D$ ssi $C \sqsubseteq D$ et $D \sqsubseteq C$*
2. *C et D sont incompatibles ssi $C \sqcap D \sqsubseteq \bot$*
3. *C est consistant ssi $C \not\sqsubseteq \bot$*

1.5.1.4 Base de connaissances et mécanismes d'inférence

La subsomption est le principal mécanisme utilisé dans les logiques de description pour raisonner garantissant un raisonnement logique sur les concepts. Elle ne permet de prendre en compte ni les connaissances *terminologiques* qui correspondent aux relations entre les concepts, ni les connaissances *factuelles* qui définissent les relations entre individus et concepts. Les connaissances terminologiques définissent le domaine d'application et portent sur les concepts, alors que les connaissances factuelles correspondent à une base de connaissances particulière et concernent les objets.

Définition 1.5.4 (base de connaissances) *Etant donnée la signature $S = (O, C_a, R_a, Cstr)$. Une base de connaissances sur cette signature se définit comme une paire $\Sigma = (T, A)$ où :*
— *T est la composante Terminologique (appelée aussi T-Box), c-à-d. un ensemble d'axiomes de la forme $C_i \sqsubseteq C_j$ (où C_i et C_j sont des expressions conceptuelles) ;*
— *A est la composante Assertionnelle (appelée aussi A-Box), c-à-d. un ensemble d'assertions de la forme $o : C$ (où $o \in O$ et C est une expression conceptuelle) ou $(o_i, o_j) : R$ (où $o_i, o_j \in O$ et R est une expression de rôle).*

Informellement, $C_i \sqsubseteq C_j$ signifie que toute instance de C_i est aussi une instance de C_j ; $o : C$ signifie que o est une instance de C ; et $(o_i, o_j) : R$ signifie que l'arc liant o_i à o_j est une instance du rôle R. Plus formellement, on définit les *modèles* d'une base de connaissances comme les interprétations qui satisfont les axiomes et les assertions.

Définition 1.5.5 (modèle) *Soit une base de connaissances $\Sigma = (T, A)$ et une interprétation $\mathcal{I}$, pour une même signature S. On dit que $\mathcal{I}$ est un modèle de Σ, et on note $\mathcal{I} \models \Sigma$, ssi*
— *pour tout $C_i \sqsubseteq C_j$ dans T, $C_i^{\mathcal{I}} \subseteq C_j^{\mathcal{I}}$;*
— *pour tout $o : C$ dans A, $o^{\mathcal{I}} \in C^{\mathcal{I}}$;*
— *pour tout $(o_i, o_j) : R$ dans A, $(o_i^{\mathcal{I}}, o_j^{\mathcal{I}}) \in R^{\mathcal{I}}$*

Nous donnons en figures 1.1 et 1.2 de l'exemple 1.5.1.4 qui suit un exemple de T-Box et de A-Box modélisant différents concepts ainsi que les individus (instances) pour un domaine d'interprétation. La variante des logiques de description qui nous intéresse est $\mathcal{SHOIQ}(\mathcal{D})$ car elle est décidable et son raisonnement est complet. Ce point sera détaillé en section 3.4.2 du chapitre 3.

Exemples L'exemple qu'on peut donner ici est celui des vehicules possèdant des roues, un moteur, une marque, un modèle. Un individu de ce concept est identifié par son *immatriculation*. Les concepts et les rôles définis sont les suivants :

— Vehicles, Wheels Motors, Highway, Owners.
— hasWheels, hasMotor, hasaccessHighway

$$
\begin{aligned}
&\mathsf{Vehicles} \sqsubseteq \top && (1)\\
&\mathsf{Wheels} \sqsubseteq \top && (2)\\
&\mathsf{Motors} \sqsubseteq \top && (3)\\
&\mathsf{Highway} \sqsubseteq \top && (4)\\
&\mathsf{Owners} \sqsubseteq \top && (5)\\
&\top \sqsubseteq \leq 1\ \mathsf{hasWheels} && (6)\\
&\geq 1\ \mathsf{hasWheels} \sqsubseteq \mathsf{Vehicles} && (7)\\
&\top \sqsubseteq \forall \mathsf{hasWheels.Wheels} && (8)\\
&\forall \mathsf{hasWheels} : y = \mathsf{Vehicles} && (9)\\
&\top \sqsubseteq \leq 1\ \mathsf{hasWheels}^{-1} && (10)\\
&\top \sqsubseteq \leq 1\ \mathsf{hasMotor} && (11)\\
&\geq 1\ \mathsf{hasMotor} \sqsubseteq \mathsf{Vehicles} && (12)\\
&\top \sqsubseteq \forall \mathsf{hasMotor.Motors} && (13)\\
&\forall \mathsf{hasMotor} : y = \mathsf{Vehicles} && (14)\\
&\top \sqsubseteq \leq 1\ \mathsf{hasaccessHighway} && (15)\\
&\geq 1\ \mathsf{hasaccessHighway} \sqsubseteq \mathsf{Vehicles} && (16)\\
&\top \sqsubseteq \forall \mathsf{hasaccessHighway.Highway} && (17)
\end{aligned}
$$

FIGURE 1.1 – Exemple de T-Box

hasWheels est une fonction partielle (axiome (6)), ayant comme domaine Vehicles (axiome (7)), et comme co-domaine Wheels (axiome (8)) et est totale (axiome (9)) avec la relation inverse définie aussi comme étant une fonction partielle (axiome (10)) ; hasMotor est une fonction partielle (axiome (11)), ayant comme domaine Vehicles (axiome (12)), qui est aussi totale (axiome (14)) et ayant comme co-domaine Motors (axiome (13)) ; hasaccessHighway est une fonction partielle (axiome (15)), ayant comme domaine Vehicles (axiome (16)), comme co-domaine Highway (axiome (17)).

On suppose que le vehicule immatriculé 114-AA est sur l'autoroute $A1$. Le domaine d'interprétation $\Delta^{\mathcal{I}}$ est défini par la A-Box comme suit :

$$
\begin{aligned}
\text{114-}AA &: Vehicles\\
4w &: Wheels\\
V8 &: Motors\\
A1 &: Highway\\
personX &: Owners\\
(\text{114-}AA, 4w) &: hasWheels\\
(\text{114-}AA, V8) &: hasMotor\\
(\text{114-}AA, A1) &: hasaccessHighway
\end{aligned}
$$

FIGURE 1.2 – Exemple de A-Box

1.5.1.5 Quelles vérifications ?

Les raisonnements qu'on peut appliquer par rapport à une Base de Connaissances Σ composée d'une T-Box et d'une A-Box ($\Sigma = (T, A)$) sont les suivants :

- **La consistance :** une base de connaissances est consistante si elle admet au moins un modèle. Un concept C est dit satisfiable en respect avec la base Σ, s'il existe un modèle $\mathcal{I}$ de Σ avec $C^{\mathcal{I}} \neq \varnothing$;
- **La classification :** un concept C_i est subsumé par un concept C_j dans une base Σ, noté par $\Sigma \models C_i \sqsubseteq C_j$, si pour modèle $\mathcal{I}$ de Σ on a $C_i^{\mathcal{I}} \subseteq C_j^{\mathcal{I}}$;
- **L'instanciation :** un objet o est instance d'un concept C, noté $\Sigma \models o : C$, si pour tout modèle $\mathcal{I}$ de Σ on a $o^{\mathcal{I}} \in C^{\mathcal{I}}$ [108] ;
- **L'équivalence :** Deux concepts C_i et C_j sont dits équivalents en respect avec la base de connaissances Σ , noté par $\Sigma \models C_i \equiv C_j$, si ils se subsument mutuellement en respect avec Σ.

1.5.2 Logiques pour la spécification en vérification et validation

Si les logiques utilisées pour la représentation des connaissances cherchent à réduire les complexités pour trouver des mécanismes de raisonnements décidables et une représentation sans ambiguïtés de la connaissance, elles ne permettent cependant pas de gérer les problèmes exposés en vérification et validation. Ces dernières cherchent en plus des modèles formels d'exprimer des propriétés liées au temps, telles que la vivacité en plus de la sûreté. L'explicitation de contraintes liées au temps est indispensable pour garantir en terme de temps de réponses des services efficaces qui assurent une cohérence temporelle. Il existe de nombreux formalismes pour exprimer des propriétés liées au temps telles que les propriétés de vivacité. Ces logiques sont généralement des extensions de la logique propositionnelle aux opérateurs temporels qui décrivent les changements dans le temps. Citons parmi ces logiques : les logiques PLTL, CTL, TLA. Nous exposons dans ce qui suit une brève description de TLA [174].

1.5.2.1 Le formalisme TLA

Développée par Lamport [174], la logique TLA (pour Temporal Logic of Action en anglais) est une logique qui spécifie des systèmes réactifs. Ce sont des propriétés temporelles qui peuvent être exprimées et vérifiées avec cette logique. On peut citer la vivacité, l'équité, la fatalité etc. En TLA, les comportements sont exprimés par des formules qui illustrent l'ensemble des exécutions possibles du système étudié.

Le comportement d'un système est modélisé par des états (s) moyennant un ensemble infini dénombrable de variables *Var*, et un ensemble de valeurs *Val*. Ainsi, la sémantique de cette logique est définie en termes d'états. Un état est une affectation de valeurs aux variables, c'est-à-dire une fonction de l'ensemble *Var* de noms de variables vers la collection *Val* de valeurs. La fonction définit les états du système ($s \in Var \rightarrow Val$). Le comportement du système est une suite infinie $s_0, s_1, ..., s_i, ...$, d'états qui est définie dans l'ensemble de tous les états possibles noté S. Une fonction d'état f définie de l'ensemble des états possibles S vers l'ensemble des valeurs *Val*, à laquelle la valeur qui lui est associée à l'état s notée $s[\![f]\!]$ est définie sémantiquement comme suit :

$$s[\![f]\!] \triangleq f(\forall\ `v' : s[\![v]\!]/v)$$

où : $f(\forall$ 'v' : $s[\![v]\!]/v)$ est la valeur obtenue de f par substitution de $s[\![v]\!]$ pour v pour toutes les variables v. Une variable x est une fonction d'état, une fonction d'état qui assigne la valeur $s[\![x]\!]$ à l'état s. La définition de $[\![f]\!]$ pour la fonction d'état f étend donc la définition de $[\![x]\!]$ pour la variable x.

L'expression booléenne qu'on peut construire à partir de variables et constantes est un prédicat P. $s[\![P]\!]$ est la valeur obtenue lorsqu'on remplace les variables x dans le prédicat P par leurs valeurs $s[\![x]\!]$ à l'état s. On dit qu'un état s satisfait un prédicat P, ssi, $s[\![P]\!]$ est vraie. Les fonctions d'état correspondent à la fois aux expressions dans les langages de programmation ordinaires et aux sous-expressions des assertions utilisées dans la vérification de programme ordinaire.

Les actions définissent des expressions booléennes qui contiennent des variables non-primées, primées et des symboles de constantes. Les actions définissent des relations entre des paires d'états successifs entre un état ancien et un nouvel état. Les anciens états sont exprimés par des variables non-primées et des constantes, alors que les nouveaux états sont exprimés par des variables primées et des constantes. Soient la paire d'états consécutifs $\langle s, t\rangle$; les variables x ; la valeur des variables à état s $s[\![x]\!]$ et la valeur des variables à l'état t $t[\![x]\!]$. L'action A est satisfaite par la paire $\langle s, t\rangle$, *ssi*, $s[\![t]\!]$ est vraie. $s[\![t]\!]$ est la valeur obtenue en remplaçant les variables non-primées v par leurs valeurs $s[\![v]\!]$ à l'état s et celles primées v' par les valeurs de x, $t[\![v]\!]$ à l'état t comme suit :

$$s[\![A]\!]t \triangleq A(\forall\text{'}v\text{'} : s[\![v]\!]/v, t[\![v]\!]/v')$$

Les actions peuvent être :

— $ENABLEDA$ qui définit un prédicat vrai dans un état s, si et seulement s'il est possible à partir de cet état s d'effectuer un pas moyennant l'action A. Ceci s'explique par l'existence d'un état t accessible à partir de s en exécutant l'action A :

$$s[\![ENABLEDA]\!] \triangleq \exists t \in \mathsf{S} : s[\![A]\!]t$$

— $\langle A_x\rangle$ qui consiste à changer les valeurs des variables x par l'action A par de nouvelles valeurs $x'.\langle A_x\rangle$:

$$\langle A\rangle_x \triangleq A \wedge (x' \neq x)$$

— $[A]_x$ cette notation exprime que soit l'action A est observée, soit il y a eu un bégaiement. Dans ce dernier cas, il n'y a aucun changement de valeurs des variables $x : x' = x$ qu'on note *unchanged* x

Deux opérateurs booléens sont utilisés pour exprimer des formules temporelles en TLA, à savoir, l'opérateur $\Box$ pour exprimer « toujours », et l'opérateur $\Diamond$ pour exprimer la fatalité.

Le comportement σ défini par la suite $\langle s_0, s_1, ...\rangle$ satisfait la formule temporelle F, si la valuation $\sigma[\![F]\!]$ est vraie, sachant que $[\![F]\!]$ définit une fonction de l'ensemble des comportements vers les booléens. La validité de composants d'une formule d'un comportement défini par rapport à $\sigma \triangleq \langle s_0, s_1, ...\rangle$ exprime la satisfaction d'une formule par un comportement comme l'indiquent les cas suivants :

1. $\sigma[\![F \wedge G]\!] \triangleq \sigma[\![F]\!] \wedge \sigma[\![G]\!]$
2. $\sigma[\![\neg F]\!] \triangleq \neg\sigma[\![F]\!]$
3. $\sigma[\![F \Rightarrow G]\!] \triangleq \sigma[\![F]\!] \Rightarrow \sigma[\![G]\!]$

4. $\Box F$ qui exprime « toujours la formule F » :

$$\langle s_0, s_1, ...\rangle [\![\Box F]\!] \triangleq \forall n \in \mathbb{N} : \langle s_n, s_{n+1}, ...\rangle [\![F]\!]$$

   $\langle s_0, s_1, ...\rangle [\![\Box F]\!]$ veut dire que la formule F est vraie pour toutes les suites qui viennent après s_0, s_1 ;

5. $\Diamond F$: Fatalement F : $\Diamond F \triangleq \neg\Box\neg F$:

$$\langle s_0, s_1, ...\rangle [\![\Diamond F]\!] \triangleq \exists n \in \mathbb{N} : \langle s_n, s_{n+1}, ...\rangle [\![F]\!]$$

   Le comportement $\langle s_0, s_1...\rangle$ satisfait $\Diamond F$ ssi, la formule F est vérifiée par une suite des états à partir de s_0, s_1.

6. $\Box\Diamond F$ qui veut dire : Infiniment souvent F :

$$\langle s_0, s_1, ...\rangle [\![\Box\Diamond F]\!] \triangleq \forall n \in \mathbb{N} : \exists m :\in \mathbb{N} : \langle s_{n+m}, s_{n+m+1}, ...\rangle [\![F]\!]$$

   La formule exprime que la suite $\langle s_0, s_1, ...\rangle$ satisfait la formule temporelle $\Box\Diamond F$ ssi, F est vérifiée par des préfixes infinis du comportement $\langle s_0, s_1, ...\rangle$.

Les actions et les prédicats sont aussi exprimés selon des cas.

La logique TLA exprime aussi des hypothèses d'équité sur le système étudié, des propriétés de sûreté, de vivacité, ...

- La sûreté est exprimée comme suit : $\Box P$: le prédicat P est toujours vrai.
- La vivacité est exprimée par : $\Diamond$*property* qui signifie que la propriété *property* arrivera fatalement.
- L'équité définit des contraintes sur les comportements infinis et peut être définie comme :
  - équité faible : $WF_x(A) \triangleq \Diamond\Box ENABLED\langle A\rangle_x \Rightarrow \Box\Diamond\langle A\rangle_x$ signifiant qu'à partir d'un état, si l'action A est toujours activable, alors elle sera activée infiniment souvent ;
  - équité forte : $SF_x(A) \triangleq \Box\Diamond ENABLED\langle A\rangle_x \Rightarrow \Box\Diamond\langle A\rangle_x$ signifiant que l'action A est infiniment souvent activée si elle est activable infiniment souvent.

Spécification TLA d'un système Un système est modélisé par une spécification en logique TLA par la formule :

$$Init \wedge \Box[Next]_x \wedge L$$

- *Init* : définit le prédicat d'initialisation des comportements du système ;
- *Next* : est la relation définie sur les paires d'états consécutifs, alors que $\Box[Next]_x$ la relation *Next* est satisfaite par les paires d'états consécutifs ou bien les variables x ne changent pas de valeurs ce qui correspond au bégaiement.
- L : exprime une conjonction d'hypothèses d'équité fortes et/ou faibles sur les comportements du système.

En TLA il existe un opérateur appelé *leads to*. Cet opérateur définit qu'une propriété P conduit à une autre propriété Q qu'on note par $P \rightsquigarrow Q$ et qu'on définit comme suit :

Définition 1.5.6 ($P \rightsquigarrow Q$) *Pour toute formule temporelle P et Q, la formule temporelle $P \rightsquigarrow Q$ est définie par : $P \rightsquigarrow Q \triangleq \Box(P \Rightarrow \Diamond Q)$, et signifie qu'à chaque fois que la propriété P est vérifiée, la propriété Q sera fatalement vérifiée.*

L'opérateur $\rightsquigarrow$ posséde des règles de vérification ayant des propriétés telles que la transitivité, la disjonction, la déduction, ... et définit les règles de preuve suivantes :

— **LATTICE** :

$$\frac{F \wedge c \in S \Rightarrow (H_c \rightsquigarrow (G \vee \exists d \in S.(c \succ d) \wedge H_d))}{F \Rightarrow ((\exists c \in S.H_c) \rightsquigarrow G)}$$

où $\succ$ définit un ordre partiel bien fondé sur l'ensemble $S(S, \succ)$.
Cette règle stipule que la formule F avec la valeur $c \in S$, et sous l'hypothèse qu'une occurence de H_c, conduit à soit G, soit à une nouvelle occurence de H, avec une valeur d dans S strictement inférieure à c ($c \succ d$). Avec la contrainte qui assure qu'aucune chaîne descendante infinie n'existe dans S (car ordre partiel bien fondé), alors G sera fatalement vérifiée ;

— **WF1** :

$$\begin{array}{ll} P \wedge [N]_x \Rightarrow (P' \vee Q') & (1) \\ P \wedge \langle N \wedge A \rangle_x \Rightarrow Q' & (2) \\ P \Rightarrow ENABLED \langle A \rangle_x & (3) \\ \hline \Box[N]_x \wedge WF_x(A) \Rightarrow (P \rightsquigarrow Q) & (4) \end{array}$$

Sachant que : P et Q sont des prédicats, $[N]_x$ est la relation définie entre deux états successifs, A est une action sur laquelle l'hypothèse d'équité faible est posée ($WF_x(A)$), la règle exprime qu'avec les hypothèses qui expriment que : (1) : le successeur d'un état qui satisfait P, satisfait soit P soit Q ; (2) : le successeur d'un état qui satisfait P, après activation de l'action $\langle A \rangle_x$, doit satisfaire Q et (3) : lorsque P est vrai, alors l'action $\langle A \rangle_x$ est activable : la conclusion qu'avec chaque pas incluant A est comme un pas $[N]_x$ et qu'une hypothèse d'équité faible est posée sur A alors on peut déduire que $P \rightsquigarrow Q$ (4).

— **SF1** :

$$\begin{array}{ll} P \wedge [N]_x \Rightarrow (P' \vee Q') & (1) \\ P \wedge \langle N \wedge A \rangle_x \Rightarrow Q' & (2) \\ \Box P \Box[N]_x \wedge \Box F \Rightarrow \Diamond ENABLED \langle A \rangle_x & (3) \\ \hline \Box[N]_x \wedge SF_x(A) \wedge \Box F \Rightarrow (P \rightsquigarrow Q) & (4) \end{array}$$

Les hypothèses (1) et (2) sont les mêmes que dans la règle précédente. En plus des deux hypothèses (1) et (2) et sous l'hypothèse (3) indiquant que si P est toujours vrai et que des pas $[N]_x$ sont activables, alors l'action $\langle A \rangle_x$ est fatalement activable. Et la conclusion dit que si chaque pas est considéré comme un pas $[N]_x$ et qu'une hypothèse forte est posée sur A alors on peut déduire $P \rightsquigarrow Q$.

L'auteur dans ses travaux [16] a défini un cadre basé sur la logique TLA pour le formalisme Event-B et qui sera abordé au chapitre 2.

1.5.3 Synthèse

La particularité des logiques de description est de pouvoir correspondre à un sous-ensemble de la logique des prédicats [55], sans faire apparaître la moindre variable, mais tout en gardant des quantifications. Le principe est que les atomes de la logique dénotent des prédicats unaires (*concepts*) et binaires (*rôles*), et qu'une sémantique des modèles est définie de façon analogue à celle des logiques modales. La relation entre la logique du premier ordre et les logiques de description est donnée en annexe A. Lorsqu'une approche se base sur la logique, le langage de représentation est habituellement une variante du

calcul des prédicats du premier ordre, et le raisonnement revient à vérifier la conséquence logique. Il est donc possible de traduire une base de connaissances en logique classique en associant un prédicat unaire à chaque concept, un prédicat binaire à chaque rôle et des règles d'inférence pour la subsomption. De plus, ces logiques offrent des constructeurs supplémentaires tels que $\neg$, $\sqcup$, ajouté à cela leur style déclaratif. Par ailleurs, la sémantique *définitionnelle*, et non descriptive permet une classification des concepts et une instanciation automatique des individus.

Habituellement, les logiques dénotent une valeur de vérité, alors qu'il est préférable pour la conception d'un système d'avoir une description ayant des objets correspondant à la spécialisation d'autres objets représentés sous forme d'une relation de spécialisation/-généralisation plutôt qu'une implication logique [185]. Ce constat est réalisé en recherche d'information, mais qui peut être appliqué pour la conception des systèmes car il est plus facile de reconnaître des propriétés que de les décrire en particulier en utilisant le formalisme Event-B comme nous le verrons dans les chapitres qui suivent. L'interprétation par des ensembles d'individus plutôt que par des valeurs de vérité a aussi l'avantage d'être explicite pour permettre de donner une sémantique bien définie aux objets décrits, car la conception d'un système suit plusieurs points de vue. De plus, les connaissances en logiques de description peuvent être complètes, et fidèles à la réalité (correction) et consistante ; les règles d'inférence sont bien fondées et compréhensibles. Le maintien des connaissances au cours du processus de développement logiciel implique l'acquisition de nouvelles connaissances. Ces logiques sont plus adaptées au maintien des spécifications et disposent de mécanismes pour soutenir les différentes activités du génie logiciel. La difficulté consiste alors à trouver un équilibre entre expressivité et pouvoir d'inférence pour réduire la complexité. Enfin, elles offrent un maintien de la consistance dans le processus de construction des configurations.

En contre-partie, les logiques utilisées pour la représentation des connaissances, (les logiques de description) décrivent les connaissances d'un domaine indépendamment du temps et des changements qui peuvent affecter les objets décrits [4]. Les travaux sur la représentation des connaissances montrent qu'une action est aussi définie par un concept. Celui-ci est considéré comme *intentionnel* [143]. Par exemple, le concept intentionnel « poser »pour exprimer l'action de « poser quelque chose ». Mais, les logiques pour représenter et raisonner sur les connaissances ne permettent pas d'exprimer le changement d'états car le raisonnement sur les axiomes définissant les contraintes est monotone. En effet, l'ajout de nouvelles connaissances ne fera qu'augmenter les déductions dans les raisonnements, et les connaissances initiales ne peuvent être modifiées ou enlevées. L'ajout de nouvelles règles dans une base de connaissances ne fait qu'étendre les conclusions, i.e., n'infirme pas les conclusions déjà obtenues, ce qui permet de conclure que la représentation de l'évolution des états d'un système n'est pas permise. Les logiques temporelles expriment les changements au moyen de traces telles que illustré par la logique TLA ci-dessus. Les logiques de description ne permettent pas d'exprimer des propriétés liées au temps telles que la vivacité, l'équité, ... Ces logiques ne raisonnent pas sur les traces et ne disposent pas d'opérateurs temporels. Par ailleurs, le raisonnement ontologique en logique de description est réalisé sur l'ensemble du modèle de l'ontologie (tous les concepts et toutes leurs propriétés sémantiques) et ne peut être effectué sur un séquencement particulier de règles.

4. Bien qu'il existe des variantes qui permettent de raisonner sur les actions et le temps telles que [21], mais celles-ci ne sont pas abordées dans cette thèse car les raisonneurs ontologiques construits sont basés dans leur majorité sur des variantes des logiques de description décrites dans le présent document.

1.6 Ontologies pour la conception de systèmes

La conception de système fournit des réponses sur la question qui consiste à savoir ce qui a été conçu comme produit final, mais ne donne pas de réponse sur pourquoi le produit a été tel qui l'est pour savoir pourquoi certaines solutions ont été rejetées ou s'il était possible de considérer d'autres solutions, connaître quelles justifications ont été apportées pour rejeter ou adopter telles ou telles solutions, et si les exigences spécifiées ont été rempliées ou pas par la solution de conception finale, ... Répondre à de telles interrogations permettrait de comprendre en profondeur la solution pour pouvoir la maintenir et l'enrichir. La croissance en permanence de la complexité des systèmes à concevoir conduit les entreprises à rendre les connaissances plus accessibles au public visé. Citons à titre indicatif le domaine de l'aérospatial. L'activité de vérification et de validation nécessite souvent plus d'information que celle contenue dans le seul modèle pour avoir une preuve complète, ce qui nécessite de comprendre et de décrire cette information supplémentaire pour ouvrir le champs de vision d'un concepteur [74].

Pour toutes ces raisons le partage de la connaissance entre ingénieurs et clients est une nécessité qui améliore les justifications et les explications des choix de conception. Cette justification renvoie à une description d'une solution qui inclut des détails de raisonnements et des justifications des décisions ou choix de conception. Ces détails doivent incarner les raisons pour lesquelles le système a été conçu de la sorte. Cette justification définit une explication de la raison pour laquelle un design a été adopté. Ceci est particulièrement important dans le cas de systèmes qui doivent être certifiés, comme les systèmes de vote étudiés dans cette thèse, puisque par définition la sûreté de fonctionnement [175] d'un système doit établir des propriétés qui permettent de placer une *confiance justifiée* dans les services fournis par ce système. En effet, la certification consiste à s'assurer que le modèle respecte une norme et peut servir de base à l'établissement d'un référentiel réutilisable et générique à un domaine. Les ontologies et plus généralement l'ingénierie des connaissances permettent ce type de raisonnement. Elles ont été appliquées dans différents domaines tels que les avions [52, 90, 131], l'aérospatial [171], la sécurité [182, 236], le ferroviaire [249, 250], et dans bien d'autres domaines.

Dans les domaines tels que l'aérospatial, et dans ce cas on peut citer parmi les projets qui ont été mis en place et qui ont adopté ces approches le projet CRYSTAL [5], les ontologies peuvent aider à l'amélioration de la qualité de création des exigences grâce à l'utilisation de langages contrôlés et de modèles variés. À ce niveau, les ontologies peuvent traiter de la description des modèles, des relations axiomatiques entre les variables de modèle et de la ressource lexicale à utiliser en connexion. Elles permettent également de trouver la manière d'homogénéiser les outils de gestion des exigences et les bases de données grâce à la fourniture de modèles conceptuels standardisés pour la gestion des exigences qui, à leur tour permettront la définition de format d'échange (INCOSE, OMG, etc.).

Les méthodes d'ingénierie conventionnelles sont basées sur des descriptions textuelles des exigences du système. Un grand nombre de documents interconnectés décrivant un énorme volume d'exigences exprimées dans un langage naturel sont produits dans le processus de conception. Avec l'introduction de l'ingénierie des exigences, les modèles de ges-

5. pour CRitical SYSTem Engineering AcceLeration. Voir le lien des livrables sur : `http://www.crystal-artemis.eu/`. Dans ce projet, la gestion des besoins et la vérification et validation englobent plusieurs cas d'utilisation du projet CRYSTAL pour permettre la traçabilité entre les exigences et les modèles ainsi que le soutien de la vérification et la validation de la conception par rapport aux exigences.

tion des exigences ont été définis et mis en œuvre dans de nombreux outils. De nombreuses vérifications de cohérence peuvent être automatisées en utilisant les techniques issues du web sémantique par exemple [193]. Ceci peut être réalisé en établissant des normes pour l'échange de données qui définissent des attributs d'objets d'exigences modifiables à titre indicatif, on peut citer : titre, description, type, URL, statut ... pour structurer les besoins, et pouvoir gérer des versions ainsi que des configurations différentes du projet en question [6].

De nombreux travaux (citons à titre indicatif [10, 71, 111, 162, 227, 258, 264]) dans la littérature se sont penchés sur cette question et intéressés à l'utilisation des ontologies pour la conception des systèmes. Les ontologies peuvent intervenir à des niveaux différents dans la conception de systèmes, pour l'élicitation des exigences [162], pour l'extraction et la traçabilité des exigences issues de cahier de charges [227], pour la modélisation et la simulation, et parmi les auteurs qui travaillent sur cet aspect, on peut citer Giancarlo Guizzardi, et parmi ses travaux, on peut citer [111, 143]. Ces travaux pour leur grande partie, interviennent en général au niveau des exigences écrites (textuelles) ou bien en utilisant des modèles semi-formels tels que UML et nous ne pouvons tous les décrire.

Des travaux ont été réalisés pour transformer des ontologies vers des contextes Event-B, citons par exemple [7] [12, 194]. Ces travaux traduisent des ontologies en contextes Event-B, soit en passant par un langage intermédiaire tel que [12], soit en appliquant une traduction directe tel que [194]. Les travaux tels que [194] favorisent une dérivation aisée sinon intuitive des éléments définis dans la composante terminologique d'une ontologie (T-Box) sans pour autant exploiter les mécanismes de raisonnement ontologique et le raffinement en Event-B. De plus, ces travaux ne se basent pas sur la sémantique des formalismes cibles. Cependant, ils permettent de dériver un langage pivot qui est succeptible d'exploiter des ontologies de type OWL ou PLIB. En effet, le travail réalisé dans [194] concerne le développement d'un outil qui permet de générer des contextes Event-B à partir d'ontologies PLIB ou OWL. Les auteurs dans ces travaux définissent un modèle pivot fédérant les langages des ontologies OWL et ou des ontologies PLIB. L'outil est intégré dans Rodin [4, 6] et donne la possibilité d'utiliser deux approches *Shallow* et *Deep*. La première approche encode les concepts d'ontologies directement en tant qu'éléments de contexte dans Event-B, alors que la seconde approche utilise la modélisation profonde. Dans ce dernier cas, les concepts génériques des ontologies sont formalisés dans un premier contexte Event-B, et les concepts spécifiques des ontologies sont définis dans un second contexte Event-B comme des instances spécifiques des contextes génériques. La subsomption entre sous-concepts et concepts est définie comme l'inclusion entre constantes Event-B. Cette approche, par son modèle pivot, permet de récupérer des propriétés après avoir traduit l'ontologie en contexte Event-B. C'est donc à l'utilisateur d'aller chercher manuellement les résultats de la traduction pour les exploiter. Ces travaux ne donnent pas de précisions sur la manière dont les propriétés du domaine sont exploitées, ne distinguent pas le contexte et ne montrent pas comment exploiter par raffinement les contraintes ainsi que la sémantique du domaine pour faire des preuves pour la vérification et la validation de modèles en Event-B.

Sadoun [227] dans sa thèse a présenté des travaux significatifs qui traitent les deux phases, à savoir, l'extraction à partir de texte, puis la génération d'un modèle formel. Ces travaux proposent de définir une ontologie comme un modèle de représentation pivot,

6. pour plus de détails consulter les livrables du projet CRitical SYSTem Engineering AcceLeration. sur : `http://www.crystal-artemis.eu/`

7. Les travaux des auteurs [194] ont été réalisés dans le cadre du projet IMPEX.

ayant pour but de faciliter le passage de spécifications en langage naturel vers des spécifications formelles. Ce travail se focalise sur l'identifcation des exigences en se basant sur les deux composantes d'une ontologie (T-Box et A-Box) et en analysant les besoins dans les spécifications écrites en langage naturel, puis en définissant un ensemble de notations et de contraintes. Pour guider cette extraction à partir de texte, l'auteur fait recours à l'identification des instances des propriétés (instances de rôles) sémantiques pour peupler l'ontologie construite préalablement permettant ainsi à l'utilisateur de configuer le comportement d'un système à l'aide de descriptions écrites en langage naturel. Le peuplement d'une ontologie consiste à y ajouter un ensemble d'instances de concepts et de propriétés. L'auteur choisit d'utiliser le langage OWL-DL, une version décidable de OWL pour définir l'ontologie utilisée comme lien entre spécifications en langage naturel et formelles. Ce choix est justifié non seulement, par la popularité du langage, mais aussi parce qu'il favorise la création de propriétés et des annotations nécessaires à une représentation efficace et compréhensible des connaissances à partir des textes pour un peuplement automatique d'ontologie. Par ailleurs, OWL permet de maintenir le lien entre les éléments du texte et leur représentation sémantique. Des unités linguistiques peuvent ainsi être liées aux instances de l'ontologie auxquelles elles réfèrent. Cette première étape est donc une association par intégration entre texte écrit en langage naturel et ontologie OWL.

Cette approche possède plusieurs avantages :

— Premièrement, elle permet de faire appel aux mécanismes d'inférence de l'ontologie pour classer et identifer de manière univoque les individus qui participent aux instances de propriétés extraites des spécifications textuelles écrites en langage naturel.

— Deuxièmement, elle possède la capacité de reconnaître non seulement des mentions d'instances de propriétés sémantiques dans les textes, mais précise également chaque type d'instances de propriétés. Celles-ci sont définies entre un domaine et une image qui représentent les deux ensembles d'individus pouvant être liés par la propriété pour reconnaître le contexte qui dénote la présence d'une instance de propriété modélisée dans l'ontologie. Cela suppose l'inclusion des connaissances qui caractérisent cette propriété dans la définition de son contexte d'apparition dans les textes en lui donnant accès à un ensemble de connaissances contextuelles suffisantes et formelles. Une analyse syntaxique du corpus d'apprentissage donne lieu à la création d'un arbre de dépendances syntaxiques. L'obtention des termes qui dénotent chaque propriété de l'ontologie, des ensembles de termes dénotant ses domaine et image est acquise à partir de la termino-ontologie SKOS [8] (pour *Simple Knowledge Organization System* en anglais). Une fois l'étape de reconnaissance achevée, l'étape de classification peut commencer. Cette étape consiste à classer les individus dans leur classe sémantique (concept ontologique) en exploitant les propriétés définissantes des concepts et des individus. Les vérifications réalisées sont celles d'applicabilité des règles définies dans l'ontologie, de cohérence et d'appartenance des individus à leur classe sémantique.

— Puisque le raisonnement ontologique est monotone et ne permet donc pas de réaliser toutes les vérifications souhaitées et qui concernent le changement d'états, l'auteur a proposé de traduire le résultat en spécifications *Maude* [79] (langage basé sur la logique de réécriture) qui est bien adapté à la vérification de systèmes à base de règles concurrentes privilégiant ainsi les techniques de vérification basées sur le *model-checking* et garantissant une association par dérivation. Cette association exploite les aspects orientés objet du langage Maude pour faire des transformations

8. `http://www.w3.org/TR/skos-reference/`

nécessaires du résultat de l'ontologie peuplée. Un autre avantage de cette approche est qu'elle ne traduit pas l'ensemble de l'ontologie en spécifications Maude, mais seulement de dériver du modèle OWL des spécifications suffisantes à la vérification du comportement dynamique du système modélisé et défini par des règles utilisateurs identifiées à partir de l'analyse des spécifications d'exigences. Ces règles sont représentées par des individus du concept *Règle-utilisateur*.

1.7 Vérification et validation pour les ontologies

Très peu de travaux qui abordent la représentation des connaissance en vérification et validation de systèmes existent dans la littérature. Citons à titre indicatif les travaux [117, 187, 191, 215, 216, 240]. Ces travaux pour leur grande partie, analysent et discutent de l'applicabilité les techniques utilisées en génie logiciel comme la modélisation, le test, la vérification et la validation pour fiabiliser les systèmes intelligents experts. À titre indicatif, nous pouvons citer très brièvement quelques travaux significatifs dans cet axe :

Sur le plan test, des outils tels que Temporal Rover [110] ont été instrumentés pour des programmes pour exécuter des fragments de code insérés en fonction de conditions complexes exprimées sous forme de formules logiques temporelles. De nouveaux algorithmes peuvent détecter des modèles de programmation concurrents suspects susceptibles de provoquer une erreur, même si aucune erreur ne se produit sur la trace observée.

Dans [22], la surveillance de l'exécution est appliquée (conjointement avec la génération automatisée de cas de test) pour vérifier le contrôleur du mobile planétaire K9. Le contrôleur est un grand programme multithread qui contrôle le mobile en fonction d'un plan flexible généré par un programme de planification. Chaque cas de test correspond à un plan, pour lequel un ensemble de propriétés temporelles sont dérivées, selon la sémantique du plan. Le système est ensuite utilisé pour surveiller ces propriétés. Ce système est entièrement automatisé et a dévoilé une faille dans l'interprétation du plan, qui s'est effectivement produite lors des essais sur le terrain avant d'être fixée dans le contrôleur. Un blocage potentiel et une course de données ont également été découverts.

Les techniques de vérifications telles que Spin ont été utilisées dans des projets de la NASA pour vérifier certaines parties de l'exécutif RAX [146]. De plus, les démonstrateurs de théorèmes ont été aussi appliqués dans de nombreuses applications de la NASA [88]. Toutefois, l'auteur [187] indique que ces démonstrateurs nécessitent beaucoup d'efforts et de compétences de la part de leurs utilisateurs pour piloter la preuve, ce qui les rend aptes à l'analyse de conceptions à petite échelle par des experts en vérification seulement.

1.8 Synthèse

En résumé, les ontologies permettent : par leur consensus, leur partage et leur capacité de raisonnement explicite et systématique, de gérer les variabilités et la traçabilité des exigences diverses ; par leur capacité de justifications de l'existant, de justifier les choix et les raisons d'être des concepts adoptés pour la conception des systèmes, car la certification de systèmes critiques impose la justification des choix des conceptions. En effet, à partir du moment où l'obtention d'une conception rationnelle est la principale approche adoptée pour le traitement de la problématique, la compréhension de la *raison d'être* d'un concept

en tant que tel et ses relations avec d'autres concepts est la première étape à suivre durant ce processus de conception et de compréhension. La compréhension d'une telle conception a recours aux explications qui font appel aux justifications du contexte des théories philosophiques de l'explication et du sens ainsi que la prise de décision qui offre une certaine justification [237, 238].

En outre, la représentation de la connaissance est fondamentalement un substitut à la chose elle-même [93], utilisée pour permettre à une entité de déterminer les conséquences en *pensant* plutôt qu'en *agissant*, c'est-à-dire en *raisonnant* sur le monde plutôt que d'« *agir* » en lui. En effet, le raisonnement est un processus qui se déroule en interne, mais la plupart des choses sur lesquelles on raisonne n'existent qu'en externe. Un programme (ou une personne) participant à la planification de l'assemblage d'un vélo, par exemple, pourrait devoir raisonner sur des entités telles que des roues, des chaînes, des pignons et des guidons, mais de telles choses n'existent que dans le monde extérieur. Cette inévitable dichotomie est une justification et un rôle fondamentaux pour une représentation : elle fonctionne comme un substitut à l'intérieur du raisonnement, un substitut aux choses qui existent dans le monde. Les opérations sur et avec des représentations se substituent aux opérations sur la chose réelle, c'est-à-dire qu'elles se substituent à une interaction directe avec le monde. Dans cette optique, le raisonnement lui-même est en partie un substitut de l'action dans le monde où nous ne pouvons pas ou ne voulons pas (encore) prendre cette mesure. Examiner des représentations en tant que substituts conduit naturellement à deux questions importantes. La première question à propos de toute substitution est son *identité* prévue : à quoi sert-elle de substitut ? Il doit y avoir une forme de correspondance spécifiée entre le substitut et son *référent* prévu dans le monde ; la correspondance est la sémantique de la représentation.

De plus, comme nous l'avons expliqué, les logiques de description ayant une sémantique bien définie, sont aussi un bon moyen qui permet de décrire de manière non-ambigüe et de raisonner sur les connaissances d'un domaine. Une distinction importante [237] entre la logique des prédicats et les ontologies est qu'en logique bien que la quantification existentielle ($\exists$) soit une notation pour affirmer que quelque chose existe, mais la logique en soi n'a pas de vocabulaire pour décrire la chose qui existe. L'ontologie apporte des réponses à ce sujet, puisqu'une ontologie est l'étude de l'existence, de toutes les sortes d'entités, abstraites et concrètes, qui composent le monde. Une ontologie a pour source l'*observation* et le *raisonnement*, l'observation fournit des connaissances du monde physique et le raisonnement donne du sens aux observations en générant un cadre d'abstraction qu'on appelle *métaphysique*.

Notons enfin, que l'aspect descriptif en ingénierie des connaissances est plus favorisé pour la compréhension des spécifications et des conceptions en génie logiciel [187]. En revanche, le raisonnement ontologique est monotone et ne permet pas d'exprimer le changement d'états et des propriétés liées au temps.

1.8.1 Lien entre domaine, conception ou modélisation d'un système

Les travaux dans la littérature ont montré les liens explicites qui peuvent exister entre le domaine d'application, le système (ou le modèle) et les besoins des utilisateurs (ou les

	Ontologies	**Event B**
Représentations	Mondes ouverts ou fermés	Systèmes fermés
Objectif	Description	Prescription
Approches	Déclaratives	Constructives
Aspect statique/dynamique	Pas de séparation	Prise en compte (context et machine)
Sémantique formelle	Raisonnement/capacité d'inférence (démonstration et justifications)	Obligations de preuve
Référencement	Oui (explicite)	Non(implicite)
Formalismes	Logiques DLs , F-logic, FOL-DC ... Nombreux (PLIB, OWL, graphes conceptuels...)	Logique FOL et théorie des ensembles. Event B
Intégration de besoins	Génréralisation/Spécialisation, a priori/a posteriori	Abstraction/raffinement
Types de propriétés	Propriétés de bon sens.	Propriétés de Sûreté, d'invariance ...
Inférence vs OPs	Capacité à inférer de nouveaux faits. Raisonnement monotone. Pas de capacité d'exprimer le chagement d'état car les règles du raisonneur s'exécutent sans ordre d'enchainement. L'ajout de nouvelles connaissances ne modifie pas les anciennes. Connaissances initiales explicitées pour le raisonneur	Validation par animation (model checking). Capacité à exprimer le chagement d'état via des prédicats *BA* dans le système, il existe un ordre d'exécution des événements (ordre implicite). Modification possible car les événements peuvent supprimer des valeurs des variables. Pas précisées (implicites).
Contextes	Domaine : famille de contextes	Logiques classiques : indépendances de tout contexte
Interprétations du contexte	Ensemble d'individus DLs, ACL, ...	Espaces de valeurs infinies et affirmations par des valeurs de vérité

FIGURE 1.3 – Bilan comparatif entre ontologies et modélisation pour la V&V

exigences). On parle alors de relation ternaire [46, 112, 154] composée du domaine, des besoins de l'utilisateur et des spécifications et ou du modèle ou du système selon la phase dans laquelle se situe la conception. Au niveau système, le développement progressif d'un logiciel doit toujours commencer par la description du domaine ; se poursuivre par une étape d'évolution des exigences ; se terminer par une conception du logiciel tel que le la relation triptyque (*triptych* en anglais) : $\mathcal{D}$, $\mathcal{S} \models \mathcal{R}$ soit valide [46, 154]. La correction du logiciel peut ainsi être prouvée vis-à-vis des exigences qui s'appuient sur le domaine tel que décrit.

Le même constat est réalisé du point de vue modélisation [112, 154], et le problème est aussi abordé comme un problème de recherche d'un modèle devant satisfaire implicitement, non seulement les propriétés du domaine de connaissances de l'ingénierie système, mais aussi les propriétés et les exigences exprimées par les utilisateurs. Cette analyse des propriétés est réalisée à partir de ces connaissances par le concepteur : *Propriétés du domaine* $\wedge$ *Spécifications* $\Rightarrow$ *Besoins exprimés*. Premièrement, la vérification du modèle formelle requiert l'expression formelle des besoins par les propriétés liées au domaine. L'étape de validation du modèle consiste alors à s'assurer que le modèle réalise correctement les fonctions pour lesquelles il a été créé dans les diverses configurations et situations. Cette description repose sur l'observation du comportement du système sous étude tel qu'il *est* c-à-d : avec ses comportements nominaux et ceux défaillants.

Nous transposons donc cette démarche en vérification et validation à base de modèles suivant l'hypothèse sémiotique [256] comme suit : le système observé est un ensemble de phénomènes, d'éléments (concrets ou abstraits, tangibles ou intangibles) perçus par un observateur. Ces éléments sont désignés sous le terme général de signes, et le système observé est un système de signes. Le système est alors observé selon trois dimensions (cf. figure 1.4) :

— *syntaxique* : correspond à la représentation du système (modèle, propriétés, contexte d'utilisation). L'aspect syntaxique est lié au modèle faisant l'objet de l'étude. Elle définit la structure de représentation employée. On parlera de référent (cf. définition 1.8.1). Dans notre cas, il s'agit des modèles Event-B ;

— *sémantique* : le sens ou la signification donnée au modèle ainsi qu'à ses propriétés. L'aspect sémantique est défini par une ontologie de domaine qui décrit ce que le modèle *est*. L'ontologie donne la structure du système (un modèle dans notre cas), comme *étant quelque chose* ;
— enfin, l'aspect *pragmatique* : qui définit la relation entre le modèle, les propriétés ainsi que ses utilisations. La dimension pragmatique est liée au *connaissances sur le contexte* du modèle, qui correspond, selon qu'il s'agisse de vérification ou de validation, au raffinement de modèles Event-B ou niveau d'abstraction, ou aux données et aux configurations dynamiques. Cette dimension est attachée à *ce que le modèle fait*, et par conséquent, projette le système dans sa *fonction* et prescrit ses comportements ;

Définition 1.8.1 *Un référent est l'objet de l'étude. Il peut s'agir d'un modèle ou d'un ensemble de modèles élaborés selon une approche de modélisation multi vues, multi niveaux de détail et multi formalismes d'un même système [74].*

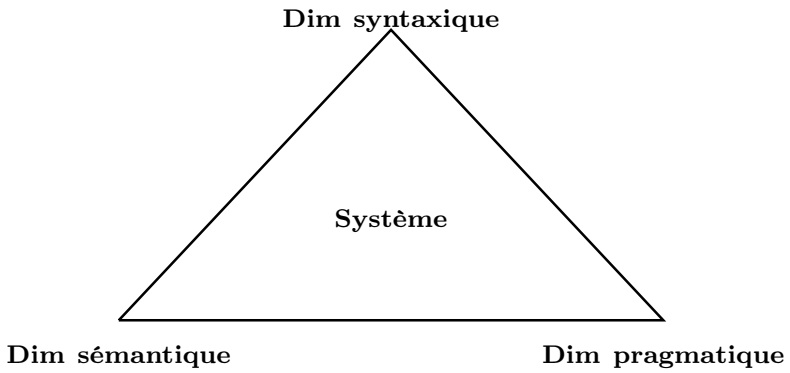

FIGURE 1.4 – Définition d'un système selon l'hypothèse sémiotique

Mais, un modèle doit être revalorisé pour juger de sa pertinence [74]. La pertinence nécessite donc de s'assurer que le modèle répond aux objectifs du concepteur (*complétude* vs. *intérêt*) pour être manipulable sans introduire d'ambiguïté ou de biais (*complétude* vs. *fidélité*). C'est un but à la fois de vérification de la construction et de validation de chaque modèle. La vérification en utilisant le formalisme Event-B est effectuée par raffinement de modèles. Pour s'assurer de la complétude (consistance) d'un modèle ce mécanisme génère des obligations de preuve. La pertinence est alors jugée par chaque niveau d'abstraction et l'intérêt est de tenir compte non seulement, des propriétés exprimées à chaque niveau de raffinement définissant chacun une vue, mais aussi de l'inclusion des traces d'exécution. Juger la pertinence de la validation consiste à s'assurer de la pertinence de chacun de ces modèles définis par raffinement et chacune de ces vues vis-à-vis de la réalité (est ce que le modèle est fidèle). Cette étape permet de juger de la cohérence dans une situation ou une configuration donnée. Juger la pertinence est un aspect lié au *degré de granularité* qu'à la connaissance du domaine d'application du système étudié. Ce grain de connaissance permet aux concepteurs de les guider pour savoir ce qu'ils doivent réellement représenter de manière judicieuse pour leur fournir des frontières et des cadres par des niveaux de détail et cela en suivant des hypothèses établies selon l'expertise du domaine.

1.9 Contextualisation et dépendance

1.9.1 Pourquoi le contexte ?

L'importance de la notion de *"contexte"* a été largement reconnue dans la littérature et les différentes disciplines de recherches témoignent de cet intérêt. Cette notion a été étudiée en représentation des connaissances (cf [54, 177]), le traitement automatique de la langue naturelle (cf [72, 53]), le web sémantique (cf [116]), la sécurité (cf [196, 178, 229, 40, 64]), etc. Nous n'abordons ici que les travaux qui nous servent dans le cadre de la vérification et la validation qui s'appuient sur les modèles pour la preuve. Nous exposons en particulier l'intérêt à considérer le contexte dans les phases de conception d'un système et les différentes représentations existantes en particulier celles qui se fondent sur la logique. Pour un état de l'art complet sur le contexte consulter [28, 92], ainsi que tous les travaux de Brézillon (citons par exemple [62, 60, 63, 61]).

Le besoin d'étudier le contexte s'est fait ressentir dans plusieurs domaines. Citons la sécurité par exemple, où l'intégration des connaissances contenues dans les patrons d'attaques accompagnée des connaissances limites liées aux vulnérabilités du système cible et des menaces potentielles dépend fortement du contexte de son application [122]. Le contexte est utilisé pour évaluer l'impact et la *plausibilité* de ces attaques. La fiabilité d'un système repose sur une approche systématique des exigences et des stratégies de rétablissement [169], en identifiant, détectant, et corrigeant les risques et les menaces, et en développant des mécanismes sécurisés tout en aidant les concepteurs à choisir les contre-mesures appropriées pour réduire les attaques. Ainsi, la validation des hypothèses faites par les concepteurs est réalisée sur la modélisation des menaces associées à leurs informations contextuelles pour protéger le système de modifications non autorisées des données ou de divulgation d'informations. Le contexte constitue donc un élément déterminant dans le choix des patrons de sécurité à appliquer.

Au niveau conceptuel, les techniques fondées sur l'abstraction [200] facilitent la conception de systèmes, en donnant des vues plus ou moins précises de ces systèmes. En effet, un modèle conceptuel doit intégrer l'intention du concepteur et donner une vue claire correcte et complète par une sémantique non ambigüe. La contextualisation met précisément l'accent sur les détails de chaque application particulière, ainsi que sur le processus de modélisation lui-même. Ces détails définissent un contexte et constituent l'identité unique de chaque tâche de modélisation. Le contexte définit donc le domaine du problème auquel l'utilisateur est confronté pour atteindre un objectif d'utilisation. Cette notion décrit la nature du problème ainsi que l'ensemble des entités concernées par une application dans la conception d'un système. La contextualisation est un mécanisme d'abstraction qui offre une séparation entre les données collectées. Elle permet de résoudre les problèmes liés aux différences de perception entre les différents acteurs dans le système favorisant ainsi l'organisation et la rationalisation des perspectives d'une même réalité.

En ingénierie des exigences c'est à Mullery [198] que revient l'utilisation de cette notion de vues. Ces travaux présentent des approches pour collecter des exigences en divisant le système en vues définies en fonction des parties prenantes du système. Ce principe définit que toutes les informations nécessaires pour la spécification d'un système ne peuvent être découvertes à partir d'une unique vue. Les vues sont définies pour identifier, organiser, modéliser et valider les exigences d'un système. Cette notion de vue a été utilisée dans l'ingénierie des besoins [205] comme un moyen pour séparer la perception qui pose problème

dans la modélisation.

L'utilisation du contexte dans les phases de développement en amont permet de situer dans l'espace et dans le temps les systèmes développés [241, 104]. Le contexte capture et permet une analyse des compromis pour décider comment les exigences personnelles doivent être mises en œuvre. Le contexte dans les phases en aval du processus de développement améliore la validation des systèmes produits [103, 224]. Une telle validation se détermine par la mesure de la relation entre la représentation du système réel et un modèle confronté à des cas d'utilisations prévues du modèle. Ces cas d'utilisation correspondent à des configurations et des scénarios.

Le contexte définit l'ensemble des configurations spécifiques qui font référence aux comportements attendus des acteurs de l'environnement vis-à-vis du système étudié [224]. Ces travaux définissent le contexte comme environnement et utilisent des automates de contexte spécifiques et intrusifs qui simulent des scénarios de comportements de l'environnement en vue de limiter l'espace d'exécution du modèle lors de la preuve. Ces automates sont composés avec le modèle à valider ce qui permet de restreindre l'ensemble des exécution du modèle. Pour répondre aux sollicitations de son environnement, un système doit interragir avec cet environnement à travers des actions. Les configurations spécifiques définies dans ces travaux [224] permettent d'analyser le comportement d'un système. La réduction par exploration de contextes consiste à réduire l'espace des états du système en indiquant les comportements de l'environnement avec lequel le système interagit pour guider le *model-checker* à concentrer ces efforts non plus sur l'exploration de l'automate global du système, mais plutôt sur une restriction pertinente de ce dernier pour la vérification des propriétés spécifiques.

Alors cas d'utilisation, scénarios ou configurations ? La littérature donne différentes visions de ces notions. Certains travaux considèrent les cas d'utilisation comme des représentations, alors que les scénarios sont des descriptions des étapes à suivre pour compléter un cas d'utilisation (pour plus de détails consulter [220]). Pour notre part, et pour les spécificités qu'a le formalisme Event-B, nous ne rentrerons pas dans ce débat. En effet, le formalisme Event-B est basé sur la description d'événements et les scénarios et les cas d'utilisation sont définis par des configurations lors de la validation des modèles. Celles-ci sont implicites. Dans le cas de la vérification, il s'agit d'exploiter les patrons de conception afin d'y appliquer des raffinements supplémentaires ou des instanciations pour un cas d'utilisation défini précis.

De tout ce qui précède, nous retenons qu'au niveau de la conception d'un système, le contexte correspond aux différentes vues définies pour constituer un système. Au niveau de la validation, le contexte se définit par des scénarios, des cas d'utilisation, des configurations qui restreignent l'espace des états. Il définit l'environnement avec lequel il interagit.

Sur le plan de preuve, le contexte est aussi défini par l'ensemble des hypothèses qui établissent des vérités admises constituant les prémisses des règles de déduction utilisées par exemple en déduction naturelle pour prouver la conclusion d'une règle.

1.9.2 Définition du contexte

De nombreuses définitions du contexte ont été données dans la littérature. Parmi celles qui reviennent dans tous les documents consultés, on trouve celle de l'auteur dans ses

travaux [101, 102]. Cette définition met en évidence toute information pouvant caractériser les entités utilisées (personnes, objets, endroits, etc.) selon une perception particulière. Néanmoins, lorsqu'on s'intéresse à un domaine particulier, cette perception ou facette ressort liée au domaine d'intérêt [43]. Elle se définit comme un moyen parmi un ensemble fini de moyens génériques de l'analyse d'un domaine [43, 44] : une vue du domaine, telles que les différentes facettes couvrent conceptuellement différents points de vue, et de telle sorte que ces points de vue, ensemble, couvrent le domaine. Un point de vue peut être défini comme un objet local qui encapsule des connaissances partielles du système et du domaine étudié [119].

De manière générale, il existe deux grands axes de recherche où le contexte a pris une grande place, l'un est basé sur la logique et trouve son origine dans les travaux de McCarthy [183], et l'autre s'appuie sur les ontologies, citons par exemple [259].

Le contexte a été étudié en intelligence artificielle depuis les années 1990, où McCarthy [183] introduisit cette notion comme une généralisation d'une collection d'hypothèses. Cette définition met en exergue une collection d'hypothèses. Dans ses travaux, McCarthy montre que cette notion implique des situations ou le temps. Parmi les définitions qui emploient également la notion d'hypothèse, on trouve la définition donnée dans [61], où les auteurs définissent le contexte comme étant *"un ensemble infini et partiellement connu d'hypothèses"*.

Brézillon considère le contexte comme un état des connaissances vis-à-vis d'un focus d'attention. Cette notion est toujours relative à quelque chose : le contexte d'un objet, le contexte d'une action, le contexte d'une interaction [63]. Par ailleurs, l'auteur [61] distingue données, informations, et connaissances. Les données sont des symboles perçus par un observateur au travers de capteurs, desquelles émergent des informations qui sont des données avec un fort contenu sémantique. L'association du contenu sémantique aux données donne des connaissances. Et les connaissances sont de deux types : les connaissances explicites et les connaissances tacites. Les premières sont facilement formalisables et donc communicables alors que les dernières sont hautement personnelles.

L'analyse de l'auteur [28] a montré qu'il n'existe pas de consensus autour de cette notion, que le contexte est intimement lié à la notion de situation [11, 109]. Celle-ci représente un ensemble de faits physiques, localisation spatiale, temporelle, ou fonctionnelle, par exemple une tâche en cours de réalisation. L'auteur propose de définir le contexte comme étant "*un agrégat de connaissances utilisé par un agent pour exécuter une action contextuelle dans une situation donnée, à un temps t avec un but précis. Une action contextuelle est une fonction qui met en correspondance un ou plusieurs arguments extraits du contexte avec un symbole utilisé par une action*".

Il souligne par ailleurs que le contexte ne peut être considéré indépendamment de son intention qui est définie par l'action dont la sémantique est contrainte par l'information minimale étant donné un concept intentionnel. Les concepts intentionnels définissent des actions. L'auteur conclut par donner la définition suivante [92] :

Définition 1.9.1 (fait) *Un fait dénote les objets en relation.*

Définition 1.9.2 (situation) *Étant donné un environnement valide* Γ *où* $\equiv$ *désigne l'égalité syntaxique et* $N : \mathbb{N}$*, une situation* $\mathcal{S}$ *est décrite par une séquence finie de certains objets de preuve* v_i *où chaque* v_i *est une preuve de la variable correspondante* x_i*. Une situation est décrite par une collection de faits* f_i*, telle que* $\mathcal{S} = \{f_1, ..., f_n\}$*.*

Par exemple, un domaine composé de trois blocs a, b, et c posés sur une table et empilés sur une collection de faits pourrait être :

$$On(a, table), On(c, table), On(b, c), Clear(a), Clear(b)$$

où $On(a, table)$ indique que le bloc a est sur la table alors que $Clear(a)$ signifie que le bloc a ne contient rien. Une action est un symbole de prédicat restreint aux verbes d'action (par exemple, déplacer, prendre, lire, tourner, etc.). Par exemple, l'action $Move(b, c, a)$ déplace un bloc b de c à a si b et a sont effacés.

Définition 1.9.3 (Contexte) *Étant donné une situation et un ensemble d'actions, le concept de contexte est la connaissance minimale qui affecte le comportement de toute action liée à ce contexte dans une situation donnée.*

Dans ces travaux l'action possède un type (concept) et est contextualisée (cf. 1.9.4).

1.9.3 Contexte et logique

Un modèle prescriptif d'un système cherche et sert à formaliser le problème et les besoins, et l'objectif est de fournir une représentation d'un système à créer qui mette en évidence les propriétés souhaitées de ce système. Quelle représentation et quel formalisme utilise-t-on alors pour représenter les connaissances du contexte ? Le premier problème auquel les travaux dans la littérature sont confrontés est celui de la modélisation du contexte. Il était question de trouver des formalismes qui offrent une représentation simple qui repose sur un raisonnement rationnel. Ici, la logique est la référence centrale pour fournir des modèles formels de contexte.

Historiquement ce sont les travaux de McCarthy [183] qui introduisent une logique qu'on appelle *calcul des situations*. Le calcul des situations est une logique définie pour représenter le contexte. L'auteur décrit le contexte comme étant un objet de première classe qui peut être l'argument d'un prédicat. Celui-ci est vu comme un méta-prédicat $ist(c, p)$ qui affirme que l'assertion p est vraie dans le contexte c. Des règles d'extension du contexte appelées *lifting rules* ont été définies pour relier une valeur de vérité dans un contexte à une valeur dans un autre contexte. Des opérations pour entrer et sortir d'un contexte ont été aussi définies. Son modèle du contexte utilise une logique qui trouve son origine dans le calcul des situations [130]. L'axiomatisation des contextes dans son approche est étendue en définissant une relation de spécialisation entre deux contextes *specializes*(c_1 , c_2) dont la valeur de vérité est vraie si le contexte c_1 a plus d'hypothèses que c_1.

Les travaux dans [69] ont proposé une extension des travaux de McCarthy à la logique propositionnelle. La logique propositionnelle du contexte étend la logique propositionnelle classique par l'ajout d'un opérateur modal $ist(\kappa, \phi)$ exprimant que la déclaration ϕ est vraie dans le contexte κ. Chaque contexte posséde son propre vocabulaire, ie : un ensemble de propositions atomiques. Etant donné un ensemble de contextes $\mathbb{K}$, un ensemble $\mathbb{P}$ de propositions atomiques, l'ensemble des formules bien formées $\mathbb{W}$ est construit à partir des propositions $\mathbb{P}$ faisant recours aux connecteurs propositionels usuels (négation, implication) auxquels s'ajoute l'opérateur ist.

D'autres extensions à la logique du premier ordre ont aussi vu le jour dans [68]. Les contextes sont des objets dans leur sémantique, ceux-ci peuvent donc être des termes dans

leur langage sur lesquels la quantification est rendue possible. Sur la base de l'extension à la logique propositionnelle [69], l'ensemble des propositions est étendu en ajoutant les quantificateurs universel et existentiel. À noter que de nouvelles interprétations des opérateurs d'entrer et de sortie d'un contexte de McCarthy ont été aussi fournies dans les deux extensions.

Il existe des logiques qu'on appelle *logiques contextualisées* [118] construites dans l'objectif de raisonner sur certaines propriétés contextualisées pour permettre leur classification dans un contexte particulier, puisque certaines classifications ne sont pas nécessairement vraies dans l'absolu. L'exemple donné ici est celui des oiseaux qui volent dans un contexte donné. La propriété "oiseau" est contextuellement subsumée par la propriété "vole", mais on n'a pas toujours $oiseau \sqsubseteq vole$. Dans ce type de raisonnement, c'est le principe du modus tollens qui est utilisé. Par exemple, appliquer un détournement au modus tollens qui revient à déduire une prémisse à partir d'une conclusion. Sur l'exemple des oiseaux qui volent, citons à titre indicatif : « *Les oiseaux savent voler* », et le fait « *Un avion sait voler* », alors on devrait déduire que « *Un avion est oiseau* ». Or, dans ce type de situations, la classification et l'extraction ne sont pas vraies, d'où l'intérêt d'introduire cette logique contextualisée car elle est adaptée pour faire de l'extraction en fonction des données ou des faits existants définis de manière extensionnelle.

En effet, le formalisme adopté est celui de l'analyse de concepts formelle dans sa variante logique appelée ACL. Ce formalisme intègre cette forme de relation à la logique appelée *relation de subsomption contextualisée*. Cette relation consiste à généraliser les implications entre attributs utilisés dans l'analyse de concepts formelle à des fins d'extraction de connaissances [126, 235]. Dans ce travail [118], la relation de subsomption contextualisée[9] $\sqsubseteq_K$ est un pré-ordre, et la relation d'équivalence associée est notée $\equiv_K$. La logique contextualisée se définit alors comme suit :

Définition 1.9.4 (logique contextualisée) *Soit $K = (\mathcal{O}, \mathcal{L}, d)$ un contexte ACL. La logique contextualisée est définie par l'ensemble partiellement ordonné $\mathcal{L}_K = \langle L; \sqsubseteq_K \rangle$ modulo la relation d'équivalence $\equiv_K$.*

Cette logique est de nature différente de la *logique contextuelle* introduite par l'auteur dans [214] : la première est une logique sur les formules, alors que la seconde est une logique sur les contextes (au sens de l'analyse de concepts formelle). Cette dernière est basée sur la notion de théorie de modèles pour l'implication des graphes conceptuels pour établir un ensemble de règles d'inférence solides et complètes. Son interprétation est basée sur les concepts (les *intensions*) et les objets (les *extensions*) également issus de l'analyse de concepts formelle.

Par ailleurs, l'auteur [118] a montré que le contexte ne modifie pas les propriétés des objets car il n'ajoute et ne supprime pas de propriétés dans un domaine. Citons à titre indicatif, l'utilisation du concept de téléphone. Dans le contexte où c'est un smartphone qui est utilisé, alors le domaine d'application doit nécessairement inclure les smartphones ainsi que tout autre type de téléphone. Dans ce cas, le contexte joue le rôle d'une théorie qui étend la relation de subsomption et permet de déduire de nouvelles connaissances. Sa logique contextualisée facilite l'extraction, à partir d'un contexte, des connaissances sous deux formes :

9. La subsomption contextualisée se définit comme suit : Soient $K = (\mathcal{O}, \mathcal{L}, d)$ et $f, g \in \mathcal{L}$. On dit que f est contextuellement subsumée par g dans le contexte K, notée $f \sqsubseteq_K g$, ssi $\beta_K(f) \subseteq \beta_K(g)$, c-à-d, tout objet qui satisfait f satisfait aussi g.

— *par déduction* pour les connaissances sur le contexte (dans ce cas, on pourrait prendre l'exemple où "tous les smartphones sont des téléphones, dans le contexte précis") ;
— et *par induction*, sur les connaissances du domaine, par exemple, "tous les smartphones sont des téléphones, dans un domaine précis".

Les connaissances du domaine permettent donc d'induire des propriétés propres à ce dernier, alors que le contexte permet de faire des déductions supplémentaires selon les faits ou les données fournis dans ce contexte.

1.9.4 Le contexte comme théorie : une sémantique basée sur les situations

Quelle interprétation donne-t-on alors au contexte ? Les logiques doivent avoir des théories de référence qui permettent de déterminer comment les constantes et les variables sont associées aux choses dans l'univers du discours, pour donner du sens aux propositions [237]. De plus, une région arbitraire de l'espace-temps n'a pas de *signification intrinsèque* [155]. Il faut donc donner une sémantique au contexte. Les travaux en théorie des situations cherchent à définir une interprétation du contexte basée sur les situations. Des travaux dans ce sens ont vu le jour [11, 29, 30, 31, 100]. Ces travaux se basent sur la théorie des situations pour formaliser le contexte. Celui-ci est traité comme une amalgamation de situations de base et des règles qui gouvernent les relations avec le contexte. Le contexte est représenté par un type de situation qui supporte deux types d'infons : les infons factuels qui représentent des faits, et les contraintes. Les contraintes sont des paramètres conditionnels [11]. Les principaux concepts primitifs de la théorie des situations [100] sont donc les infons et les situations. L'infon est l'unité de base qui incarne les éléments d'information. Ces derniers sont représentés par $\ll R, a_1, a_2, ...a_n, i \gg$, où R dénote une relation à n-arguments ; les $a_1, ..., a_n$ sont des objets appropriés représentants les arguments respectifs de R, ceux-ci dénotent des individuels, des situations, des temps, des lieux, des types (paramètres) et sont représentés par $\dot{t}$; i dénote la polarité de l'infon qui caractérise l'état de vérité de l'unité d'information (1 si la relation est vraie, 0 sinon). A titre d'exemple, $\langle\langle On, ACsystem, \dot{t}, 1\rangle\rangle$ représente un infon qui définit un système d'air conditionné à un temps $\dot{t}$. L'infon indique les situations où le système est en fonctionnement (On).

Les situations rendent certains infons factuels i.e., $s \vDash \iota$: stipule que l'infon ι est vrai dans la situation s. Le symbole $\vDash$ est une relation de validité entre les situations et les infons qui permet ainsi de capturer la sémantique d'une situation. Une telle validité exprime une relation de dépendance entre situation et infons. En théorie des situations la représentation du contexte est conçue uniquement pour fournir des informations de base. Le contexte définit entre autre le domaine de quantification [100] qui correspond aux uniformités de Barwise partagées par les infons. Les contraintes systématiques entre les types des situations sont ce qui permet à une situation donnée de contenir des informations sur une autre situation. En d'autres termes, le sens est ce qui permet à un événement d'un type particulier d'avoir une sémantique.

Étant donné le paramètre $\dot{x}$, un ensemble I fini d'infons contenant $\dot{x}$, ainsi le type abstraction défini par $[\dot{x}|\ \dot{s} \vDash \bigwedge_{\iota \in I} \iota]$. Ce type correspond au type de toute situation s à laquelle $\dot{x}$ est fonction des objets d'un type particulier de sorte que toutes les conditions dans I sont obtenues. Les abstractions de tpes concernent les paramètres, $\dot{x}$, les situations $\dot{s}$ et l'ensemble I. Par exemple, les infons $\langle\langle Sullivan, Drofnats\ University, 1\rangle\rangle$, $\langle\langle Suzanne, Drofnats\ University, 1\rangle\rangle$, $\langle\langle John, Drofnats\ University, 1\rangle\rangle$ partagent l'in-

formation $Drofnats\ University$. La différence réside dans le nom des étudiants qui prend des valeurs différentes. Cet infon peut être représenté par l'abstraction type $[\dot{s} \mid \dot{s} \vDash \langle\langle \dot{t},\ Drofnats\ University, 1\rangle\rangle]$, où $\dot{t}$ est un type paramètre qui définit un étudiant ($Student$).

L'exemple suivant illustre la manière dont une situation peut être représentée. Étant donnée une personne invitée à un séminaire situé dans un salon particulier d'un hôtel.

$$S_0 = [\dot{s} \mid \dot{s} \vDash << Located, seminar28, Hotel\_Pierre\_le\_Grand, \dot{t}, 1 >>]$$
$$S_1 = [\dot{s} \mid \dot{s} \vDash << Located, Pierre, Hotel\_Pierre\_le\_Grand, \dot{t}, 1 >>]$$
$$B = [\dot{s} \mid \dot{s} \vDash << Participates, Pierre, seminar28, \dot{t}, 1 >>]$$
$$C = \{S_0 \Rightarrow S_1 | B\}$$

Les conditions paramétriques qui expriment une contrainte sont définies par C. Cette contrainte signifie que si un séminaire a lieu dans un hôtel donné (S_0), alors il est possible de déduire qu'une personne est également dans cet hôtel (S_1) car elle participe au séminaire (B). B est un ensemble de conditions pour lesquelles la contrainte C exprime une information.

Barlatier [28, 92] unifie les deux approches basées sur la logique et celles basées sur les ontologies. Sa notion principle est un concept ontologique dénoté par un type. L'action contextualisée est considérée comme une fonction qui met en correspondance un ou plusieurs arguments extraits du contexte avec un symbole utilisé par une action. Ainsi, la structure du contexte est décrite par une agrégation de prédicats exprimant des propriétés et des contraintes reliées à une action contextualisée au sein d'une situation. Celle-ci dénote un ensemble de faits.

L'auteur s'inspira des travaux de [11, 100] sur la théorie des situations pour décrire le contexte comme étant un type de situation supportant à la fois des infons factuels et des contraintes. Une situation est le résultat d'exécution d'un événement. L'auteur définit un formalisme fondé sur la logique intuitionniste constructive et le calcul des constructions étendu [181] (qu'il a étendu pour prendre en compte les constantes et le sous-typage) ainsi que la théorie des situations [100]. La possibilité de réutiliser le contexte est concrétisée par la notion de types dépendants [82, 212]. L'auteur avance que la dynamicité peut être assurée par le fait que, dans une situation donnée, n'existent que les objets (instances) appartenant à des types de contexte. Il explique par ailleurs que l'utilisation des types rend le contexte indépendant des applications envisagées. Les types dépendants sont des types où un type peut être défini en fonction d'un habitant d'un autre type. Par exemple, le type des tableaux d'entiers où la taille est représentée par une information indiquée dans le type, i.e., alors il n'existe pas un type tableau, mais une famille : $tableau(0), tableau(1), ...$ pour des tableaux de taille 0, 1, etc. Cette correspondance entre les types dépendants et les contextes est justifiée par le fait qu'un contexte n'est pas une notion absolue [89, 109, 168], mais est relative à une activité, une action ...

Par ailleurs, le contexte est perçu comme un *moment universel*[10] [105] d'un point de vue ontologiquement, i.e., un concept dont l'existence dépend de *concepts intentionnels*. Ceux-ci sont concrétisés par des actions. La notion de *dépendance existentielle* [143] énonce un principe général comme suit : *Étant donné le prédicat ϵ dénotant l'existence. Un individu*

10. Notons seulement que l'origine de cette notion de *moment* réside dans la théorie des accidents individuels développés par Aristote dans sa métaphysique et ses catégories. Pour lui, un accident est une propriété, un événement ou un processus individualisé qui ne fait pas partie de l'essence d'une chose. *Un moment est un endurant qui est inhérent à, et, par conséquent, dépend de manière existentielle d'un autre endurant. Pour plus de détails consulter [143, 244].*

x est existentiellement dépendant d'un autre individu y ssi, par nécessité, y doit exister à chaque fois que x existe. Formellement, $ed(x, y) \stackrel{def}{=} \Box(\epsilon(x) \Rightarrow \epsilon(y))$ [11].

Étant donné que le contexte ne peut être considéré en dehors de l'utilisation d'une action, la description d'un contexte doit s'appuyer sur cette dépendance en établissant un lien *explicite* entre l'action et le contexte. Ainsi, la structure du contexte est décrite par une agrégation de prédicats exprimant des propriétés et des contraintes liées à des actions contextualisées dans une situation. Le lien entre un contexte et une intention s'exprime par une dépendance fonctionnelle représenté par un type dépendant. Partant du constat que les concepts de base se justifient par des observations ou des axiomes, et en assimilant les concepts ontologiques à des types, dans ce travail la subsomption ontologique correspond au sous-typage et la relation partonomique (la relation Parti-Tout issue des méréologies) qui s'applique entre des types sommes correspond à ce que McCarthy appelle "*context lifting*". Le *context lifting* illustre le fait que les prédicats définissant le contexte sont paramétrés par des situations. Ainsi, au niveau représentation un contexte peut être en relation partonomique défini comme suit :

Définition 1.9.5 (contextes partonomiques) *Un contexte type* C' *est une partie d'un autre contexte type* C *si* C' *contient au moins tout les types vérifiés dans* C *et si les types communs sont dans une relation de sous-type selon le plus petit ordre partiel défini sur les termes* [12].

1.10 Synthèse

Il apparaît clairement que l'explicitation de connaissances issues du domaine est le meilleur moyen de donner une justification et une sémantique sans ambiguïtés au modèles et aux conceptions de système. C'est à partir de l'établissement des fondements ontologiques des concepts de base d'un langage de modélisation conceptuelle que la clarification de la sémantique du monde dans lequel le système est positionné peut être obtenue [144]. En effet, une sémantique bien définie du modèle conceptuel d'un domaine conduit à une qualité supérieure du système logiciel construit sur ce modèle de domaine. Une ontologie est considérée comme la capitalisation de connaissances sur un domaine, qui doivent être formelles, consensuelles, où les aspects plus subjectifs liés à la tâche en question doivent être évalués par les experts du domaine [247] de manière coopérative. Dans notre cas, la vérification et la validation de systèmes pour s'assurer de leur adéquation aux exigences exprimées dans les cahiers de charge sont les principales tâches à accomplir. La description du domaine est la première étape à suivre avant de résoudre les problèmes liés à l'élicitation des exigences et à la conception des systèmes [46]. Les ontologies peuvent apporter :

- des descriptions explicites et précises par leur représentation et utilisation déclarative des unités de connaissance ;
- des explications et des vérifications de cohérence par leur mécanisme de raisonnement ;

11. De manière générale, ces travaux se basent sur l'étude des relations méréologiques des partie-tout. Ces relations peuvent être de différents types. Dans cette thèse nous ne détaillerons pas ces travaux car ces relations impliquent des détails sur les logiques épistémiques et les modalités ainsi que d'autres types de raisonnements non abordés dans cette thèse. Toutefois, les relations qui nous intéressent sont les méréologies de dépendance telles que définies par Barlatier. Pour plus de détails consulter [143, 244].
12. cf. corrolaire 2 donné dans [28].

De plus, pour lever les ambiguïtés au cours du processus de conception d'un système, le contexte est la première problématique à laquelle il faut faire face tout au long du processus de conception d'un système. L'absence de consensus [60] apparaît lorsque l'on considère la nature du contexte considéré comme statique ou dynamique, discret ou continu, la connaissance ou le processus. La question posée est alors : le contexte est-il connu a priori ou a posteriori dans la modélisation et la conception de systèmes ? Nous montrons que la vision a priori est liée à la vérification et est définie à un niveau donné de raffinement dans Event-B, que la vision a posteriori est liée plutôt à la validation de modèles dans ce formalisme dans le cas d'une vision dynamique, mais peut être considérée aussi par les raffinements a posteriori dans la chaîne de conception (vérification Event-B).

Puisque :

— la preuve de propositions en mathématiques est considérée comme un objet et que les travaux exposés dans notre état de l'art montrent que les preuves sont perçues comme le résultat d'un événement, à savoir, des situations ;
— de plus, les preuves peuvent être le résultat de l'existence d'une fonction [28] effectuant une action, ou encore le résultat obtenu par un démonstrateur de théorème étant donnée une hypothèse sur des entités, des propriétés ou des contraintes ;

De ce fait, du point de vue des logiques, il est connu que :

— les logiques classiques comme la logique du premier ordre ont des interprétations dans des espaces de valeurs infinies, et où le contexte est juste une collection d'affirmations consistant en des assertions significatives sous la forme de textes [201]. Sa fermeture est la collection de toutes les vérités qui tiennent dans toute situation où ces déclarations sont valides. En particulier, puisque l'on peut dériver n'importe quoi d'une seule fausse déclaration, la fermeture d'un ensemble contenant une déclaration universellement fausse est l'ensemble de toutes les déclarations ;
— les logiques qui représentent un domaine ont des interprétations dans des ensembles d'objets connus et finis. On peut citer les logiques de description [24, 25], et l'analyse de concepts logique (ACL) [118] ;
— les logiques initiées par McCarthy ont des interprétations dans des ensembles de situations [183].

Nous nous inspirons de ce dernier type de logique pour définir le mécanisme de dépendance pour le formalisme Event-B. Les mécanismes de raisonnement logique peuvent être utilisés pour raisonner formellement sur le contexte soit pour la vérification de la cohérence du contexte ; soit pour la déduction du contexte de niveau plus élevé, ce qui correspond au contexte implicite, à partir du contexte de niveau plus faible connu sous le nom de contexte explicite [115]. Le contexte implicite ne peut être acquis directement, mais inféré à partir des connaissances explicitées [115]. Nous montrons que le raffinement résout en grande partie ce problème. L'utilisation du domaine et du contexte permet en revanche de fournir des mécanismes configurables et extensibles pour effectuer des vérifications et des inférences par rapport à ces propriétés de manière inductive en fonction du domaine, et déductivement selon le contexte.

Habituellement, la connaissance n'est pas systématiquement décrite par les langages de modélisation (pour les raisons données ci-dessus i.e., détails sans intérêt, abstractions, objectif de modélisation, réutilisation, etc.) car l'ingénierie système a pour première préoccupation de *construire un système ou une machine* [154]. Un problème, un système ou un phénomène en génie logiciel suit une vision constructiviste (et donc prescriptive) qui ne peut être indépendante de l'observateur (l'ensemble des parties prenantes dans le processus de conception d'un système) de ce phénomène. Ces approches sont donc définies

comme le résultat issu de l'action humaine qu'est la perception et l'interprétation par cet observateur de la réalité. Un système est donc créé, inventé et constitué au cours du processus de conception, de développement et d'utilisation. Un domaine décrit une réalité qui signifie qu'elle est indépendante de l'observateur et de sa perception. Un système est donc décrit par un couplage, une association de ses propres propriétés avec les effets caractérisés par les propriétés de son domaine dont il est dépendant. Ces associations doivent manipuler les concepts nécessaires pour garantir et donner accès à des mécanismes de raisonnement selon les niveaux d'abstraction et les détails de modélisation.

La formalisation des objets de l'étude appelés *référent* (cf. définition 1.8.1) et du grain de la connaissance (ou degré de granularité de la connaissance) pour définir les niveaux des détails qui correspondent aux niveaux d'abstraction d'un modèle faisant l'objet de l'analyse est fondamentale pour caractériser chaque unité de connaissance spatiale ou temporelle [74]. Elle l'est d'autant plus vraie, non seulement pour établir les propriétés à spécifier et à vérifier pour le système étudié, mais aussi pour garantir l'inclusion de traces dans les modèles construits. Nous définissons des stratégies basées sur cette notion de grain et de hiérarchisation des unités de connaissance afin de mieux mécaniser le raffinement en Event-B et à valider la pertinence du modèle (niveau de raffinement) jugée par l'information nécessaire et suffisante (complète) pour exprimer les propriétés invariantes et de sûreté d'un système. Cette association utilise les logiques de description [24, 25] pour extraire des propriétés issues du domaine. L'approche défendue ici est celle d'une *association par intégration* qui peut être réalisée par un processus d'extraction partiellement automatisé. Nous montrons en perspectives de nos travaux comment il serait possible d'exploiter la logique ACL [118] pour formaliser cette extraction par raffinement.

Le chapitre suivant aborde le formalisme de modélisation Event-B.

Chapitre 2

Le formalisme Event-B

Sommaire

2.1 Introduction 56
2.2 Représentation et modélisation des connaissances en Event-B 57
2.2.1 Description d'un contexte Event-B 58
2.2.2 Description d'une machine Event-B 59
2.2.3 Raffinement de modèles Event-B 62
2.2.4 Sémantique implicite des modèles Event-B : Obligations de preuves 63
2.3 Les différentes approches de structuration des modèles Event-B 65
2.3.1 Composition/décomposition de modèles 66
2.3.2 Patrons de conception 67
2.4 Extension Temporelle pour le formalisme Event-B 68
2.4.1 Extension TLA pour Event-B 68
2.5 Outils Event-B utilisés 73
2.6 Synthèse 73

« Le domaine de la liberté commence là où s'arrête le travail déterminé par la nécessité. »
- Karl Marx

2.1 Introduction

La modélisation est un moyen pour parvenir à spécifier et vérifier les systèmes informatiques. Le formalisme Event-B [3] est une méthode parmi d'autres qui adopte cette approche. Cette technique repose sur un langage basé sur une description événementielle des systèmes par le biais de formules logiques. Les propriétés exprimées dans ce formalisme concernent la sûreté des systèmes au moyen de la logique du premier ordre et la théorie des ensembles. La construction de modèles repose sur une relation qu'on appelle raffinement, et des conditions de vérification. La section 2.2 du présent chapitre introduit les notions de base du formalisme Event-B.

Le raisonnement de façon automatique sur les preuves dans la conception des systèmes est difficile. Ceci est dû au fait que les preuves dans ces systèmes s'attachent surtout à définir les notions et les concepts basés sur la faisabilité des schémas de conception où les risques sont évalués dans un monde fermé, en posant des hypothèses et des contraintes. En revanche, un système n'est pas nécessairement lié à une contrainte ni même à un type particulier d'hypothèses. Le but est de s'inscrire dans une démarche d'élaboration en adoptant un processus d'automatisation des preuves d'une part, et de couvrir un large spectre des propriétés de sûreté à vérifier d'autre part. Au niveau conceptuel, le *contexte* dépend de l'*environnement* dans lequel les systèmes vont s'exécuter. C'est donc sur la base de l'analyse du contexte que l'on peut définir les contraintes et établir les propriétés de sûreté. Et les résultats souhaités en terme de conception des systèmes prouvés corrects émanent d'une volonté de formalisation et de factorisation des efforts à fournir en terme de preuves. Ceci est garanti par le développement de *patrons de conception*. La section 2.3 fait un tour d'horizon sur les approches qui permettent de structurer le développement et résoudre les difficultés citées ci-dessus dans le formalisme Event-B.

L'utilisation du formalisme Event-B [3] pour spécifier et prouver la correction des systèmes a gagné de plus en plus du terrain, car il offre un pouvoir d'expressivité élevé et un raisonnement formel sur les modèles mécanisé par le raffinement. Plusieurs systèmes ont été modélisés en utilisant la méthode Event-B, et son application touche des domaines de nature différente. Ce formalisme a été utilisé pour la modélisation de nombreux problèmes et systèmes, tels que les algorithmes distribués [16], les systèmes multi-agents [137], la sécurité et en particulier, les protocoles d'authentification, d'établissement des clefs, et d'accès [34], les protocoles de population [189], le pacemaker [234], . . . et dans bien d'autres travaux.

Néanmoins, ce formalisme ne devrait pas se réduire à l'expression de propriétés de sûreté. Le besoin de gérer et d'exprimer les propriétés liées au temps s'est fait ressentir. En effet, les propriétés telles que la vivacité, l'équité sont des exigences importantes dans la spécification des systèmes, en particulier dans les systèmes étudiés dans cette thèse, à savoir, les systèmes de vote. Cet aspect ou caractéristique est devenu d'un grand intérêt dans la communauté des chercheurs des langages de spécification en général, et en particulier du langage Event-B. Il existe des travaux qui ont abordé cette notion tels que [16, 147, 188, 189, 231]. Seule l'approche de Andriamiarina dans ses travaux [16] donne

une solution exhaustive et complète qui raisonne par raffinement, et dont la sémantique est basée sur les traces d'exécution qui est bien définie. Ces travaux définissent un cadre formel basé sur la logique temporelle des actions (Temporal Logic of Action) TLA [174]. La différence est que le cadre défini pour Event-B raisonne sur les événements dans les machines d'un modèle, alors que dans TLA c'est l'action qui prend place dans la structure sur laquelle ce formalisme raisonne. La section 2.4 décrit le cadre temprel à la TLA défini pour le formalisme Event-B. La section 2.6 présente une synthèse du chapitre.

2.2 Représentation et modélisation des connaissances en Event-B

La spécification en Event-B [3] fournit des modèles dont la sémantique est exprimée au moyen d'objets formels. Un modèle formel en Event-B se bâtit à partir de trois principaux ingrédients : un langage de spécification basé sur la théorie des ensembles et la logique des prédicats ; un système de vérification basé sur la génération d'un ensemble d'obligations de preuves ; et un mécanisme de raffinement qui permet un développement correct par construction. Un tel développement permet d'obtenir des modèles de systèmes réactifs à événements discrets prouvés qui peuvent être implémentés. Le tableau 2.2 donne les notations ensemblistes utilisées dans la conception de modèles en Event-B.

Nom	**Syntaxe**	**Définition**
Relation binaire	$s \leftrightarrow t$	$\mathbb{P}(s \times t)$
Composition	$r_1 ; r_2$	$\{x, y \mid x \in a \wedge y \in b \wedge \exists\, z.(z \in c \wedge x \mapsto z \in r_1 \wedge z \mapsto y \in r_2)\}$
Domaine	$\mathrm{dom}(r)$	$\{a \mid a \in s \wedge \exists\, b.(b \in t \wedge a \mapsto b \in r)\}$
Codomaine	$\mathrm{ran}(r)$	$\mathrm{dom}(r^{-1})$
Identité	$\mathrm{id}(s)$	$\{x \mapsto x \in s\}$
Restriction	$s \lhd r$	$\mathrm{id}(s) ; r$
Co-restriction	$r \rhd t$	$r ; \mathrm{id}(t)$
Anti-restriction	$s \lhd\!\!\!- r$	$(\mathrm{dom}(r) - s) \lhd r$
Anti-co-restriction	$r \rhd\!\!\!- t$	$r \rhd (\mathrm{ran}(r) - t)$
Image	$r[w]$	$\mathrm{ran}(w \lhd r)$
Surcharge	$q \lhd\!\!+ r$	$(\mathrm{dom}(r) \lhd\!\!\!- q) \cup r$
Fonction partielle	$s \nrightarrow t$	$\{r \mid r \in s \leftrightarrow t \wedge (r^{-1} ; r) \subseteq \mathrm{id}(t)\}$
Fonction totale	$s \rightarrow t$	$\{f \mid f \in s \nrightarrow t \wedge \mathrm{dom}(f) = s\}$
Injection totale	$s \rightarrowtail t$	$\{f \mid f \in s \rightarrow t \wedge f^{-1} \in t \nrightarrow s\}$

FIGURE 2.1 – Les notations ensemblistes du formalisme Event-B

Un modèle formel $\mathcal{M}$ dans le formalisme Event-B est composé de contextes et de machines. Un contexte B $\mathcal{T}h$ spécifie la partie statique d'un modèle et comprend des ensembles supports s, des constantes s, des axiomes et théorèmes (Axm et Thm) qui établissent des contraintes et des propriétés des éléments statiques. Une machine décrit la dynamique du système au moyen d'une liste finie de variables $vars$ décrivant l'état du système, éventuellement modifiés par une liste d'événements $E = \{e_0, ..., e_n\}$. Le changement d'état doit respecter des propriétés appelées invariants et théorèmes Inv qui doivent être maintenues à chaque activation des événements dans le système. Ces invariants sont exprimés par des prédicats et doivent indiqués les types de chaque variables dans la machine. La Figure 2.2 illustre sommairement les éléments qui forment un contexte et une machine Event-B. L'utilisation par une machine des éléments définis dans un contexte en

Event-B est rendue possible via la relation « *voit* », ainsi une machine peut « *voir* » (*sees* en anglais) un contexte.

Les spécifications formelles dans ce formalisme sont conçues de manière incrémentale en transformant une spécification abstraite en spécification plus concréte jusqu'à obtention d'un modèle à partir duquel une implémentation peut être générée. Cette concrétisation s'effectue à travers une ou plusieurs étapes de raffinement. Ce processus est validé par des obligations de preuves qui garantissent la préservation des invariants abstraits. Chaque modèle qui établit une étape du mécanisme de raffinement est défini dans une machine. Un composant est défini par une machine abstraite et un ensemble de machines qui raffinent cette première machine abstraite.

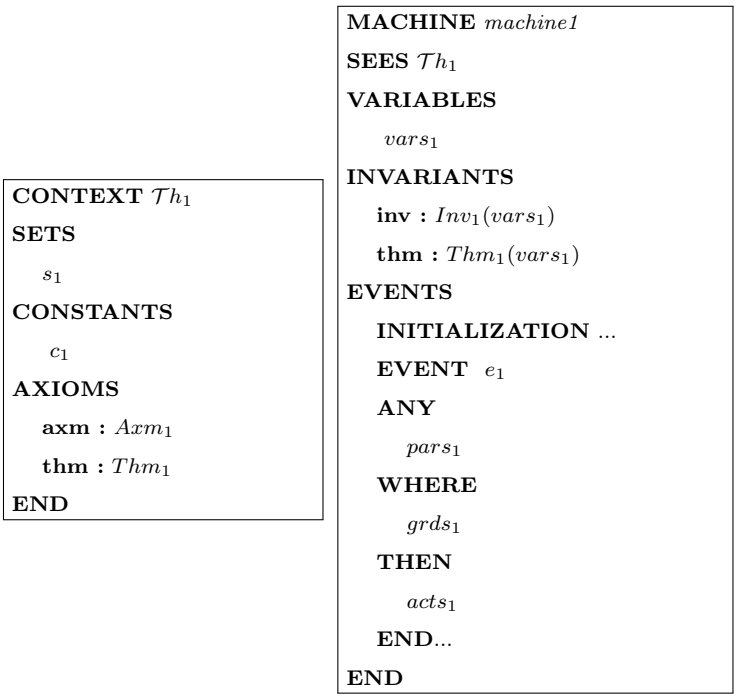

FIGURE 2.2 – Structure d'un modèle Event-B

2.2.1 Description d'un contexte Event-B

Un contexte Event-B regroupe :

— des ensembles non vides par défaut. Ces derniers servent pour typer les éléments définis dans le modèle et sont introduits dans la clause SETS.

— des constantes qui ne changent pas de valeurs dans la spécification et sont introduites dans la clause CONSTANTS.

— des axiomes et des théorèmes exprimés sous forme de prédicats et introduits dans la clause AXIOMS établissent des propriétés sur les ensembles et les constantes (particulièrement leur typage) définis. Les théorèmes doivent être prouvés en s'aidant des axiomes préalablement définis.

Les contextes peuvent être définis par extension. Un contexte Th_2 qui étend un autre contexte Th_1 peut utiliser les éléments définis dans ce dernier (Th_1). Une telle extension est exprimée par la clause *extends* dans le contexte Th_2.

2.2.2 Description d'une machine Event-B

La dynamique du système est décrite dans des machines qui modélisent au moyen de variables, propriétés et événements, les comportements du système étudié. Une machine est constituée :

— d'un ensemble de variables qui définissent l'état du système modélisé introduit dans la clause VARIABLES ;
— d'un ensemble d'événements introduit dans la clause EVENTS faisant évoluer les valeurs des variables. Il existe un événement appelé INITIALIZATION qui permet d'initialiser les valeurs dans la machine ;
— d'un ensemble de propriétés établissant l'invariant du système introduites dans la clause INVARIANT, et des théorèmes ;

2.2.2.1 Exemple de modèles Event-B

L'exemple donné dans ce qui suit illustre nos propos. Il s'agit d'un modèle qui spécifie le comportement d'un système d'accès des véhicules sur autoroute. Dans le premier modèle, tous les véhicules ont un identifiant unique exprimé dans l'ensemble des véhicules $VEHICULES$. Implicitement, il s'agit des immatriculations. Similairement, toutes les autoroutes sont exprimées dans l'ensemble $AUTOROUTES$ défini dans le contexte de ce modèle :

SETS
$VEHICULES$, $AUTOROUTES$
AXIOMS
axm1 : $VEHICULES \neq \varnothing$
axm2 : $AUTOROUTES \neq \varnothing$
END

L'état du système est caractérisé par la variable *acces* qui donne l'accès aux automobilistes à une autoroute donnée. Celle-ci est représentée par une fonction partielle de l'ensemble des véhicules vers celui des autoroutes (*inv1*) :

VARIABLES
acces
INVARIANTS
inv1 : $acces \in VEHICULES \nrightarrow AUTOROUTES$

Initialement, aucun véhicule n'est sur l'autoroute. Ceci est exprimé par l'inialisation de la variable *acces* à vide.

INITIALIZATION
THEN
act1 : $acces := \varnothing$
END

Le changement d'état à ce niveau est réalisé par deux événements qui modifient cette variable, soit en donnant l'accès à un véhicule donné sur une autoroute ($acces := acces \cup \{v \mapsto auto\}$) à condition que ce dernier n'a pas eu l'accès au préalable $v \notin dom(acces)$, soit en lui retirant l'accès ($acces := acces \setminus \{v \mapsto auto\}$) une fois qu'il y est ($v \notin dom(acces)$). À ce niveau d'abstraction, tous les comportements sont possibles.

Acces
ANY
$v, auto$
WHERE
grd1 : $v \in VEHICULES$
grd2 : $auto \in Autoroutes$
grd3 : $v \notin dom(acces)$
THEN
act1 : $acces := acces \cup \{v \mapsto auto\}$
END

Sortie
ANY
$v, auto$
WHERE
grd1 : $v \in VEHICULES$
grd2 : $auto \in Autoroutes$
grd3 : $v \mapsto auto \in acces$
THEN
act1 : $acces := acces \setminus \{v \mapsto auto\}$
END

2.2.2.2 Les événements

Un événement dans la méthode Event-B est défini par : *1)* des gardes qui expriment la condition sous laquelle l'événement devient déclenchable ; *2)* des actions qui définissent l'évolution des valeurs des variables d'état dans le modèle. Les changements d'état indiqués par les valeurs des variables dans une machine sont exprimés via des *substitutions généralisées*. Dans sa forme la plus simple, $var := Exp(var)$, une substitution généralisée est assimilée à une instruction d'affectation. Dans cette construction, var dénote le vecteur bâti sur l'ensemble des variables d'état du modèle, et $Exp(var)$ l'ensemble des expressions. L'interprétation d'une telle construction est une substitution de chaque variable du vecteur var par son expression correspondante dans $Exp(var)$. $var : |P(var, var')$ dénote une forme de substitution plus générale, telle que var est modifiée de sorte que le prédicat $P(var, var')$ devienne vrai, et où var' exprime la nouvelle valeur du vecteur de variables et var dénote l'ancienne valeur. Cette forme de substitutions est non déterministe. D'autres substitutions existent telles que : $var :\in Exp(var)$ qui indique que la variable var est modifiée et prend sa valeur dans $Exp(var)$; Notons qu'il est possible d'exprimer toutes les substitutions citées au travers de la substitution généralisée $var : | P(var, var')$.

Les événements ont différentes formes :

— une forme indéterministe où var est une variable du modèle, t est un paramètre. Le déclenchement de l'événement n'est possible que s'il existe une valeur de la variable locale t qui satisfait le prédicat $grd(var, p)$. L'action $var : |P(var, var', p)$ peut être alors exécutée.

EVENT evt
ANY
p
WHERE
$grd(var, p)$
THEN
$var : \mid P(var, var', p)$
END

— une forme gardée caractérisée par l'absence de variables locales (paramètres). Une telle forme ne dépend que des variables globales du modèle.

EVENT evt
  WHERE
    $grd(var)$
  THEN
    $var :| P(var, var')$
END

— une forme simple où la garde est vraie en permanence.

EVENT evt
  THEN
    $var :| P(var, var')$
END

Les événements sont décrits par les prédicats nommés prédicat *Avant-Après* (*Before-After BA*) et dénotés par *BA(var, var')*. Un tel prédicat exprime la relation qui lie les valeurs des variables avant (*var*) et après (*var'*) l'observation d'un événement. Les obligations de preuves sont produites dans l'objectif d'indiquer qu'une condition $Inv(var)$ est préservée. Leur forme générale résulte immédiatement de la définition du prédicat avant-après (*BA(var, var')*). Le tableau 2.3 résume les gardes et les prédicats avant-après.

Événement e	**Garde** $grd(e)$	**Prédicat avant-après**
BEGIN $var :\|P(var, var')$ END	$TRUE$	$P(var, var')$
WHEN $grd(var)$ THEN $var :\|P(var, var')$ END	$grd(var)$	$grd(var) \wedge P(var, var')$
ANY p WHERE $grd(p, var)$ THEN $var :\|P(var, var', p)$	$\exists p.(grd(p, var))$	$\exists p.(grd(p, var) \wedge P(var, var', p))$

FIGURE 2.3 – Les gardes et les prédicats avant-après

Les événements peuvent avoir trois status : *ordinary*, *convergent* et *anticipated*. Le premier définit un status ordinaire pour les événements où aucune contrainte n'est imposée quant à leur observabilité. Le second type définit un événement convergent dont l'observabilité n'est pas infinie. Ceci revient à définir un *variant* qui sera décrémenté à chaque observation de l'événement convergent. Cette expression de variant définit un ordre bien fondé qui exprime qu'il n'existe pas de séquence infinie décroissante d'éléments dans l'ensemble défini par le variant et décrémenté à chaque observation de l'événement ayant le status convergent. Un événement anticipated exprime que la convergence de l'événement en question ne peut être prouvée au niveau de raffinement en question, mais elle le sera dans des raffinements ultérieures. L'événement anticipé permet exprime qu'il ne modifie pas les valeurs des variants qui concernent d'autres événements convergents dans le même modèle Event-B.

Notons enfin que les machines Event-B peuvent aussi être interprétées comme des systèmes de transition [34], où les transitions sont définies par les événements avant-après.

2.2.3 Raffinement de modèles Event-B

La modélisation des connaissances dans le formalisme Event-B est réalisée par raffinement. Ce mécanisme permet de construire des modèles par étapes successives, en commençant généralement par un modèle très abstrait ne contenant que très peu de variables d'état et très peu d'événements. Au fur et à mesure de l'avancement dans ce processus, les modèles se retrouvent ou se terminent avec une étape face à de nombreux événements et de nombreuses variables, d'où le nom de *développement incrémental* ou *correct-par-construction*

Le raffinement entre les modèles Event-B conserve les propriétés déjà prouvées au niveau des modèles les plus abstraits dans la chaîne de raffinement. Ce mécanisme autorise l'ajout de nouvelles variables dans les modèles concrets, la suppression d'autres variables, l'ajout de nouveaux événements qui raffinent l'événement spécial « *skip* », le renforcement des gardes d'événements dans les modèles abstraits et l'affinement de leurs actions. Les événements nouvellement ajoutés ne modifient pas les variables abstraites. Les notations $cvar$, et $avar$ réfèrent respectivement aux variables nouvellement introduites et celles abstraites dans les modèles concrets de la chaîne de raffinement. Les *invariants de collage* $J(avar, cvar)$ offrent, par leur ajout dans les modèles concrets, les relations entre les variables concrètes $cvar$ et abstraites $avar$. Ainsi, leur introduction assure le typage des variables $cvar$.

La structure d'un modèle raffinant un modèle abstrait est la même que ce dernier à l'exception de la clause EXTENDS indiquée au niveau des contextes pour exprimer qu'un contexte Event-B étend un autre contexte plus abstrait, et la clause REFINES qui doit être indiquée au niveau des machines pour exprimer qu'une machine Event-B raffine une autre machine plus abstraite.

2.2.3.1 Un exemple de raffinement

L'exemple qu'on peut prendre ici est celui du modèle abstrait précédent et qui concerne le système d'accès des véhicules sur autoroute. L'accès sur l'autoroute est contraint par des tickets. Pour exprmier le fait qu'un véhicule prend un ticket, on introduit deux nouvelles variables pour illustrer le fait de distribuer des tickets, et de caractériser l'état des véhicules qui ont pris un ticket. Un nouvel ensemble porteur $TICKETS$ est introduit dans le contexte pour typer les tickets. Chaque distribution de tickets est unique ($tickets\_dist \in TICKETS \nrightarrow VEHICULES$), et le sous-ensemble des véhicules qui ont pris un ticket est inclus dans l'ensemble des véhicules $veh\_a\_ticket \subseteq VEHICULES$. Les accès et les sorties restent identiques aux premiers événements dans le modèle abstrait. Deux nouveaux événements sont introduits pour distribuer des tickets et pour illustrer le fait qu'un automobiliste a payé son accès. Dans un modèle abstrait, aucune précision n'est donnée quant à la somme à régler. L'observation de l'événement pour distribuer des tickets est gardée par le fait qu'un véhicule n'a pas encore eu l'accès et que le ticket qui lui sera distribué n'existe pas dans l'historique des tickets déjà distribués. La variable $tickets\_dist$ permet donc d'enregistrer l'historique des distributions des tickets.

takeTicket
ANY
$v, auto, t$
WHERE
$grd1: \ v \in VEHICULES \setminus veh\_a\_ticket$
$grd2: \ auto \in Autoroutes$
$grd3: \ v \notin dom(accesautoroute)$
$grd4: \ t \in TICKETS$
$grd5: \ t \notin dom(tickets\_dist)$
THEN
$act1: \ tickets\_dist := tickets\_dist \cup \{t \mapsto v\}$
$act2: \ veh\_a\_ticket := veh\_has\_ticket \cup \{v\}$
END

Le paiement d'un ticket est illustré simplement par le retrait d'un véhicule du sous-ensemble des véhicules qui ont pris un ticket. A ce niveau également aucune contrainte n'est imposée dans la spécification.

Pay
ANY
$v, auto, t$
WHERE
$grd1: \ v \in VEHICULES$
$grd2: \ auto \in Autoroutes$
$grd3: \ v \mapsto auto \in accesautoroute$
$grd4: \ t \in TICKETS$
$grd5: \ t \mapsto v \in tickets\_dist$
$grd5: \ v \in veh\_a\_ticket$
THEN
$act1: \ veh\_a\_ticket := veh\_a\_ticket \setminus \{v\}$
END

2.2.4 Sémantique implicite des modèles Event-B : Obligations de preuves

Si les formules de la logique du premier ordre définissent les assertions d'une structure d'un modèle Event-B, et si la structure d'un modèle Event-B est définie par les contextes et les machines tels que exposés ci-dessus, cependant, le raisonnement sur les preuves en Event-B et leur construction se fait par raffinement des modèles Event-B, et leur concrétisation est transposée par les obligations de preuve et les traces d'exécution d'un modèle Event-B. Les obligations de preuve sont aussi des formules logiques qui prennent en considération les changements d'état du système. Les événements peuvent être reformulés sous forme des prédicats *Avant-Après*. Les obligations de preuves sont établies afin de garantir la correction des modèles. Elles servent à : prouver les propriétés d'invariance ; prouver les propriétés d'un raffinement ; prouver ou déduire des propriétés théorèmes.

Nous notons $\mathcal{CT}h$ les contextes concrets ajoutés par extension Event-B dans la chaîne de raffinement, $cvar : \mathcal{I}nit(cvar')$ la substitution de l'initialisation pour attribuer une valeur aux variables nouvellement introduites ainsi qu'à celles du modèle abstrait conservées, $BAA(e)(avar, avar')$ le prédicat avant-après de l'événement abstrait e et $BAC(e)(cvar', cvar')$

le prédicat avant-après de l'événement concret e. $\mathcal{CT}h(s,c)$ est le contexte concret qui étend (extends) le contexte abstrait $\mathcal{T}h(s,c)$.

	Obligations de preuve des modèles Event-B
INIT	$\mathcal{T}h(s,c), \mathcal{I}nit(var,s,c) \vdash \mathcal{I}nv(s,c,var)$
FIS	$\mathcal{T}h(s,c), \mathcal{I}nv(s,c,var), grd(e)(var) \vdash \exists\ var'.BA(e)(s,c,var,var')$
INV	$\mathcal{T}h(s,c), \mathcal{I}nv(s,c,var), BA(e)(s,c,var,var') \vdash \mathcal{I}nv(var')$
S-THM	$\mathcal{T}h(s,c), \mathcal{I}nv(s,c,var) \vdash Thm(s,c,var)$
R-INIT	$\mathcal{CT}h(s,c), \mathcal{I}nit(cvar',s,c) \vdash \exists\ avar'.(\mathcal{I}nit(avar') \wedge J(avar,cvar'))$
R-INV	$\mathcal{CT}h(s,c), \mathcal{I}nv(avar), J(avar,cvar), BAC(e)(cvar,cvar')$ $\vdash \exists\ avar'.(BAA(e)(avar,avar') \wedge J(avar',cvar'))$
R-SKIP	$\mathcal{CT}h(s,c), \mathcal{I}nv(avar), J(avar,cvar), BAC(e)(cvar,cvar') \vdash J(avar',cvar')$
R-VAR-FIN	$\mathcal{CT}h(s,c), \mathcal{I}nv(avar), J(avar,cvar) \vdash \text{finite}(V(cvar))$
R-VAR-NAT	$\mathcal{CT}h(s,c), \mathcal{I}nv(avar), J(avar,cvar) \vdash V(cvar) \in \mathbb{N}$
R-VAR	$\mathcal{CT}h(s,c), \mathcal{I}nv(avar), J(avar,cvar), BAC(e)(cvar,cvar') \vdash V(cvar') < V(cvar)$

FIGURE 2.4 – Les obligations de preuve de la chaîne de raffinement de modèles Event-B

Le tableau 2.4 donne la liste des obligations de preuve d'un modèle Event-B et de son raffinement.

Faisabilité d'un événement : (FIS) Les événements dans une machine doivent être faisables. Cela revient à dire que lorsque l'invariant est vrai, il existe toujours une nouvelle valeur var' du vecteur de variables var satisfaisant le prédicat $P(var,var')$ de la forme normale $var : |P(var,var')$, et où var correspond à la valeur de la variable avant l'observation d'un événement qui satisfait l'invariant dans le système, et dont la garde est vraie.

Invariant d'un modèle : À chaque observation de tout événement dans le système, l'invariant doit être préservé :

— en premier lieu par l'initialisation du système définie par l'événement INITIALISATION, et l'obligation de preuve concernée est **INIT**, qui stipule que la substitution $var : |\mathcal{I}nit(var')$ doit maintenir l'invariant de la machine ;

— par les autres événements définis dans chaque machine et correspondant à l'obligation de preuve **INV**. Cette obligation exprime que le changement d'état lié à l'observation d'un événement et exprimé par le prédicat avant-après respecte ou maintient l'invariant du système.

L'utilisation des axiomes et théorèmes définis dans les contextes est donc utile pour la réalisation de ces preuves. Aussi, le même principe est appliqué pour la preuve des théorèmes dans les machines Event-B (**S-THM**). Ceux-ci définissent les propriétés de sûreté. Les propriétés de sûreté sont des propriétés exprimées déductivement sur les invariants dans les machines. Nous reviendrons sur ce principe dans la suite de ce chapitre (cf section 2.4).

Obligation de preuve pour le raffinement : cas de l'initialisation L'attribution dans l'initialisation de nouvelles valeurs aux variables (conservées ou nouvellement introduites) au niveau des raffinements ne doit pas contredire l'initialisation dans le modèle abstrait. L'obligation de preuve concernée est **R-INIT**.

Obligation de preuve pour le raffinement : cas de l'invariant L'obligation de preuve concernée est **R-INV**. Elle permet d'établir que la modification des variables abstraites dans les événements concrets doit se faire de manière identique à celle des l'événements abstraits. Cette formule exprime que l'observation d'un événement concret e (prédicat before-after $BAC(e)(cvar', cvar')$) en respect avec les invariants concrets et de collage, doit établir l'observation de l'événement abstrait $(BAA(e)(avar', avar'))$ en respect avec l'invariant de collage entre les variables abstraites et concrètes.

Obligation de preuve pour le raffinement : cas de l'événement skip (bégaiement) En Event-B, de nouveaux événements peuvent être introduits par raffinement. Dans ce cas, ces derniers raffinent l'événement spécial *skip*. Une obligation de preuve qui établit que l'observation de ces événements ne modifient pas les variables abstraites via le collage avec les variables concrètes $J(avar, cvar')$, telles que $BAA(skip)(avar, avar') = avar = avar'$ doit être prouvée (**R-SKIP**).

Obligation de preuve pour le variant Les variants sont des expressions qui définissent un ordre bien fondé sur un ensemble afin d'être décrémenté par les événements convergents. La première contrainte que doit satisfaire cette expression est qu'elle doit être finie (**R-VAR-FIN**). La deuxième condition établit que le variant est entier positif (**R-VAR-NAT**). La dernière condition stipule qu'à chaque observation de l'événement convergent, la valeur du variant décroît (**R-VAR**). Lorsqu'elle atteindra sa valeur minimale, l'observabilité de l'événement convergent en lien n'est plus possible.

Obligation de preuve pour le non-blocage Il existe une obligation de preuve qui permet d'établir l'absence de blocage dans les machines. Celle-ci est définie comme suit :

$$\mathcal{CT}h(s,c), \mathcal{I}nv(avar), J(avar, cvar), (cgrd_1(avar), \vee ... \vee cgrd_n(avar)) \vdash (agrd_1(cvar) \vee ... \vee agrd_m(cvar)))$$

et exprime qu'il n'y a pas de blocage dans les machines concrètes et abstraites. *agrd* et *cgrd* expriment respectivement, les gardes dans les machines abtraites et concrètes.

2.3 Les différentes approches de structuration des modèles Event-B

La méthode Event-B a donné aux développeurs la possibilité de construire des modèles de systèmes complexes qui sont corrects par construction. Le raffinement est une primitive qui permet de construire des spécifications Event-B structurées. Mais, cette technique n'est pas la seule qui offre des mécanismes de structuration de spécifications, notamment en termes de réutilisation. La composition et décomposition de modèles ainsi que les patrons

de conception sont des techniques qui permettent également de factoriser les preuves. Les sous-sections suivantes décrivent brièvement ces techniques.

2.3.1 Composition/décomposition de modèles

La composition et la décomposition de modèles est un autre moyen pour construire des modèles de taille considérable car elles permettent une structuration des spécifications par assemblage de composants dans le cas d'une composition, et par désassemblage dans le cas d'une décomposition. Le défi consiste alors à exhiber les différentes parties sujettes à composer ou à décomposer en vue de prouver des propriétés séparement. L'idée de la décomposition de modèles est ainsi clairement très attrayante : elle consiste à découper un système d'événement lourd en petits morceaux qui peuvent être manipulés plus confortablement que l'ensemble. Plus précisément, chaque partie peut être raffinée indépendamment des autres. Mais, la contrainte que doit satisfaire cette décomposition est que ces parties raffinées indépendamment pouvaient toujours être facilement recomposées. Ce processus de re-composition devrait ensuite aboutir à un système qui aurait pu être obtenu directement, sans la décomposition.

La décomposition de modèles dans la méthode Event-B a été abordée de deux manières. Une approche proposée par Abrial [7, 150] et basée sur les variables d'état et une autre proposée par Butler [66, 91] et fondée sur la décomposition des événements.

— La technique d'Abrial consiste à décomposer les modèles en parties, puis de développer chaque partie séparement. Se pose alors le problème de décomposition en cas de partage de variables ie : une variable est utilisée dans deux événements appartenant chacun à une partie ou composant issue de la décomposition. En effet, cela ne garantit pas la préservation de ses propriétés dans les raffinements de chaque composant. Pour remédier à ce problème, l'auteur propose d'introduire des variables externes et des événements externes. Les événements externes introduits dans chaque partie (composant indépendant) simulent les actions entreprises sur les variables partagées (externes) par l'autre partie. Des obligations de preuve qui permettent de prouver que l'événement externe est bien une abstraction de l'événement qu'il simule dans l'autre partie sont alors nécessaires. Le raffinement des variables externes (partagées) dans chaque partie doit se faire de la même manière. Une telle restriction s'avère trop contraignante.
— L'approche de Butler consiste à décomposer un événement en plusieurs sous-événements. Cette approche peut être vue comme un raffinement et les possibles événements introduits peuvent raffiner l'événement *skip*.

D'autres réflexions inspirées des deux approches ci-dessus ont été menées pour la composition de modèles en Event-B. À l'exemple des travaux réalisés par Poppleton [135, 213] qui développa une approche par fusion des événements [26]. Cette fusion impose un non-déterminisme dans les gardes de chaque événement y participant. L'auteur démontra que le modèle de composition est bien un raffinement de chaque modèle impliqué dans la fusion. Cette propriété a été utilisée plus tard par [34] dans la composition des mécanismes d'authentification et d'établissement des clefs des protocoles de sécurité. La mise en oeuvre de cette technique a nécessité l'ajout d'événements externes inspirés de la technique d'Abrial pour fusionner les événements en cas de partage de variables dans le but est de préserver leur propriétés. Afin de garantir et de conserver des preuves d'invariance déjà prouvées par les composants d'un système, ces derniers doivent réaliser des interactions faiblement couplées [34]. C'est-à-dire que les propriétés devant être satisfaites par

l'environnement global doivent être aussi faibles que possible.

2.3.2 Patrons de conception

Les patrons de conception [8, 148, 149], sont composés de spécifications abstraites ainsi que de modèles qui raffinent cette spécification. Cette technique formalise des développements génériques, une solution générale, qui peuvent être utlisés systématiquement dans d'autres problèmes plus spécifiques avec comme soucis, en plus de réutiliser les constituants (contextes : constantes, machines : variables, invariants, événements, ...) d'un modèle abstrait ou des ses raffinements, la réutilisation des preuves formelles déjà réalisées. L'idée est de fournir une solution prédéfinie, puis de l'incorporer dans le développement ciblé en opérant quelques modifications et/ou instanciations.

Un premier développement générique doit être réalisé, puis des patrons peuvent être introduits afin de détailler la spécification générale. En résumé, cette construction peut être vue comme une construction de modèles et raffinements de modèles. Vient ensuite, l'étape d'instanciation du patron de conception. Le principe ici est de construire une spécification à partir du patron de conception déjà construit. Il s'agit de construire un premier modèle du problème à résoudre, puis d'appliquer des correspondances entre ce dernier et le patron de conception. Cette comparaison permettrait d'extraire des similitudes entre les parties impliquées. Ensuite, il faut incorporer les différents éléments dans le modèle cible. Il s'agit d'extraire et de renommer les différentes entités extraites : les élements du contexte, des machines (variables, noms d'événements, paramètres d'événements,... etc.), les modèles de raffinements.

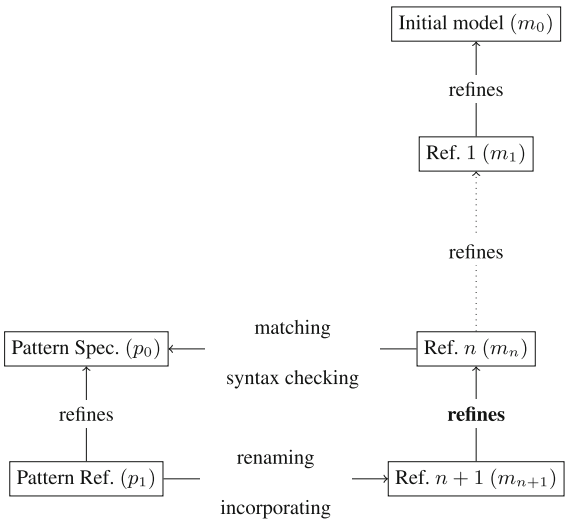

FIGURE 2.5 – Utilisation des patrons de conception dans Event-B [149]

Les étapes d'utilisation des patrons de conception peuvent être résumées comme suit :
— Établir une correspondance entre le problème et le modèle ;
— Vérification syntaxique de la correspondance pour voir si le problème est applicable ;
— Renommage des variables et événements dans le raffinement de motif qui conduirait à un conflit de noms, puisqu'il est possible d'instancier le motif plusieurs fois. Il est aussi possible de renommer les variables et les événements non conflictuels ;

— Enfin, vient l'étape d'incorporation du raffinement renommé du motif pour créer un raffinement du problème ;

Notons qu'un motif est un simple développement de modèles Event-B éventuellement liés par raffinement $(p_0, p_1 \ldots)$.

Lors d'un développement normal dans Event-B, au niveau du raffinement m_n, les développeurs peuvent associer une partie du modèle à la spécification de modèle p_0. À la suite de cette correspondance, le raffinement p_1 peut être incorporé pour créer le raffinement $m_n + 1$ de m_n (avec la possibilité de "renommer" pour éviter les conflits de noms).

2.4 Extension Temporelle pour le formalisme Event-B

Nous présentons dans ce qui suit un résumé de l'extension au temps définie dans [16, 17] basée sur TLA, et qui permet de prendre en considération les traces équitables par raffinement.

2.4.1 Extension TLA pour Event-B

L'auteur dans cette extension fait intervenir les modèles relationnels définis par l'auteur dans [107]. En premier lieu une structure de services fait intervenir des phases identifiées pour caractériser le service dont la description et la découverte est réalisée par raffinement.

Définition 2.4.1 (phase) *La phase est définie comme étant une étape algorithmique issue de la décomposition d'un service en étapes plus simples à analyser et à résoudre. Les phases sont synchronisées et coordonnées les unes par rapport aux autres.*

Cette définition met en exergue l'existence d'un ordre dans l'exécution des phases qui composent le système.

Un modèle relationnel abstrait $\mathcal{M}$ est une structure

$$(\mathcal{T}h(s,c), x, Val, \mathcal{I}nit(x), \{e_0, e_1, ...e_n\})$$

caratérisée par : $\mathcal{T}h(s,c)$ est le contexte qui définit les ensembles s et les constantes c par les propriétés statiques données par les axiomes et les théorèmes (Axm et Thm) ; un espace d'états Σ défini de l'ensemble des variables dans l'ensemble des valeurs possibles d'un modèle ($Var \rightarrow Val$) ; un ensemble d'états initiaux $\mathcal{I}nit$; un ensemble d'états terminaux pouvant être vide ; et une relation binaire $NEXT$ définie sur Σ. La relation $NEXT$ définit les transitions possibles entre états par les prédicats *avant-après* BA (Before-After) de la liste des événements $\{e_0, e_1, ..., e_n\}$. Une transition par la relation $NEXT(x, x')$ est relative à un événement dans la machine, et où x, x' correspond au valeurs des variables avant et après l'observation de l'événement en question. Il est possible d'avoir dans cette liste d'événements l'événement *skip*, qui correspond à la relation d'identité sur l'ensemble des valeurs $Id[Val]$, défini par $BA(skip)(x, x')$. La relation $NEXT$ relie les valeurs des variables avant (x) et après (x') l'observation d'un événement $NEXT \mathrel{\widehat{=}} e_0 \vee e_1 \vee ... \vee e_n$.

Ainsi, les propriété de sûreté sont définies déductivement en fonction d'invariants inductifs comme suit :

Propriété 2.4.1 (principe d'induction) *Étant donné $\mathcal{M}$ un modèle relationnel défini par $(\mathcal{T}h(s,c), x, Val, \mathcal{I}nit(x), \{e_0, e_1, ...e_n\})$. Une propriété $P(x)$ est une propriété de sûreté ssi, il existe une propriété d'état $\mathcal{I}nv(x)$, telle que : $\forall x, x' \in Val$:*

1. $\mathcal{I}nit(x) \Rightarrow \mathcal{I}nv(x)$;
2. $\mathcal{I}nv(x) \Rightarrow P(x)$;
3. $\mathcal{I}nv(x) \wedge NEXT(x, x') \Rightarrow \mathcal{I}nv(x')$

La sûreté est une propriété déduite d'invariants inductifs plus forts. Nous rappelons que les preuves sont données dans [107] et rappelées dans [16].

Sur cette base l'auteur définit les traces d'exécution engendrées par un modèle Event-B $\mathcal{M}$. Les traces sont définies comme une suite infinie de valeurs comme suit :

Définition 2.4.2 (traces) *Une trace engendrée par $\mathcal{M}$ est une suite de valeurs :*

$$\sigma_0, \sigma_1, ..., \sigma_i\sigma_{i+1}$$

où $\sigma_0 \in \mathcal{I}nit$ et $\forall i \in \mathbb{N}.(\sigma_i, \sigma_{i+1}) \in NEXT$ ou $\sigma_i = \sigma_{i+1}$.

L'ensemble des traces du modèle $\mathcal{M}$, noté $\mathcal{T}r(\mathcal{M})$, est un sous-ensemble de $\mathbb{N} \rightarrow Val$. Un événement correspond à l'observation d'une transition et est une substitution généralisée. L'équité définit les contraintes sur les comportements infinis du modèle $\mathcal{M}$. Ces contraintes garantissent que certains événements peuvent être observés s'ils sont observables à partir d'un certain moment. L'équité est définie sur les événements e de $\mathcal{M}$ comme suit :

— Son observabilité $ENABLED\langle e\rangle_x$ est : $ENABLED\langle e\rangle_x \mathrel{\widehat{=}} \exists y.BA(e)(x, y)$, ce qui signifie que la garde de e notée $grd(e)$, est vraie, et qu'il existe une valeur après y des variables du modèle $\mathcal{M}$, telle que les actions de e sont observables ;

— son observation $\langle e\rangle_x$ telle que : $\langle e\rangle_x \mathrel{\widehat{=}} BA(e)(x, x') \wedge (x \neq x')$. Ce qui signifie que les valeurs des variables sont modifiées

Deux types d'équité sont distingués pour un événement e :

— l'équité faible :

$$WF_x(e) \mathrel{\widehat{=}} \Diamond\Box ENABLED\langle e\rangle_x \Rightarrow \Box\Diamond\langle e\rangle_x$$

Si l'événement e est observable indéfiniment à partir d'un état, alors e est observé infiniment souvent ;

— l'équité forte :

$$SF_x(e) \mathrel{\widehat{=}} \Box\Diamond ENABLED\langle e\rangle_x \Rightarrow \Box\Diamond\langle e\rangle_x$$

Si l'événement e est observable infiniment souvent, alors e est observé infiniment souvent.

Ainsi, les conditions d'équité L qui conditionnent les exécutions du modèle $\mathcal{M}$ sont données comme la conjonction d'hypothèses d'équité faibles et fortes sur les événements de $\mathcal{M}$:

$$L \mathrel{\widehat{=}} WF_x(e_i) \wedge ... \wedge SF_x(e_j),$$

avec e_i et e_j sont des événements différents de $\mathcal{M}$.

Cette extension permet d'exclure les traces non-équitables par raffinement, et de spécificifier les propriétés de vivacité à l'aide de l'opérateur *leads to* ($\rightsquigarrow$). Un cadre temporel pour un modèle $\mathcal{M}$ est défini par la spécification TLA $\mathcal{S}pec(\mathcal{M})$ comme suit :

Définition 2.4.3 (spécification temporelle pour Event-B) *La spécification $\mathcal{S}pec(\mathcal{M})$ TLA est définie comme suit :*

$$\mathcal{I}nit(x) \wedge \Box[NEXT]_x \wedge L$$

avec :

- *$\mathcal{I}nit(x)$ le prédicat définissant les conditions initiales du modèle $\mathcal{M}$;*
- *$NEXT \equiv \exists e.e \in E \wedge BA(e)(x, x')$, où E est l'ensemble des événements de $\mathcal{M}$. La formule $\Box[NEXT]_x$ exprime que chaque paire d'états consécutifs satisfait la relation $NEXT$, et les valeurs des variables d'état x changent de valeurs ou restent les mêmes ;*
- *L est une conjonction d'équité fortes et faibles sur les événements $e \in E$.*

Une trace d'exécution équitable σ simule le modèle $\mathcal{M}$ sous les hypothèses d'équité L notée $\sigma \in tfair(\mathcal{M}, L)$, ssi, $\sigma \models \mathcal{S}pec(\mathcal{M})$. Cela signifie que soit $\sigma \in \mathcal{I}nit$ et que pour tout $i \in \mathbb{N}, (\sigma_i, \sigma_{i+1}) \in NEXT$ ou $\sigma_i = \sigma_{i+1}$ et que σ respecte les hypothèses d'équité L.

La vivacité est une propriété exprimée par le biais de l'opérateur $\rightsquigarrow$ pour le modèle $\mathcal{M}$, où la formule $P \rightsquigarrow Q$ est définie par :

$$\Box(P \Rightarrow \Diamond Q)$$

et signifie qu'à chaque fois que la propriété P est vérifiée, la propriété Q est vérifiée ou le sera *fatalement* dans le futur.

L'ensemble des traces équitables sous les hypothèses d'équité L du modèle $\mathcal{M}$ est noté $tfair(\mathcal{M}, L)$.

Définition 2.4.4 (formule temporelle $P \rightsquigarrow Q$) *Sous les hypothèses d'équité L, le modèle $\mathcal{M}$ satisfait $P \rightsquigarrow Q$, si pour toutes les traces $\sigma \in tfair(\mathcal{M}, L)$, la propriété suivante est vérifiée :*

$$\forall i.(i \geq 0 \wedge P(\sigma_i) \Rightarrow \exists j.(j \geq i \wedge Q(\sigma_j)))$$

Dans ce cadre un service est défini comme un événement *service-as-event* (cf figure 2.6). Cette approche utilise les propriétés de l'opérateur $\rightsquigarrow$ comme la transitivité, la disjonction, la déduction etc pour guider le raffinement par décomposition des services en *phases* ou *étapes* locales aux processus qui composent le système. La synchronisation et la coordination des phases sont assurées par les propriétés de vivacité qui introduisent la notion de *contrôle* dans un modèle Event-B.

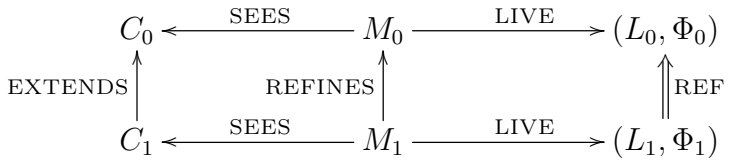

FIGURE 2.6 – La méthodologie du paradigme service-as-event

Ainsi, les relations REF et LIVE illustrées sur la figure 2.6 enrichissent la sémantique des modèles Event-B. LIVE signifie que la spécification de la machine M_0 est donnée par : $\mathcal{S}pec(M_0) \mathrel{\widehat{=}} \mathcal{I}nit(x) \wedge \Box[NEXT] \wedge L_0$, et chaque propriété $P \rightsquigarrow Q$ de Φ_0 est dérivable à partir de $\mathcal{S}pec(M_0)$, c'est-à-dire : $\mathcal{S}pec(M_0) \vdash (P \rightsquigarrow Q)$. REF est définie comme suit : chaque propriété de vivacité $P_1 \rightsquigarrow Q_1$ de Φ_1 est dérivable à partir de $\mathcal{S}pec(M_1)$

($\mathcal{S}pec(M_1) \vdash (P_1 \rightsquigarrow Q_1)$), et le modèle M_1 raffine M_0, et la spécification $\mathcal{S}pec(M_1)$ de la machine M_1 ainsi que l'ensemble des propriétés de vivacité Φ_1 associées permettent de dériver toutes les propriétés $P \rightsquigarrow Q$ de Φ_0, ce qui signifie que : $\mathcal{S}pec(M_1), \Phi_1 \vdash (P \rightsquigarrow Q)$.

Dans ce qui suit, les notations respectives, $P \xrightarrow{e} Q$ et $P \overset{e}{\Rightarrow} Q$ indiquent quand P est vraie et conduit à Q sous l'hypothèse d'équité faible posée sur l'événement e, respectivement, sous l'hypothèse d'équité forte posée sur l'événement e. définies comme suit :

Définition 2.4.5 ($P \xrightarrow{e} Q$) *Soit un modèle $\mathcal{M}$ définissant les variables x, un invariant $\mathcal{I}nv(x)$ sur ces variables, un événement e et d'autres événements re_0, re_1, re_n. Les événements $re_i (0 \leq i \leq n)$ modélisent soit d'autres actions possibles du système, soit des actions de l'environnement de ce système. Soit P et Q deux propriétés sur les valeurs prises par les variables x. P est la condition d'observation de l'événement e et Q est la propriété établie après l'observation de ce dernier. $P \xrightarrow{e} Q$ est l'expression de la propriété de vivacité $P \rightsquigarrow Q$ par l'événement e du modèle $\mathcal{M}$. Les événements e et $re_i (0 \leq i \leq n)$ doivent établir les propriétés suivantes :*

- *e doit satisfaire :*
  - *la propriété $P(x) \wedge BA(e)(x, x') \Rightarrow Q(x')$,*
  - *la condition de faisabilité : $P(x) \Rightarrow (\exists y.BA(e)(x, y))$*
- *Les événements re_i doivent satisfaire les propriétés : $P(x) \wedge BA(re_i)(x, x') \Rightarrow (P(x') \vee Q(x'))$, ce qui veut dire que l'observation d'un événement re_i dans un état satisfaisant la propriété P, ne rend pas fausse la propriété P, la condition d'observation attendue par l'événement e ou conduit à Q.*
- *Une condition d'équité faible est posée sur l'événement e ($WF_x(e)$).*

Définition 2.4.6 ($P \overset{e}{\Rightarrow} Q$) *Soit un modèle $\mathcal{M}$ défini ave ses variables x, un invariant $\mathcal{I}nv(x)$, un événement e et d'autres événements $re_0, re_1, ...re_n$. Soit P et Q deux propriétés sur les valeurs prises par les variables x de ce modèle. L'événement e modélise une action du système et les événements re_i $(i \leq i \leq n)$ modélisent soit d'autres actions possibles du système, soit des actions de l'environnement. Un ou plusieurs événements peuvent conduire à une propriété R différente de P et de Q lorsque P est vraie. P est la condition d'observation de l'événement e et Q est la propriété satisfaite après l'observation de e. $P \overset{e}{\Rightarrow} Q$ est l'expression de la propriété de vivacité $P \rightsquigarrow Q$ par l'événement e de $\mathcal{M}$. Les événements e et re_i $(0 \leq i \leq n)$ doivent satisfaire les conditions suivantes :*

- *e doit établir :*
  - *la propriété : $P(x) \wedge BA(e)(x, x') \Rightarrow Q(x')$;*
  - *la condition de faisabilité : $P(x) \Rightarrow (\exists y.BA(e)(x, y))$;*
- *Les événements re_i doivent établir les propriétés suivantes :*
  - *il existe un événement re_j, tel que $P(x) \wedge BA(re_j)(x, x') \Rightarrow R(x')$ conduisant à un état satisfaisant R, telle que $R \neq P \wedge R \neq Q$, et $R \rightsquigarrow P$ satisfaite par $\mathcal{M}$;*
  - *re_j satisfait aussi la condition de faisabilité : $P(x) \Rightarrow (\exists y.BA(re_j)(x, y))$;*
  - *tous les événements re_i, incluant re_j satisfont la propriété $(P(x) \vee R(x)) \wedge BA(re_i)(x, x') \Rightarrow (P(x') \vee R(x') \vee Q(x'))$;*
- *Une condition d'équité forte doit être posée sur l'événement e ($SF_x(e)$).*

La relation REF est définie similairement au raffinement dans Event-B, où les propriétés de vivacité Φ_0 du modèle abstrait $\mathcal{M}_0$ sont détaillées à l'aide des règles d'inférence de l'opérateur $\rightsquigarrow$. Cette relation est basée sur la déduction et la preuve des propriétés de vivacité de Φ_0 à partir de celles de Φ_1. Cela signifie que toute propriété de vivacité $P \rightsquigarrow Q$ de Φ_0 peut être dérivée de Φ_1 et de la machine concrète $\mathcal{M}_1$ ayant les variables y un prédicat d'initialisation $\mathcal{I}nit_1(y)$, un invariant de collage $J(x, y)$. L'étape suivante

détermine une hypothèse d'équité L_1 du modèle concret $\mathcal{M}_1$ de sorte que les propriétés de Φ_1 , permettent de dériver les hypothèses abstraites de Φ_0. La relation REF exprime que le modèle concret $\mathcal{M}_1$, sous ses hypothèses d'équité L_1, satisfait toutes les propriétés de vivacité $P \rightsquigarrow Q$ abstraites Φ_0.

La première technique de raffinement est appelées *superposition* et est inspirée de UNITY [73]. Cette technique est résumée comme suit :

— La variable abstraite xa d'un modèle abstrait $\mathcal{M}_0$ n'est pas modifiée par les nouveaux événements $ec_1, ec_2, ..., ec_l$ introduits lors du raffinement de $\mathcal{M}_0$ en $\mathcal{M}_1$. Les propriétés d'équité dans ce cas, restent identiques. Si un événement ae du modèle abstrait $\mathcal{M}_0$ était défini avec une équité donnée $SF_x(ae)$ (respectivement $WF_x(ae)$), alors le nouvel événement étendu ec par h est défini avec une hypothèse $SF_y(ec)$ (respectivement $WF_y(ec)$), où $y = x$.

La seconde technique de raffinement est basée sur le raffinement de données, où l'espace de données change. La relation entre les variables x du modèle abstrait, et les variables y du modèle concret est définie par une fonction r, telle que $r(x) = y$. Cette fonction est bijective et peut remplacer le nom par un autre, à condition que les nouveaux événements concrets ec soient équivalents aux anciens abstraits ae à r près. Une relation d'équivalence doit être définie entre ae qui modifie les variables x, et l'événement concret ec modifiant les variables g à f $ae =_f ec$, suivant les cas :

1. en remplaçant l'action abstraite de l'événement ae par une action concrète équivalente ;
2. en remplaçant les gardes et actions abstraites par des gardes et des actions équivalentes ;
3. en remplaçant les paramètres t_a d'un événement abstrait ae par des paramètres concrets t_c qui doivent être reliés par une fonction bijective h, telle que $t_c = h(t_a)$.

Nous disposons alors de :
— $BA(ae)(x, x') \equiv BA(ec)(y, y')$, où ec est obtenu en remplaçant une occurence de x par y ;
— $r(x') = y'$;
— $r(x) = y$.
Ces contraintes donnent la fonction de raffinement définie comme suit

Définition 2.4.7 (fonction de raffinement) *Une fonction de raffinement $f : \Sigma_1 \to \Sigma_0$ est basée sur les conditions suivantes :*
— *La fonction de raffinement préserve l'état visible d'un composant ;*
— *Les états initiaux de $\mathcal{M}_1$ sont associés par f aux états initiaux de $\mathcal{M}_0$;*
— *Chaque événement ec_1 de $\mathcal{M}_1$ raffine un événement ae_0 du modèle abstrait $\mathcal{M}_0$ ou est un bégaiement de $\mathcal{M}_0$. Et l'on déduit qu'avec f, $ec_1 \Rightarrow ae_0$;*
— *$f(L_1) \subseteq L_0$*

Et le raffinement d'une spécification est défini par l'existence d'une fonction de raffinement exprimant que $\mathcal{M}_1$ implémente $\mathcal{M}_0$ qui signifie l'inclusion des traces de $\mathcal{M}_1$ (au bégaiement près) dans les traces de $\mathcal{M}_0$.

Il faut rappeler que cette extension est dédiée pour les algorithmes distribués.

2.5 Outils Event-B utilisés

Nous utilisons l'outil Rodin [4, 6] développé dans le cadre d'un projet européen. Cette plateforme est développée avec les environnements Eclipse. Pour décharger les obligations de preuve, des prouveurs automatiques et interactifs ont été intégrés dans cette plateforme comme les SMT [94]. L'outil utilisé pour la validation en Event-B et qui est intégré dans la plateforme Rodin comme plugin est appelé ProB [38]. Cet outil offre un environnement intégré pour l'édition, la vérification, l'animation et la vérification de modèles Event-B. C'est un avantage considérable pour le concepteur, car il lui permet de basculer entre les différents outils pour valider, déboguer et améliorer les modèles conçus et ceci est réalisé par raffinement.

2.6 Synthèse

Il faut dire que le langage du formalisme Event-B est très expressif puisqu'il se base sur la logique des prédicats, et où les spécifications sont réalisées par raffinement. Néanmoins, cette expressivité rend aussi difficile la tâche de modélisation. D'autres parts, le besoin d'un niveau sémantique bien défini dans les applications pour les modèles Event-B est nécessaire. En effet, un système formel définissant des règles d'inférences qui permettent de mieux comprendre la description du système à spécifier requiert une représentation et des structures standardisées du sens qui établissent des compréhensions entre personnes et logiciels.

Les travaux exposés dans le chapitre précédent ont montré que l'utilisation des ontologies de domaine permettent : d'approfondir la compréhension du produit logiciel par les concepteurs du système cible par le biais d'annotations par exemple ; de justifier les choix conceptuels et d'effectuer des raisonnements et des déductions supplémentaires liés à la cohérence pour permettre un bon typage ; de valider une conception ou un modèle par le biais de configurations valides. Aussi l'association des ontologies au formalisme Event-B a été abordée. Toutefois, ces travaux ne traitent pas les aspects dynamiques d'Event-B et le raisonnement ontologique et ne traduisent pas la partie assertionnelle (A-Box) d'une ontologie, d'où leur rigidité. Nous verrons dans le chapitre suivant que les propriétés dynamiques peuvent aussi être extraites d'ontologies, que le raisonnement déductif ontologique peut correspondre à l'implication contextualisée en logique dans le formalisme Event-B si l'on considère la composante assertionnelle d'une ontologie. De plus, ces travaux ne donnent pas de précisions sur la manière dont les propriétés du domaine peuvent être exploitées, ne distinguent pas le contexte et ne montrent pas comment exploiter par raffinement des contraintes du domaine comme nous le verrons dans nos contributions.

La vision défendue dans cette thèse est qu'une ontologie doit être *associée par intégration* au modèle définissant le système à concevoir pour pouvoir exploiter le raisonnement ontologique. Et ce n'est qu'en automatisant partiellement ce processus d'intégration qu'il est possible d'exploiter les ontologies de domaine pour faire de l'*extraction par raffinement* dans la partie dynamique qu'est la machine Event-B. Nous ne cherchons donc pas à traduire une ontologie en modèles Event-B, mais à exploiter la conceptualisation et les mécanismes de raisonnement ontologique afin de mieux guider le concepteur dans sa démarche de modélisation, de vérification et de validation justifiées. Nous montrons en pespectives de nos travaux qu'il est possible d'extraire ces propriétés du domaine par raf-

finement, mais en adoptant des stratégies préalablement définies en fonction du domaine et du contexte. Cette vision donnerait lieu à un nouveau formalisme basé sur l'analyse de concept logique. Nous tentons de répondre aux interrogations suivantes : *1)* Niveau d'abstraction et ou raffinement ou grain de connaissances (hiérarchisation) ? *2)* Raisonnement a priori ou a posteriori ? *3)* Quelles connaissances et quelles propriétés du domaine et du contexte et pour quelles vérifications ? Nous étudions les répercussions sur les obligations de preuve.

Les chapitres qui suivent définissent un nouveau mécanisme de preuve en Event-B basé sur les dépendances exposées dans le chapitre précédent (cf. chapitre 1), et des stratégies pour l'extraction de propriétés du domaine (cf. chapitre 3) après avoir défini une nouvelle logique avec contraintes du domaine basées sur les logiques de description. Ces techniques sont ensuite appliquées aux patrons de conceptions (cf chapitre 4) pour les systèmes de vote. Nous montrons l'intérêt d'une telle démarche sur les deux plans, à savoir la vérification et la validation en Event-B.

Chapitre 3

Contextualisation et intégration de contraintes du domaine en Event-B

Sommaire

3.1 Introduction 76
3.2 Modélisation du contexte dans Event-B 78
3.3 Des modèles Event-B dépendants 81
3.3.1 Étude de cas : système de gestion ERP 86
3.3.2 Modèle Event-B équivalent aux modèles dépendants 91
3.4 Stratégies d'intégration de contraintes du domaine pour la conception de systèmes 93
3.4.1 Exemple de motivation 93
3.4.2 Logique du premier ordre avec les contraintes du domaine pour Event-B : BL-DC 95
3.4.3 Contextualisation versus. raffinement 99
3.4.4 Application à l'étude de cas 102
3.4.5 Analyse et discussions 114
3.5 Travaux en annexe 115
3.6 Synthèse 117

« La vie n'est facile pour aucun de nous.
Mais quoi, il faut avoir de la persévérance,
et surtout de la confiance en soi.
Il faut croire que l'on est doué pour quelque chose,
et que, cette chose, il faut l'atteindre coûte que coûte. »
- Marie Curie

3.1 Introduction

Ce chapitre aborde nos travaux sur l'importance du contexte et de la contextualisation, de la conceptualisation de domaine dans le cadre de la conception des systèmes, et en particulier, dans la preuve de correction de systèmes reposant sur les méthodes de vérification et de validation formelles. Pourquoi le contexte ? Comment exploiter le domaine et comment la conceptualisation d'un domaine peut aider à concevoir un modèle Event-B ? et leurs impacts sur le raisonnement pour la preuve reposant sur les formalismes logiques ? Les travaux exposés dans le présent chapitre ont fait l'objet des publications [165].

Nous avons montré dans ce qui précède que la notion de contexte n'est pas absolue et est relative à une activité, une situation, un focus, etc. Plus précisément, le contexte se définit par rapport à un concept intentionnel i.e., l'*action* [28, 92]. Le contexte est considéré comme étant la connaissance minimale qui affecte le comportement de toute action liée à ce contexte dans une situation donnée [92]. Cette connaissance contraint la sémantique d'une action.

Définir une sémantique pour le contexte est indispensable, car une région arbitraire de l'espace-temps n'a pas de *signification intrinsèque* [155]. Par ailleurs, pour avoir des propositions ayant du sens, la logique doit avoir une théorie de référence qui détermine comment les constantes et les variables sont associées aux choses dans l'univers du discours [237], d'où l'intérêt d'avoir une sémantique du contexte. Plusieurs travaux ont donné des sémantiques pour le contexte. Les situations dans les travaux de Barwise et Perry [31] en 1983 définissent une sémantique ou une interprétation pour le contexte comme une théorie qui relie le sens et l'interprétation des propositions à des fragments plus faciles à gérer i.e., les situations. Chaque situation définit une configuration des aspects du monde dans une région limitée de l'espace et du temps. En théorie des situations, la notion du contexte est définie comme un type de situation qui supporte à la fois des infons factuels et des contraintes. Une telle représentation est exprimée par une dépendance entre situation et contraintes.

La plupart de ces approches tentent de représenter le contexte, pas de le modéliser [60]. Rappelons que Brézillion [60] dinstingue *modélisation* et *représentation* du contexte. Le but d'un modèle est de donner une image cohérente du contexte qui peut être utilisé pour expliquer et prédire par simulation. Le but d'une représentation du contexte est seulement de rendre compte de ce qui est observé indépendamment de la façon dont cela est fait. Un modèle est incarné dans une théorie, alors qu'une représentation dépend du formalisme de représentation choisi.

Nous étudions la contextualisation et le contexte de preuve dans le formalisme Event-B. Au niveau de la preuve, le contexte permet d'établir et de valider des relations de confiance qui fournissent des interprétations valides dans un domaine précis à des fins de certification. La méthode Event-B est un cadre de structuration qui spécifie, au travers

de son langage, une description formelle des systèmes *réactifs*. L'usage du contexte fait dans nos travaux suit le point de vue de l'intelligence artificielle. L'intelligence artificielle fait de la représentation de l'information et plus précisément, de ce qui caractérise l'information elle-même, la principale préoccupation ayant une *description intensionnelle* du contexte [28]. Plutôt que de considérer les contextes comme des objets formels à la McCarthy [183], nous adoptons l'approche de Barlatier [92] qui considère le contexte comme connaissance minimale faisant partie d'un système centré sur la preuve, en raisonnant sur les modèles Event-B. Les *connaissances contextualisées* sont des connaissances qui sont *explicitement* prises en compte dans la résolution de problèmes. La *connaissance contextuelle* est une connaissance utilisée *implicitement* dans la résolution de problèmes. Plusieurs questions sont posées : quelles sont les connaissances minimales pour faire des preuves ? Comment est représenté et comment est modélisé le contexte dans le formalisme Event-B ? Quelles sont les connaissances explicites et les connaissances implicites ? Comment sont explicitées les connaissances de preuve ? Quel est le rôle que joue le raffinement vis-à-vis des connaissances liées au domaine et au contexte ?

Nous proposons de répartir le contexte en Event-B en *contraintes*, *hypothèses* et *dépendances*, selon que les connaissances sont *acquises*, *admises* ou *déduites* dans un système de preuve. Nous définissons ce nouveau mécanisme de dépendance entre les modèles Event-B, où la propriété de terminaison stable est une condition nécessaire pour son établissement. Une correspondance avec la théorie des situations est établie : une situation correspond à l'état du système en Event-B et les faits établissent une interprétation logique qui définit l'état du système. Ceux-ci représentent les valeurs dans les ensembles (le contenu des ensembles Event-B), les valeurs des constantes, ainsi que les valeurs des variables définies dans les machines. Les contraintes correspondent aux propriétés statiques définies dans les contextes Event-B. Ce mécanisme se définit alors par une combinaison d'états dans les machines Event-B et de propriétés statiques définies dans les contextes Event-B. La dépendance est une relation qui prend des valeurs à partir des faits existants dans une situation qui nécessite la définition d'une nouvelle obligation de preuve. La section 3.2 présente une description détaillée du contexte, donne un cadre formel du mécanisme de dépendances dans ce formalisme.

Les spécifications des comportements dans le formalisme Event-B suivent une approche assertionnelle décrite au moyen de contextes Event-B et de machines Event-B. Le contexte est codé selon le formalisme de représentation de manière implicite [60]. Le formalisme de représentation des connaissances adopté dans Event-B est la logique du premier ordre munie de la théorie des ensembles. Puisque qu'il faut donner une sémantique universelle à une logique donnée, celle-ci est rendue indépendante de tout contexte [118]. Alors que les *connaissances logiques* portent sur tous les contextes possibles, et les *connaissances factuelles* portent sur un contexte donné qui correspond au contexte de données (aux faits), les *connaissances du domaine* portent sur une famille de contextes, ces connaissances se situent donc à un niveau intermédiaire entre la logique et les faits (données). Les auteurs dans [201] ont montré que la logique du premier ordre est une *logique contextuelle*. Par ailleurs, le raisonnement sur des systèmes fermés dans le formalisme Event-B nécessite l'introduction des environnements Γ dans les obligations de preuve. Les obligations sont donc des formules logiques *contextuelles*.

Par ailleurs, selon McCarthy l'utilisation des connaissances dans différents contextes, requiert un processus de *décontextualisation*, permettant d'abstraire un ensemble de contextes dans un contexte plus général qui couvre les contextes initiaux. Le processus inverse à l'abstraction est le raffinement. Le raffinement est au coeur de la méthode Event-B qui

permet à la fois de construire des systèmes corrects et de résoudre des conflits liés à la conception des systèmes. Ce processus contextualise donc les modèles Event-B construits. La section 3.4 définit une logique avec les contraintes du domaine pour Event-B en se basant sur les logiques de description, établit des stratégies d'intégration par raffinement de la sémantique des propriétés liées à la fois aux connaissances du domaine et du contexte. Les sections 3.5 et 3.6 présentent respectivement, les travaux en lien avec la problématique abordée dans le présent chapitre, et une synthèse.

3.2 Modélisation du contexte dans Event-B

L'activité de vérification établit qu'un système répond à certaines propriétés. Un système est décrit par sa spécification qui peut être abstraite ou concrète. La sémantique d'une action est nécessaire mais non suffisante, car en terme de preuve les garanties des propriétés de sûreté sont que les actions dans le système ne doivent pas produire quelque chose de mauvais, ce qui constitue la définition même d'un système *sûr*. En outre, en plus d'être descriptive, une spécification formelle doit également être prescriptive. La prescription des comportements est réalisée par la spécification d'invariants qui sont évalués à chaque changement d'état en prennant en compte ces connaissances minimales i.e., le contexte. Suivant les travaux de Brézillon, le contexte est associé à un centre d'attention ou au focus [60]. Notre focus du contexte est la preuve et l'intention correspond aux actions définies dans le système de preuve qui concrétisent l'objectif du système.

Les connaissances [13] sont toujours liées à un domaine $\mathcal{D}$. Rappelons que le domaine est l'univers du discours auquel le système est associé. Les connaissances dans le formalisme Event-B sont toutes les informations et les données qui s'illustrent par le biais des définitions données dans les contextes ainsi que les machines Event-B. Les définitions sont données par les éléments statiques (ensembles, constantes, et axiomes dans les contextes Event-B), ainsi que les événements et les invariants dans les machines Event-B. Rappelons aussi que la connaissance (cf. définition 1.2.3) est toujours liée à deux aspects : sa finalité qui correspond à l'objectif qu'est la preuve dans notre cas, et sa capacité générative pour produire ou générer de nouvelles propriétés et inférences dans notre cas.

La *connaissance minimale* est toute information de base qui permet de caractèriser un modèle Event-B pour lui donner une sémantique, un sens et une interprétation dans un domaine particulier.

Définition 3.2.1 (Contexte) *Étant donné le système de preuve Event-B, le contexte Cxt est la connaissance minimale qui affecte l'observation de toute action dans le système satisfaisant les propriétés de sûreté dans une situation donnée.*

Pour notre part, les faits établissent une interprétation logique qui définit l'état du système représenté par les variables i.e., une situation. Dans le formalisme Event-B une situation représente l'état du système et les faits représentent les valeurs dans les ensembles (le contenu des ensembles Event-B), les valeurs des constantes ainsi que les valeurs des variables définies dans les machines. Dans ce formalisme les changements d'état sont

13. Dans le formalisme Event-B, nous distinguons les connaissances de représentation du contexte qui sont liées au langage de formules utilisé qu'est la logique du premier ordre et la théorie des ensembles, et les connaissances de modélisation du contexte qui sont liées aux caractéristiques du système à modéliser. Les connaissances indiquées ici concernent le second type (contexte de modélisation).

réalisés par l'observation des événements observables permettant ainsi de mettre à jour ces les valeurs d'état du système.

Les connaissances à modéliser pour faire des preuves sont acquises lorsqu'il s'agit de contraintes, supposées, ou déduites. Les connaissances peuvent donc faire référence à des contraintes, des hypothèses, ou aux faits qui peuvent être déduits suite à l'exécution d'événements. Le contexte se décline ainsi en *hypothèses*, *contraintes* ou une combinaison de situations et de contraintes. Cette combinaison de situations et de contraintes exprime une *dépendance* entre les parties impliquées dans le système.

La méthode Event-B est un cadre de structuration qui assure au travers de son langage une description formelle des systèmes. Cette description inclut également la spécification de l'environnement dans lequel ce système est plongé. Les hypothèses sont faites sur l'environnement du système :

— soit sur le comportement des parties corrompues en introduisant des variables, des événements et des propriétés dans les machines Event-B qui modélisent les différentes intrusions que le système prend en compte. Cette modélisation comprend également une prescription via les propriétés de sûreté prévue comme contre mesure ;

— ou bien sur les contraintes. Ces dernières sont définies dans les contextes Event-B.

La preuve dans le formalisme Event-B porte sur l'étude des propriétés correspondant à des actions *observables*, i.e., sur la visibilité des événements dans le système. Ce sont donc des connaissances déduites dans un modèle Event-B qui résultent des échanges entre chaque partie dans le système qui permettent d'exprimer cette dépendance.

Partant de ce constat, en tant que connaissances minimales pour un système de preuve, nous pouvons établir la répartition suivante du contexte de preuve dans le formalisme Event-B :

Le contexte comme contraintes - La sémantique ou la signification d'un modèle et de ses propriétés dépend du contexte de sa conceptualisation. L'aspect sémantique d'une conceptualisation est le point de vue à partir du modèle, de ses propriétés et de son contexte qui aborde les fondements du système selon la forme, la structure et les faits existants d'un domaine particulier. On appelle ces structures de faits des contraintes. Elles correspondent au différents concepts, rôles et valeurs des attributs de concepts du domaine en question. Les contraintes peuvent être physiques i.e., qui existent dans la nature comme par exemple la structure du corps humain, ou bien elles sont entièrement conçues par l'humain. A titre indicatif, la monnaie, les différents types de vote (vote majoritaire, vote à préférences, ...) qui permettent de calculer et de donner un sens aux résultats d'une élection sont des contraintes qui définissent le contexte dans lequel la preuve est réalisée. Les contraintes sont exprimées en Event-B par des structures définies dans le contexte, à savoir, des ensembles, des constantes ainsi que des axiomes et théorèmes qui établissent leur typage ainsi que leurs caractéristiques ou propriétés statiques. On notera ces contraintes sous la forme de $C\text{-}Ax$ et $C\text{-}Thm$ pour illustrer les axiomes et les théorèmes dans un contexte d'un modèle Event-B.

Le contexte comme hypothèses - Les hypothèses sur l'environnement situent également les systèmes à vérifier. Celles-ci sont établies par les concepteurs. Les hypothèses ne sont pas toujours vérifiées mais seulement supposées ou admises. Elles se traduisent par

une restriction sur les capacités des comportements corrompus (celles-ci sont exprimées dans les machines Event-B) et les contraintes qui peuvent être définies dans les contextes Event-B. Dans l'exemple des protocoles de vote, les objectifs de chaque protocole de vote ainsi que le type des attaques décrites déterminent si les preuves sont réussies ou non. Le recours aux primitives cryptographiques, afin d'assurer une construction de protocoles de vote sûrs garantissant les propriétés exigées repose sur des hypothèses relatives aux difficultés à réaliser des calculs ou à résoudre certains problèmes [121]. Dans le contexte du système Belenios [83] dont le développement est basé sur une version d'Helios [84] avec des *Credentials* (accréditations), le schéma de preuves est construit sous l'hypothèse que les autorités responsables de l'enregistrement des signatures et les tableaux d'affichage des bulletins ne peuvent être malhonnêtes simultanément.

Nous avons montré [128] que ces types d'hypothèses peuvent être exprimés dans des modèles Event-B, éventuellement par raffinement, en ajoutant de nouvelles variables, événements et propriétés dans les machines. Les hypothèses sur les contraintes sont des restrictions sur les éléments statiques définis dans les contextes Event-B. De nombreux travaux sur les protocoles de vote ont réussi à prouver les propriétés en formulant des hypothèses sur le format des bulletins de vote. Ainsi, les preuves ne sont possibles qu'en utilisant les valeurs binaires des bulletins de vote (c'est-à-dire 1/0, oui / non, ...). Ces restrictions sont exprimées en ajoutant de nouvelles constantes et axiomes dans les contextes Event-B. Nous notons les hypothèses sur les contraintes dans les contextes Event-B par *H-Ax* et *H-Thm* pour illustrer les axiomes et les théorèmes supplémentaires dans les contextes d'un modèle Event-B. Les hypothèses sur l'environnement sont illustrées par des variables, des propriétés invariantes et des événements supplémentaires dans les machines.

Le contexte comme dépendances - Les travaux de McCarthy montrent que les prédicats qui définissent le contexte sont paramétrés par des situations. En outre, le point de vue ontologique démontre qu'un contexte est un « *moment universel* » [14] [105]. Barlatier [28] a établi une correspondance avec la théorie des ensembles, à savoir, les concepts ontologiques (i.e. les types) sont interprétés comme des ensembles d'éléments et les rôles comme des relations entre les éléments de ces différents concepts. Partant de ce constat, et en faisant le parallèle avec la théorie des situations, où les situations sont interprétées comme des états dans le formalisme Event-B, et les contraintes comme les éléments définis dans les contextes Event-B, nous pouvons exprimer ce type de contexte par une relation de dépendances entre les modèles Event-B qui résultent de la combinaison des situations (des valeurs des états dans le formalisme Event-B) et de contraintes exprimées dans les contextes Event-B, en confondant ce *moment universel* à la terminaison qui nécessite la stabilité. Ce type de contexte peut parfaitement être illustré par les systèmes de vote.

La mise en œuvre du processus de vote passe principalement par trois phases : *1) La phase de préparation du vote* : à l'issue de cette phase, sont établies les listes des candidats nominés sur lesquels portera le choix des électeurs, ainsi que les électeurs enregistrés ayant le droit de voter. L'établissement de ces listes dépend des lois locales de chaque pays ; *2) La phase d'enregistrement du vote* : cette phase permet aux électeurs tous éligibles d'exprimer leur choix sur la base de la liste des représentants nominés lors de la phase précédente. Ainsi, par l'utilisation de la liste électorale l'électeur doit s'authentifier comme un électeur admissible et déposer son vote individuellement ; *3) La phase de dépouillement du vote* :

14. un moment est un individu qui dépend existentiellement d'autres individus.

elle couvre le comptage et le résultat des rapports issus de la phase d'enregistrement. À la fin de ces phases, l'interprétation des résultats dépend de la méthode de vote adoptée qui contraint la décision finale.

Les contraintes dépendantes sont exprimées par sous la forme de *D-Ax* et *D-Thm* pour illustrer les axiomes et les théorèmes dépendants dans les contextes d'un modèle Event-B.

3.3 Des modèles Event-B dépendants

Dans ce qui suit, nous adoptons les notations de [107, 16], qui expriment des modèles relationnels et leur extension aux aspects temporels suivant le formalisme TLA.

Soient deux modèles Event-B $\mathcal{M}_i$, avec $i \in 1..2$ définis par :

$$\mathcal{M}_i \mathrel{\widehat{=}} (\mathcal{T}h_i(s_i, c_i), x_i, Val_i, \mathcal{I}nit_i(x_i), \{e_{i0}, ..., e_{in}\})$$

- $Ax(s_i, c_i)$ sont les axiomes définis sur les ensembles s_i et les constantes c_i ;
- $Thm(s_i, c_i)$ sont les théorèmes définis sur les ensembles s_i et les constantes c_i ;
- $\mathcal{T}h_i(s_i, c_i) \mathrel{\widehat{=}} Ax(s_i, c_i) \cup Thm(s_i, c_i)$ est le contexte qui définit les éléments et les propriétés statiques d'un modèle $\mathcal{M}_i$;
- l'espace d'états Σ_i est l'ensemble des valeurs possibles Val_i des variables $x_i : x_i \rightarrow Val_i$; Nous noterons $\mathcal{C}_i(s_i, c_i)$ pour exprimer les contraintes dépendantes définies dans un contexte B exprimant les axiomes $D\text{-}Ax(s_i, c_i)$ et les théorèmes $D\text{-}Thm(s_i, c_i)$ dépendants ;
- un ensemble d'états initiaux $\mathcal{I}nit_i$;
- un ensemble d'événements $\{e_{i0}, ..., e_{in_i}\}$.

$$\mathcal{S}pec(\mathcal{M}_i) \mathrel{\widehat{=}} \mathcal{I}nit_i(x_i) \wedge \Box[NEXT_i]_{x_i} \wedge L_i$$

avec

- $NEXT_i \mathrel{\widehat{=}} \exists e.e \in \{e_{i0}, ..., e_{in_i}\} \wedge BA(e)(x_i, x_i')$;
- x_i' est la valeur des variables après l'observation de l'événement e_i ;
- $\Box[NEXT_i]$ signifie que toute paire d'états satisfait la relation $NEXT$ et que les valeurs des variables restent les mêmes ou changent ;
- L_i est une conjonction de contraintes d'équité faibles et fortes sur des combinaisons d'événements e_i du modèle $\mathcal{M}_i$;

La spécification $\mathcal{S}pec(\mathcal{M})$ d'un modèle $\mathcal{M}$ décrit l'ensemble des traces équitables qui respectent les hypothèses d'équité L, et qui définissent les comportements autorisés du modèle $\mathcal{M}$.

Une trace d'exécution équitable $tfair(\mathcal{M}_i, L_i)$ engendrée par un modèle $\mathcal{M}_i$ est définie par une suite infinie de valeurs :

$$\mathcal{T}race(\mathcal{M}_i) \mathrel{\widehat{=}} \{\sigma_0\sigma_1\sigma_2...\sigma_j\sigma_{j+1}... \mid \sigma_0 \in \mathcal{I}nit_i \wedge \forall j \in \mathbb{N}.(\sigma_j, \sigma_{j+1}) \in \ NEXT_i \vee \sigma_j = \sigma_{j+1}\}$$

satisfaisant la spécification $\mathcal{S}pec(\mathcal{M}_i)$ et notée par $\sigma \vDash \mathcal{S}pec(\mathcal{M}_i)$.

La configuration initiale d'un modèle Event-B se définit par rapport à un domaine $\mathcal{D}$ comme suit :

Définition 3.3.1 (configuration initiale d'un modèle Event-B) *La configuration initiale, dénotée par conf , d'un modèle Event-B $\mathcal{M}$ par rapport au domaine $\mathcal{D}$, est une substitution θ du domaine $\mathcal{D}$ pour les valeurs des ensembles s, et constantes c du contexte*

Event-B définie telle que, ces valeurs satisfont les contraintes définies par les axiomes Axm et les théorèmes Thm définies dans le contexte Event-B du présent modèle, notées par P.

La configuration terminale stable est donnée comme suit :

Définition 3.3.2 (configuration stable d'un modèle Event-B) *La configuration terminale d'un modèle Event-B $\mathcal{M}$ est définie par la fin d'observation des événements définis dans le modèle $\mathcal{M}$. Cela signifie que plus aucun événement n'est observable dans les machines du modèle Event-B i.e. plus aucune progression dans les machines du modèle $\mathcal{M}$ n'est possible.*

Pour définir la propriété de terminaison stable, il faut ajouter à la définition de l'opérateur *Leads to* [16] (cf. définition 2.4.4) la propriété de stabilité qui suit : Il s'agit d'exprimer qu'on arrivera fatalement à un futur où la propriété Q sera vérifiée et restera vraie sur tous les chemins qui suiveront. La propriété de terminaison stable se définit comme suit :

Définition 3.3.3 (terminaison stable) *Une spécification $\mathcal{S}pec(\mathcal{M})$ d'un modèle $\mathcal{M}$ sous les hypothèses d'équité L satisfait la propriété de stabilité en $\mathcal{T}$ à partir de $\mathcal{I}nit$, si pour toutes les traces $\sigma \in \mathcal{T}race(\mathcal{M})$, la propriété suivante est vérifiée :*

$$\forall i.(i \geq 0 \wedge P(\sigma_i) \Rightarrow \exists j.(j \geq i \wedge \mathcal{T}(\sigma_j) \wedge \forall k.(k \geq j \wedge \mathcal{T}(\sigma_k))))$$

La stabilité implique que le prédicat qui caractérise les états finaux restera vrai pour tous les états suivants. Une preuve de progression, en définissant un variant sur un ordre bien fondé est nécessaire pour montrer la convergence dans le système [16].

Définition 3.3.4 (Phase d'un modèle Event-B) *Une phase d'un modèle Event-B $\mathcal{M}$ est définie comme étant une étape qui débute par la configuration initiale du modèle $\mathcal{M}$ et se stabilise lorsque la configuration terminale stable de ce même modèle est détectée.*

Chaque phase dans notre cas est donc modélisée par un modèle (machines et contextes B) indépendamment des autres phases, et l'on demande à ce que le niveau de raffinement pour chaque phase soit suffisamment élaboré pour que l'ensemble des constantes dans le contexte B de la phase qui suit la phase en cours trouve son correspond (variable) dans le dernier raffinement de cette phase dont il dépend.

Notons que dans le cas de décomposition par dépendances, un composant est défini par un modèle Event-B abstrait et éventuellement un ensemble de modèles qui raffinent ce modèle abstrait. Ce composant est dédié à une phase dans le système considéré. La structure d'un modèle Event-B en contextes et machines qui voient ces contextes liés par extension et qui définissent une phase donne lieu donc à un composant.

Nous soulignons que les définitions de phases et de composants données dans cette section sur les dépendances ne concernent que ce mécanisme de dépendances, et que dans le cas où cette relation n'est pas satisfaite, nous utilisons les définitions données par l'auteur dans [16] aussi rappelées au chapitre 2. Ainsi, l'étude de cas donnée en section 3.4 utilise les définitions données par l'auteur [16].

La preuve par dépendance repose essentiellement sur l'existence de situations dans lesquelles une partie peut dépendre de l'autre. Une telle approche se traduit par la donnée d'un ensemble de valeurs d'état d'un composant dans le formalisme Event-B qui satisfait les contraintes définies dans les contextes Event-B de l'autre partie. En d'autres termes, les valeurs des variables du premier composant correspondent à une interprétation valide des contraintes du second composant dans le système. Nous fondons notre idée sur le

fait que les composants impliqués dans le système ne vérifient pas les mêmes propriétés de sûreté, et n'ont pas les mêmes objectifs ou *"tâches"* à réaliser, mais ont leur propore existence dans le système.

Définition 3.3.5 (Dépendances entre deux modèles Event-B) *Deux modèles Event-B $\mathcal{M}_i$, avec $i \in 1..2$ (i.e. : $\mathcal{M}_1$ et $\mathcal{M}_2$) sont dépendants, et on dit que le modèle $\mathcal{M}_2$ dépend contextuellement du modèle $\mathcal{M}_1$ en respect avec la propriété de stabilité $\mathcal{T}_1$ qu'on note $Dep(\mathcal{M}_1, \mathcal{T}_1, \mathcal{M}_2, v_1, c_{21})$, ssi :*

1. *pour le modèle $\mathcal{M}_1$ sous les hypothèses d'équité L_1, la propriété de vivacité $\mathcal{I}nit_1 \rightsquigarrow \mathcal{T}_1$ est dérivable à partir de la spécification $\mathcal{S}pec(\mathcal{M}_1)$ i.e., $\mathcal{S}pec(\mathcal{M}_1) \vDash \mathcal{I}nit_1 \rightsquigarrow \mathcal{T}_1$ et $\mathcal{T}_1$ est le prédicat caractérisant l'ensemble des états terminaux stables du modèle $\mathcal{M}_1$;*
2. *Cette propriété est maintenue sur tous les chemins atteignables après :*
$$\mathcal{S}pec(\mathcal{M}_1) \vDash (\forall x_1, x_1'.(\mathcal{T}_1(x_1) \wedge NEXT_1(x_1, x_1') \Rightarrow \mathcal{T}_1(x_1'))) ;$$
3. *le contexte dépendant étend le contexte source : $\mathcal{T}h_2(s_2, c_2)$ extends $\mathcal{T}h_1(s_1, c_1)$;*
4. *il existe un sous-ensemble non vide v_1 de l'ensemble des variables x_1 ($v_1 \subseteq x_1$) du modèle $\mathcal{M}_1$ tel que la propriété suivante est vérifiée :*
$$\mathcal{T}h_2(s_2, c_2) \vDash (\mathcal{T}_1(x_1) \wedge v_1 = c_{21} \wedge (c_2 = c_{21} \cup c_1 \cup c_{22})) \Rightarrow \mathcal{C}_2(s_2, c_2),$$
*avec c_2 est l'ensemble des constantes c_1 définies dans le contexte du premier modèle $\mathcal{M}_1$ et est obtenue par extension du contexte $\mathcal{T}h_1$ selon le point **3**, union les variables v_1 issues de la machine du premier modèle $\mathcal{M}_1$, auxquelles s'ajoutent les constantes c_{22} nouvellement introduites dans le contexte Event-B du second modèle $\mathcal{M}_2$. Les valeurs de ces dernières (c_{22}) peuvent être définies en fonction des valeurs des variables v_1 du premier modèle.*

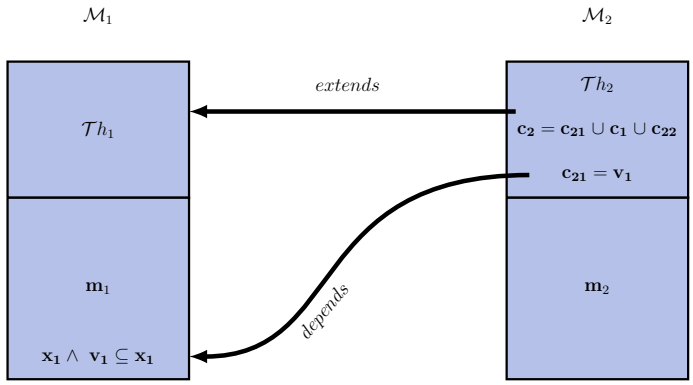

FIGURE 3.1 – Dépendance entre modèle Event-B

En résumé, la dépendance entre deux modèles Event-B $\mathcal{M}_1$ et $\mathcal{M}_2$ se définit ainsi : *i)* les contextes B $\mathcal{M}_1$ et $\mathcal{M}_2$ sont communs i.e. : font partie du contexte B $\mathcal{T}h(s_2, c_2)$; *ii)* une transformation d'un certain nombre de variables d'un modèle source $\mathcal{M}_1$ en des constantes dans le modèle cible $\mathcal{M}_2$; *iii)* le prédicat caractérisant la stabilité du premier modèle satisfait les contraintes définies dans le contexte B du deuxième composant.

Concrètement, un modèle $\mathcal{M}_2$ dépend d'un autre modèle $\mathcal{M}_1$ si la stabilité donnée par le prédicat caractérisant l'ensemble des états terminaux du modèle $\mathcal{M}_1$ est détectée, alors la ***"configuration terminale"*** de ce modèle satisfait la ***"configuration initiale"*** du modèle $\mathcal{M}_2$ qui est définie par : le contenu des ensembles porteurs ainsi que les valeurs des constantes du contexte $\mathcal{T}h_2(s_2, c_2)$.

$\mathcal{T}h_2(s_2, c_2)$ est la structure définissant le contexte B du second modèle dans sa totalité i.e., toutes les propriétés (axiomes et théorèmes) statiques en conjonction qui établissent

le typage ainsi que les contraintes dépendantes et indépendantes des situations, alors que $\mathcal{C}_2(s_2, c_2)$ sont les contraintes qui doivent être satisfaites par l'ensemble des états finaux du premier modèle i.e., qui sont dépendantes des situations. Celles-ci s'ajoutent donc dans le contexte du second modèle B en conjonction avec les propriétés indépendantes des situations liées au premier modèle. Cette approche reflète le fait qu'à la stabilisation de la première phase, aucune modification ne peut être faite sur ces éléments en tant que variables, puisque ces dernières dans le premier composant conservent leur valeurs à la terminaison. On applique alors à l'opérateur définissant le prédicat des contraintes statiques qui prend la liste des ensembles et des constantes définis dans le contexte dépendant les nouvelles valeurs des variables définies dans le premier modèle.

La consistance en Event-B est définie par l'ensemble des obligations de preuve. L'obligation de preuve relative au mécanisme de dépendance entre deux modèles $\mathcal{M}_1$ et $\mathcal{M}_2$ qui doit être déchargée est définie comme suit :

Définition 3.3.6 (Obligation de preuve de dépendance entre les modèles $\mathcal{M}_1$ et $\mathcal{M}_2$)

$$\mathcal{T}h_1(s_1, c_1), Inv(s_1, c_1, x_1), v_1 \subseteq x_1, c_{21} \subseteq c_2, \mathcal{T}_1(x_1) \vdash \mathcal{C}_2(s_2, c_2)(v_1/c_{21})$$

L'obligation de preuve montre bien le paramétrage des constantes par des valeurs de variables du modèle Event-B source.

Dans l'idéal, nous avons l'égalité où toutes les variables sont constantes du modèle dépendant ($v_1 = x_1$). Et les contextes[15] deviennent en relation *partonomique* telle que définie par Barlatier[16] dans [28, 92], et où un contexte $\mathcal{C}xt_1$ est une partie d'un autre contexte $\mathcal{C}xt_2$ si $\mathcal{C}xt_1$ contient au moins toutes les contraintes vérifiées dans $\mathcal{C}xt_2$. Le second modèle devient dépendant du premier (cf. définition 1.9.5). Nous disposons donc de constantes qui vérifient des propriétés axiomatiques tout en garantissant leur *bonne construction.*

$\mathcal{M}_1$/ ∗ modèle source ∗ /
variables x_1...
invariants...
variant
  V_1
initialization1...
event ce_1 **status convergent**
any t_1 **where**
  $G_1(t_1, x_1)$
then
  $S_1(t_1, x_1)$
end...

$\mathcal{M}_2$/ ∗ modèle dépendant ∗ /
variables x_2...
invariants...
variant
  V_2
initialization2...
event ce_2 **status convergent**
any t_2 **where**
  $G_2(t_2, x_2)$
then
  $S_2(t_2, x_2)$
end...

FIGURE 3.2 – Modèles Event-B décomposés par dépendance

Preuve de stabilité des phases séparées

Les preuves sont identiques pour toutes les phases liées par dépendance. Nous donnons ici la preuve de la stabilité pour une phase, et l'on reproduit les mêmes preuves pour

15. On parlera indifféremment de contextes ou de modèles B en relation partonomique.
16. Dans ses travaux, Barlatier s'est intéressé uniquement aux relations *Partie-Tout* basées sur les dépendances.

toutes les phases qui puevent être liées par dépendance.

La propriété de terminaison souhaitée est notée comme suit : $\mathcal{T}_1$ et l'ensemble des variables est donné par : x_1.

Les hypothèses d'équité posées dans la présente machine sont :

$$L_{\mathcal{M}_1} \ \widehat{=} \ WF_{x_1}(convergent\_event)$$

$$\begin{array}{rcl} NEXT_1 & \widehat{=} & \vee BA(convergent\_event)(x_1, x_1') \\ & & \vee BA(\textit{finish})(x_1, x_1') \\ & & \vee (x_1 = x_1') \ldots \end{array}$$

Les autres hypothèses d'équité qui peuvent être posées sur les événements du modèle doivent respecter les contraintes données dans le chapitre de l'état de l'art. Aucun des autres événements ne peut modifier la valeur du variant. Seul l'événement convergent est autorisé à le faire.

La propriété de vivacité à montrer est la suivante :

$$\Phi_{\mathcal{M}_1} \widehat{=} \mathcal{I}nit_1(x_1) \rightsquigarrow \mathcal{T}_1$$

qui signifie qu'à partir de l'initialisation des variables du modèle, on arrivera fatalement à atteindre un état où le temps évoluera jusqu'à atteindre la propriété de stabilité voulue.

Nous posons : $\mathcal{I}nit_{x_1} \ \widehat{=} \ P_0$, $V(t) \ \widehat{=} \ variant = t$. La propriété de stabilité $\Phi_{\mathcal{M}_1}$ est exprimée comme suit :

$$\begin{array}{rcll} \Phi_{\mathcal{M}_1} & \widehat{=} & \{V(t) \rightsquigarrow V(t-1) & \Phi_1 \\ & & , (\exists t.V(t)) \rightsquigarrow V(0) & \Phi_2 \\ & & \} & \end{array}$$

— Φ_1 stipule qu'on arrivera fatalement par atteindre un état où la valeur du variant diminuera d'un pas de 1 ;
— Φ_2 avance qu'on arrivera fatalement à un état où la différence convergera vers la valeur pour laquelle le variant vaudra zéro ;

En posant $T_1 \widehat{=} V(t)$ et $T_2 \widehat{=} V(t-1)$, nous avons :

$$P_0 \wedge [NEXT_1]_{x_1} \Rightarrow (T_1' \vee T_2')$$

Nous divisons la preuve de cette implication en les preuves suivantes, selon la définition $NEXT_1$:

— $T_1 \wedge BA(convergent\_event)(x_1, x_1') \Rightarrow (T_1' \vee T_2')$
— $T_1 \wedge BA(\textit{finish})(x_1, x_1') \Rightarrow (T_1' \vee T_2')$
— $T_1 \wedge x_1 = x_1' \Rightarrow (T_1' \vee T_2') \ \ldots$

Le seul événement qui fait évoluer le temps est $convergent\_event$, donc la valeur n'est modifiée que par les actions prévues dans cet événement. Cet événement est observable jusqu'à ce que la valeur de la variable convernée atteindra la valeur zéro ou $\varnothing$.

L'hypothèse d'équité faible posée sur l'événement $convergent\_event$ nous permet de dire que **(a)** : $T_1(x_1) \wedge BA(convergent\_event)(x_1, x_1') \Rightarrow T_2(x_1')$, et de la condition de faisabilité de l'événement $convergent\_event$, nous pouvons déduire que **(b)** : $T_1(x_1) \Rightarrow (\exists x_1'.BA(convergent\_event)(x_1, x_1'))$ sont satisfaites par l'événement $convergent\_event$. Et l'on peut déduire de **(a)**, $T_1 \wedge \langle NEXT \wedge convergent\_event \rangle_{x_1} \Rightarrow T_2$ et de **(b)** que

$T_1 \Rightarrow ENABLED\langle convergent\_event\rangle_{x_1}$, avec
$ENABLED\langle convergent\_event\rangle_{x_1} \widehat{=} (\exists y_1.BA(convergent\_event)(x_1, y_1))$.

La règle WF1 permet de déduire que le temps finira par progresser pour faire décroître le variant et puisqu'aucun événement ne viole la propriété $V(t)$, à l'exception de l'événement *convergent_event*. En appliquant la règle LATTICE sur l'ensemble bien fondé des entiers naturels, nous pouvons déduire que le système convergera vers la valeur $V(1)$. Nous pouvons donc déduire que : $\mathcal{S}pec(\mathcal{M}_1) \vdash \Phi_1$.

Nous avons alors : $T_1 = 1 = V(1)$.

L'observation de l'événement *convergent_event* arrivé à $V(1)$, et l'hypothèse d'équité faible posée sur ce même événement, nous permettent de déduire que : $\mathcal{S}pec(\mathcal{M}_1) \vdash \Phi_2$, d'où la terminaison avec stabilité est atteinte, puisque une fois le variant atteint sa valeur minimale, seul l'événement *finish* sera activable dans le système, et aucun changement ne peut être fait sur les variables dans le système, à partir du moment où aucun des événements dans la présente machine ne sera observable.

3.3.1 Étude de cas : système de gestion ERP

L'exemple donné ici concerne la gestion de stocks par achats et ventes d'articles, et une comptabilisation en différé des recettes et des dépenses. Cette modélisation est une représentation simplifiée et ne décrit pas tous les détails de gestion dans un système ERP [17]. Le premier modèle noté $\mathcal{M}\_sp$ est décrit par un contexte *sales_purchases_cxt* et une machine *sales_purchases_machine*. Le système gère les achats (l'événement *buy*) et les ventes (l'événement *sale*) d'articles. Les ventes s'effectuent selon les prix fixés dans le contexte, alors que les achats se font à des prix fixés par le marché.

L'imputation des écritures comptabilisées en différé permet de ne comptabiliser les opérations qu'au moment de l'exécution des opérations de transfert. Le journal des écritures comptables pour les opérations de ventes et d'achats est configuré en différé selon une periode de clôture notée *deferred_period*.

Les achats et les ventes peuvent se réaliser durant cette periode (*grd3*). Cette periode est décrémentée par l'événement convergent *"forward_time"* qui fait décrémenter l'expression du variant définie dans cette première machine. L'impression des différentes opérations de ventes et d'achats se fait à travers les deux variables *incomings* et *expenses* respectivement.

CONTEXT $sales\_purchases\_cxt$
SETS
  $ARTICLES$
CONSTANTS
  $deferred\_period, prices\_art$
AXIOMS
  $axm1 : deferred\_period \in \mathbb{N}1$
  $axm2 : prices\_art \in ARTICLES \rightarrow \mathbb{N}1$

17. ERP pour Enterprise Ressource Planning en Anglais.

MACHINE *sales_purchases_machine*
SEES *sales_purchases_cxt*
INVARIANTS
inv1 : $period \in 0 \mathinner{..} deferred\_period$
inv2 : $sold\_art \subseteq ARTICLES$
inv3 : $incomings \in sold\_art \rightarrow \mathbb{N}$
inv4 : $purchased\_art \subseteq ARTICLES$
inv5 : $expenses \in purchased\_art \rightarrow \mathbb{N}$
inv6 : $purchased\_art \cap sold\_art = \varnothing$
inv7 : $\forall art, p \cdot (art \mapsto p \in incomings \Rightarrow art \mapsto p \in prices\_art)$
VARIANT
$deferred\_period - period$
EVENTS
EVENT forward_time *convergent*
WHEN
grd1 : $period \in 0 \mathinner{..} (deferred\_period - 1)$
THEN
act1 : $period := period + 1$
END

EVENT sale
ANY *art*
WHERE
grd1 : $art \in ARTICLES$
grd2 : $art \notin sold\_art \wedge art \notin purchased\_art$
grd3 : $period \in 0 \mathinner{..} (deferred\_period - 1)$
THEN
act1 : $sold\_art := sold\_art \cup \{art\}$
act2 : $incomings(art) := prices\_art(art)$
END
EVENT buy
ANY art, p
WHERE
grd1 : $art \in ARTICLES \wedge p \in \mathbb{N}_1$
grd2 : $art \notin purchased\_art \wedge art \notin sold\_art$
grd3 : $period \in 0 \mathinner{..} (deferred\_period - 1)$
THEN
act1 : $purchased\_art := purchased\_art \cup \{art\}$
act2 : $expenses(art) := p$
END ...

Les variables définies dans cette machine, à l'exception de la variable *period*, sont initialisées à l'ensemble vide. La variable *period* est initialisée à 0.

Preuve de stabilité de la phase des achats et des ventes

La propriété de stabilité souhaitée est définie comme suit :

$$\mathcal{T}_1 \;\widehat{=}\; period = deferred\_period$$

Mais l'on souhaite également montrer qu'à la stabilité de la présente phase, on ait des ventes et des achats (des dépenses et entrées).

Alors la propriété de vivacité souhaitée est :

$$\mathcal{T}_1 \;\widehat{=}\; period = deferred\_period \wedge (purchased\_art \neq \varnothing \wedge sold\_art \neq \varnothing)$$

Nous posons : $x_1 = \{period,\ sold\_art,\ incomings,\ purchased\_art,\ expenses,\ purchased\_art\}$.

Les hypothèses d'équité posées dans la présente machine sont :

$$L_{\mathcal{M}_1} \;\widehat{=}\; WF_{x_1}(forward\_time), WF_{x_1}(sale), WF_{x_1}(buy)$$

Les hypothèses d'équité faible sur les événements d'achat et de vente sont justifiées par le fait qu'on ne peut comptabiliser dans la phase dépendante qu'en ayant des recettes et des dépenses en entrée. Cela justifie donc la propriété de vivacité souhaitée et ainsi l'axiome 11 (**axm11**) défini au niveau du contexte dépendant (*cxt_compt*). Nous verrons que dans l'étude de cas des systèmes cette contrainte d'équité forte n'est pas imposée puisque ces systèmes sont régis par des modalités dans certains pays où le vote n'est pas obligatoire.

$$
\begin{array}{rcl}
NEXT_1 & \widehat{=} & \vee BA(forward\_time)(x_1, x_1') \\
& & \vee BA(sale)(x_1, x_1') \\
& & \vee BA(buy)(x_1, x_1') \\
& & \vee BA(finish)(x_1, x_1') \\
& & \vee (x_1 = x_1')
\end{array}
$$

$$
\begin{array}{rcl}
\mathcal{I}nit_1(x_1) & \widehat{=} & period = 0 \\
& & \wedge sold\_art = \varnothing \\
& & \wedge incomings = \varnothing \\
& & \wedge purchased\_art = \varnothing \\
& & \wedge expenses = \varnothing
\end{array}
$$

La propriété de vivacité à montrer est la suivante :

$$\Phi_{\mathcal{M}_1} \widehat{=} \mathcal{I}nit_1(x_1) \rightsquigarrow \mathcal{T}_1$$

qui signifie qu'à partir de zéro, heure de début de la période des ventes et des achats, à l'initialisation, où aucun achat ou vente ne sont effectués, on arrivera fatalement à atteindre un état où le temps évoluera jusqu'à atteindre l'heure de début de comptabilisation avec une modification des valeurs des recettes et des dépenses.

Nous posons :

— $P_0 \widehat{=} \mathcal{I}nit_{x_1}$;
— $P_1 \widehat{=} (sa \in ARTICLES \wedge sa \notin sold\_art)$, qui signifie qu'un article sa n'a pas encore été vendu ;
— $P_2 \widehat{=} sa \in sold\_art$, signifiant qu'un article sa a été vendu ;
— $P_3 \widehat{=} ia \in ARTICLES \wedge ia \notin purchased\_art$, signifiant qu'un article ia n'a pas encore été acheté ;
— $P_4 \widehat{=} ia \in purchased\_art$, signifiant qu'un article ia a été acheté ;
— $V(t) \widehat{=} deferred\_period - period = t$.

De plus, nous posons les propriétés de vivacité suivantes :

— $P_1 \rightsquigarrow P_2$: pour exprimer qu'un article qui n'a pas encore été vendu le sera fatalement ;
— $P_3 \rightsquigarrow P_4$: pour exprimer qu'un article qui n'a pas encore été acheté le sera fatalement ;

La propriété de stabilité du modèle $\mathcal{M}_1$ représentant la machine $sales\_purchases\_machine$ est exprimée comme suit :

$$
\begin{array}{rcll}
P_s & \widehat{=} & \{V(t) \rightsquigarrow V(t-1) & \Phi_1 \\
& & , (\exists t.V(t)) \rightsquigarrow V(0) & \Phi_2 \\
& & \} &
\end{array}
$$

— Φ_1 stipule qu'on arrivera fatalement par atteindre un état où la valeur du compteur diminuera d'un pas de 1 ;
— Φ_2 avance qu'on arrivera fatalement à un état où la différence convergera vers l'heure de comptabilisation en differé, valeur pour laquelle le variant vaudra zéro ;

Les propriétés de vivacité du modèle sont alors : $P_{\mathcal{M}_1} \widehat{=} \{P_s, P_1 \rightsquigarrow P_2, P_3 \rightsquigarrow P_4\}$

Vivacité des événements d'achats et de ventes : Pour montrer que la machine $\mathcal{M}_1$ satisfait $P_1 \rightsquigarrow P_2$, il faut montrer que $\mathcal{M}_1$ satisfait $(P_s \vee P_1) \rightsquigarrow P_2$. L'hypothèse d'équité faible posée sur l'événement *sale* nous permet de nous retrouver dans le cas : **(1)** : l'événement convergent va nous ramener, d'un état qui satisfait Φ_1 vers un état qui satisfait Φ_1 à nouveau, et par conséquent ne falsifie pas P_1, tel que $P_1 \rightsquigarrow P_2$, ou à état qui satisfait Φ_2, où P_1 est falsifiée. L'événement *buy* ne falsifie pas la propriété P_1. Nous avons donc :

$$(P_s \vee P_1) \wedge [NEXT_1]_{x_1} \Rightarrow ((P'_s \vee P'_1) \vee P_2)$$

Soit :
— $(P_s \vee P_1) \wedge BA(forward\_time)(x_1, x'_1) \Rightarrow ((P'_s \vee P'_1) \vee P_2)$
— $(P_s \vee P_1) \wedge BA(sale)(x_1, x'_1) \Rightarrow ((P'_s \vee P'_1) \vee P_2)$
— $(P_s \vee P_1) \wedge BA(buy)(x_1, x'_1) \Rightarrow ((P'_s \vee P'_1) \vee P_2)$
— $(P_s \vee P_1) \wedge (x_1 = x'_1) \Rightarrow ((P'_s \vee P'_1) \vee P_2)$

La propriété **(3)** : $P_1(x_1) \wedge BA(sale)(x_1, x'_1) \Rightarrow P_2(x'_1)$, et la condition de faisabilité **(4)** : $P_1(x_1) \Rightarrow (\exists x'_1.BA(sale)(x_1, x'_1))$ sont satisfaites par *sale*. De **(3)** nous pouvons déduire que $P_1 \wedge \langle NEXT_1 \wedge sale\rangle_{x_1} \Rightarrow P'_2$ et de **(4)** nous avons : $P_1 \Rightarrow ENABLED\langle sale\rangle_{x_1}$, où $ENABLED\langle sale\rangle_{x_1} \widehat{=} (\exists y.BA(sale)(x_1, y))$. La règle $WF1$ nous permet de déduire que la machine $\mathcal{M}_1$ ($sales\_purchases\_machine$) satisfait la propriété $P_1 \rightsquigarrow P_2$.

Les mêmes preuves sont faites pour l'événement *buy*. Nous pouvons aussi déduire que la machine $\mathcal{M}_1$ ($sales\_purchases\_machine$) satisfait la propriété $P_3 \rightsquigarrow P_4$.

Progression et stabilité de l'événement convergent : Une hypothèse d'équité faible est posée sur l'événement convergent ($WF_{x_1}(forward\_time)$), **cas (1)** si les autres événements différents de cet événement convergent sont aussi observables la propriété Φ_1 reste vraie, car l'observabilité de l'un des événements d'achat ou de vente ne falsifie pas la valeur du compteur, condition d'observation de l'événement convergent, ou conduit à Φ_2. Nous avons donc **(2)** :

$$\Phi_1 \wedge [NEXT_1]_{x_1} \Rightarrow \Phi_1$$

Soit :
— $\Phi_1 \wedge BA(forward\_time)(x_1, x'_1) \Rightarrow (\Phi'_1)$
— $\Phi_1 \wedge BA(sale)(x_1, x'_1) \Rightarrow (\Phi'_1)$
— $\Phi_1 \wedge BA(buy)(x_1, x'_1) \Rightarrow (\Phi'_1)$
— $\Phi_1 \wedge (x_1 = x'_1) \Rightarrow (\Phi'_1)$

La propriété **(3)** $\Phi_1(x_1) \wedge BA(forward\_time)(x_1, x'_1) \Rightarrow \Phi_1(x'_1)$, et la condition de faisabilité **(4)** : $\Phi_1(x_1) \Rightarrow \exists x'_1.BA(forward\_time)(x_1, x'_1)$ sont satisfaites par l'événement convergent $forward\_time$. De la propriété **(3)**, nous déduisons $\Phi_1 \wedge \langle NEXT_1 \wedge forward\_time\rangle_{x_1} \Rightarrow \Phi'_1$, et de **(4)** nous déduisons $\Phi_1 \Rightarrow ENABLED\langle forward\_time\rangle_{x_1}$, et $ENABLED\langle forward\_time\rangle_{x_1} \widehat{=} (\exists y.BA(forward\_time)(x, y))$. L'application de la règle d'hypothèse faible $WF1$ permet de déduire que la machine $\mathcal{M}_1$ ($sales\_purchases\_machine$) satisfait la propriété $\Phi_1 \rightsquigarrow \Phi_1$.

Pour montrer que la machine se stabilise vers la valeur qui rend le variant vide (Φ_2), la règle WF1 permet de déduire que le temps finira par progresser pour faire décroître le variant et puisqu'aucun événement ne viole la propriété Φ_1, à l'exception de l'événement

forward_time. En appliquant la règle LATTICE sur l'ensemble bien fondé des entiers naturels, nous pouvons déduire que le système convergera vers la valeur Φ_2. Nous pouvons donc déduire que : $\mathcal{S}pec(\mathcal{M}_1) \vdash \Phi_2$.

L'observation de l'événement *forward_time* arrive à satisfaire Φ_2, et l'hypothèse d'équité faible posée sur ce même événement, nous permettent de déduire que : $\mathcal{S}pec(\mathcal{M}_1) \vdash P_{\mathcal{M}_1}$, d'où la stabilité, puisque une fois le variant atteint sa valeur minimale, seul l'événement *finish* sera activable dans le système, et aucun changement ne peut être fait sur les variables dans le système.

Les mêmes preuves sont faites pour le modèle dépendant en posant une hypothèse d'équité faible sur l'événement *accounting*.

La *clôture* d'une période implique que la comptabilisation des recettes et des dépenses peut *commencer* lorsque la valeur de la variable *period* vaudra *deferred_period* : lorsque le premier composant se stabilise. Cela est exprimé par la relation de dépendance entre les deux modèles. La comptabilisation est décrite par le modèle $\mathcal{M}\_accounting$ défini par la machine *accounting_machine* qui voit le contexte *accounting_cxt* dépendant de la machine *sales_purchases_machine*. Ce contexte étend le premier *sales_purchases_cxt* et contient toutes les constantes définies comme variables dans la machine *sales_purchases_machine*. avec la valeur de la constante *period* qui vaut *deferred_period* dans le contexte dépendant. Nous notons la dépendance entre les deux modèles $Dep(\mathcal{M}\_sp, period = deferred\_period, \mathcal{M}\_accounting, v_1, c_{21})$, avec : v_1 toutes les variables définies par les invariants $inv1$,...,$inv7$ dans la première machine ; et c_{21} correspond aux mêmes éléments, mais définis comme constantes dans le contexte dépendant *accounting_cxt*, auxquelles s'ajoutent la contrainte : $axm11 : period = deferred\_period \Rightarrow (purchased\_art \neq \varnothing \wedge sold\_art \neq \varnothing)$. La contrainte de dépendance dans cet exemple est : $period = deferred\_period \wedge axm11$.

La comptabilisation des opérations consiste à calculer à travers la variable *balance*, initialisée à 0, la différence entre les recettes *incomings* et les dépenses *expenses* en utilisant la constante *total* définie dans le contexte dépendant comme une fonction qui permet de sommer les valeurs des arguments qu'elle prend.

CONTEXT *cxt_compt* **EXTENDS** *cxt_achat_vente*
CONSTANTS
$period, sold\_art, total$
AXIOMS
axm1 : $period = deferred\_period$
axm2 : $sold\_art \subseteq ARTICLES$
axm3 : $incomings \in sold\_art \rightarrow \mathbb{N}$
axm4 : $purchased\_art \subseteq ARTICLES$
axm5 : $expenses \in purchased\_art \rightarrow \mathbb{N}$
axm6 : $purchased\_art \cap sold\_art = \varnothing$
axm7 : $\forall art, p \cdot (art \mapsto p \in incomings \Rightarrow art \mapsto p \in prices\_art)$
axm8 : $\forall sa, a \cdot (sa \in ARTICLES \rightarrow \mathbb{N} \wedge a \mapsto 0 \in sa \Rightarrow total(sa) = total(sa \setminus \{a \mapsto 0\}))$
axm9 : $\forall sa, a, soa \cdot \left(\begin{array}{l} sa \in ARTICLES \rightarrow \mathbb{N} \wedge a \mapsto soa \in sa \\ \Rightarrow (total(sa) = soa + total(sa \setminus \{a \mapsto soa\})) \end{array} \right)$
axm11 : $period = deferred\_period \Rightarrow (purchased\_art \neq \varnothing \wedge sold\_art \neq \varnothing)$
...

MACHINE *accounting_machine* / ∗ *dependent machine* ∗ /
SEES *accounting_cxt* / ∗ *dependent context* ∗ /
INVARIANTS
inv1 : $balance \in \mathbb{Z} \wedge accounted \in BOOL$
inv2 : $period = deferred\_period \wedge accounted = FALSE \Rightarrow balance = 0$
EVENT accounting
WHEN
grd1 : $accounted = FALSE$
THEN
act1 : $balance := total(incomings) - total(expenses)$
act2 : $accounted := TRUE$
END

Cette relation peut être généralisée à n'importe quel nombre de modèles Event-B, comme l'indique le diagramme de la figure 3.3. Si l'on prend par exemple l'exemple d'un système de gestion, il est alors possible de définir une dépendance entre le service *Human Ressources Service* et le service *Payroll Service*, et entre ce dernier service et le service *Accounting Service*. La dépendance est relation irréflexive et transitive comme le souligne Barlatier dans ses travaux, puisque les extensions entre les contextes Event-B n'ont de sens que dans un seul sens : si c_2 *extends* c_1, alors c_1 n'étend pas c_2.

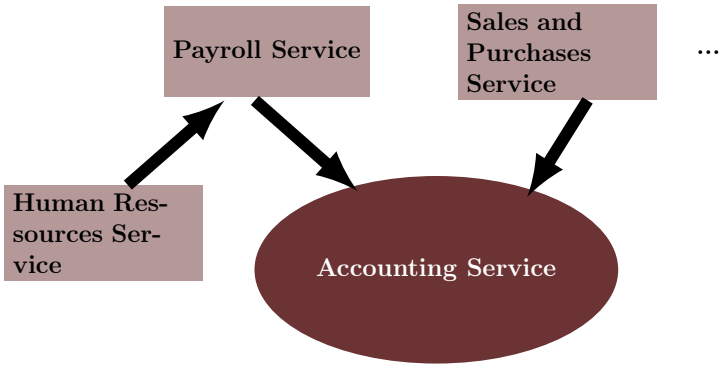

FIGURE 3.3 – Système de gestion ERP

3.3.2 Modèle Event-B équivalent aux modèles dépendants

Le principe des dépendances exposé dans le présent chapitre est fondé sur l'existence de situations qui valident des axiomes appelés contraintes statiques définies au niveau des contextes Event-B du modèle dépendant. Cette validation est conditionnée par la stabilité du composant source $\mathcal{M}_1$. Dans ce type de décomposition, les variables du modèles Event-B source sont des constantes dans le modèle Event-B dépendant. La condition de stabilité indique que l'événement convergent pour chaque phase doit faire décroître le variant à chaque observation de cet événement ; et que toutes les variables impliquées dans la phase ne changeront plus leur valeurs à la stabilité du composant. Les preuves de stabilité de chaque modèle dépendant (à savoir les modèles $\mathcal{M}_1$ et $\mathcal{M}_2$ sur la figure 3.2) sont réalisées en posant des hypothèses d'équité faible sur l'événement convergent, et en reprenant les mêmes preuves que celles réalisées sur l'exemple ci-dessus, avec l'opérateur $\rightsquigarrow$.

Le meilleur moyen de composer les phases impliquées dans la construction du système par dépendances est de le faire par raffinement.

Pour pouvoir recomposer les deux modèles Event-B afin d'obtenir le modèle équivalent des deux modèles Event-B dépendants, il est nécessaire d'introduire dans le même modèle (machine) Event-B du premier composant un événement anticipé (ayant un statut *anticipated*) qui représente la phase dépendante $\mathcal{M}_2$. La stabilité du premier composant $\mathcal{M}_1$ est assurée par l'événement convergent ce_i ainsi que l'expression du variant le concernant V_1. Et ce n'est que dans les prochains raffinements que la preuve de convergence du second modèle Event-B $\mathcal{M}_2$ doit être établie. Pour ce faire, il faut définir un ordre lexicographique [184]. Si pour la première phase $\mathcal{M}_1$ ayant un ordre bien fondé $\prec_1$ sur l'ensemble S_1, et la deuxième phase $\mathcal{M}_2$ ayant un ordre bien fondé $\prec_2$ sur l'ensemble S_2, alors l'ordre lexicographique de la combinaison $\prec_{lex}$ est défini comme suit :

$$(x_1', x_2') \prec_{lex} (x_1, x_2) \Leftrightarrow (x_1' \prec_1 x_1) \vee (x_1' = x_1 \wedge x_2' \prec_2 x_2)$$

Ainsi, *la combinaison lexicographique* $\prec_{lex}$ est la relation bien fondée sur l'ensemble $S_1 \times S_2$. Les variants lexicographiques peuvent être généralisés à n-tuples où chaque variant est dédié à un composant tel que $V = (V_1, \ldots V_n)$ décroît à chaque itération d'une boucle pour chaque composant en respect avec l'ordre lexicographique [184].

$\mathcal{M}_0$/ * modèle abstrait résultant * /
variables x_1...
invariants...
variant
  V_1
initialization1...
event ce_1 **status convergent**
any t_1 **where**
  $G_1(t_1, x_1)$
then
  $S_1(t_1, x_1)$
end...
event ce_2 **status anticipated**
any t_2 **where**
  $G_2(t_2, x_2)$
then
  $S_2(t_2, x_2)$
end...

$\mathcal{RM}$/ * raffinement de $\mathcal{M}_0$ * /
refines $\mathcal{M}_0$
... l'événement ce_1 est le même défini dans $\mathcal{M}_0$
variables x_2...
invariants...
variant
  V_2
initialization2
when condition stabilité
de l'événement convergent dans $\mathcal{M}_0$
then...
event ce_2
refinies ce_2
  status convergent
any u_2 **where**
  $G_2(u_2, x_2)$
then
  $S_2(u_2, x_2)$
end...

FIGURE 3.4 – Modèles Event-B recomposés par dépendance

Dans cette recomposition, il faut définir dans un premier temps un modèle abstrait $\mathcal{M}_0$ qui introduit : *1)* l'événement convergent ce_1 qui décrémente $V_1(x_1)$ en respect avec l'ordre bien fondé $\prec_{x_1}$ sur l'ensemble S_1 ; *2)* l'événement anticipé ce_2. Ensuite, dans les prochains raffinements de ce modèle abstrait $\mathcal{M}_0$, l'événement anticipé ce_2 est raffiné dans $\mathcal{RM}$ en événement convergent qui décrémente $V_2(x_2)$ en respect avec l'ordre bien fondé $\prec_{x_2}$ sur l'ensemble S_2.

L'introduction d'un événement anticipé pour le second composant dans la première phase assure que le variant de la première phase ne subira pas de modification. Cette approche de composer les modèles Event-B dépendants laisse les variables définies dans le second modèle Event-B, et qui sont typées par les constantes dans le contexte Event-B dépendant, avec un typage tel que les variables définies dans le premier composant.

La création de copies des variables du premier modèle pour la deuxième phase n'est qu'une affectation via un événement introduit à cet effet des valeurs de ces variables aux variables définies pour le second modèle. Il s'agit de l'événement **initialization2** du modèle dépendant. Cet événement est gardé par la condition de stabilité de la phase du modèle source. Aussi, tous les événements qui seront introduits et qui sont dédiés pour la seconde phase ne doivent pas modifier les valeurs des variables du premier composant.

Les mêmes hypothèses d'équité faible doivent être posées sur les événements convergents recomposés dans la machine $\mathcal{RM}$. Il faut noter qu'il est indispensable de poser une hypothèse d'équité forte (SF) sur l'événement **initialization2** introduit dans le modèle recomposé ($\mathcal{RM}$) correspondant à l'événement d'initialisation de la machine dépendante.

3.4 Stratégies d'intégration de contraintes du domaine pour la conception de systèmes

La composition de services [18] peut être confrontée à de nombreuses hétérogénéités liées aux données échangées [197], à leur représentation ainsi qu'à leur interprétation. Il est évident que l'acquisition correcte des connaissances du domaine permet une meilleure compréhension et communication du problème à traiter. En effet, un modèle conceptuel doit capturer l'intention du concepteur et faire passer un message précis avec une sémantique sans ambiguïté. Ceci est particulièrement important si les modèles conceptuels doivent être utilisés efficacement en tant que base pour la génération de code pour des systèmes. La sémantique implicite joue un rôle important dans l'identification et l'évaluation des fonctionnalités des systèmes étudiés. La conception d'un système doit adopter des stratégies qui facilitent la résolution de ces conflits. Nous fournissons dans un premier temps les étapes à suivre qui permettent la réalisation d'une architecture laquelle facilite la tâche de conception aux concepteurs pour une intégration du contexte et une utilisation des ontologies de domaine optimales. Ces stratégies sont définies selon trois facteurs ou paramètres dont la détermination nous semble essentielle : 1) la *granularité de l'ontologie* ; 2) le *raisonnement a priori/a posteriori* effectué au niveau des ontologies et au niveau de la modélisation en Event-B ; 3) l'*extraction des connaissances inductive et déductive*.

Nous donnons dans ce qui suit, un exemple fil conducteur que nous utiliserons pour illustrer nos propos.

3.4.1 Exemple de motivation

L'exemple qu'on suivra pour illustrer nos propos dans ce qui suit est un exemple extrait des travaux de M. Mrissa dans sa thèse [197]. Cette étude de cas concerne les services de réservation de billets d'avion et de location de voiture par internet.

- le premier service effectue des réservations de billets d'avion ;
- le second service effectue des locations de voitures une fois le voyageur arrivé à destination ;
- il existe un dernier service chargé d'additionner les sommes d'argent pour que le client puisse règler les frais de l'avion et de voiture.

18. Dans cette section, nous utilisons les définitions de composants, de services, et de phases données par l'auteur dans [16], aussi rappelées au chapitre 2.

Les coopérations entre ces différents services s'effectuent selon une hétérogénéité sémantique des données échangées lors de leur composition et qui sont liées au contexte de chaque service. La représentation des données pour ces services constitue la première hétérogénéité entre les services. Le contexte de traitement des données est différent, car les réservations des billets d'avion sont effectuées en Europe qui utilise l'Euro et un facteur multiplicateur de 1, alors que le service de location de voiture facture les frais de location avec la devise du pays destinataire, à savoir le Yen Japonais, et utilise un facteur multiplicateur de 1000. D'autres hétérogénéités existent aussi entre ces services telles que la représentation des dates de réservation et de location des véhicules. Pour le premier service, les dates suivent une représentation selon le format européen : (`dd.mm.yyyy` et 12 : 00 `AM/PM`). Pour le second service, les dates sont formatées selon la notation Japonaise (`yy.mm.dd` et 24 :00).

Les concepts sémantiques utilisés sont les mêmes dans les deux représentations (le concept prix, le concept monnaie, billets d'avion, ticket de location, etc.), mais leur interprétation est différente selon le service et, par conséquent, selon le contexte dans lequel le service opère. Nous avons montré que ce type de contexte est un contexte de contraintes qui permettent de situer les différents développements et d'interpréter ces derniers selon la vision considérée (le point de vue) du service en question.

La prise en compte de cette sémantique et du contexte de chaque service est indispensable car elle permet non seulement une conceptualisation cohérente et complète, mais aussi une interprétation correcte des données vis-à-vis de chaque acteur et point de vue dans le système dans sa globalité. Et l'objectif étant de s'assurer que cette composition est cohérente avec l'intention du concepteur. La conceptualisation habituelle est du ressort du domaine (ontologie) considéré, alors que la conception d'un système qui réalise ces traitements doit suivre le domaine auquel elle s'associe.

Nous avons développé les modèles Event-B pour cette composition de services (cf. section 3.4.4). Un tel schéma pourrait être représenté comme le montre la figure 3.5. Pour une compatibilité sémantique de cette composition, notre modélisation en Event-B modélise chaque service par un événement dédié à cet effet, à savoir : *i)* un événement pour le service « Flight_Booking » pour assurer la réservation des billets d'avion ; *ii)* un événement « Car_Rental » pour prendre en charge la location de voiture pendant la durée du séjour ; *iii)* un événement « Addition_Service »pour le calcul du montant total généré par les deux services. Le service qui additionne les frais des deux services produit un résultat dans la monnaie du client qui est celle utilisée pour les transactions bancaires.

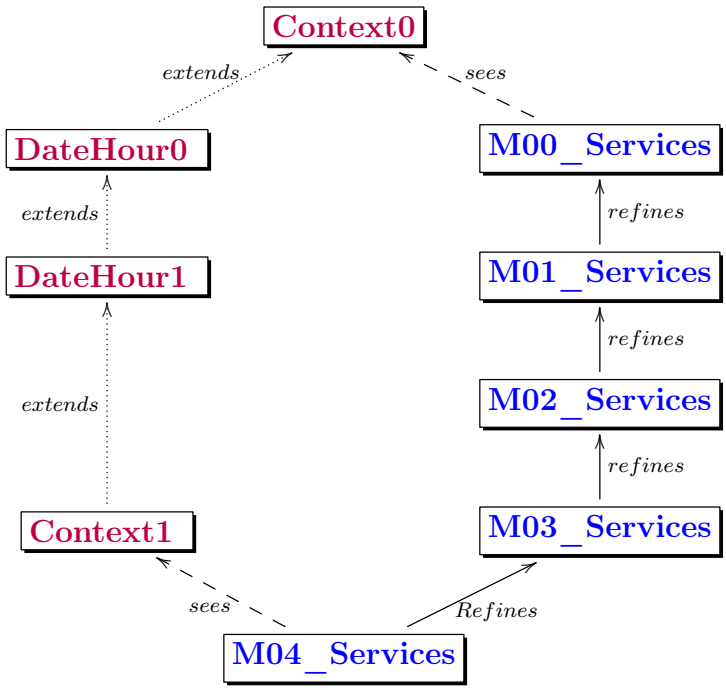

FIGURE 3.5 – Structure des services de réservation (avions et voitures)

L'utilisation de types de données corrects par les différents services doit être vérifiée par les typages définis dans les contextes Event-B. En d'autres termes, les axiomes et les théorèmes définissant les contraintes imposées sur les ensembles et les constantes définis dans la partie statique doivent être cohérents et correspondent bien aux types de données manipulées. L'association avec l'ontologie doit pouvoir vérifier les incompatibilités, ainsi que les types déduits qui peuvent être impliqués par les contraintes des composants. Un système de comportement doit vérifier l'utilisation correcte par rapport à la structure des données manipulées. Une telle utilisation se manifeste par les *concepts intensionnels* concrétisés par les actions dans le système. Cette exigence doit donc être établie dans la partie dynamique du système, à savoir, les machines Event-B.

Dans cet exemple, nous ne traiterons que l'aspect lié à la *monnaie* et nous considérons les représentations japonaises des dates et heures pour des raisons de simplification. Les conversions entre chaque type de représentation sont simples. Le fonctionnement d'un modèle sur des tarifs de « billets d'avion » et des tarifs de « tickets de location de voiture » est une conséquence du domaine d'application pour lequel le modèle est construit. Ainsi, contrairement aux typages, il est essentiel que les constructeurs de modèles ou les concepteurs puissent construire leur propre système de propriétés spécifiques au domaine. Aussi, les nuances des types ressortent du contexte car ces dernières sont dépendantes des cas d'utilisation situés selon les parties impliquées dans le système.

3.4.2 Logique du premier ordre avec les contraintes du domaine pour Event-B : BL-DC

Les chapitres de l'état de l'art montrent que les notions d'identité concrétisée par les objets, de structure concrétisée par les rôles, des classes qui correspondent aux concepts, de sous-typage qui s'illustre par la subsomption, de la T-Box ainsi que la A-Box qui font apparaître l'héritage et les instances pour une formalisation offrant une sémantique pour une expressivité au choix, donnent aux logiques de description un avantage pour leur

utilisation. De plus, leur interprétation des formules logiques par des ensembles d'objets plutôt que par des valeurs de vérité rendent ces logiques plus appropriées à l'extraction [19] de connaissances, où les formules logiques jouent le rôle de requêtes dont le résultat attendu est un ensemble d'objets pouvant être utilisés dans le cadre de la validation des modèles Event-B.

Dans cette section, nous donnons une extension de la traduction des logiques de description en logique du premier ordre en prenant en compte les spécificités de ce formalisme. Les tableaux donnés en annexes A et B définissent les correspondances entre les logiques de description utilisées comme sémantique pour le formalisme OWL DL, et les formules exprimées dans le langage Event-B.

Étant donnés :
— une base de connaissances $\mathcal{SHOIQ}(\mathcal{D}) \widehat{=} (T\text{-}Box, A\text{-}Box)$;
— un modèle Event-B défini (possiblement par raffinement) par ses contextes et machines :

$$\mathcal{M}_i \widehat{=} (\mathcal{T}h_i(s_i, c_i), x_i, Val_i, \mathcal{I}nit_i(x_i), \{e_{i0}, ..., e_{in}\});$$

Le résultat de l'association des formules issues des modèles Event-B et celles issues de la base de connaissances est noté *BL-DC* pour *Event-B First Order Logic with Domain Constraints*. Nous notons les transformations des logiques de description vers Event-B T, et inversement, les transformations de Event-B vers les logiques de description T^{-1} qui est définie partiellement.

Il faut noter que dans notre cas, on ne cherche pas à traduire l'ensemble d'une ontologie en modèles Event-B, mais seulement à récupérer ou extraire ou intégrer des propriétés et des objets du domaine pour pouvoir les exploiter soit en vérification, soit en validation, soit pour justifer des représentations ou des choix de conception tels que notre état de l'art l'a montré. Les correspondances établies à ce niveau cherchent, étant donnée une propriété exprimée à un certain niveau de raffinement dans Event-B, de récupérer des propriétés, des objets ou des concepts ontologiques qui subsument ces propriétés exprimées en Event-B. Cela est particulièrement intéressant dans le cas d'utilisation de prédicats n-aires comme nous le montrons dans l'étude de cas des systèmes de vote.

Dans ce qui suit, nous utilisons la police textsf comme par exemple Tickets, Concept..., lorsqu'il s'agit d'une variable du domaine (un concept ontologique) ou d'un rôle ontologique. Nous adoptons la police italique lorsque nous nous situons au niveau des modèles Event-B, et nous garderons cette police lorsque il s'agit du résultat de l'association d'ontologies avec les modèles Event-B (BL-DC).

Étant donné que la première préoccupation dans notre travail est l'extraction de connaissances du domaine pour leur utilisation dans Event-B, la fonction de traduction T est la fonction qui nous intéresse. Mais, pour ce faire il faut formuler les requêtes définies dans le langage des logiques de description utilisé, il faut donc la fonction inverse qui nous permet de traduire les formules Event-B vers les formules des logiques de description utilisées.

Rappelons que les concepts représentent des prédicats unaires, alors que les relations entre les objets ou les propriétés représentent des prédicats binaires. Dans tous les cas, ceux-ci peuvent correspondre à des constantes ou des variables Event-B en fonction des propriétés ajoutées extrinsèquement dans les modèles Event-B. Les individus sont aussi des

19. Par manque de temps, nous n'avons pas pu aller jusqu'au bout de l'approche à définir pour extraire ces connaissances de manière automatique, mais nous donnons les grandes lignes ainsi que les stratégies d'extraction. Nous reprenons ce point en perspectives de notre thèse.

constantes ou des variables selon leur définition dans Event-B. Ces individus se distinguent donc dans leur domaine d'individus [20] $\Delta^{\mathcal{I}}$ comme étant des entités à part entière et jouent le rôle de références uniques dans la conceptualisation du domaine. Les contraintes sur le domaine des valeurs $\mathcal{D}$ ont aussi les mêmes correspondances à la différence que ces valeurs peuvent être identiques et qu'elles sont différentes des individus qu'elles caractérisent dans leur domaine $\Delta^{\mathcal{I}}$. Les éléments dans l'ensemble des valeurs du domaine $\Delta^{\mathcal{I}}_{\mathcal{D}}$ (noté *SetData*) définissent les valeurs des attributs des classes C des individus $o : C$.

L'hypothèse des identifiants uniques peut être assurée avec la propriété OWL DifferentIndividuals. La propriété SameIndividual n'est donc plus utile. Dans OWL DL, les littéraux sont disjoints de la classe *Thing*, il faut donc définir un ensemble dans Event-B pour les données dans $\Delta^{\mathcal{I}}_{\mathcal{D}}$. Toutes les classes qui sont incluses soit dans l'ensemble *Thing* soit dans l'ensemble des littéraux de $\Delta^{\mathcal{I}}_{\mathcal{D}}$ doivent être déclarées comme étant des constantes dans le cas des définitions statiques dans Event-B car l'inclusion entre ensembles dans la clause *Set* en Event-B n'est pas permise. C'est l'approche adoptée par le partenaire Supelec [194] du projet dans lequelle s'inscrit notre thèse. Mais ce travail réalise une traduction pure et dure d'une terminologie box en contexte Event-B. Cette restriction oblige à déclarer les sous-ensembles comme étant des constantes si elles concernent la partie statique dans les modèles Event-B.

Cette transformation n'est pas unique. En effet, il peut être plus commode pour un concepteur en Event-B d'avoir des ensembles dans la clause *Set* d'un contexte Event-B plutôt que d'avoir un seul ensemble dans lequel tous les autres ensembles doivent être des constantes incluses dans cet ensemble. Dans ce cas, ce sont tous les concepts atomiques de haut niveau C_a dans l'ontologie qui deviennent des ensembles dans un contexte Event-B, et l'ensemble *Thing* disparaîtra. Suivant cette transformation des concepts atomiques de haut niveau en ensembles Event-B, aucune précision quant à la propriété de disjonction des ensembles n'est utile. Dans le cas où la première transformation en constantes incluses dans un ensemble *Thing* est adoptée, les axiomes définissant cette disjonction sont nécessaires. À titre indicatif, dans l'exemple des services de réservation de billets d'avion et de location de voiture, les concepts atomiques de haut niveau *Countries* et *Transport_Devices* peuvent être transformés en constantes Event-B incluses dans l'unique ensemble *Thing*, avec l'axiome suivant : $axmD : Countries \cap Transport\_Devices = \varnothing$. Dans le cas, où l'on transforme les concepts atomiques de haut niveau en ensembles Event-B, les deux concepts précédents sont des ensembles Event-B, mais dans ce cas ni l'ensemble *Thing*, ni l'axiome précédent ne sont utiles.

En OWL les deux catégories significatives de propriétés sont : les Object properties qui lient les objets entre eux ; et les Datatype properties qui lient les individus et les valeurs des données.

Notons qu'aucune distinction n'est faite quant aux transformations en Event-B des valeurs des données du domaine $\Delta^{\mathcal{I}}_{\mathcal{D}}$ (les litéraux) et les individus des concepts dans $\Delta^{\mathcal{I}}$ (les références). Les seules différences résident dans leur caractérisation des objets et des données ou des valeurs car les individus permettent une identification univoque ce qui peut correspondre aux éléments dans un ensemble en Event-B, puisque, par définition, les éléments dans un ensemble Event-B sont distincts et uniques. À titre indicatif, l'identifiant AF1448 des vols de la classe Flights est transformé en constante incluse dans cet ensemble ($Flights$) dans le contexte Event-B. Et le litéral euro de la classe Currency se transforme également en constante incluse dans l'ensemble *Currency* dans le contexte Event-B. La

20. Nous parlerons indifféremment d'individus, d'entités ou d'objets.

différence est que les individus peuvent correspondre aux valeurs de variables dans les machines Event-B, ce qui n'est pas le cas des littéraux comme l'indiquent les deux dernières lignes du tableau B.1.

La notation `restri` est l'abréviation de `restriction` dans le premier tableau, et const/var signifie que la transformation résultante est soit une constante soit une variable Event-B selon sa définition dans les modèles Event-B puisque le langage des requêtes sera guidé par les formules définies dans ce formalisme. Le tableau B.3 complète les transformations de tout type de relations, les restrictions des domaines et des co-domaines des relations Event-B vers les logiques de description. Le langage du formalisme Event-B est plus expressif que celui des logiques de description, mais en reformulant certaines formules, nous arrivons donc à obtenir les traductions de toute formule Event-B vers les logiques de description. Par exemple, la notation Event-B $(x \mapsto y) \mapsto (z \mapsto h) \in p \parallel q \Leftrightarrow$ est équivalente à $x \mapsto z \in p \wedge y \mapsto h \in q$. La transformation $T^{-1}(x \mapsto z \in p \wedge y \mapsto h \in q)$ devient $(\mathsf{x}, \mathsf{z}) \in \mathsf{p} \sqcap (\mathsf{y}, \mathsf{h}) \in \mathsf{q}$ dans les logiques de description utilisées, avec p et q deux relations ou deux rôles ontologiques. La traduction inverse T^{-1} n'est pas complète. Par exemple, les prédicats n-aires ne sont pas pris en compte dans la variante des logiques de description utilisée. Cette restriction empêche d'avoir une transformation complète des formules Event-B vers la logique de description $\mathcal{SHOIQ}(\mathcal{D})$. Cela dit, cette transformation a en revanche son interêt en particulier pour extraire des relations binaires qui participent dans la définition de relations n-aires au niveau des contextes Event-B pour aider à la validation de ces derniers. Un autre point, est la preuve des obligations de preuve qui font recours à des propriétés en utilisant des prédicats n-aires. Notre experience montre que dans certains cas, les preuves ne peuvent être faites qu'en décomposant ces relations n-aires en relations binaires. Ce point sera détaillé dans le chapitre qui traite des protocoles de vote.

Les quantificateurs *existentiel* et *universel* en Event-B se transforment respectivement, en *disjonction* et *conjonction* sur l'ensemble des individus concernés par la portée du quantificateur, à savoir le concept ou le sous-concept ontologique correspondant à l'élément (constante, ensemble, variable) défini dans Event-B.

$$\forall x.x \in X ==> \bigwedge_{\mathsf{x} \in \mathsf{X}}$$

$$\exists x.x \in X ==> \bigvee_{\mathsf{x} \in \mathsf{X}}$$

Exemples Pour mieux illustrer nos propos considérons l'exemple où l'on définit une constante Event-B *aircrafts* qui définit l'ensemble des avions comme un sous-ensemble des moyens de transport $Transport\_Devices$ comme suit $aircrafts \subseteq Transport\_Devices$. La traduction de cette axiome Event-B se fait comme suit :

$$T^{-1}(aircrafts \subseteq Transport\_Devices) = \mathsf{aircrafts} \sqsubseteq \mathsf{Transport\_Devices}$$

À noter que cette transformation concerne la terminologie box T-Box.

Si l'on prend l'exemple de la fonction partielle en Event-B définie comme suit : $bills\_customers \in Bills \nrightarrow customers$, avec $Bills$ un ensemble de factures et $customers$ une constante qui identifie les clients et est incluse dans l'ensemble des acteurs $Actors$, la transformation en logique de description de cette fonction selon les tableaux précédents

donne :

$$T^{-1}(bills\_customers \in Bills \nrightarrow customers) = \begin{cases} \top \sqsubseteq \leq 1\ \mathsf{bills\_customers}, \\ \geq 1\ \mathsf{bills\_customers} \sqsubseteq \mathsf{Bills}, \\ \top \sqsubseteq \forall \mathsf{bills\_customers.customers} \end{cases}$$

et concerne aussi la T-Box.

3.4.3 Contextualisation versus. raffinement

Pour mieux illustrer notre démarche, reprenons notre exemple de services de réservation de billets d'avion et de location de voiture. Dans ce qui suit, nous notons les éléments liés au domaine (concepts, rôles ontologiques) en utilisant la police textsf lorsque nous nous situons au niveau du domaine. Par exemple, Tickets indique le concept des tickets issu du domaine. L'ontologie de domaine de ce système fournit les concepts atomiques de haut niveau C_a suivants :

- Transport_Devices : ce sont les dispositifs de transport existants ;
- Tickets : le concept des tickets qui peuvent être utilisés par l'ensemble des services ;
- Flights : les numéros des vols ou les codifications possibles pour les vols ;
- Airports : la liste des aéroports existants ;
- Actors : ce concept représente les entités susceptibles d'effectuer des actions dans un système ;
- Flight_Category : les catégories existantes des vols ;
- Rentals : correspond à l'ensemble des identifications des locations de vehicules ;
- Cars_Types : correspond au type des véhicules utilisés ;
- Cars_Classes : correspond aux classes des véhicules utilisés ;
- IDCars : ce concept identifie l'ensemble des véhicules et contient leurs immatriculations ;
- Countries : contient les noms des pays ;
- Bills : identifie l'ensemble des factures ;

Ces concepts sont subsumés par le concept global $\top$. La transformation de ces concepts peut être une transformation en constantes Event-B incluses dans l'ensemble *Thing* avec les contraintes qui indiquent que l'intersection entre chaque paire de constantes est vide comme expliqué dans ce qui prècède ; ou bien une traduction en ensembles Event-B dans le cas où l'on souhaiterait supprimer l'ensemble *Thing*.

Dans ce cas, nous considérons par exemple, le concept Tickets comme étant une généralisation des deux sous-concepts flight_Tickets et renter_Tickets, avec les contraintes qui indiquent que leur intersection est vide.

$$\begin{gathered} (\mathsf{flight\_Tickets} \sqsubseteq \mathsf{Tickets}) \\ \sqcap\ (\mathsf{renter\_Tickets} \sqsubseteq \mathsf{Tickets}) \\ \sqcap\ (\mathsf{flight\_Tickets} \sqcap \mathsf{renter\_Tickets} = \bot) \end{gathered}$$

Ces contraintes du domaine se transforment selon les correspodances données sur les tableaux en annexe B (cf. tableaux B.1, B.2, et B.3) en suivant la fonction de traduction T donnée en axiomes dans un contexte Event-B comme suit :

$$\begin{gathered} (flight\_Tickets \subseteq Tickets) \\ \wedge\ (renter\_Tickets \subseteq Tickets) \\ \wedge\ (flight\_Tickets \cap renter\_Tickets = \varnothing) \end{gathered}$$

Le même principe est appliqué au concept Transport_Devices, où il sera spécialisé en vehicles et aircrafts. Ces deux sous-concepts sont transformés en constantes dans le contexte Event-B correspondant avec les mêmes axiomes que ceux définis pour le concept Tickets et ses sous-concepts. Les deux sous-concepts sont disjoints car un véhicule ne peut être un avion. Le domaine n'exclut pas l'existence d'autres catégories ou moyens de transport comme les bateaux, les TGV etc. Cette contrainte du domaine n'impose aucune restriction sur le concept subsumant Transport_Devices justifiant ainsi l'absence de la contrainte d'égalité quant aux deux sous-concepts au concept Transport_Devices c-à-d : $Tickets = flight\_Tickets \cup renter\_Tickets$. Cette représentation des concepts n'est pas unique, et il peut être aussi intéressant de ne considérer que ces sous-concepts dans l'ontologie auquel cas le concept général Tickets disparaîtrait, et les deux sous-concepts deviennent des concepts atomiques de haut niveau. Le concept Actors n'est qu'une généralisation du concept customers défini comme constante dans le contexte Event-B et qui correspond aux clients qui se servent des services offerts sur internet.

Cette classification offre une extensibilité des définitions dans le cas où il y aurait d'autres types d'interactions à considérer dans le système. Cette classification est basée sur une identification unique des individus au sein de leur concept ou sous-concept qui permettrait par la suite de raisonner correctement sur les comportements des clients dans cet exemple à titre indicatif. En effet, cette classe ou concept n'est autre que l'ensemble des identifiants qui permettent de distinguer de façon unique un client lors de son utilisation des services de réservation et de location. Les objets sémantiques concernés dans ce cas sont les immatriculations mentionnées sur les passeports des personnes. Implicitement, il s'agit des noms et prénoms des personnes concernées. Dans la conceptualisation d'un domaine, ces deux informations ne peuvent en aucun cas distinguer les clients faisant recours aux deux services car un même nom peut être attribué à plusieurs personnes. Cette situation n'est pas permise car elle peut être source d'ambiguïtés. Le domaine définit donc des relations, ou dans ce cas précis les attributs, Noms, qui contient les noms des personnes ainsi que l'attribut Prenoms, contenant les prénoms possibles d'un domaine. Leur domaine est le concept des acteurs alors que leur co-domaine est l'ensemble des chaînes de caractères. Un numéro de passeport n'est alors attribué qu'à une et une seule personne constituant ainsi les propriétés sémantiques *définissantes* c-à-d, *nécessaires* et *suffisantes*. D'autres attributs sont aussi définis tels que les dates de naissances, etc.

La conceptualisation d'une ontologie de domaine dépend de sa profondeur hiérarchique ou de sa granularité. Cette représentation est un facteur qui peut être aussi déterminant dans la conception d'un système, et déterminer ainsi le niveau de détail des raffinements des modèles en Event-B. Les visions sur ce point divergent selon les objectifs et les utilisations faites de cette représentation la rendant ainsi subjective. Notre souhait est à la fois de pouvoir identifier de façon unique les individus de manière à les classer dans des ensembles Event-B à des fins de typage, et de trouver les propriétés nécessaires et suffisantes qui les définissent pour produire des représentations sans ambiguïtés sémantiques d'une part, mais aussi de pouvoir déterminer les concepts intensionnels qui définissent les actions dans la partie dynamique d'un système, permettant de produire les comportements attendus tout en respectant les propriétés qui établissent un raffinement correct vis-à-vis des traces d'exécution dans un modèle Event-B d'autre part. Un autre objectif est celui de faciliter l'adoption des conceptions réalisées par un grand nombre de concepteurs dans le but de limiter les efforts de développement à faire. Une représentation guidée par les propriétés sémantiques (rôles du domaine) d'un concept du domaine favorise une structuration des états dans une machine, et leur granularité permet de définir des niveaux

de raffinements des modèles d'un système. Les rôles du domaine définissant plus précisément les propriétés extrinsèques sont ceux qui définissent les structures des états dans les machines Event-B.

Les ontologies de domaine ont pour but d'atteindre une représentation commune des connaissances selon des accords préalablement établis, alors que la médiation et les alignements permettent d'effectuer des transformations selon le contexte pour le modifier, selon les trois axes d'hétérogénéité donnés dans ce qui suit (valeurs, structures et sémantiques). En Event-B, le raisonnement s'effectue sur des modèles qui décrivent les services et leurs environnements comme des événements dans un modèle, et ce n'est que par raffinement que les détails ainsi que les propriétés établies pour chaque partie sont intégrés. Une telle démarche constitue donc bien une forme de médiation dans notre cas, où la résolution de ces traitements coincident soit avec des raffinements supplémentaires, soit avec des représentations différentes des structures de données, soit au niveau de la validation des modèles.

Nous postulons qu'une approche *a priori* dans les modèles Event-B est plus intéressante à employer pour décrire les concepts et les propriétés liés au domaine en respectant les niveaux de granularité de l'ontologie, et qu'une intégration *a posteriori* par raffinement est plus appropriée pour contextualiser les systèmes selon le point de vue considéré, car le raffinement offre une séparation des vues abstraites et concrètes pour décrire les données. Cette séparation donne un pouvoir d'utilisation de types de représentations différents pour les instances d'un même concept. À titre indicatif, dans l'exemple précédent, le concept Tickets est traduit en ensemble (ou constante selon le choix du concepteur comme décrit dans ce qui précède). C'est ce même concept qui est utilisé dans le premier modèle abstrait en Event-B. Pourtant, ce sont les raffinements [21] qui suivent qui montrent bien qu'il existe des tickets qui correspondent aux billets d'avion et des tickets de location de voitures exploités de façon différentes selon le service en question. Le même principe s'applique au concept Transport_Devices qui est aussi spécialisé, selon son utilisation dans la partie dynamique du système, en sous-concepts aircrafts et vehicles. Cette généralisation/spécialisation est donc importante à considérer dans la conception d'un système pour profiter d'une meilleure factorisation dans la spécification et la preuve de comportements corrects d'un système. Cette approche permet d'appliquer des raisonnements *a priori* sur le domaine pour extraire inductivement des propriétés sur le domaine et *a posteriori* pour déduire des propriétés sur le contexte.

Les propriétés sémantiques décrivent les divers aspects et caractéristiques sémantiques des concepts en étant identifiées par un nom et possédant une valeur instance d'un type (entier, chaîne de caractères, ...etc.), selon les hétérogénéités de valeurs, les hétérogénéités des structures ainsi que les hétérogénéités sémantiques [197].

— **hétérogénéités des valeurs** : Les hétérogénéités des valeurs peuvent s'illustrer dans l'exemple des réservations de billets d'avion et de locations de voiture, par les valeurs de la devise utilisée. Ainsi, dans le contexte d'une réservation de billets d'avion, celle-ci est interprétée dans la devise en Euros, alors que dans le cas de location de voiture, les traitements s'opèrent en Yen japonais. Cette distinction est faite au niveau des valeurs des attributs des concepts en question. De façon générale, dans le cas de la modélisation en Event-B, nous considérons ce type de résolution au niveau de la validation des modèles. Car ce type d'hétérogénéité n'a d'effets ou de

21. Notre modélisation définit ces concepts dans un même contexte Event-B, mais leur utilisation est réalisée par raffinement.

conséquences sur les preuves, mais joue un rôle déterminant lors de l'interprétation des modèles.

— **hétérogénéités des structures** : Les hétérogénéités des structures peuvent s'illustrer dans les représentations des propriétés ou des rôles, leurs cardinalités, etc. Dans notre exemple de réservation, cela peut concerner des représentations différentes des dates et heures selon le pays concerné comme expliqué dans la section précédente. Cette différence doit aussi être explicitée dans la description d'un système avec l'objectif de déterminer leur compatibilité dans les traitements. La résolution de ce type de contexte est souvent effectuée par raffinement, sous ses formes horizontales et verticales, des modèles via les extensions différentes ou pas entre les contextes Event-B. Cette hétérogénéité est illustrée sur les représentations des dates selon leur contexte : représentations européennes ou japonaises. Ainsi, il peut être utile d'établir des correspondances entre les différentes représentations dans l'objectif de résoudre les problèmes liés à leur sémantique pour trouver des terrains d'entente pour les traitements et les échanges entre chaque service. Ce cas a bien été illustré par le développement des protocoles de vote que nous aborderons dans les chapitres suivants.

— **Hétérogénéités sémantiques** : Enfin, les hétérogénéités sémantiques sont liées au vocabulaire utilisé pour la description des concepts et des propriétés. Celles-ci peuvent impacter les noms donnés aux éléments définis dans un modèle Event-B. Si l'on considère l'exemple pris dans [197], cette hétérogénéité peut s'illustrer par l'emploi du mot «VATIncluded», par un fournisseur, tandis que d'autres fournisseurs peuvent utiliser le mot «TVAIncluse», pour décrire la propriété sémantique qui indique si la taxe sur la valeur ajoutée (TVA) est incluse dans le prix. La résolution au niveau des ontologies s'effectue par la définition des équivalences sémantiques en adéquation avec les concepts et les objets en question. Nous estimons que ce cas d'hétérogénéité doit être résolu au niveau du domaine, ainsi le résultat n'est que exploité dans le cas de la modélisation en Event-B. Toutefois, il est aussi possible d'opérer des instanciations des modèles Event-B et des renommages des objets de conception concernés (ensembles, constantes et variables Event-B) dans le cas où les équivalences sont extensionnelles.

Nous rappelons que cette répartition ne concerne que le *contexte des données*. Une classification plus appropriée du contexte est donnée dans ce qui précède, où le contexte de preuve est défini comme étant des connaissances minimales réparties en contraintes, hypothèses et dépendances en raisonnant sur les preuves réalisées au moyen du formalisme Event-B. Le contexte des données est dépendant de cette classification et est nécessaire pour une interprétation correcte des données. Il s'agit en l'occurence de la validation des modèles Event-B dans notre cas.

3.4.4 Application à l'étude de cas

Dans ce qui suit, nous donnons les modèles Event-B de l'exemple des réservations de billets d'avions et de locations de voitures. La modélisation de cette étude de cas illustre bien nos propos. En effet, dans ce modèle abstrait, le seul événement défini est celui relatif aux réservations, mais aucune précision n'est donnée quant aux réservations des billets d'avion, ou de tickets de locations de voitures, et cette représentation n'exclut pas les opérations d'annulation de réservations. Ces opérations sont définies dans les raffinements suivants. Aucune indication n'est donc donnée concernant les tickets employés dans cet

événement et les dispositifs de transports utilisés. Ceci permet donc l'utilisation d'une représentation abstraite commune pour des types et des services différents. Cette démarche correspond bien à une intégration *a posteriori* par rapport aux spécifications données en Event-B, mais *a priori* au niveau abstrait les concepts primitifs du domaine Tickets et Actors sont les seuls objets sémantiques exploités. Nous notons ici que les variables définies dans les machines peuvent être aussi définies dans le domaine. Il est d'ailleurs souhaitable de procéder ainsi, car le caractère holistique des ontologies permet la mention de toutes les entités, les relations et les propriétés définissantes et d'unicité d'individus qui peuvent être d'une grande utilité, particulièrement dans le cadre de la validation des modèles Event-B.

CONTEXT $Context0$
SETS $Thing...$
CONSTANTS
$...aircrafts, vehicles, customers, flight\_Tickets,$
$renter\_Tickets, Prices$
AXIOMS
axm1 : $aircrafts \subseteq Transport\_Devices$
axm2 : $vehicles \subseteq Transport\_Devices$
axm3 : $vehicles \cap aircrafts = \varnothing$
axm4 : $customers \subseteq Actors$
axm5 : $flight\_Tickets \subseteq Tickets$
axm6 : $renter\_Tickets \subseteq Tickets$
axm8 : $flight\_Tickets \cap renter\_Tickets = \varnothing$
axm9 : $Prices = \mathbb{N}...$

MACHINE $M00\_Services$
SEES $Context0$
VARIABLES
$assigned\_tickets$
INVARIANTS
inv1 : $assigned\_tickets \in Tickets \leftrightarrow Actors$
EVENTS
INITIALISATION
act1 : $assigned\_tickets := \varnothing$
END
EVENT Reservation
THEN
act1 : $assigned\_tickets :| assigned\_tickets' \in (Tickets \leftrightarrow Actors)$
END

Au niveau abstrait, les définitions des sous-concepts définis dans le premier contexte Event-B "Contexte0", et non utilisées au niveau de la machine abstraite peuvent être introduites dans un contexte Event-B qui étend le présent contexte, et cette seconde solution est privilégiée. La définition d'un contexte unique est adoptée dans l'unique objectif d'organisation des modèles et n'a aucune conséquences sur les preuves à réaliser. Par l'adoption d'une représentation *top-down*, l'intégration de la sémantique orientée contexte est confiée au mécanisme de raffinement pour résoudre les hétérogénéités liées au contexte, sans pour autant surcharger les modèles abstraits. Ainsi, les preuves peuvent être aussi plus faciles à décharger. Seule la réquisition d'une ontologie de domaine est nécessaire pour le niveau de raffinement considéré pour l'extraction *a priori* de propriétés du domaine de manière

inductive[22]. Le principe de généralisation/spécialisation correspond donc au principe de modélisation par abstraction/raffinement, si l'on considère les modèles Event-B comme objets sur lesquels on peut raisonner (référent) (cf. définition 1.8.1).

3.4.4.1 Premier raffinement

Dans ce premier raffinement, nous distinguons les réservations effectives de leurs annulations. Les seules variables introduites sont des variables qui précisent les tickets réellement affectés aux clients (*reserved_tickets*) (**inv1**), les relations entre ces tickets et leurs clients se spécialisent par la définition d'une fonction *assigned_tickets* (**inv3**) à ce niveau de raffinement et est donc subsumée par la relation définie dans le modèle abstrait. Le sous-concept des clients effectivement enregistrés dans la base de données du système est également une spécialisation du concept *customers* (**inv2**). Nous notons *customer* pour abréger *effective_customers* pour une visibilité meilleure du code des modèles Event-B.

MACHINE $M01\_Services$
REFINES $M00\_Services$
SEES $Context0$
VARIABLES
$reserved\_tickets, assigned\_tickets, customer$
INVARIANTS
inv1 : $reserved\_tickets \subseteq Tickets$
inv2 : $customer \subseteq customers$
inv3 : $assigned\_tickets \in reserved\_tickets \rightarrow customers$
inv4 : $ran(assigned\_tickets) = customer$
EVENTS
INITIALISATION
act1 : $reserved\_tickets := \varnothing$
act2 : $customer := \varnothing$
act3 : $assigned\_tickets := \varnothing$
END

22. Nous donnons quelques pistes sur comment cette extraction peut se définir, notamment en utilisant des vues logiques définies dans le formalisme de l'analyse de concepts formelle (ACL) [118], et comment elle peut être guidée par les propriétés définies pour cette vue (niveau de raffinement considéré). Cette extraction est orientée but car les obligations de preuve définissent les requêtes d'extraction. Les vues logiques correspondent au raisonnement ontologique *a posteriori*.

EVENT Reservation
REFINES Reservation
ANY p, t
WHERE
grd1 : $p \in customers$
grd2 : $t \in Tickets$
grd3 : $t \notin reserved\_tickets$
grd4 : $t \notin dom(assigned\_tickets)$
grd5 : $t \mapsto p \notin assigned\_tickets$
grd6 : $\forall pp \cdot pp \in customer \Rightarrow t \mapsto pp \notin assigned\_tickets$
THEN
act1 : $reserved\_tickets := reserved\_tickets \cup \{t\}$
act2 : $assigned\_tickets(t) := p$
act3 : $customer := customer \cup \{p\}$
END
EVENT Cancellation
REFINES Reservation
ANY p, t
WHERE
grd1 : $t \subseteq reserved\_tickets$
grd2 : $p \in customer \ \wedge t = assigned\_tickets^{-1}[\{p\}]$
THEN
act1 : $reserved\_tickets := reserved\_tickets \setminus t$
act2 : $assigned\_tickets := t \lhd assigned\_tickets$
act3 : $customer := customer \setminus \{p\}$
END

3.4.4.2 Deuxième raffinement

Les différents environnements des services définissent les contextes des interactions entre ces services. Par raffinement des transformations des données échangées entre les différents services impliqués dans ces interactions peuvent être opérées. Ces transformations concernent les interprétations des données du contexte émetteur vers le contexte récepteur, car la séparation des vues abstraites et concrètes pour la description de l'état du système est confiée au mécanisme de raffinement. Le présent raffinement spécialise les tickets selon leur utilisation en tickets correspondant aux billets d'avion (**inv1**) employés par le service de réservation de billets d'avion, et ceux consacrés pour la location de voitures (**inv2**). Ainsi, les réservations des billets d'avion se lient aux passagers des avions (**inv3**), alors que la propriété *car_rentals* est une propriété liée aux tickets de location de voiture et les loueurs des véhicules (**inv5**), en gardant à l'esprit que ces réservations sont instantanées (**inv4** et **inv5**) définissant ainsi des bijections entre le domaine et le co-domaine de chaque relation.

INVARIANTS
inv1 : $passengers \subseteq customers$
inv2 : $renters \subseteq customers$
inv3 : $booking\_plane\_ticketsID \in flight\_Tickets \nrightarrow passengers$
inv4 : $ran(booking\_plane\_ticketsID) = passengers$
$\wedge\ booking\_plane\_ticketsID^{-1} \in passengers \nrightarrow flight\_Tickets$
inv5 : $car\_rentals \in renter\_Tickets \nrightarrow renters$
$\wedge\ ran(car\_rentals) = renters \wedge car\_rentals^{-1} \in renters \nrightarrow renter\_Tickets$
inv6 : $ran(booking\_plane\_ticketsID) \cup ran(car\_rentals) = ran(assigned\_tickets)$
inv7 : $reserved\_tickets = dom(booking\_plane\_ticketsID) \cup dom(car\_rentals)$
inv8 : $assigned\_tickets = booking\_plane\_ticketsID \cup car\_rentals$
thm9 : $dom(booking\_plane\_ticketsID) \cap dom(car\_rentals) = \varnothing$
...

Nous notons que cette spécification est également liée au domaine, mais n'est pas réaliste car un client peut effectuer plusieurs réservations de billets d'avion, sous la condition que les dates des différentes réservations soient différentes. La connaissance implicite dans cette représentation est que la date de réservations des vols est instantanée. Pour considérer cette contrainte, les dates et les rôles définis pour les billets d'avion et leur date de réservation doivent être introduits ou explicités. Les dates font l'objet du dernier raffinement. Cependant, nous ne traitons pas cette contrainte car la priorité est donnée aux représentations du contexte et des propriétés issues du domaine et les aspects abordés dans cette étude de cas couvrent pratiquement tous les points pertinents pour définir et justifier des stratégies de représentation et d'extraction des connaissances du domaine.

À ce stade de raffinement, les deux variables *assigned_tickets* et *reserved_tickets* sont abstraites et substituées par les nouvelles variables accompagnées des invariants de collage **inv6**, **inv7**, ainsi que **inv8**. Ces propriétés sont aussi définies au niveau du domaine car la spécialisation de ces variables abstraites est interprétée selon les intensions (actions) faites des objets sémantiques définis par chaque service c-à-d : les concepts intensionnels. Il est ainsi évident que les domaines des rôles concernés par la réservation des billets d'avion (*booking_plane_ticketsID*) et la location de voiture (*car_rentals*) soient disjoints ce qui est illustré par le théorème **thm9**. C'est une propriété déduite sur le contexte des deux services car l'on dispose préalablement de la propriété illustrée par l'axiome **axm8** ($flight\_Tickets \cap renter\_Tickets = \varnothing$) défini dans le premier contexte Event-B ci-dessus.

Les variables telles que $booking\_plane\_ticketsID^{-1}$ et $car\_rentals^{-1}$ sont également construites selon le domaine puisqu'une approche holistique de Quine considère qu'une structure (les concepts, les objets et les relations) ne se réduit pas à elle même, mais est considérée dans son ensemble n'ayant du sens qu'en entretenant des relations avec la totalité des structures (concepts, les objets et les relations) observationnelles qui composent un schéma conceptuel. Ces invariants sont utiles à considérer pour l'événement « Cancellation » et permettent de préciser par exemple la propriété du sous-concept *passengers* par la relation $booking\_plane\_ticketsID^{-1}$ qui définit les individus de ce sous-concept en relation avec des billets d'avion.

L'événement « Reservation » consacré aux réservations est raffiné en deux événements qui modélisent les réservations de billets d'avion « Flight_Booking »et les locations de voitures « Car_Rental ». L'événement « Cancellation » d'annulation concerne à la fois l'annulation du vol ainsi que la location de voiture en question comme illsutré dans la

suite :

EVENT Flight_Booking
REFINES Reservation
ANY p, f
WHERE
grd1 : $p \in customers$
grd2 : $f \in flight\_Tickets$
grd3 : $f \notin dom(booking\_plane\_ticketsID)$
grd4 : thm $f \mapsto p \notin booking\_plane\_ticketsID$
grd5 : thm $f \notin renter\_Tickets$
grd6 : $p \notin passengers$
with t : $t = f$
THEN
act1 : $booking\_plane\_ticketsID(f) := p$
act2 : $passengers := passengers \cup \{p\}$
end

EVENT Car_Rental
REFINES Reservation
ANY c, t
WHERE
grd1 : $c \in customers \land c \notin renters \land c \in passengers$
grd2 : $t \in renter\_Tickets$
grd3 : $t \notin dom(car\_rentals)$
grd4 : thm $t \notin flight\_Tickets$
with p : $p = c$
THEN
act1 : $car\_rentals(t) := c$
act2 : $renters := renters \cup \{c\}$
end

EVENT Cancellation
REFINES Reservation
ANY p, tf, tc
WHERE
grd1 : $tf \in dom(booking\_plane\_ticketsID) \land p \in passengers$
grd2 : $p \in ran(booking\_plane\_ticketsID)$
grd3 : $booking\_plane\_ticketsID(tf) = p$
grd4 : $\{tf\} = booking\_plane\_ticketsID^{-1}[\{p\}]$
grd5 : $tc \in dom(car\_rentals) \land p \in renters \land car\_rentals(tc) = p$
grd6 : $\{tc\} = car\_rentals^{-1}[\{p\}]$
with t : $t = \{tf, tc\}$
THEN
act1 : $booking\_plane\_ticketsID := \{tf\} \lhd booking\_plane\_ticketsID$
act2 : $passengers := passengers \setminus \{p\}$
act3 : $renters := renters \setminus \{p\}$
act4 : $car\_rentals := \{tc\} \lhd car\_rentals$
end

Notons que la clause *with* définie au niveau de l'événement « Cancellation » sert à lier les paramètres abstraits de l'événement raffiné « Reservation » à leur paramètres concrets donnés par l'événement « Cancellation ». Dans cet exemple, il s'agit de définir deux nouveaux paramètres (tf, tc) qui remplacent l'unique paramètre t donné au précédent raffinement.

3.4.4.3 Troisième raffinement

Ce raffinement introduit les numéros des vols (**inv1**) ainsi que ceux consacrés aux locations de véhicules (**inv2**). Ainsi, les variables *booking_plane_flights* et *car_rentals_vehicles* lient respectivement, les billets d'avion à leur numéros de vol (*Flights*), et les tickets de location des véhicules à leur numéro de voyage (*Rentals*). *Flights* et *Rentals* sont définies au niveau du premier contexte Event-B « Context0 ». Par transitivité, les billets réservés sont liés au dispositifs pour lesquels les numéros des vols sont affectés (**inv11**) et cette propriété *flight_Tickets_aircraft* est équivalente à la composition de la variable *booking_plane_flights* et la constante *booking_plane_tickets_flights* définie dans le contexte « Context0 »(**thm13**). Cette propriété est déduite de l'invariant (**inv12**). Cet invariant est extrait du domaine et est nécessaire pour la preuve de l'invariant principal défini dans le raffinement suivant. L'extraction de cet invariant est donc effectuée par

induction sur le domaine, mais selon un raisonnement ontologique déductif suivant une identification et une classification des individus et des instances des propriétés définies dans cet invariant. Nous expliquons ce point plus en détail dans le dernier raffinement, car il y existe des subtilités concernant ce point. Nous parlons d'extraction inductive ou déductive au niveau des modèles Event-B, mais il peut s'agir d'une déduction au niveau des raisonneurs ontologiques pour une extraction inductive. Les numéros des sièges occupés des dispositifs de transport (avions et véhicules) sont aussi introduits à ce stade de raffinement. Les mêmes propriétés sont aussi définies pour les locations de véhicules.

INVARIANTS

inv1 : $booking\_plane\_flights \in flight\_Tickets \nrightarrow Flights$

inv2 : $car\_rentals\_vehicles \in renter\_Tickets \nrightarrow Rentals$

...

inv11 : $flight\_Tickets\_aircraft \in flight\_Tickets \nrightarrow aircrafts$
$\wedge dom(booking\_plane\_flights) = dom(flight\_Tickets\_aircraft)$
$\wedge booking\_plane\_flights \in flight\_Tickets \nrightarrow Flights$

inv12 : $\forall f, air \cdot f \mapsto air \in flight\_Tickets\_aircraft$
$$\Rightarrow \left(\begin{array}{l} \exists F \cdot F \in Flights \\ \wedge F \in ran(booking\_plane\_flights) \\ \wedge f \mapsto F \in booking\_plane\_flights \\ \wedge F \in dom(booking\_plane\_tickets\_flights) \\ \wedge F \mapsto air \in booking\_plane\_tickets\_flights \end{array} \right)$$

thm13 : $flight\_Tickets\_aircraft = booking\_plane\_flights; booking\_plane\_tickets\_flights$

inv14 : $\forall f, air \cdot f \notin dom(flight\_Tickets\_aircraft) \wedge air \notin ran(flight\_Tickets\_aircraft)$
$$\Rightarrow \neg \left(\begin{array}{l} \exists F \cdot F \in Flights \wedge F \in ran(booking\_plane\_flights) \\ \wedge f \mapsto F \in booking\_plane\_flights \\ \wedge F \in dom(booking\_plane\_tickets\_flights) \\ \wedge F \mapsto air \in booking\_plane\_tickets\_flights \end{array} \right)$$

...

L'événement responsable des réservations de billets d'avion est renforcé par les gardes suivantes :

grd8 : $nmbr\_occupied\_seats\_aircraft(air) < seats\_number\_aircrafts(air)$

grd9 : $f \notin dom(booking\_plane\_flights) \wedge f \mapsto fid \notin booking\_plane\_flights$

grd10 : $fid \in dom(booking\_plane\_tickets\_flights) \wedge booking\_plane\_tickets\_flights(fid) = air$

où *seats_number_aircrafts* est une constante qui définit les nombre de sièges pour chaque appareil de navigation aérienne. Dans ce même événement de réservations des billets d'avion, le nombre des sièges est incrémenté par 1 pour chaque réservation (**act4**) et le billet en question est attaché au numéro de vol choisi (**act3**) et au dispositif de transport concerné (**act5**).

act3 : $booking\_plane\_flights(f) := fid$

act4 : $nmbr\_occupied\_seats\_aircraft(air) := nmbr\_occupied\_seats\_aircraft(air) + 1$

act5 : $flight\_Tickets\_aircraft(f) := air$

Nous notons que d'autres propriétés sont aussi définies dans cette machine pour par exemple restreindre le nombre de sièges affectés aux sièges disponibles donnés par la

constante *seats_number_aircrafts*, pour lier les domaines des relations *booking_plane_flights* et *flight_Tickets_aircraft* ...

3.4.4.4 Quatrième raffinement

C'est à ce niveau de raffinement où sont introduits les tarifs des billets d'avion ainsi que ceux des locations des véhicules et les factures qui leur correspondent. Chaque facturation est effectuée par le service dédié : les billets d'avion (**inv1**) et les tickets des véhicules (**inv2**). Le tarif à règler par le client est représenté par la variable *price_to_pay* (**inv3**), et les factures sont attribuées aux clients (**inv7**) via la variable *bills_customers* qui a comme domaine le sous-concept *paid_bills* (**inv6**) représentant des factures acquittées (**inv8**). Ainsi, chaque facture a un prix (**inv18**) ayant *paid_bills* comme domaine (**inv19**), et inversement, chaque billet d'avion n'est lié qu'à une et une seule facture (**inv11**). Les mêmes propriétés sont aussi définies pour les locations de voitures. D'autres propriétés liées aux définitions des domaines et des images de chaque propriété sont aussi intégrées à ce niveau de raffinement.

Le service responsable d'additionner les tarifs des billets d'avion et de locations de voitures est concrétisé par l'événement « Addition_Service ». L'addition des tarifs est définie par la somme des prix des billets d'avions et des tickets de location de voitures (**act1**). Cette addition est gardée particulièrement par les gardes **grd4**, **grd6** et **grd10** pour montrer que cet événement ne peut être observé que si les billets et les tickets sont préalablement facturés et que la facture n'a pas été attribuée (acquittée). Les gardes **grd18**, **grd19 grd20** ainsi que la condition ($\forall\ cust2 \cdot cust2 \neq cust \Rightarrow f \mapsto cust2 \notin booking\_plane\_ticketsID$) de la garde **grd4** sont des propriétés extraites inductivement sur le domaine. Ces gardes sont le résultat d'une extraction inductive de propriétés du domaine, et elles sont inductivement construites dans l'ontologie de domaine par définition. Il s'agit donc de propriétés ontologiques inductives également.

L'invariant principal dans cette machine est l'invariant **inv21** qui permet d'établir que toute facture affectée à un client et établie avec un prix est toujours définie par la somme des tarifs des billets d'avion et de location de voiture de ce même client. Cet invariant donne une précision sans perte sur les prix que le client paie. Il ne nous est pas possible de prouver ou de déduire cet invariant sans la propriété du domaine définie par l'invariant (**inv20**), car il est évident que pour chaque facture établie, il existe toujours un client à qui elle a été affectée. Ce type de propriété est donc relatif à l'holisme expliqué ci-dessus stipulant qu'une approche qui considère les parties dans leur ensemble (le tout) montrant ainsi que tous les objets sont reliés entre eux pour constituer des faits. L'implication logique n'est pas définie dans les formalismes des logiques de description. Cette propriété n'est qu'une déduction suivant le mécanisme de raisonnement si l'on considère l'ensemble des individus donnés dans la composante assertionnelle A-Box d'une ontologie. En effet, cette propriété est une classification dans leur relation (propriété sémantique) des instances de propriété vérifiant la relation indiquée entre factures, prix et clients. Ce sont ces propriétés que nous pouvons extraire inductivement sur le domaine afin de réussir les preuves des modèles Event-B. L'existence d'un client vérifiant cette propriété n'est qu'une identification des instances de la propriété *bills_customers* dans l'ensemble des individus du concept *customers* pour les clients et des individus *Bills* du côté des factures. En conclusion, il s'agit d'une identification d'un individu du sous-concept *customers* ayant été classé par les propriétés *bills_customers* et *price_to_pay* suivant l'hypothèse d'exis-

tence d'une facture ayant été classée par la propriété *bills_price* et par ce même prix et cette même facture.

Cette vérification permettrait non seulement de vérifier la cohérence des propriétés écrites ou modélisées en Event-B, mais de déduire de nouvelles connaissances sur le domaine et ou le contexte. Ces raisonnements ne changent pas les objets, mais les identifient et les classent seulement. L'identification se fait au niveau assertionnelle, alors que la classification est réalisée au niveau terminologique de l'ontologie. Cette propriété est une déduction au niveau des raisonnements ontologiques et elle est inductivement extraite au niveau des modèles Event-B selon le niveau de raffinement en question. Cette déduction au niveau de l'ontologie correspond au contexte des données ou des objets présents dans la composante assertionnelle A-Box, mais il s'agit d'une propriété vraie pour tous les comportements décrits dans la machine au niveau de la modélisation en Event-B, d'où leur classification comme étant des propriétés du domaine.

Nous pouvons donc établir les règles suivantes pour un processus d'intégration de propriétés ontologiques pour la vérification :

1. L'extraction définie par induction dans les machines en Event-B est fonction de la propriété à extraire :
   — S'il s'agit de propriétés quantifiant de manière existentielle (utilisant le quantificateur $\exists$) telles que c'est le cas pour l'invariant **inv20**, alors celles-ci sont déduites par le raisonneur ontologique selon le contexte (les faits ou les données existants dans la composante assertionnelle d'une ontologie), mais elles sont induites dans le processus d'extraction en Event-B car ces propriétés sont vraies pour tous les comportements définis dans une machine. Il peut s'agir de propriétés ontologiques de dépendance, citons par exemple dans l'exemple : Un client ne peut avoir une facture que s'il a réglé une somme d'argent (cf . invariant 20). Dans ce cas, il existe bien une dépendance entre ces deux propriétés ontologiques (ou relations sémantiques) ;
   — S'il s'agit d'une propriété qui quantifie universellement (utilisant le quantificateur $\forall$), alors celle-ci est inductivement extraite dans le processus d'extraction, mais elle est aussi définie inductivement car elle est vraie indépendamment de tout comportements dans un système. Elle peut également correspondre à une garde d'un événement ;
2. Pour la partie statique d'un modèle Event-B, à savoir, le contexte Event-B :
   — Les contraintes du domaine sont à la fois induites des raisonnements ontologiques et dans le processus d'extraction. Il peut s'agir d'extraire des concepts, sous-concepts ou des relations mais également des individus selon les besoins dans les contextes Event-B, mais tout en respectant le degré de granularité des connaissances. Ce degré peut être guidé par les éléments définis à chaque niveau de raffinement. Les propriétés extraites pour un contexte Event-B peuvent être des propriétés intrinsèques qui définissent les individus de manière à les identifier et les classer dans leur concept auquel ils appartiennent ;
   — Les contraintes déduites selon le contexte par le raisonneur ontologique sont aussi déduites (des théorèmes) dans le processus d'extraction pour les modèles Event-B.

Il faut noter que cette approche suit le raisonnement ontologique a posteriori, où il est possible de ne considérer que les propriétés des concepts selon une vue considérée. Ce mécanisme n'est pas défini au niveau des raisonneurs des logiques de description étudiées

dans cette thèse. Nous préconisons de définir ce mécanisme d'extraction en utilisant des vues logiques définies dans l'analyse de concepts logique. C'est une association par intégration partiellement automatisée, où selon l'hypothèse sémiotique présentée dans les chapitres concerant l'état de l'art (cf. sous-section 1.8.1), le modèle Event-B correspond au référent définissant la dimension syntaxique, l'ontologie définit les aspects declaratifs sémantiques du modèle et les comportements prescrits dans les machines correspondent à la dimension pragmatique.

INVARIANTS

inv1 : $price\_trip \in flight\_Tickets \pfun \mathbb{N}_1$

inv2 : $price\_renter \in renter\_Tickets \pfun \mathbb{N}_1$

inv3 : $price\_to\_pay \in customers \pfun \mathbb{N}_1$

inv6 : $paid\_bills \subseteq Bills$

inv7 : $bills\_customers \in Bills \pfun customers$

inv8 : $dom(bills\_customers) = paid\_bills$ ...

inv11 : $bills\_ftickets^{-1} \in flight\_Tickets \pfun Bills$ ...

inv18 : $bills\_price \in Bills \pfun \mathbb{N}$

inv19 : $dom(bills\_price) = paid\_bills$

inv20 : $$\forall b, n \cdot \left(\begin{array}{l} b \in Bills \land n \in \mathbb{N}_1 \land b \mapsto n \in bills\_price \\ \Rightarrow \left(\begin{array}{l} \exists cust \cdot cust \in customers \land b \mapsto cust \in bills\_customers \\ \land cust \mapsto n \in price\_to\_pay \end{array} \right) \end{array} \right)$$

inv21 : $$\forall cust, price, f, t, bil \cdot \left(\begin{array}{l} price \in \mathbb{N}_1 \land bil \mapsto cust \in bills\_customers \land bil \mapsto f \in bills\_ftickets \\ \land bil \mapsto t \in bills\_rtickets \land price\_to\_pay(cust) = price \\ \land bil \mapsto price \in bills\_price \land bil \in paid\_bills \\ \Rightarrow \left(\begin{array}{l} \forall p1, p2 \cdot \left(\begin{array}{l} p1 \in \mathbb{N}_1 \land p2 \in \mathbb{N}_1 \land f \mapsto p1 \in price\_trip \\ \land t \mapsto cust \in car\_rentals \\ \land f \mapsto cust \in booking\_plane\_ticketsID \\ \land t \mapsto p2 \in price\_renter \\ \Rightarrow p1 < price \land p2 < price \land price = p1 + p2 \end{array} \right) \end{array} \right) \end{array} \right)$$

EVENTAddition_Service
ANY $cust, f, t, bil, cost1, cost2$
WHERE
grd1 : $cust \in ran(booking\_plane\_ticketsID) \land cust \in ran(car\_rentals)$
grd2 : $f \in flight\_Tickets \land f \in dom(booking\_plane\_ticketsID)$
grd3 : $t \in renter\_Tickets$
grd4 : $booking\_plane\_ticketsID(f) = cust$
$\land(\forall cust2 \cdot cust2 \neq cust \Rightarrow f \mapsto cust2 \notin booking\_plane\_ticketsID)$
grd5 : $t \in dom(car\_rentals)$
grd6 : $car\_rentals(t) = cust$
grd7 : $f \in dom(price\_trip)$
grd8 : $t \in dom(price\_renter)$
grd9 : $bil \in Bills$
grd10 : $bil \notin paid\_bills$
grd11 : $cost1 \in \mathbb{N}_1 \land f \in dom(price\_trip) \land cost1 = price\_trip(f)$
grd12 : $cost2 \in \mathbb{N}_1 \land t \in dom(price\_renter) \land cost2 = price\_renter(t)$
grd13 : $bil \notin dom(bills\_customers) \land bil \mapsto cust \notin bills\_customers$
grd14 : $bil \notin dom(bills\_ftickets) \land bil \mapsto f \notin bills\_ftickets$
grd15 : $bil \notin dom(bills\_rtickets) \land bil \mapsto t \notin bills\_ftickets$
grd16 : $f \notin ran(bills\_ftickets)$
grd17 : $t \notin ran(bills\_rtickets)$
grd18 : thm $\forall cc \cdot cc \neq cust \Rightarrow bil \mapsto cc \notin bills\_customers$
grd19 : thm $\forall ff \cdot ff \neq f \Rightarrow bil \mapsto ff \notin bills\_ftickets$
grd20 : thm $\forall tt \cdot tt \neq t \Rightarrow bil \mapsto tt \notin bills\_rtickets$
grd21 : $bil \notin dom(bills\_price)$
THEN
act1 : $price\_to\_pay(cust) := cost1 + cost2$
act2 : $paid\_bills := paid\_bills \cup \{bil\}$
act3 : $bills\_customers := bills\_customers \cup \{bil \mapsto cust\}$
act4 : $bills\_ftickets(bil) := f$
act5 : $bills\_rtickets(bil) := t$
act6 : $bills\_price(bil) := cost1 + cost2$
end
...

Ce raffinement introduit également les dates des réservations de billets d'avion et de location de véhicule. Ainsi, chaque vol et chaque trajet est programmé selon des dates précisées dans le contexte Event-B « Context1 »(**axm1**, **axm2**) qui étend les deux contextes « DateHour0 »et « DateHour1 ». Et il est évident que la programmation des heures de départ des vols soit en avance par rapport aux heures d'arrivée (**axm3**).

axm1 : $programedtripsdate \in Flights \rightarrow (convertdatetonumber \rightarrowtail\!\!\!\!\!\!\rightarrow convertdatetonumber)$
axm2 : $programedrentalsdate \in Rentals \rightarrow convertdatetonumber$
axm3 : $\forall d1, d2, f \cdot d1 \mapsto d2 \in programedtripsdate(f) \Rightarrow d1 < d2$

convertdatetonumber est définie dans le contexte « DateHour1 »comme suit :

axm7 : $convertdatetonumber = \{((yy * 10000) + (mh * 100) + dd) \mid Dates(yy \mapsto mh \mapsto dd) \in Hours\}$

Elle permet d'effectuer des conversions des dates en nombres pour pouvoir faire des com-

paraisons lors des locations des véhicules (voir **grd12** suivante). *Dates* et *Hours* correspondent aux dates et heures définies par les axiomes 1 et 6 selon les formats japonnais dans le contexte « DateHour »comme suit :

axm1 : $Hours = \{hh \mapsto mm \mapsto ss | hh \in 0 \mathinner{..} 24 \land mm \in 0 \mathinner{..} 60 \land ss \in 0 \mathinner{..} 60\}$
axm2 : $incrementhh = (\lambda hh \mapsto mm \mapsto ss \cdot (hh \mapsto mm \mapsto ss) \in Hours \land mm = 60 \land hh < 24 \land ss = 60 | ((hh + 1) \mapsto 0 \mapsto 0))$
axm3 : $incrementmm = (\lambda hh \mapsto mm \mapsto ss \cdot (hh \mapsto mm \mapsto ss) \in Hours \land mm < 60 \land hh < 24 \land ss = 60 | (hh \mapsto (mm + 1) \mapsto 0))$
axm4 : $incrementss = (\lambda hh \mapsto mm \mapsto ss \cdot (hh \mapsto mm \mapsto ss) \in Hours \land mm < 60 \land hh < 24 \land ss < 60 | (hh \mapsto mm \mapsto (ss + 1)))$
axm5 : $Years = \mathbb{N}_1$
axm6 : $Dates = \{(yy \mapsto mh \mapsto dd) \mapsto (hh \mapsto mm \mapsto ss) | (hh \mapsto mm \mapsto ss) \in Hours \land yy \in Years \land mh \in 1 \mathinner{..} 12 \land dd \in 1 \mathinner{..} 31\}$
axm7 : $\forall y, d, m, h \cdot (((y \mapsto m \mapsto d) \mapsto h \in Dates \land h \in Hours \land m \in \{1, 3, 5, 7, 8, 10, 12\}) \Rightarrow d \in 1 \mathinner{..} 31)$
axm8 : $\forall y, d, m, h \cdot (((y \mapsto m \mapsto d) \mapsto h \in Dates \land h \in Hours \land m \in \{4, 6, 9, 11\}) \Rightarrow d \in 1 \mathinner{..} 30)$
axm9 : $\forall y, d, m, h \cdot (((y \mapsto m \mapsto d) \mapsto h \in Dates \land h \in Hours \land m = 2 \land (\exists x \cdot (x \in \mathbb{N} \land y = 4 * x))) \Rightarrow d \in 1 \mathinner{..} 29)$
axm10 : $\forall y, d, m, h \cdot (((y \mapsto m \mapsto d) \mapsto h \in Dates \land h \in Hours \land m = 2 \land (y/4) \notin \mathbb{N}) \Rightarrow d \in 1 \mathinner{..} 28)$
...

Rappelons que ces contraintes sont liées au *contexte de contraintes* de chaque utilisation et sont notées par *C-Axm* et *C-Thm*. Notons que l'utilisation dans les définitions du symbole λ correspond à des définitions des ensembles en compréhension.

Ainsi, l'événement de location de véhicules est renforcé par la garde suivante pour tenir compte des contraintes des heures à respecter afin de n'effectuer des locations de véhicule qu'après les heures d'arrivées des vols programmés :

grd12 : $\forall d1, d2 \cdot \left(\begin{array}{c} d1 \mapsto d2 \in programedtripsdate(booking\_plane\_flights(booking\_plane\_ticketsID^{-1}(c))) \\ \Rightarrow programedrentalsdate(idv) \geq d2 \end{array} \right)$

Nous notons ici que les conversions entre les représentations japonnaises et européennes des dates et heures sont simples. Il suffit de décaler les positions entre chaque élément qui définit les dates, mais qu'ensuite, il va falloir reprendre les représentations japonaises pour effectuer les comparaisons des dates. Ces conversions et les contraintes définies sur ces objets sémantiques doivent également être établies au niveau des ontologies de domaine.

Cette analyse liée à l'utilisation du contexte pour poser les fondements de notre approche de construction, et qui repose sur une contextualisation des modèles par raffinement possède l'avantage de résolution des hétérogénéités sémantiques liées aux données des différents services en interaction pour obtenir une interprétation correcte de ces données de la part de ces services mis en jeu dans la composition.

Ce faisant, notre démarche de spécification hétérogène combinée au mécanisme de raffinement permet d'être conforme aux véritables besoins de description du contexte de chaque composant tout en permettant la combinaison de ces spécifications à des fins de vérification.

3.4.5 Analyse et discussions

Il faut rappeler que l'association entre ontologie et modèle Event-B est basée sur les dimensions données dans notre état de l'art. Ces dimensions sont, selon l'hypothèse sémiotique, celles du modèle Event-B défini comme référent, du domaine donné par une ontologie définissant l'aspect sémantique descriptif du système (modèle dans notre cas), et enfin l'aspect pragmatique constructiviste donné par les fonctions à définir et les propriétés prescrites dans les comportements pour définir les états corrects en conformité avec les descriptions données par le domaine. Un système entretient donc des liens avec l'ensemble de sa théorie observationnelle (son domaine d'application). C'est pourquoi lorsque une anomalie surgit ne permettant pas de rendre compte de l'expérience, il est alors indispensable d'insérer ce système dans son ensemble, dans sa structure globale qu'est son domaine, puisque ces anomalies n'ont de sens que dans leur domaine. C'est donc à l'holisme ontologique [219] que l'on fait appel pour pouvoir comprendre les phénomènes et ainsi situer où les répercussions doivent être déterminantes.

L'exemple qu'on peut donner ici est celui de la sécurité [41]. Sa théorie est basée sur des constructions mathématiques, des analyses et des preuves. Ses systèmes sont construits conformément aux pratiques d'ingénierie, où le raisonnement inductif et déductif pour examiner la sécurité d'un système à partir d'axiomes clefs pour découvrir ses principes sous-jacents peut être appliqué à des situations non traditionnelles et à de nouvelles théories et de nouveaux mécanismes. Leur validation ne peut se faire qu'en lien avec cette théories définissant les hypothèses du modèle compatibles avec l'environnement auquel la théorie est appliquée.

Les stratégies définies dans le présent chapitre sont basées sur des points importants à considérer lors de la conceptualisation d'un domaine, et par la même occasion la conception d'un système. Le rapprochement entre conceptualisation et conception ou modélisation est basée sur une étude approfondie des deux disciplines, qui puisse améliorer l'exploitation d'un domaine et du raisonnement ontologique pour les techniques de preuve. En particulier, l'exploitation des mécanismes d'identification et de classification pour pouvoir déduire de nouvelles propriétés implicites, pourtant indispensables pour faire des preuves ou valider des modèles en Event-B. Pour ce faire, il est nécessaire de formaliser cette intégration. Notons que c'est pour une intégration partiellement automatisée que nous optons. Cela conduirait à définir un nouveau formalisme. Nous préconisons d'utiliser l'analyse de concepts formelle dans sa variante logique pour les raisons que nous exposerons en conclusion de cette thèse. Nous donnons les grands axes de cette formalisation en perspectives de nos travaux (cf. chapitre 5).

Trois principaux fondements constituent notre approche :

1. En premier lieu, une propriété est implicite lorsqu'elle est extraite inductivement ou déductivement selon le domaine et le contexte ;
2. Ensuite, l'implicite est situé selon le niveau de raffinement défini par une machine et un contexte Event-B et qui correspond au degré de granularité de l'ontologie. Cet aspect permet de juger la petinence du niveau de détails intégrés dans les modèles Event-B, et ce pour les propriétés (invariants et théorèmes) exprimées, mais également pour l'inclusion des traces d'exécution ;
3. Enfin, un autre point est celui d'établir des correspondances entre les deux formalismes étudiés, à savoir, le formalisme Event-B et les logiques de description (variante $\mathcal{SHOIQ}(\mathcal{D})$). Cette correspondance ne traduit pas les formules Event-B vers

les formules de la logique $\mathcal{SHOIQ}(\mathcal{D})$, mais essaie à partir des concepts, à savoir, les constantes, les ensembles et les variables définis au niveau du modèle Event-B, d'extraire des propriétés qui subsument ces trois éléments Event-B. Notons aussi que le choix de la variante des logiques de description a son importance dans la mesure où nous considérons que le problème est un problème d'extraction de connaissances et non une traduction directe. Ces orientations nous obligent à choisir une variante qui soit décidable ;

4. L'extraction sera réalisée par raffinement en définissant des vues logiques comme suit :

   Vue logique = $\{set_n, const_n, var_n\}$, où n est le niveau de raffinement considéré.

Nous montrons donc en perspectives qu'il est possible de définir un modèle Event-B, associé à son domaine comme étant un concept au sens de l'analyse de concepts logique. Un concept modèle se définit par son ensemble d'obligations de preuve étendu avec la composante terminologique (T-Box) d'une ontologie, et de l'ensemble des traces d'exécution de ce même modèle Event-B définissant ainsi la structure qui permettrait par la suite de faire de l'extraction en appliquant des règles définies dynamiquement et guidées par les obligations de preuve au niveau du modèle Event-B.

Le fait de récupérer des concepts, ou des propriétés subsumant les éléments (constantes, ensembles et variables) définis à un niveau de raffinement dans le modèle Event-B a toute son importance. Un constat a été fait durant notre expérience de modélisation avec Event-B : Il est plus facile de reconnaître des propriétés des concepts ou des axiomes que de les écrire ou de les définir par un concepteur. L'avantage que peut avoir cette aproche est qu'elle permet de suggérer ou de proposer à un niveau donné de raffinement des éléments (concepts, propriétés et ou des données) qui peuvent être pertinents pour son analyse sans pour autant l'obliger à les prendre. Si l'on considère l'étude de cas donnée ci-dessus, il peut être judicieux pour un concepteur de lui proposer par exemple au deuxième raffinement 3.4.4.2, un sous-concept *rentertikcets* subsumé par le sous-concept $renter\_Tickets$, au lieu de la relation $car\_rentals$ définie par l'invariant **inv5** pour des raisons qui peuvent être liées aux propriétés exprimées souhaitées à ce niveau. Ensuite, dans les raffinements qui suivent il serait possible d'établir l'invariant de collage avec la variable $car\_rentals$ et la variable *rentertikcets* proposée. Il peut également s'agir d'une liste de propriétés. On peut aussi donner le cas où une relation ternaire définie au niveau du modèle Event-B. Dans ce cas, la suggestion des éléments définissant cette relation ternaire pourrait être importante pour la vérification et la validation des contextes Event-B. Un exemple de ce cas sera traité au chapitre suivant sur les systèmes de vote.

Si ces extractions peuvent être suffisantes pour la vérification des modèles en Event-B, elles le sont moins pour la validation. Dans ce cas, il peut arriver que d'autres propriétés sémantiques d'autres concepts doivent être intégrés.

3.5 Travaux en annexe

Un modèle conceptuel est une vision d'un concepteur de système. Il n'est pas exclu que deux concepteurs utilisent des modèles différents pour concevoir le même système, d'où l'hétérogénéité des conceptions. De ce fait, la conception d'un système doit faire face au différences qui existent dans les représentations du monde réel qu'elle a à traiter. Une approche de conception par abstraction considère que les choses, les concepts, les systèmes

sont indépendants du contexte où ils se situent. Le raffinement est donc un mécanisme inverse à l'abstraction qui contextualise les systèmes étudiés. Les différents cas d'étude que nous avons traités illustrent nos fondements. Les exemples traités dans [9, 14] et [190] viennent s'ajouter pour appuyer notre réflexion. Les travaux dans [9] traitent la conception d'un système avionique, où la partie qui produit les informations telles que l'altitude et la vitesse sur le vol les communique à la partie responsable de leur affichage via un canal unidirectionnel. L'échange de ces données s'effectue en opérant des conversions des valeurs exprimées en pouces, en mètre et en kilomètre via des constantes, des axiomes définis dans des contextes B. Alors que le composant responsable du calcul de ces informations exprime l'altitude en pouce et la vitesse en mètre par heure, leur affichage est effectué en mètre et en kilomètre par heure respectivement. Le système considéré dans [190] est un train d'atterrissage dont la vitesse affichée de la roue est estimée en kilomètre par heure, tandis que le calcul effectué pour déterminer la vitesse sur le sol est estimé en miles par heure. Cet exemple introduit également des constantes de conversion entre les différentes unités employées pour exprimer les exigences.

Nous avons aussi développé des patrons de conception pour les systèmes de péage sur autoroute [164]. Cette étude considère le cas de la monnaie. Dans ce cas d'étude le tarif des tickets est un nombre qui prend ses valeurs dans l'ensemble des entiers. Cependant, cette valeur possède une interprétation pour le concepteur du système et pour les usagers des services offerts sur l'autoroute, elle correspond à la monnaie utilisée pour payer le service d'accès à l'autoroute. Nous avons montré par raffinement que cette sémantique peut être explicitée car il existe une relation qui établit une équivalence entre les différentes valeurs des différentes monnaies qui peuvent être utilisées pour régler la somme à payer. Il s'agit du *"change"* qui permet de convertir la valeur en une monnaie en une valeur définie dans une autre monnaie. Ceci s'entreprend en ajoutant de nouveaux ensembles, constantes ainsi que des axiomes par extension des contextes B ; en modifiant les événements concernés et en introduisant les invariants de collage adéquats pour mieux exprimer la sémantique de ce contexte. Le contexte traité dans les différents cas cités, correspond au *contexte de contraintes*.

La composition par dépendance que nous avons proposée n'a pas de liens avec les travaux sur la composition et la décomposition de modèles Event-B [150, 7, 91, 66, 135, 213, 26, 34]. Ces travaux adoptent une approche de composition et décomposition soit avec partage de variables, soit en partageant des événements dans chaque composant. Les dépendances entre modèles Event-B ne sont pas confrontées à ces conditions. Les modèles Event-B sont définis comme des structures ayant leur propres contextes et machines, et où le modèle dépendant possède les constantes définies comme variables dans le modèle source, dont le contexte dépendant étend le contexte source du premier modèle.

Le contexte de preuve dans les travaux de [103, 104] fait référence à l'environnement dans lequel le système cible est conçu et interragit avec ce dernier. Cet environnement est construit comme un composant ayant un ensemble de comportements finis mais exhaustifs. Le contexte est le périmètre ie : contraintes et conditions, lesquelles caractérisent un ensemble d'interactions du modèle avec son environnement. Un tel composant est un modèle représenté par des automates de contexte. La modélisation du contexte est réalisée dans un langage de description du contexte appelé *CDL* pour *Context Description Language*. Alors que dans ce formalisme l'analyse des propriétés (de sûreté et de vivacité) est réalisée par une composition du modèle à vérifier ainsi que celui comportant les spécifications de l'environnement, dans le formalisme Event-B , cette notion fait partie des modèles développés car les systèmes considérés sont des systèmes fermés où l'environnement est

inclus dans leur modélisation.

Notons enfin que les travaux réalisés dans le cadre du projet IMPEX sur l'utilisation des ontologies de domaine sont orientés vers une traduction directe par dérivation d'une ontologie en contexte Event-B. Ces approches supposent donc l'existence de ces ontologies préalablement conçues. Notre approche favorise l'exploitation du raisonnement ontologique par intégration de manière interactive. Il serait alors possible de manipuler les mécanismes des deux côtés, à savoir le raisonnement ontologique afin d'améliorer les mécanismes de vérification et de validation, mais la réciproque est tout aussi envisageable, où l'activité de vérification pourrait être exploitée pour aider à la conceptualisation d'un domaine. Les interactions peuvent alors être considérées dans les deux sens.

3.6 Synthèse

Le contexte étudié est un contexte lié à la preuve. Notre définition du contexte est une adaptation de celle proposée par l'auteur dans [28, 92] qui définit le contexte dans un cadre plus général et qui, selon notre point de vue, est la définition la plus précise et formelle qui traite l'aspect dynamique manquant aux ontologies.

Le *context lifting* de McCarthy implique des situations ou le temps. De plus, les relations partonomiques entre les contextes de [28, 92], expriment un changement de structures qui correspond à un changement de modèles dans notre cas. Le contexte comme connaissances étant une notion dépendante de l'espace et du temps, elle peut donc être définie comme dépendance entre modèles en Event-B, où la terminaison telle que définie i.e., avec persistance, doit être établie. Celle-ci se fonde sur le principe que les prédicats sont validés par des situations dans le cas du Event-B, il s'agit de contraintes définies dans les contextes B qui doivent être validées par des états qui sont représentés dans les machines. Le principe des dépendances est un principe dual du principe d'invariance dans les machines Event-B, affirmant que les états sont contraints par des invariants afin d'établir les propriétés de sûreté dans un système de preuve.

Le principe de dépendance peut être appliqué à tout système décrit par des phases qui ne se chevauchent pas, pourvu que la stabilité soit une caractéristique de chaque phase impliquée dans le système. Dans ce cas, le système de transition représenté par les machines Event-B qui est relatif à la phase source est construit explicitement. Il est à noter que la notion de dépendances ne cherche pas à générer ou construire un système de transition (machines Event-B) à partir de contraintes statiques du modèle dépendant, mais seulement à valider des axiomes par les états à la stabilité du modèle Event-B source. Ces axiomes sont supposés vérifiés. Ces deux problématiques ont chacune leur importance selon le problème à traiter. Ce point sera discuté en perspectives.

Le mécanisme de dépendances a été appliqué aux patrons de conception pour les systèmes de vote que nous avons développés (voir chapitre 4). L'objectif était de savoir vérifier des systèmes caractérisés par une exécution phasée. Notre développement vient appuyer l'opinion des auteurs dans [95] qui soulignent l'importance de prendre en considération le déroulement phasé des protocoles de vote qui écarte certains comportements corrompus permettant ainsi d'apporter des garanties sur certaines propriétés de sécurité [170].

Au-delà de la définition du mécanisme de dépendances, se pose la question de l'étude de la contextualisation dans le formalisme Event-B. Trois répartitions ont été établies :

le contexte de preuve comme contraintes, hypothèses ou dépendances. Ces connaissances relatives au contexte de preuve donnent une justification aux comportements des systèmes concrétisés par les actions. Les approches théoriques existantes considèrent le contexte explicitement ou implicitement. Ces connaissances se répercutent à la fois sur les deux fronts, à savoir, la vérification et la validation des modèles Event-B. Les *connaissances contextualisées* sont des connaissances qui sont *explicitement* prises en compte dans la vérification pour exprimer et prouver des propriétés (statiques et dynamiques) dans les modèles Event-B. L'utilisation *implicite* relative aux *connaissances contextuelles* importe peu à la vérification, mais est indispensable pour la validation et l'interprétation des modèles Event-B dans l'objectif de les certifier.

Le choix des structures pour représenter l'état du système a aussi un impact sur l'interprétation des modèles construits. Les études de cas présentées dans cette thèse illustrent parfaitement nos propos. Ensembles et quantificateurs logiques permettent également un bon paramétrage des modèles pour la vérification par une démarche de construction d'invariants automatique sur des systèmes couvrant suffisamment de comportements pour concevoir des patrons de conception. L'instanciation des patrons obtenus consiste à préciser des valeurs des ensembles dans les contextes B, ce cas n'engendre pas d'obligations de preuves supplémentaires ; ou bien à introduire d'autres raffinements pour les besoins spécifiques des concepteurs.

La conception des systèmes nécessite des connaissances de plus en plus nombreuses liées à des domaines diversifiés, et leur spécifications sont un vecteur conducteur important. Il est plus facile de reconnaître que de décrire ce que l'on modélise, dans le cas de la vérification et la conception des systèmes au moyen du formalisme Event-B. Il est indispensable de disposer de techniques qui permettent à la fois une construction de modèles Event-B adaptables à de nouveaux problèmes, mais aussi qui peuvent être enrichis et maintenus. Un point important dans la modélisation en Event-B est la capacité à trouver les premiers modèles abstraits sur lesquels le concepteur peut se baser de sorte à exprimer des propriétés en limitant les retours aux niveaux les plus abstraits. Nous montrons (voir chapitre 5) que le problème peut être considéré comme un problème de recherche et d'extraction de connaissances issues du domaine lequel est concrétisé par une ontologie. Cette extraction repose sur les fondements mathématiques de l'analyse de concepts formelle dans sa variante logique appelée ACL [118] (pour Analyse de Concepts Formelle), où ces techniques ont montré leur pertinence.

Par ailleurs, le formalisme de représentation utilisé en Event-B est la logique du premier ordre. C'est la raison pour laquelle nous avons défini les stratégies en suivant la démarche donnée en section 3.4.3 du présent chapitre dans l'objectif de guider les concepteurs dans leur cycle de développement incrémental. Ces stratégies sont orientées domaine et suivent les trois axes donnés et qui concernent 1) la *granularité de l'ontologie* ; 2) le *raisonnement a priori/a posteriori* effectué au niveau des ontologies et au niveau de la modélisation en Event-B ; 3) l'*extraction des connaissances inductive selon le domaine et déductive en fonction du contexte.*

Le second point important à considérer est que les méthodes qui doivent être proposées pour la recherche et l'extraction de connaissances doivent être sensibles au domaine auquel sera lié le système à concevoir, d'où l'intérêt d'une forte contextualisation. Et la contextualisation des développements cependant doit être réalisée par raffinement. Le raffinement peut être vu comme une approche de vérification *a posteriori*, où les connaissances implicites peuvent être explicitées par intégration dans les modèles abstraits permettant ainsi

de lever certains conflits et ambiguïtés. La relation de abstraction/raffinement entre modèles Event-B correspond donc à une relation de généralisation/spécialisation selon les assertions définies par les obligations de preuves et les traces d'exécution données (voir chapitre 5). Juger la pertinence d'une modélisation selon la granularité de l'ontologie est lié aux détails introduits par raffinement, dans l'intérêt de préserver l'inclusion de traces d'exécution.

Chapitre 4

Patrons de conception par dépendance-Application aux systèmes de vote

Sommaire

4.1 Introduction 122
4.2 Conception des systèmes de vote 124
4.2.1 Caractéristiques des systèmes de vote 125
4.2.2 Exigences et propriétés d'un système de vote électronique . . . 127
4.2.3 Analyse des systèmes et des protocoles de vote électronique . . 130
4.3 Patrons de développement pour les systèmes de vote 131
4.3.1 Phase du vote 134
4.4 Relation de dépendance entre les phases du vote 159
4.4.1 Contexte dépendant pour la phase de dépouillement 160
4.4.2 Phase de dépouillement 166
4.5 Extensions possibles des patrons des protocoles de vote . . . 173
4.5.1 Composition des modèles des protocoles de vote et le modèle de l'attaquant Dolev-Yao 177
4.6 Exploitation des connaissances du domaine pour concevoir des systèmes de vote 179
4.6.1 Extraction pour la vérification 179
4.6.2 Extraction pour la validation 182
4.7 Conclusion 183

« J'ai appris que la voie du progrès n'était ni rapide ni facile. »
- Marie Curie

4.1 Introduction

Un système électroral [265], mode de scrutin ou régime électoral, désigne tout type de processus qui permet de désigner l'expression du choix d'un corps électoral donné, souvent la désignation d'élus pour exercer un mandat en tant que représentants de ce corps (élection), ou le choix direct (référendum) d'une option parmi plusieurs. Les systèmes de vote n'échappent pas au processus d'automatisation et la volonté de traitement plus rapide qui devient de plus en plus d'actualité. On parle alors de vote électronique (appelé communément e-voting en anglais). Le vote électronique se définit selon la recommendation du comité des ministres aux états membres sur les normes juridiques, opérationnelles, et techniques relatives au vote électronique, en septembre 2004, comme une élection ou un référendum électronique qui implique le recours à des moyens électroniques au moins lors de l'enregistrement du suffrage [251, 252]. Cette définition inclut tout moyen pouvant être utilisé afin de donner la possibilité aux électeurs de faire leur choix et de l'enregistrer. Elle se réfère au vote par Internet, à l'utilisation de machines de vote électronique qu'on appelle kiosques de vote. De plus, le vote par Internet se fait soit par les systèmes de vote à distance, où les électeurs peuvent émettre leurs suffrages en dehors des bureaux de vote, et on peut citer dans ce cas l'exemple des systèmes utilisés en Estonie ; soit par en donnant la possibilité aux électeurs d'émettre leur suffrage en se servant d'un système de vote par Internet ne pouvant être utilisé qu'à partir de certains points d'accès. Ces derniers sont installés dans les bureaux de vote. À titre indicatif, ces systèmes sont utilisés en Belgique, aux Etats-Unis, aux Pays-Bas.

La diversité des sources d'informations collectées relatives à la configuration des systèmes de vote, ainsi que toutes sources et instructions liées à l'utilisation de ces systèmes, donnent naissance à diverses exigences de sécurité basées sur des hypothèses liées à l'environnement et aux procédures juridiques très spécifiques selon le pays et le type de vote. De telles caractéristiques rendent ces systèmes très complexes. Par conséquent, les systèmes de vote électronique nécessitent l'utilisation de méthodes formelles pour bien comprendre les aspects liés à la sécurité.

Par ailleurs, l'intégration de connaissances et le support automatique assisté par ordinateur sont les principaux défis à relever dans la communauté des chercheurs en génie logiciel [70, Preface]. Le premier défi implique une importante perte de temps, et cela est dû à l'une des difficultés dans les relations humaines, à savoir, le manque de *connaissances explicites* partagées entre les membres du groupe/projet, avec d'autres groupes et avec d'autres parties prenantes. Le deuxième défi vient du fait que de nombreux projets comprennent la conception/construction d'outils avancés pour supporter différentes activités d'ingénierie logicielle. La simplification et l'établissement de normes des systèmes de vote rend plus simple leur spécification et par la même occasion, leur compréhension. L'objectif est la proposition de modèles de spécifications génériques adaptables et extensibles pour le besoin spécifique de chaque utilisateur et utilisation finaux.

Les problèmes de conception concernent l'identification de modèles et de mécanismes

appropriés pour faciliter le processus de vote. Le déploiement d'un système de vote électronique est inévitablement associé à un système distribué, qui introduit des problèmes spécifiques de sécurité et de confidentialité. Cela nécessite la sélection d'un protocole de vote spécifique, la conception et la mise en œuvre et l'intégration des mécanismes pertinents [15].

La première étape de l'analyse de sécurité d'un système consiste à définir un modèle de menace à l'aide de ces concepts clés. Le modèle de menace d'un système décrit les objectifs qu'un attaquant peut avoir, les types d'attaquants susceptibles d'attaquer le système et les capacités disponibles pour chaque type d'attaquant [260, 153]. Le modèle de menace d'un système décrit les objectifs que pourrait avoir un attaquant (par exemple, pour manipuler le nombre de votes), les types d'agresseurs qui pourraient tenter d'attaquer le système (par exemple, les électeurs, les agents de vote, etc.) et les capacités disponibles pour chaque type d'attaquant. Il est tout aussi important de décrire les menaces hors de portée de l'analyse, car les risques ne dépendent pas seulement des niveaux de sécurité des systèmes, elles sont aussi le résultat de contournement des procédures juridiques qui régissent ces systèmes de vote. Une telle démarche caractérise le contexte de chaque produit final, d'où l'intérêt d'étudier cette notion dans les systèmes de vote.

Ce chapitre présente une première tentative de formalisation de patrons de conception pour les systèmes de vote en utilisant l'approche de développement correct-par-construction qu'est le raffinement au moyen du formalisme Event-B. Cette construction de modèles Event-B a été élaborée à partir de collecte d'informations et des lectures de documents officieux d'auteurs experts du domaine. Les patrons de conception ont donné lieu à la publication [128]. Le développement est réalisé à base de modèles Event-B structurés par raffinement, où chaque étape de raffinement introduit soit de nouveaux événements, soit de nouvelles propriétés.

L'approche adoptée est une approche ensembliste pour décrire les concepts ainsi que leurs relations dans ces systèmes. Nous définissons un système de vote par phases, où chaque phase est décrite par des contextes et des machines Event-B séparés. Chaque phase est caratérisée par la propriété de vivacité qu'est la stabilité, où plus ancun événement lié à cette phase ne peut plus être observable, une fois une certaine propriété est atteinte suivant la phase en question.

De nombreuses propriétés et exigences sont exprimées dans la littérature sur les systèmes de vote électronique. Les auteurs dans [84, 85], indiquent que dans le cas idéal, un système de vote devrait garantir l'*anonymat* et la *vérifiabilité* qui sont le maillon fort dans la sécurité de tout système de vote :

— la vérifiabilité garantit que tous les électeurs peuvent faire confiance au résultat proclamé sans avoir à s'appuyer sur une autorité particulière. En outre, cette propriété garantit l'existence d'un algorithme pouvant exhiber la preuve du résultat du décompte et de l'intégrité et l'honnêteté des autorités quant à leur comportement (de dépouillement). Dans notre cas, ce sont les obligations de preuve générées qui montrent que le système s'est bien comporté. Notre modélisation combine ces deux propriétés de vérifiabilité car la modélisation en utilisant le formalisme Event-B raisonne sur des systèmes fermés, où les modèles incluent le système et l'environnement ;
— l'anonymat d'un vote n'est pas défini par le secret du vote papier (le contenu des bulletins papier est connu) de chaque électeur. Cette propriété exprime que le lien entre l'électeur et son vote n'est connu que de lui, ou en d'autres termes, que la

connaissance de chaque électeur est limitée seulement à son choix. En général, cette propriété est basée sur la destruction du lien entre l'électeur et son vote.

L'*éligibilité* des électeurs est aussi une propriété importante à exprimer, car elle détermine si un électeur a le droit de voter ou non. Notre modélisation exprime cette propriété comme une condition concernant l'*authentification* et l'*autorisation* des électeurs pour effectuer le vote qui conditionne l'enregistrement de chaque vote. L'authentification implique l'identification des électeurs au moyen de *credentials* et de *mots de passe* préalablement fournis à cette fin. Notre modélisation exprime également la propriété de *Pas de vote double* ou d'unicité du vote, et la propriété de *Pas de vote partiel*.

Les systèmes de vote dont les caractéristiques de leurs spécifications ne peuvent être établies séparément, mais doivent être connues dans l'ensemble dans sa totalité, et non en étudiant chaque partie séparément. Cette caractéristique est relative à l'holisme ontologique de Quine [219] défini comme pour un être est entièrement ou fortement déterminé par le tout dont il fait partie. Les conditions *nécessaires* et *suffisantes* doivent être indiquées pour toutes les propriétés de l'élément étudié. C'est-à-dire, qu'un système complexe est considéré comme une entité possédant des caractéristiques liées à sa totalité, et des propriétés non déductibles de celles de ses éléments. Notre état de l'art a montré l'intérêt d'utiliser les ontologies comme association un couplage avec le système étudié ou modélisé. Le développement de modèles en collaboration avec les développeurs de logiciels et des ingénieurs permet aux législateurs, à la fois, de comprendre l'impact des lois et des changements des processus qui résultent des changements de technologies et des processus, et de maintenir la traçabilité des produits logiciels. Nous montrons, en utilisant les stratégies exposées au chapitre 3, comment il est possible d'obtenir la traçabilité entre les modèles et les connaissances (les lois ainsi que les procédures) liées au domaine d'expertise en question. Une telle démarche permet d'obtenir des modèles ayant une sémantique non ambiguë, standard, en utilisant des ontologies de domaine, car la précision est une propriété importante dans ces systèmes.

La section 4.2 donne un aperçu sur les systèmes de vote ainsi qu'à leur conception, leur caractéristiques et résume les propriétés et les exigences les plus importantes dans le processus de construction des systèmes de vote électronique. La section 4.3 détaille le développement des patrons de conception pour les systèmes de vote. La section 4.4 détaille l'utilisation du mécanisme de dépendances entre les phases dans le développement de ces patrons. L'objectif est d'obtenir la spécification d'un réel système de vote. La section 4.5 montre comment, à partir de nos spécifications, il serait possible d'étendre le développement afin d'intégrer de nouvelles exigences. La section 4.6 présente l'application des stratégies d'intégration des connaissances liées au domaine pour mettre à profit les connaissances sémantiques définies dont le but est d'identifier précisément et de manière univoque les propriétés *définissantes* liées aux comportements du système en question.

4.2 Conception des systèmes de vote

Dans le domaine du vote, nous pouvons lier les principaux facteurs qui rendent complexe la compréhension des phénomènes à : *i)* la complexité liée aux données, *ii)* la complexité conceptuelle, *iii)* la complexité ontologique et sémantique *iv)* la complexité technique.

4.2.1 Caractéristiques des systèmes de vote

L'intelligence humaine a établi de nombreux modes pour recueillir les préférences des électeurs selon le vote et le pays concernés, Parmi ces modes on trouve :

— **Le scrutin majoritaire** :
Ce mode de scrutin s'insère dans la catégorie des systèmes majoritaires à un ou deux tours. Chaque électeur ayant le droit pour voter glisse dans l'urne un seul bulletin. Un bulletin représente une voix pour un et un seul candidat. Le candidat ayant recueilli le plus de voix devient vainqueur du tour ou de l'élection en question selon que le scrutin est à un ou deux tours. Ce type de vote est réputé pour sa facilité à dépouiller les suffrages.

— **Le vote « cumulatif » ou « à points »** :
On met à disposition de chaque électeur un nombre de points donnés qu'il répartira librement sur chaque candidat présent dans la liste électorale. Le candidat vainqueur est le candidat ayant récolté le maximum de points. Ce mode de vote se place dans la catégorie des systèmes semi-proportionnels.

— **Le vote « par classement » ou « alternatif »** :
Une classification de chaque candidat est établie par les électeurs par ordre de préférence. Ainsi, les candidats ayant obtenu la majorité absolue est élu. Le processus est réitéré en éliminant des bulletins le candidat ayant recueilli le moins de voix. Cette méthode modifie le rang des candidats placés après le candidat éliminé.

Il existe d'autre modes de scrutins comme par exemple, le vote par évaluation, où des mentions (Excellent, Très bien, Bien, Passable, Insuffisant, À rejeter, notes de 1 à une échelle fixée, à titre indicatif) sont attribuées à chaque candidat. Le candidat ayant récolté la médianne la plus élevée est le candidat vainqueur.

De manière générale, tout système de vote met en exergue les notions suivantes [23] :

— La liste des électeurs éligibles pour voter préalablement définie ;
— La liste des candidats ou des représentations sur lesquels vont porter les choix des électeurs ;
— Les bulletins qui sont représentés selon le système adopté (un bulletin compte une voix, un bulletin contient tous les candidats avec une mention qui dépend du mode de scurtin, ...) ;
— Le nombre de tours à considérer pour chaque élection ;
— Le dépouillement qui dépend de la règle instaurée qui détermine le ou les vainqueurs des élections, permettant d'établir ainsi une décision, laquelle concrétise l'intension du système.

Dans le cas du vote électronique, on parle aussi de certificats électroniques attribués aux électeurs avant de voter, de clefs de chiffrement et de déchiffrement, de tableaux d'affichage (bulletin board en anglais), de serveurs, de blocks ou de bureaux virtuels ...

La mise en oeuvre du processus de vote passe principalement par trois phases :

— **La phase de préparation du vote** : à l'issu de cette phase, sont établies les listes des candidats nominés sur lesquels portera le choix des électeurs, ainsi que les électeurs enregistrés ayant droit de vote. L'établissement de ces listes dépend des lois locales de chaque pays.
— **La phase d'enregistement du vote** : cette phase permet aux électeurs tous éligibles d'exprimer leur choix sur la base de la liste des représentants nominés lors

23. Election Markup Language, `http://xml.coverpages.org/eml.html`.

de la phase précédente. Ainsi, par l'utilisation de la liste électorale l'électeur doit s'authentifier comme un électeur admissible et déposer son vote individuellement.

— **La phase de dépouillement du vote** : cette phase couvre le comptage et le résultat des rapports issus de la phase d'enregistrement.

Concevoir des systèmes de vote électronique suit des contraintes, des points de vue et des modalités différentes selon le pays, les citoyens et les lois ainsi que les réglementations du pays ainsi que le vote en question comme indiqué en introduction du présent chapitre. En outre, cette conception a recours aux technologies de l'information et de la communication (Information and Communication Technologies (ICT) en anglais). Les systèmes de vote sont divisés en deux types : à savoir, le vote *en-ligne*, et le vote *hors-ligne*. Le premier type permet aux électeurs de voter depuis chez eux en se connectant à Internet. Alors que le second type de vote donne la possibilité aux électeurs de voter dans des bureaux de vote ayant été équipés par des « machines à voter ». Concevoir un système de vote électronique se structure en couches qui contiennent :

— le noyau qui inclut les différentes contraintes liées aux lois ainsi que les différents logiciels et matériels utilisés à cette fin et les différents acteurs participants aux vote en question (serveurs, audits, tableaux d'affichage, etc.) ;

— mais aussi les différentes primitives cryptographiques utilisées pour protéger (chiffrer et sceller) les données des utilisateurs. Chaque schéma de sécurité correspond à un ensemble de contraintes de sécurité qui décrivent les exigences pour un objectif de sécurité précis (authentification, confidentialité, intégrité, ...). Celles-ci reposent sur les mécanismes de sécurité (chiffrement, signatures, ...) qui sont appliqués à un ensemble de types de données comme par exemple, aux bulletins choisis par les votants, ou aux identités des électeurs. Par exemple, des crédentials peuvent être nécessaires pour garantir l'authentification. Ces contraintes permettent donc de spécifier sur quel type de données le schéma de sécurité devrait être appliqué. L'utilisation de mécanismes cryptographiques est contrainte est obligatoire [80], recommandée également par la CNIL [24] ;

— ainsi que les interfaces homme-machine qui préconisent selon le matériel utilisé les différents moyens pour le vote, téléphones, ordinateurs, cartes perforées, écrans tactiles, disposition des touches, les couleurs, les formats utilisés [80] ... Mais elle inclut également les différents dispositifs utilisés pour les électeurs ayant un handicap particulier (écouteurs, des guides sonores ...). Ces interfaces sont aussi dépendantes des deux premières couches citées.

Parmi les dispositifs qui peuvent être utilisés dans le processus de vote électronique on peut citer : le vote par Internet, le vote dans des bureaux de vote en utilisant des machines ou des kiosques de vote installés dans ces bureaux de vote, le vote par téléphone. Le matériel utilisé varie d'un pays à un autre. On peut trouver des cartes perforées (Punchcard en anglais), des Marksenses et des DRE (Direct-Recording Electronic en anglais) ... Ces dispositifs sont mis en place dans des architectures qui peuvent inclure des serveurs web, des systèmes de gestion de bases de données, des serveurs SMS (Short Message Service en anglais) qui utilisent des postes téléphoniques mobiles des services de messagerie SMS pour accéder au système de vote électronique. Dans ce dernier cas, le vote est effectué par téléphone et les serveurs SMS communiquent au niveau le plus bas avec un certain nombre de modems GSM (pour Global System for Mobile Communication en anglais) qui reçoivent les messages SMS des électeurs via un fournisseur de services. Cette étape est suivie par la saisie du message par le serveur SMS, puis la vérification du contenu du message. Le serveur envoie ensuite une réponse à l'électeur soit pour l'enregistrement du

24. Commission Nationale de l'Informatique et des Libertés

vote dans la base de données soit en indiquant qu'il y a erreur [217].

Dans tout scénario de vote, la première étape à suivre est de s'authentifier auprès des autorités, puis de faire son choix de vote et de l'enregister. Ces étapes sont suivies par une déconnexion du service ou de l'interface qui permet d'accéder à l'espace utilisateur. Selon le système ou le protocole de vote, des autorisations de consultation de modification peuvent être réalisées. L'opération de dépouillement peut être effectuée soit par blocks au fur et à mesure que le processus de vote progresse, soit à la fin de la phase de vote selon l'approche adoptées.

D'autres caractéristiques qui doivent être prises en considération dans le développement d'un système de vote électronique sont les modalités dans les comportements des électeurs car les exigences peuvent être : obligatoires (nécessaires), ou bien interdites, encouragées (facultatives), ou optionnelles (autorisées) [80].

4.2.2 Exigences et propriétés d'un système de vote électronique

Il faut dire qu'en plus de toute cette diversité des moyens matériels et logiciels qui doivent être mis en place pour permettre aux électeurs durant le processus de vote de faire leur choix et de l'enregistrer, puis de dépouiller les résultats des suffrages, un système de vote doit garantir de nombreuses propriétés et exigences que nous pouvons classer : en exigences liées à la vérification ; en propriétés liées à la validation, le test ainsi que la simulation ; en propriétés liées à la justice et l'équité dans le processus de vote. De plus, il est aussi possible de classer ces exigences par phases de vote, par exemple les exigences liées à la phase de préparation d'un vote, les propriétés liées à la phase d'enregistrement de vote, et les propriétés liées à la phase de dépouillement de vote.

Ces exigences de vote électronique sont souvent traduites en propriétés explicites et détaillées qu'un système de vote électronique viable devrait posséder [248]. Nous ne pouvons citer toutes les exigences et tous les mécanismes déployés pour concevoir un système de vote dit sûr et certifiable. Mais nous pouvons résumer ces exigences comme suit : Selon [81, 86, 87, 204] un bon système de vote se caractérise par les propriétés que sont : la précision, la démocratie, l'anonymat, la vérifiabilité, la commodité, la flexibilité, la mobilité. Parmi ces caractéristiques, les plus importantes exigences à satisfaire sont :

- **L'exactitude** : celle-ci se traduit par l'impossibilité de :
  - modifier un vote ;
  - éliminer du décompte final un vote validé
  - compter un vote invalide
- **La démocratie** : cette propriété se traduit par :
  - seuls les électeurs éligibles ont le droit de voter ;
  - un électeur éligible ne peut voter plus qu'une fois lors d'une même élection
- **L'anonymat** : cette propriété s'exprime comme suit
  - nul n'est dans la capacité de connaitre le lien qui peut exister entre un bulletin de vote et le votant qui l'a exprimé ;
  - aucun votant ne peut prouver avoir voté d'une manière particulière en vue de se laisser corrompre
- **La vérifiabilité individuelle et universelle** : cette propriété est exrpimée comme suit :
  - chaque votant peut vérifier que son propre vote a bien été dépouillé ;

— tous les votants peuvent vérifier quant à la correction des résultats des votes comptabilisés

Il existe d'autres propriétés de sûreté qui doivent être vérifiées. Selon le système considéré, on peut trouver :

— **Equité (appelée aussi "Pas de résultat Partiel")** :
Cette propriété (*fairness property* en anglais) établit que le protocole de vote ne doit divulguer aucun résultat partiellement en vue d'influencer les électeurs qui n'ont pas encore exprimé leur vote.

— **Sans Reçu** :
Introduite par Benaloh et Tuinstra dans [37], cette propriété (*receipt-freeness* en anglais) exprime que le protocole ne doit pas être en capacité de prouver d'une quelconque manière comment le votant a exprimé sont vote. Elle garantit la non construction de reçus empêchant ainsi l'achat d'un vote ou la coercition (stipule que le votant a voté sous pression) [97, 161].

Cela signifie que tous les électeurs devraient être traités de manière égale, autrement dit avoir les mêmes opportunités d'émettre leur vote, qu'ils devraient être libres de voter, sans influence (ou coercion en anglais) indues de quelque sorte que ce soit, que le vote devrait rester secret, c'est-à-dire que les électeurs devraient avoir la possibilité d'émettre leur suffrage en secret et, qu'une fois le vote émis, le lien entre le votant et le vote devrait disparaître, qu'une seule personne ne devrait pouvoir émettre qu'un seul vote, et enfin que l'urne devrait représenter de manière fiable et sécurisée le choix du votant.

En plus des propriétés inhérentes au processus démocratique, il existe des propriétés systémiques nécessaires à la viabilité du vote électronique :

— **Solidité ou robustesse** : le système devrait veiller à ce que le processus électoral ne soit pas affecté par un comportement illégal ou des procédures défectueuses.

— **Mobilité** : les électeurs devraient pouvoir voter à partir de n'importe où sans contraintes géographiques. Le système devrait également être disponible et accessible pendant la phase d'interrogation.

— **Intégrité** : l'intégrité du système garantit qu'un système informatique doit être inviolable et l'intégrité des données garantit que les données n'ont pas été modifiées pendant le transport. Intégrité fait également référence à la résistance à la collusion lorsque les organismes électoraux sont incapables de conspirer pour introduire, modifier ou retirer des votes lors d'une élection.

— **Commodité** : un système devrait permettre aux utilisateurs de voter facilement, rapidement et avec un minimum d'instructions. La commodité se traduit par des exigences d'utilisabilité et de fiabilité. Le défi consiste toutefois à concilier commodité et sécurité.

Le processus de vote doit pouvoir être vérifiable et ouvert aux audits indépendants. De ce fait, la transparence, la justice, l'accessibilité ainsi que l'équité dans toute sorte d'élections et tout processus de vote doivent permettre aux personnes ayant un handicap d'avoir les mêmes droits qu'un électeur ordinaire, mais aussi des moyens supplémentaires qui leur facilitent cette tâche qu'est de choisir et de voter [80]. Par exemple, les électeurs qui sont malentendants mais qui doivent utiliser une interface audio peuvent également avoir besoin d'augmenter le volume de l'audio. De telles situations nécessitent des écouteurs avec une faible fuite sonore. De telles moyens leur permettent d'émettre leur vote *en totale indépendance* et de manière non-discriminante. Ceci revient à établir des normes d'utilisation informatiques mises au point par des groupes tels que World Wide Web consortium, ISO, ETSI . . . Et des machines de vote devront être achetées par adjudication ouverte [251, 252]. Aussi, des dispositions doivent être prévues pour les personnes ne

pouvant voter les jours prévus des scrutins, à titre indicatif : prévoir des procédures des votes anticipés, les votes par procuration ...

D'un point de vue conceptuel, les exigences pour les modèles du processus de vote sont dépendantes de plusieurs contraintes [13, 260] :

- *complètes* : Un modèle de processus complet doit contenir les sujets, objets, activités et contraintes pertinents des processus qui constituent l'architecture métier. C'est-à-dire que tous les éléments significatifs des processus d'administration doivent être mappés aux éléments du modèle ;
- *simples* : Tous les utilisateurs et, en particulier, les utilisateurs ayant peu ou pas de connaissances techniques doivent être capables de lire et de comprendre le modèle ;
- *formelles* : Tous les éléments du modèle devraient recevoir une signification précise et sans ambiguïté afin de permettre des simulations et/ou une vérification formelle, le cas échéant ;
- *hiérarchiques* : Les processus généraux (et informels) sont raffinés par des sous-processus ordonnés dans des niveaux d'abstraction ultérieurs. L'organisation hiérarchique est utile à des fins de présentation et pour que le modèle soit plus compréhensible.
- *maintenables* : L'organisation du modèle et de ses éléments doit respecter des conventions précises et vérifiables par la machine, afin de garantir un niveau minimum de qualité et d'uniformité.
- *traçables* : L'organisation du modèle et de ses éléments devrait permettre la synchronisation entre la représentation du modèle et les informations juridiques encodées dans le modèle, afin de garantir la gestion du changement (variabilité) entre les modèles et ces informations.

Par ailleurs, un système de vote, comme tout système critique, doit être certifié. Cette certification peut prendre différentes formes qui peuvent être uniformisées pour un système de vote électronique unique. Ce processus peut passer par des procédures de test très spécifiques, et le produit fourni doit être documenté, étant donné que les fournisseurs sont tenus de produire des documents pour prendre en charge les tests indépendants requis pour que les produits soient certifiés par les autorités compétentes [80]. Ces documents doivent répondre aux besoins des laboratoires de tests, des fonctionnaires électoraux et des techniciens de maintenance. Cette procédure suit le courant de tout produit qui doit être commercialisé et donc être conforme aux technologies de l'information et d'équipement électronique et mécanique, en commençant par l'étape de présentation du système, puis en terminant avec les étapes de déploiement, de plan de configuration, de programme d'assurance qualité, et en passant par les étapes de spécification des conceptions du logiciel, de sécurité et de vérification du système [80]. La normalisation des spécifications pourrait aider à simplifier, étendre et généraliser les résultats trouvés dans d'autres systèmes. D'où l'intérêt d'utiliser des ontologies.

Le recours aux ontologies a de nombreux avantages pour la conception d'un système de vote [167], et d'un système complexe de manière générale telle que l'explique notre état de l'art, très peu de travaux présentent de réelles ontologies qui sont complètes et prêtes à être exploitées. Citons à titre indicatif, les travaux réalisés par les auteurs dans [239] pour le e-gouvernement qui sont basés sur les annotations sémantiques de services pour la découverte de services. La combinaison des domaines de e-gouvernement et le web sémantique s'avère intéressante puisque : le domaine de l'administration électronique est unique en raison de son énorme défi en matière d'interopérabililité, compte tenu des multiples différences sémantiques d'interprétation, par exemple, des lois, des réglementations,

des services aux citoyens, des processus administratifs, des meilleures pratiques. Enfin, les différentes langues à prendre en compte dans et entre les régions, les nations, et les continents.

4.2.3 Analyse des systèmes et des protocoles de vote électronique

Les approches d'analyse des systèmes de vote sont classées selon le niveau d'abstraction et de cible d'évaluation en vérification des protocoles cryptographiques et du comportement du système [260]. Nous pouvons classer les travaux pour l'analyse des systèmes de vote électronique en travaux qui emploient des approches semi-formelles comme c'est le cas du projet ProVotE, et les approches formelles parmi lesquelles peut citer : les travaux qui se basent sur l'algèbre de processus, et les travaux basés sur les logiques modales ou épistémiques. Dans ce qui suit, nous décrivons brièvement les travaux les plus significatifs.

Les travaux dans le cadre du ProVotE [245, 257] sponsorisé par la province autonome de Trente en Italie dans le but d'évaluer le passage aux systèmes électroniques pour les élections locales. Un DRE-VVPAT dans ce projet a été développé. Ce choix a été motivé par la possibilité d'intégrer les réactions des électeurs, de guider la conception en fonction d'autres expériences et recommandations et de choisir des technologies et des composants [25]. La modélisation dans ce projet est effectuée en utilisant les diagrammes UML. Un composant critique de la machine de vote électronique est généré automatiquement à partir de diagrammes d'état (statecharts) UML. L'étape d'analyse consiste, non seulement à analyser les risques qui émanent des différentes sources liées à la technologie et au matériel employés, mais aussi à analyser les contraintes liées aux lois régissant les élections électroniques. L'étape de modélisation des exigences est aussi réalisée avec des diagrammes UML d'état et de cas d'utilisation. Les diagrammes d'état spécifient le cycle de vie de la machine de vote décrite et les cas d'utilisation fournissent les scénarios d'utilisation pour chacun du cycle de vie de la machine (par exemple, les actions nécessaires pour ouvrir le site de l'élection en question). Les interfaces utilisateurs de vote et d'administration sont spécifiées à travers un graphique décrivant leur logique et dans lequel chaque état correspond à une capture écran différente du système.

Les spécifications dans ce projet sont traduites par l'outil FSMC+ [246] en langage d'entrée NuSMV [76]. NuSMV est un vérificateur de modèles spécifiés pour des systèmes synchrones et asynchrones. Et la vérification des propriétés de sécurité et de vivacité exprimées en logique arborescente CTL et en logique temporelle linéaire LTL, en utilisant les techniques basées sur les BDD et sur SAT. Le développement dans ce projet inclut également les étapes de génération de code et de codege des tests unitaires, de tests de packaging et des essais.

Les travaux basés sur les algèbres de processus [27, 32, 97, 170] définissent les rôles de chaque entité qui compose le protocole par une grammaire qui caractérise un processus. Le protocole est spécifié comme la composition parallèle de processus en utilisant l'algèbre de processus (π-calculus) [1] qui décrivent les rôles des entités et leur interaction via l'échange de messages. Quant aux propriétés, elles sont exprimées en termes de relations d'équivalence observationnelle entre processus. De manière générale, ces travaux présentent un

25. Les machines ProVotE ont été utilisées avec des valeurs expérimentales dans diverses essais lors d'élections locales de deux petites élections à valeur juridique. Ces élections concernent l'élection des représentants des étudiants d'un lycée, en Italie, règlementé par la loi ; et l'élection qui concerne un sondage local pour réunir deux municipalités de la région Frioul-Vénétie Julienne.

cadre pour la spécification et la vérification des propriétés de type vie privée telles que le secret des votes, la propriété de sans reçu et la propriété de résistance à la coercition. À titre d'exemple, les auteurs dans leurs travaux [97, 170] ont analysé le protocole de vote FOO [125]. Les propriétés d'équité et d'éligibilité sont prouvées à l'aide de l'outil ProVerif [47, 48], tandis que la propriété de secret du vote est vérifiée par équivalence observationnelle : étant donnés le votant V_1 ayant un vote v_1, et un votant V_2 ayant un vote v_2 dans un premier système, et dans un second système le votant V_1 ayant le vote v_2 et le votant V_2 ayant le vote v_2. Si ces systèmes ne peuvent être distingués par un observateur, les électeurs et leurs votes sont indissociables. Ces approches sont basées sur le principe de permutation des CSP (Communicating Sequential Processes en anglais) [230]. Plus tard cette propriété a été généralisée aux protocoles de vote électronique [96].

Les approches basées sur les logiques modales expriment les propriétés au moyen d'une logique qui dispose de constructeurs pour s'interroger sur la possibilité que certains événements se produisent ainsi que sur leur ordre d'exécution en logique temporelle, sur la croyance des agents décrite en logiques doxastiques et sur leur connaissances exprimées en logiques épistémiques vis-à-vis de certaines propriétés. Les opérateurs de modalité utilisés sont des opérateurs qui expriment la possibilité et la nécessité. Parmi les travaux qui utilisent les logiques épistémiques on trouve [32, 160, 253]. Ces travaux utilisent la logique DEL (Dynamic Epistemic Logic en anglais). Les formules de cette logique décrivent l'évolution du protocole à analyser à chaque évolution des connaissances des agents du protocole. Ces approches sont généralement bien adaptées aux spécifications car elles sont expressives, mais elles le sont moins pour décrire le comportement du protocole [242].

4.3 Patrons de développement pour les systèmes de vote

Le processus de vote est modélisé par trois phases successives, et chacune est développée par une chaîne de raffinement de machines séparée. Nous vérifions formellement la correction de chaque phase individuelle dans ce processus dans sa globalité, ainsi que certaines propriétés de sûreté cruciales issues de la composition desdites phases par dépendance. Nous montrons que cette approche permet d'effectuer des preuves sur chaque composant indépendamment de l'autre. La phase de *preparation* du vote étant différente selon le système et le vote cible, il est difficile de proposer un patron commun pour cette partie vu les divergences des conceptions existantes. Notre développement suppose en particulier que cette phase est acquise. Ainsi, deux composants ont été développés, un premier composant décrit la phase d'enregistrement du vote, et un second correspondant à la phase du dépouillement du vote. Notre développement concerne le vote électronique sans la dimension cryptographique. Nous montrons ensuite comment, à partir de notre développement, il serait possible d'obtenir d'autres types de systèmes.

Hypothèses sur l'environnement : La description du système renferme également les conditions dans l'environnement qui expriment les comportements des électeurs et des attaques. En s'appuyant sur l'observation de l'évolution de l'état du système en fonction des actions entreprises par les différents acteurs dans le système, de leurs interactions et des changements de l'environnement, cette modélisation est une base pour prouver des propriétés de sûreté. En particulier, notre développement fait abstraction de tout rôle

et/ou acteur dans le système, en raison de l'approche de modélisation par l'abstraction mécanisée par le raffinement sur laquelle est basé le formalisme Event-B. Le seul acteur représenté dans nos modèles est l'électeur. Ce dernier est l'acteur principal avec lequel le système interagit. Pour qu'une entité corrompue puisse agir dans le système, celle-ci devrait se faire passer pour un électeur donné. En particulier, notre développement tient compte d'un intrus qui a les capacités suivantes :

1) établir une connexion ; *2) fermer une session déjà établie* ; *3) faire des choix* ; *4) ajouter des signatures* ; *5) ajouter des ballots : bourrage des urnes* ; *6) ajouter des signatures et des ballots simultanément* ; *7) supprimer des signatures* ; *8) supprimer des bulletins de vote* ; *10) obtenir des crédentials* ;

Ces hypothèses contextualisent notre développement comme nous l'avons expliqué dans les chapitres précédents (cf. chapitre 3). Nous qualifions ces hypothèses de "***contexte d'hypothèses***".

Description générale des modèles : La figure 4.1 illustre le schéma global de l'approche suivie dans notre développement.

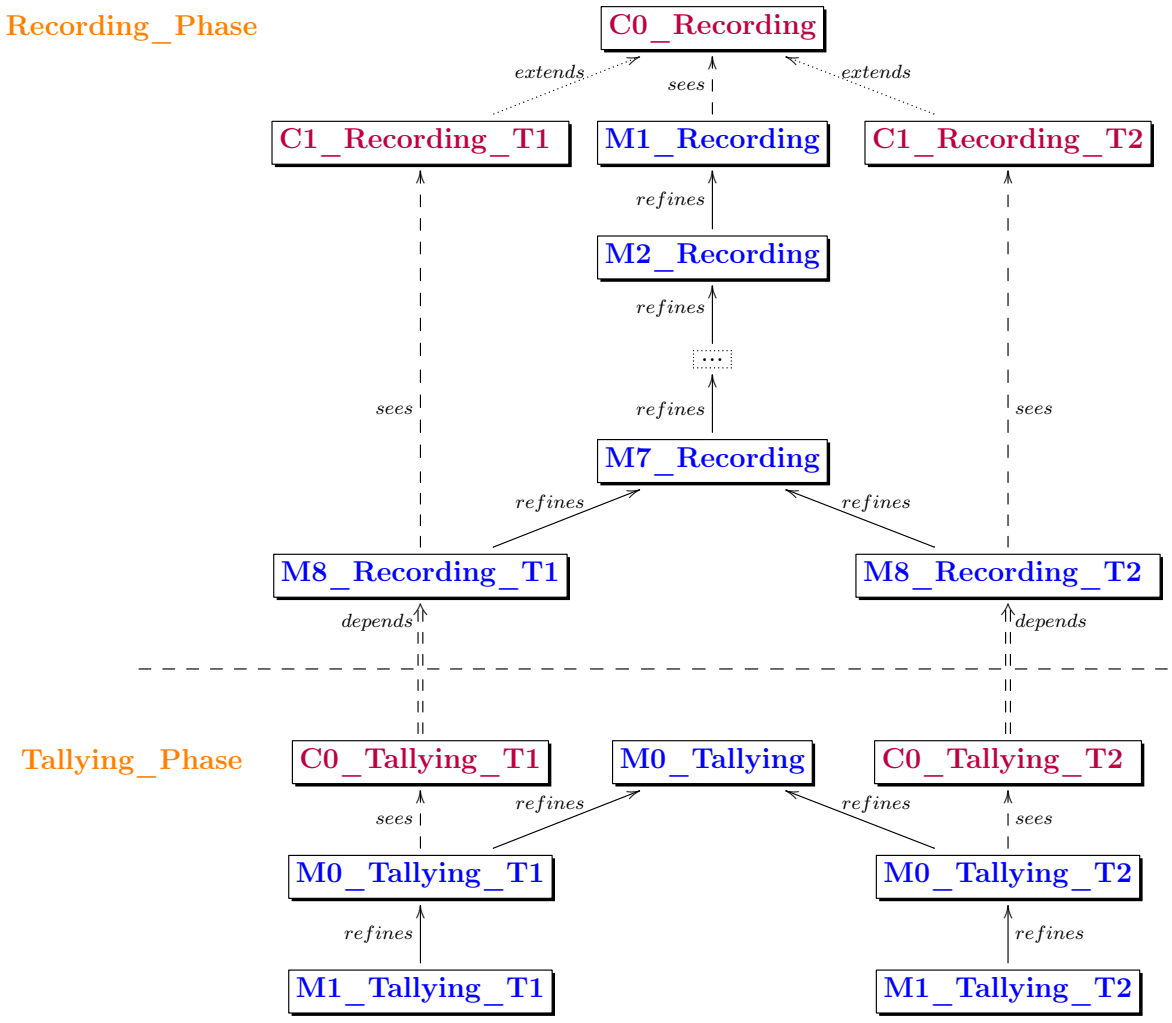

FIGURE 4.1 – Structure générale de développement des patrons pour les systèmes de vote

La première phase est décrite par un contexte Event-B $C0\_Recording$ qui définit les contraintes et les éléments statiques nécessaires pour l'ensemble des machines définies pour cette phase. Ce contexte B est établi par la phase de préparation que nous supposons vérifiée. Neuf machines ($M1\_Recording$ ... $M8\_Recording\_T1$, $M8\_Recording\_T2$) qui voient le contexte B $C0\_Recording$ sont définies.

1. Un premier modèle abstrait est introduit contenant trois événements : *i)* un événement *voter* qui illustre l'enregistrement des votes ; *ii)* un événement *forwarding_ time* qui fait évoluer le temps et converge vers l'heure de fermeture des bureaux de vote fixée par le système dans le contexte $C0\_Recording$;
   iii) un événement *finish* qui ne fait qu'observer la valeur du booléen maintenue à vrai suite à l'exécution de l'événement *forwarding_ time*. Pour gérer le temps l'événement *forwarding_ time* fait décroître un variant via un compteur qu'il incrémente ;
2. Dans le premier raffinement, on distingue l'enregistrement des votes de leur suppression. L'événement *voter* est raffiné pour exprimer qu'un vote est enregistré, en modifiant les valeurs de la variable qui illustre l'enregistrement des votes, ou bien est supprimé en supprimant la valeur d'un vote de cette même variable, à condition que l'intervalle du temps imparti pour le vote soit respecté.
3. Dans le deuxième raffinement, on introduit les trois scénarios de corruption, où des personnes se présentent pour voter à la place de quelqu'un d'autre sans avoir l'autorisation, c'est un scénario parmi d'autres scénarios de corruption qui peuvent exister. L'événement d'enregistrement du vote est raffiné en deux événements qui illustrent le vote correct et le vote malhonnête. On introduit également les scénarios où les trucages consistent à bourrer les urnes ou enregistrer des signatures sans vote réel. Deux autres événements sont introduits à cet effet à ce niveau de raffinement.
4. Dans tout système de vote il existe toujours des electeurs qui sont susceptibles de voter et faire un choix. Dans le troisième raffinement on introduit les électeurs qui représentent l'acteur principal du système en question. On distingue alors les votes des votants qui se présentent eux-mêmes pour voter et ceux effectués par procuration.
5. Le quatrième raffinement introduit les sessions qui permettent l'authentification pour chaque électeur ayant le droit de vote.
6. Le raffinement suivant précise les votes enregistrés par localisation, une nouvelle variable est introduite à cet effet et les gardes des événements pour choisir et enregistrer les votes sont renforcées par la condition d'existence des électeurs souhaitant voter dans les listes électorales préalablement établies. L'éligibilité est exprimée à ce niveau de raffinement.
7. Une manière de rendre les bulletins secrets est d'introduire des enveloppes. Le sixième raffinement introduit donc les enveloppes qu'on associe à chaque bulletin une fois le choix effectué. Le choix est réalisé par le biais d'un nouvel événement introduit à ce niveau.
8. Notre spécification est ensuite décomposée en deux machines $M8\_Recording\_T1$, $M8\_Recording\_T2$ différentes qui raffinent la machine ($M7\_Recording$) et voient deux nouveaux contextes B ($C1\_Recording\_T1$, $C1\_Recording\_T2$) incluant des représentations des suffrages (format des bulletins) selon le mode de scrutin visé. Ces deux derniers contextes B étendent le premier contexte B ($C0\_Recording$).

La phase du dépouillement (*Tallying phase*) est également modélisée par un composant Event-B.

1. Un premier modèle abstrait est introduit $M0\_Tallying$, celui-ci ne voit aucun contexte. Un contexte B est défini ($C0\_Tallying\_T1$, $C0\_Tallying\_T2$) pour chaque machine issue du premier composant. Chaque contexte B est destiné à un mode de scrutin.

2. Deux machines reliées par raffinement sont introduites pour chaque mode de scrutin qui voient chacune un contexte B différent et raffinent la machine $M0\_Tallying$. Le dépouillement est effectué par bureaux ou localisations dans la première machine.

3. Ensuite, le résultat final est calculé dans la seconde machine de chaque composant. La propriété de vérifiabilité est établie lors de cette phase comme axiomes dans les contextes B définis et comme invariant respecté par les comportements observés dans chaque machine modélisant un type de vote précis.

4.3.1 Phase du vote

4.3.1.1 Modèle abstrait de la phase d'enregistrement des votes

Nous caractérisons le premier modèle par un contexte $C0\_Recording$ et une machine $M1\_Recording$. Ce premier contexte B introduit les éléments nécessaires pour débuter une phase de vote ou d'enregistrement du vote. Dans ce premier contexte B, il nous faut tous les éléments qui permettent de donner la possibilité de choisir et de prendre en compte le vote d'un électeur. En particulier, notre modélisation considère que la phase de préparation du vote est acquise (avérée), ce premier contexte B représente donc le résultat fourni de la phase de préparation que l'on suppose vérifiée. Les ensembles porteurs, les constantes et les propriétés statiques sont introduits dans ce contexte Event-B tels que : *Electors*, *Choices*, *Envelopes*, *PollingStation*, *Representatives*, *Bulletins*, *Signatures*, *emargements_electors_list*, *votershosting*, *start_time*, *end_time* etc.

Dans ce premier modèle, l'état du système est caractérisé par deux variables qui représentent chacune les votes enregistrés et le temps qui s'écoule dans le système. Les votes sont vus comme une relation entre l'ensemble des émargements (*Signatures*) et les choix des électeurs *Choices*. Les invariants dans cette machine ne sont qu'un moyen pour typer les variables.

VARIABLES

recorded_ votes

timer

INVARIANTS

*inv*1 : $recorded\_votes \in Signatures \leftrightarrow Choices$

*inv*2 : $timer \in start\_time \mathinner{..} end\_time$

VARIANT

$end\_time - timer$

INITIALISATION

act1 : $recorded\_votes := \varnothing$

act2 : $timer := start\_time$

END

EVENT ***register_ votes***

WHEN

grd1 : $timer \geq start\_time \wedge timer < end\_time$

grd2 : $\forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow timer \notin i \mathinner{..} j$

THEN

act1 : $recorded\_votes$

$recorded\_votes' \in (Signatures \leftrightarrow Choices)$

END

EVENT forwarding_time

Status convergent

WHEN

grd : $timer < end\_time$

THEN

act : $timer := timer + 1$

END

EVENT finish

WHEN

grd1 : $timer = end\_time$

THEN

act : $skip$

END

La pré-condition à cette phase est que le temps soit égal à l'heure d'ouverture des bureaux fixés dans le contexte *C*0_*Recording* et qu'aucun vote n'a été enregistré. Cette exigence est exprimée par l'événement d'initialisation, où les votes sont initialisés à l'ensemble vide. L'enregistrement d'un vote est simplement exprimé par la modification de la variable *recorded_ votes* qui est effectuée par l'événement *register_ votes*. Dans cette machine, on ne distingue que les valeurs de la variable *recorded_ votes* qui prennent leur valeurs dans $Signatures \leftrightarrow Choices$ sans préciser les actions entreprises. L'action de l'événement *register_ votes* est non-déterministe.

L'événement *forwarding_ time* change la valeur de la variable *timer* introduite dans la présente machine pour exprimer la progression du temps dans le système. La valeur de la variable (du compteur) est incrémentée par l'action de cet événement *forwarding_ time* jusqu'à ce que l'heure de fermeture des bureaux soit atteinte *end_ time*. Il faut noter que cet événement a un statut convergent. Ainsi, l'événement *forwarding_ time*, sous lequel une hypothèse d'équité faible est faite, ne sera plus observable quand la valeur de la variable *timer* vaudra *end_ time*. Nous précisons qu'à ce niveau, l'événement pour voter ne peut être observable que lorsque le vote débute et que l'heure de fermeture n'est pas encore arrivée, la garde *grd1* est donc introduite pour illustrer cette exigence ; et qu'à l'initialisation la valeur du *timer* vaut l'heure de début fixée dans le contexte Event-B vu par la présente machine.

Le vote électronique peut être soumis à des interruptions, d'où l'intérêt d'introduire une constante dans le présent contexte Event-B pour représenter les séquences durant lesquelles le processus de vote sera interrompu. Aussi, l'événement *register_ votes* est gardé par la garde *grd2* :

$$\left(\; grd2 : \forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow timer \notin i \mathinner{..} j \; \right)$$

Cette garde garantit que le service sera interrompu durant les séquences définies, par

la constante *interrupt_ sequences* qui est définie comme suit :

$$
\begin{array}{l}
axm23 : interrupt\_sequences \in \mathbb{N}_1 \rightarrowtail\!\!\!\!\!\!\rightarrow \mathbb{N}_1 \\
axm24 : \forall i \cdot i \mapsto i \notin interrupt\_sequences \\
axm25 : \forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow i < j \\
axm26 : \forall i, j, k, l \cdot i \mapsto j \in interrupt\_sequences \wedge k \mapsto l \in interrupt\_sequences \\
\qquad\qquad \wedge i < k \Rightarrow j < k \\
axm27 : \forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow i > start\_time \wedge j < end\_time
\end{array}
$$

Notons que cette représentation du temps est abstraite, et nous montrons en section 4.5 comment il serait possible de donner une représentation concrète du temps ainsi que les opérations de comparaison des valeurs qui vont avec.

La preuve de convergence des événements introduits est réalisée par l'introduction d'un *variant* qui montre qu'effectivement l'exécution de ces événements fait décroître cette expression :

VARIANT
$end\_time - timer$

Ainsi, arrivé à la fin du vote, plus personne ne peut déposer de vote, signer (émarger). Le seul événement qui sera alors observable est l'événement *finish*, qui n'effectue aucune action dans le système.

Ainsi, la stabilité est garantie en maintenant la valeur du compteur à sa valeur à la fin, et nous avons également la preuve que toutes les valeurs de variables ne changeront pas, puisque aucun événement dans les machines ne sera activable à la fin du vote. Et aucun changement ne peut être fait sur les variables du système car aucun des événements *register_ votes* ou *forwarding_ time* ne sera observable. Cette propriété garantit la stabilité comme nous l'avons expliqué dans le chapitre précédent.

Nous rappelons aussi que dans tous ce qui suit, toutes les variables sont initialisées par l'ensemble vide, à l'exception de la variable qui concerne la liste des électeurs qui n'ont pas encore voté (*list_ electors_ notyet_ voters*) introduite au dernier raffinement de cette première phase.

Nous donnons dans ce qui suit, les preuves de stabilité en s'appuyant sur la démarche donnée dans le chapitre précédent.

Preuve de stabilité de la phase du vote : modèle abstrait

La propriété de terminaison souhaitée est définie comme suit :

$$\mathcal{T}_1 \;\widehat{=}\; timer = end\_time$$

Nous posons : $x_1 = \{recorded\_votes, timer\}$.

Les hypothèses d'équité posées dans la présente machine sont :

$$L_{\mathcal{M}_1} \;\widehat{=}\; WF_{x_1}(forwarding\_time)$$

$$\begin{aligned} NEXT_1 \;\widehat{=}\; & \vee BA(\mathit{register\_votes})(x_1, x_1') \\ & \vee BA(\mathit{forwarding\_time})(x_1, x_1') \\ & \vee BA(\mathit{finish})(x_1, x_1') \\ & \vee (x_1 = x_1') \end{aligned}$$

$$\begin{aligned} \mathcal{I}nit_1(x_1) \;\widehat{=}\; & \mathit{timer} = \mathit{start\_time} \\ & \wedge \mathit{recorded\_votes} = \varnothing \end{aligned}$$

La propriété de vivacité à montrer est la suivante :

$$\Phi_{\mathcal{M}_1} \widehat{=} \mathcal{I}nit_1(x_1) \rightsquigarrow \mathcal{T}_1$$

qui signifie qu'à partir de l'heure d'ouverture des bureaux de vote, à l'initialisation, où aucun vote n'est encore enregistré, on arrivera fatalement à atteindre un état où le temps évoluera jusqu'à atteindre l'heure de fermeture des bureaux avec une éventuelle modification des valeurs des votes enregistrés.

Nous posons : $\mathcal{I}nit_{x_1} \widehat{=} P_0$, $V(t) \widehat{=} \mathit{end\_time} - \mathit{timer} = t$. La propriété de stabilité $\Phi_{\mathcal{M}_1}$ est exprimée comme suit :

$$\begin{aligned} \Phi_{\mathcal{M}_1} \;\widehat{=}\; & \{V(t) \rightsquigarrow V(t-1) && \Phi_1 \\ & , (\exists t.V(t)) \rightsquigarrow V(0) && \Phi_2 \\ & \} \end{aligned}$$

— Φ_1 stipule qu'on arrivera fatalement par atteindre un état où la valeur du compteur diminuera d'un pas de 1 ;
— Φ_2 avance qu'on arrivera fatalement à un état où la différence convergera vers l'heure de fermeture des bureaux de vote, valeur pour laquelle le variant vaudra zéro ;

En posant $T_1 \widehat{=} V(t)$ et $T_2 \widehat{=} V(t-1)$, nous avons :

$$P_0 \wedge [NEXT_1]_{x_1} \Rightarrow (T_1' \vee T_2')$$

Nous divisons la preuve de cette implication en les preuves suivantes, selon la définition $NEXT_1$:

— $T_1 \wedge BA(\mathit{forwarding\_time})(x_1, x_1') \Rightarrow (T_1' \vee T_2')$
— $T_1 \wedge BA(\mathit{register\_votes})(x_1, x_1') \Rightarrow (T_1' \vee T_2')$
— $T_1 \wedge BA(\mathit{finish})(x_1, x_1') \Rightarrow (T_1' \vee T_2')$
— $T_1 \wedge x_1 = x_1' \Rightarrow (T_1' \vee T_2')$

Le seul événement qui fait évoluer le temps est *forwarding_time*, donc, la valeur du compteur n'est modifiée que par les actions prévues dans cet événement. Cet événement est observable jusqu'à ce que la valeur du compteur *timer* atteindra *end_time*.

L'hypothèse d'équité faible posée sur l'événement *forwarding_time* nous permet de dire que **(a)** : $T_1(x_1) \wedge BA(\mathit{forwarding\_time})(x_1, x_1') \Rightarrow T_2(x_1')$, et de la condition de faisabilité de l'événement *forwarding_time*, nous pouvons déduire que **(b)** : $T_1(x_1) \Rightarrow (\exists x_1'.BA(\mathit{forwarding\_time})(x_1, x_1'))$ sont satisfaites par l'événement *forwarding_time*. Et l'on peut déduire de **(a)**, $T_1 \wedge \langle NEXT \wedge \mathit{forwarding\_time} \rangle_{x_1} \Rightarrow T_2$ et de **(b)** que $T_1 \Rightarrow \mathit{ENABLED}\langle \mathit{forwarding\_time} \rangle_{x_1}$, avec
$\mathit{ENABLED}\langle \mathit{forwarding\_time} \rangle_{x_1} \widehat{=} (\exists y_1.BA(\mathit{forwarding\_time})(x_1, y_1))$.

La règle WF1 permet de déduire que le temps finira par progresser pour faire décroître le variant et puisqu'aucun événement ne viole la propriété $V(t)$, à l'exception de l'événement *forwarding_time*. En appliquant la règle LATTICE sur l'ensemble bien fondé des entiers naturels, nous pouvons déduire que le système convergera vers la valeur $V(1)$. Nous pouvons donc déduire que : $\mathcal{S}pec(\mathcal{M}_1) \vdash \Phi_1$.

Nous avons alors : $T_1 = 1 = V(1)$.

L'observation de l'événement *forwarding_time* arrivé à $V(1)$, et l'hypothèse d'équité faible posée sur ce même événement, nous permettent de déduire que : $\mathcal{S}pec(\mathcal{M}_1) \vdash \Phi_2$, d'où la terminaison avec stabilité est atteinte, puisque une fois le variant atteint sa valeur minimale, seul l'événement *finish* sera activable dans le système, et aucun changement ne peut être fait sur les variables dans le système, à partir du moment où aucun des événements *register_votes* ou *forwarding_time* ne sera observable. À partir du moment où tous les événements introduits dans la suite seront également gardés par les gardes *grd1* et *grd2* de l'événement *register_votes*, aucun événement ne change le temps et les événements *forwarding_time* et *finish* restent inchangés, les preuves de stabilité sont identiques dans tous les raffinements qui suivent.

La stabilité est aussi définie et prouvée pour la phase de dépouillement du vote.

4.3.1.2 Premier raffinement

Nous distinguons à ce niveau les votes qui sont soit enregistrés soit supprimés. Les deux événements raffinent le même événement *register_votes* introduit dans la précédente machine. L'enregistrement des votes est effectué par l'événement *register_votes*, alors que la suppression des votes est effectuée par l'événement *remove_votes_correctly*.

EVENT ***register_votes***
- **REFINES** *register_votes*
- **ANY**
  - s, v
- **WHERE**
  - **grd1 :** $s \in Signatures$
  - **grd2 :** $v \in Choices$
  - **grd3 :** $timer \geq start\_time \wedge timer < end\_time$
  - **grd4 :** $\forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow timer \notin i..j$
- **THEN**
  - **act1 :** $recorded\_votes :=$ $recorded\_votes \cup \{s \mapsto v\}$

END

EVENT ***remove_votes_correctly***
- **REFINES** *register_votes*
- **ANY**
  - s, v
- **WHERE**
  - **grd1 :** $s \mapsto v \in recorded\_votes$
  - **grd2 :** $timer \geq start\_time \wedge timer < end\_time$
  - **grd3 :** $\forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow timer \notin i..j$
- **THEN**
  - **act1 :** $recorded\_votes :=$ $recorded\_votes \setminus \{s \mapsto v\}$

END

4.3.1.3 Deuxième raffinement

Nous distinguons à ce niveau de raffinement les votes enregistrés correctement et les votes corrompus. Quatre nouvelles variables sont introduites.

Les votes sont identifiés dans l'ensemble des choix ($inv2$), alors que les signatures sont un sous-ensemble dans l'ensemble des émargements ($inv1$) définis dans le premier contexte B explicité ci-dessus $C0\_Recording$. La propriété $inv3$ associe à chaque choix correct une et une seule signature, et à chaque signature un et un seul choix correct, ceci est exprimé par une bijection entre les deux sous-ensembles $valid\_signatures \rightarrowtail\!\!\!\!\!\rightarrow valid\_choices$. Cette propriété exprime qu'à tout moment, le nombre de votes dans les urnes est égal au nombre de signatures sur la liste d'émargements.

Nous introduisons des scénarios d'intrusion, où les gens se présentent pour voter à la place de quelqu'un d'autre sans permission, c'est un scénario parmi d'autres scénarios

corrompus qui peuvent exister. Les votes peuvent être corrompus, mais cette corruption est détectée comme un mauvais vote. Les votes corrects et les votes corrompus sont des sous-ensembles de l'ensemble des votes enregistrés. La propriété invariante $inv7$ indique que ces deux variables n'ont pas d'éléments en commun.

VARIABLES
valid_ signatures
valid_ choices
valid_ votes
corrupt_ votes_ signatures
INVARIANTS
$inv1$: $valid\_signatures \subseteq Signatures$
$inv2$: $valid\_choices \subseteq Choices$
$inv3$: $valid\_votes \in valid\_signatures \rightarrowtail\!\!\!\!\!\!\rightarrow vote$
$inv4$: $corrupt\_votes\_signatures \in Signatures \leftrightarrow Choices$
$inv5$: $valid\_votes \subseteq recorded\_votes$
$inv6$: $corrupt\_votes\_signatures \subseteq recorded\_votes$
$inv7$: $corrupt\_votes\_signatures \cap valid\_votes = \varnothing$

L'événement pour enregistrer les votes introduit dans le premier modèle est raffiné en deux événements qui permettent le stockage des votes corrects et des votes truqués. Ce scénario de trucage est donc un scénario où des choix truqués ainsi que des signatures truquées sont introduits simultanément via l'événement *corrupt_ choices_ signatures_ simultaneously*.

EVENT *vote_ correctly*
REFINES *register_ votes*
ANY
s, v
WHERE
grd1 : $s \in Signatures \setminus valid\_signatures$
grd2 : $v \in Choices \setminus valid\_choices$
grd3 : $v \notin deleted\_votes$
grd4 : $s \notin dom(corrupt\_votes\_signatures) \land v \notin ran(corrupt\_votes\_signatures)$
grd5 : $s \mapsto v \notin recorded\_votes$...
THEN
act1 : $valid\_votes := valid\_votes \cup \{s \mapsto v\}$
act2 : $valid\_signatures := valid\_signatures \cup \{s\}$
act3 : $valid\_choices := valid\_choices \cup \{v\}$
act4 : $recorded\_votes := recorded\_votes \cup \{s \mapsto v\}$
END

EVENT ***corrupt_ choices_ signatures_ simultaneously***
REFINES *register_ votes*
  ANY
    s, v
  WHERE
    grd1 : $s \in Signatures$
    grd2 : $v \in Choices$
    grd3 : $s \mapsto v \notin valid\_votes$
    grd4 : $s \mapsto v \notin recorded\_votes$...
  THEN
    act1 : $corrupt\_votes\_signatures := corrupt\_votes\_signatures \cup \{s \mapsto v\}$
    act2 : $recorded\_votes := recorded\_votes \cup \{s \mapsto v\}$
END

D'autres variables sont également introduites pour identifier les fausses signatures, et les choix invalides introduits séparément. L'événement pour enregistrer les votes introduits au premier modèle est affiné en deux événements qui permettent le stockage des votes corrects et des votes qui sont corrompus. Ce scénario de corruption est un scénario dans lequel des choix corrompus et des signatures corrompues sont introduites simultanément via l'événement *corrupt_ choices_ signatures_ simultaneously*.

Ce raffinement introduit également deux autres scénarios de corruption consistant à bourrer des urnes ou à enregistrer des votes sans de véritables signatures. Deux nouveaux événements sont introduits à ce niveau. L'événement *stuffing_ choices* consiste à ajouter un faux choix en changeant la variable *alone_ corrupt_ choices*, tandis que l'événement *corrupt_ signatures_ only* ajoute une signature corrompue en changeant la variable *alone_ corrupt_ signatures*. La variable *alone_ corrupt_ choices* est un sous ensemble de l'ensemble des choix, alors que les signatures corrompues sont un sous ensemble de l'ensemble des *Signatures*.

L'enregistrement des choix et signatures introduits par les événements d'intrusion est simulé dans les variables corrompues dans lesquelles les intrus croient enregistrer leurs votes ou signatures ou les deux à la fois. Ces deux variables n'ont aucun élément commun avec les votes corrects et les signatures correctes introduites dans le raffinement précédent (voir *inv10* et *inv11*).

$inv10 : \ valid\_choices \cap alone\_corrupt\_choices = \varnothing$
$inv11 : \ valid\_signatures \cap alone\_corrupt\_signatures = \varnothing$

Les intrus peuvent également supprimer la signature et les choix enregistrés correctement par les électeurs honnêtes, en utilisant deux nouveaux événements qui affinent *remove_ votes_ correctly*. Les seules hypothèses que nous posons dans notre développement sont que l'intrus ne peut supprimer des signatures ou des votes (choix). Ceci est dû à la bijection qui existe entre les signatures avec les votes corrects $valid\_signatures \rightarrowtail\!\!\!\!\!\rightarrow valid\_choices$ introduite dans ce même raffinement. La suppression d'un élément de cette variable implique la connaissance des choix effectués par le votant. L'enregistrement des choix et des signatures introduits par intrusion dans les événements intrusifs est simulé dans des variables de trucage dans lesquelles l'intrus croit enregistrer son vote ou sa signature ou les deux à la fois.

EVENT ***delete_ signatures***
REFINES *remove_ votes_ correctly*
  ANY
    s
  WHERE
  grd1 : $s \in valid\_signatures \land s \in dom(valid\_votes)$
  grd2 : $timer \geq start\_time \land timer < end\_time$
  grd3 : $\forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow timer \notin i \mathinner{..} j$
  WITH $v : v = valid\_votes(s)$
  THEN
  act1 : $valid\_signatures := valid\_signatures \setminus \{s\}$
  act2 : $valid\_choices := valid\_choices \setminus \{valid\_votes(s)\}$
  act3 : $valid\_votes := \{s\} \lhd valid\_votes$
  act4 : $alone\_corrupt\_choices := votes\_truques \cup \{valid\_votes(s)\}$
  act5 : $recorded\_votes := recorded\_votes \setminus \{s \mapsto valid\_votes(s)\}$
END

EVENT ***dishonest_ delete_ choices***
REFINES *remove_ votes_ correctly*
  ANY
    c
  WHERE
  grd1 : $c \in valid\_choices \land c \in ran(valid\_votes) \land c \notin alone\_corrupt\_choices$
  grd2 : $timer \geq start\_time \land timer < end\_time$
  grd3 : $\forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow timer \notin i \mathinner{..} j$
  WITH
    $s : s = valid\_votes^{-1}(c)$
    $v : v = c$
  THEN
  act1 : $valid\_choices := valid\_choices \setminus \{c\}$
  act2 : $valid\_votes := valid\_votes \rhd \{c\}$
  act3 : $valid\_signatures := valid\_signatures \setminus \{valid\_votes^{-1}(c)\}$
  act4 : $recorded\_votes := recorded\_votes \setminus \{valid\_votes^{-1}(c) \mapsto c\}$
  act5 : $alone\_corrupt\_signatures := alone\_corrupt\_signatures \cup \{valid\_votes^{-1}(c)\}$
  act6 : $alone\_corrupt\_choices := alone\_corrupt\_choices \cup \{c\}$
END

Il faut noter qu'à ce niveau de raffinement aucune explicitation des bulletins n'est faite, ce détail est introduit dans le dernier raffinement de cette phase. Une telle structuration nous permet de factoriser au mieux notre développement. Ceci dit, chaque choix est unique dans cette représentation ce qui nous garantit par construction la propriété du *vote unique*.

4.3.1.4 Troisième raffinement

Ce raffinement introduit l'acteur principal dans le système i.e., l'électeur. Les électeurs ayant effectivement voté correctement deviennent des votants enregistrés dans la variable *honest_ voters*. Cette variable permet la sauvegarde des votants éligibles. Les votants malhonnêtes sont enregistrés dans la variable *dishonest_ voters*. Les électeurs ayant voté correctement peuvent usurper l'identité d'autres électeurs pour voter à leur place. Ceci dit, on ne peut rien affirmer quant à la relation entre ces deux variables. Les votants honnêtes

sont reliés à leur signatures correctes via la variable *honest_ voters_ signatures*, ainsi un votant honnête ne peut avoir qu'une et une seule signature à la fois, et inversement, une signature correcte n'est assignée qu'à un et seul votant à la fois ($inv3$).

Les signatures corrompues des votants sont définies de l'ensemble des électeurs dans l'ensemble des signatures (*Signatures*) ($inv4$). Notons que le domaine et le co-domaine de ces deux variables n'ont aucun élément en commun ($inv5$ et $inv6$).

$$
\begin{array}{ll}
inv1: & honest\_voters \subseteq Electors \\
inv2: & dishonest\_voters \subseteq Electors \\
inv3: & honest\_voters\_signatures \in honest\_voters \rightarrowtail\!\!\!\!\!\!\rightarrow valid\_signatures \\
inv4: & corrupt\_signatures\_voters \in Electors \nrightarrow Signatures \\
inv5: & dom(corrupt\_signatures\_voters) \cap dom(honest\_voters\_signatures) = \varnothing \\
inv6: & ran(corrupt\_signatures\_voters) \cap ran(honest\_voters\_signatures) = \varnothing
\end{array}
$$

La suppression d'un choix correct préalablement fait implique la connaissance de ce choix par le votant honnête. Pour assurer le *secret* du vote, nous introduisons une variable qui représente les connaissances du votant ($inv7$). Cette variable est une fonction totale des électeurs ayant voté dans l'ensemble des parties de l'ensemble des choix. Le secret est exprimé alors par l'invariant $inv12$, qui énonce que chaque votant ayant effectivement voté ne connaît que son propre choix.

$$
\begin{array}{l}
inv7: honest\_knowledge \in honest\_voters \rightarrow \mathbb{P}(valid\_choices) \\
inv8: \forall v, elec, sig \cdot \left(\begin{array}{l} v \in valid\_choices \wedge elec \in honest\_voters \wedge sig \in valid\_signatures \\ \wedge elec \mapsto sig \in honest\_voters\_signatures \wedge sig \mapsto v \notin recorded\_votes \end{array} \right) \\
\qquad \Rightarrow elec \mapsto \{v\} \notin honest\_knowledge) \\
inv12: \forall elec1, elec2, v1, v2 \cdot \left(\begin{array}{l} elec1 \in dom(honest\_knowledge) \wedge honest\_knowledge(elec1) = v1 \\ \wedge elec2 \in dom(honest\_knowledge) \\ \wedge honest\_knowledge(elec2) = v2 \wedge elec1 \neq elec2 \end{array} \right) \\
\qquad \Rightarrow (\forall vx \cdot (vx \in v1 \Rightarrow vx \notin v2))) \\
thm14: \forall v1, v2, elec \cdot (elec \mapsto v1 \in honest\_knowledge \wedge elec \mapsto v2 \in honest\_knowledge \Rightarrow v1 = v2)
\end{array}
$$

On peut alors déduire la propriété de sûreté qui indique que le votant ne connaît que son propre vote (*thm14*).

Tous les événements qui introduisent des votes sont renforcés par la garde :

$$grd: \ voterx \mapsto \{v\} \notin honest\_knowledge$$

et l'action qui consiste à ajouter la connaissance d'un vote à un votant est ajoutée aux événements qui concernent le vote *vote_ correctly* :

$$act: honest\_knowledge := honest\_knowledge \cup \{voterx \mapsto \{v\}\}$$

L'événement de suppression correcte des votes est renforcé par les gardes :

$$
\begin{array}{ll}
grd1: & voterx \in dom(honest\_knowledge) \wedge honest\_knowledge(voterx) = \{v\} \\
grd2: & \neg(\exists votery \cdot (votery \neq voterx \wedge votery \in dom(honest\_knowledge) \\
& \qquad \wedge honest\_knowledge(votery) = \{v\})) \\
grd3: & \neg(\exists v2 \cdot (v2 \neq v \wedge \{v2\} \in ran(honest\_knowledge) \wedge honest\_knowledge(voterx) = \{v2\}))
\end{array}
$$

La suppression d'un choix implique que le votant qui souhaite supprimer son vote doit avoir connaissance de son vote, et qu'il n'existe pas de votant qui connaît son vote (il est le seul à connaître son vote) (cf. gardes *grd6*, *grd11* et *grd12* de l'événement *remove_votes_correctly* donné au septième raffinement 4.3.1.8). Ainsi, notre modélisation autorise la suppression de votes de façon correcte. Par conséquent, la suppression d'un vote est accompagnée du retrait de la connaissance du vote du votant en question :

$$act1 : \ honest\_knowledge := honest\_knowledge \setminus \{voterx \mapsto \{v\}\}$$

Et les choix supprimés ne sont plus réutilisés :

$$act8 : deleted\_votes := deleted\_votes \cup \{v\}$$

La variable *deleted_votes* est introduite au deuxième raffinement 4.3.1.3. Elle est définie comme suit :

$$\begin{array}{l} inv12 : deleted\_votes \subseteq Choices \\ inv13 : \forall v \cdot (v \in deleted\_votes \Rightarrow \neg(\exists s \cdot s \mapsto v \in (recorded\_votes))) \\ inv14 : alone\_corrupt\_choices \cap deleted\_votes = \varnothing \wedge valid\_choices \cap deleted\_votes = \varnothing \end{array}$$

Les signatures ainsi que les choix sont également supprimés. En revanche, les suppressions malhonnêtes sont libres.

Il faut dire que cette représentation est une représentation très abstraite qui peut être détaillée lors de raffinements supplémentaires. Plutôt que de lier les connaissances des votants à un choix abstrait, nous pourrions dans les raffinements qui suivent faire ce lien avec les liens entre le vote ainsi que le votant. Le co-domaine de la variable *honest_knowledge* sera alors la relation entre le votant et les bulletins introduits dans les derniers raffinements. L'anonymat d'un vote n'est pas le secret des bulletins des votes de chaque votant, mais c'est la confidentialité du lien entre le votant et son vote et qui ne doit être connu. Cette propriété repose donc sur la destruction du lien qui existe entre le votant et son bulletins. La preuve que ce lien n'est pas connu peut être illustrée lors de la vérification en utilisant les connaissances des intrus qui n'ont justement pas de connaissances du lien qui peut exister entre le votant et son choix.

Parmi les propriétés qui sont issues du domaine on peut citer :

$$inv14 : \forall vx, c \cdot \left(\begin{array}{l} vx \in honest\_voters \wedge vx \mapsto \{c\} \in honest\_knowledge \\ \quad \Rightarrow \exists\, sig \cdot \left(\begin{array}{l} sig \in valid\_signatures \\ \wedge\ vx \mapsto sig \in honest\_voters\_signatures \\ \wedge\ sig \mapsto c \in valid\_votes \end{array} \right) \end{array} \right)$$

qui exprime que l'existence d'un votant et d'un vote connu par ce votant implique systématiquement qu'il existe une signature qui a été attachée à ce votant via la variable *honest_voters_signatures*. Cette variable est aussi définie à ce niveau de raffinement comme suit :

$$inv13 : honest\_voters\_signatures \in honest\_voters \rightarrowtail\!\!\!\!\!\!\rightarrow valid\_signatures$$

D'autres variables sont introduites à ce niveau de raffinement pour lier les votants malhonnêtes à leur signature, ...

4.3.1.5 Quatrième raffinement

L'authentification nécessite que le système dispose au préalable de moyens qui assurent la reconnaissance d'une entité souhaitant interagir avec le système (numéro d'identification personnel, mot de passe, ...), qui permettent d'associer l'entité qui réclame cette identité et son identité virtuelle. Cette propriété apporte une garantie de l'identité déclarée d'une personne ou des données. Une telle identification dans notre spécification est assurée par la donnée des deux constantes introduites dans le contexte B $C0\_Recording$ ie : $Credentials\_assignment$ ainsi que $Passwords\_assignment$ qui sont définies comme : $Credentials\_assignment \in Electors \rightarrowtail\!\!\!\!\!\rightarrow Credentials$,
$Passwords\_assignment \in Electors \rightarrowtail\!\!\!\!\!\rightarrow Passwords$. C'est une attribution de *crédentials* et de mots de passe aux électeurs éligibles. Ainsi, chaque électeur possède sa propre identification qui lui donne l'autorisation pour l'accès à son espace qui est, par définition, unique à chaque électeur (bijections entre l'ensemble des électeurs et les ensembles des *crédentials* et mots de passe).

L'authentification consiste dans notre système à effectuer une vérification en introduisant deux événements dédiés à cet effet. Le premier événement permet l'ouverture d'une session pour l'électeur qui souhaite avoir l'accès à son espace de vote (se connecter), alors que le second permet la déconnexion d'un électeur ayant déjà établi une connexion.

EVENT ***open_session***
  ANY
    $elec, cred, pw$
  WHERE
    grd1 : $elec \in Electors \setminus electors\_session$
    grd2 : $cred \in Credentials \land pw \in Passwords$
    grd3 : $elec \mapsto cred \in Credentials\_assignment$
    grd4 : $elec \mapsto pw \in Passwords\_assignment$
    grd5 : $timer \geq start\_time \land timer < end\_time$
  THEN
    act1 : $electors\_session := electors\_session \cup \{elec\}$
END

EVENT ***close_session***
  ANY
    $elec$
  WHERE
    grd1 : $elec \in Electors \land elec \in electors\_session$
    grd2 : $timer \geq start\_time \land timer < end\_time$
  THEN
    act1 : $electors\_session := electors\_session \setminus \{elec\}$
END

Une authentification modifie la variable $electors\_session$ introduite à cet effet pour illustrer qu'un électeur a établi une connexion et que sa session est ouverte. Notons que cette authentification ne permet que l'accès à l'espace de vote à des fins de consultations,

d'enregistrement de vote ... Ainsi tous les événements (enregistrement et suppression de votes) dans le système sont renforcés par la garde :

$$grd: \ voterx \in electors\_session$$

L'identification est exprimée comme suit :

$$inv6 : \forall s, v \cdot \left(\begin{array}{l} s \mapsto v \in valid\_votes \\ \Rightarrow \exists\ elec, mdp, cred \cdot \left(\begin{array}{l} ((elec \mapsto s) \in dom(votershosting) \\ \wedge\ elec \mapsto mdp \in Passwords\_assignment \\ \wedge\ elec \mapsto cred \in Credentials\_assignment) \end{array} \right) \end{array} \right)$$

qui indique que pour tout vote enregistré correctement dans le système, il existe un électeur éligible pour le faire.

L'autorisation de vote nécessite que l'électeur ayant le droit de voter n'a pas encore voté. Cette vérification est effectuée dans les raffinements 4.3.1.6, et 4.3.1.7.

Cette étape distingue également les intrus qui tentent d'établir une connexion avec des informations d'identification et des mots de passe volés. Toutes les variables qui correspondent aux connaissances des intrus, ainsi que les événements pour les usurpations des identités, les mots de passe, des signatures mal utilisées et éventuellement les choix des votants déjà faits par des électeurs honnêtes. Les électeurs ne peuvent pas établir des connexions honnêtes et malhonnêtes en même temps. Cette propriété est exprimée comme suit :

$$thm5: \ intruders\_session \cap electors\_session = \varnothing$$

L'introduction de ces détails implique que les invariants $inv8$ et $inv12$ introduits dans le précédent raffinement ne sont plus suffisants. Nous devons avoir la capacité pour exprimer que les connaissances honnêtes ne sont pas connues des intrus. La propriété suivante exprime par exemple, le fait que les choix honnêtes ne sont pas connus des intrus malhonnêtes.

$$inv14 : \forall elec1, v1 \cdot \left(\begin{array}{l} elec1 \in dom(honest\_knowledge) \wedge honest\_knowledge(elec1) = v1 \\ \Rightarrow \left(\begin{array}{l} \forall\ elec2 \cdot \left(\begin{array}{l} elec2 \in dom(dishonest\_knowledge\_choice) \\ \Rightarrow \left(\begin{array}{l} \forall\ vx \cdot \left(\begin{array}{l} vx \in v1 \Rightarrow \\ elec2 \mapsto \{vx\} \notin dishonest\_knowledge\_choice \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right)$$

Les mêmes propriétés sont données pour les signatures.

L'éligibilité des votants est donnée comme suit :

$$thm6 : \forall\ s, v \cdot \left(\begin{array}{l} s \mapsto v \in valid\_votes \Rightarrow \exists\ elec, mdp, cred \cdot \left(\begin{array}{l} (elec \mapsto s) \in dom(votershosting) \\ \wedge\ elec \mapsto mdp \in Passwords\_assignment \\ \wedge\ elec \mapsto cred \in Credentials\_assignment \end{array} \right) \end{array} \right)$$

qui stipule que pour tous les votes enregistrés correctement (choix et signatures), il existe un électeur éligible qui les a émis avec une signature identique à celle enregistrée préalablement lors de l'établissement des configurations issues de la phase de préparation.

L'introduction de l'événement pour obtenir un vote de façon malhonnête viole l'invariant $inv14$. Pour pallier ce problème, nous proposons de combiner nos patrons de conception avec les modèles de l'attaquant déjà réalisés par Benaissa dans ses travaux [33]. Cette opération nous permet l'ajout de clefs dans le premier contexte défini pour la présente phase. Ainsi, nous remplaçons les messages définis dans le modèle [33] de l'attaquant par les éléments définis dans nos modèles (signatures, crédentials, mots de passe...). Le même traitement est appliqué pour toutes les variables malhonnêtes définies à ce niveau comme la variable *dishonest_ knowledge_ credentials* qui enregistre les crédentials connus des intrus, ... Pour qu'un attaquant puisse connaître les votes et les signatures des votants, il faut qu'il ait les crédentials ainsi que les mots de passe pour s'authentifier et avoir la possibilité de le faire. Pour ces raisons et puisque la taille des modèles, et des obligations de preuve, devient de plus en plus importante, nous l'appliquerons uniquement pour les connaissances des mots de passe ainsi que celles des crédentials.

4.3.1.6 Cinquième raffinement

Les votes sont enregistrés par localisations ou bureaux de vote. On note bureaux, localités ou blocks pour illustrer ces sortes de bureaux virtuels qui coincident avec les bureaux réels dans le cas de vote à l'urne classique. Le présent raffinement introduit une nouvelle variable *registred_ votes_ offices*, qui assigne chaque vote enregistré à un et un seul bureau de vote ($inv1$).

$$
\begin{array}{l}
inv1: \ registred\_votes\_offices \in PollingStation \leftrightarrow (recorded\_votes) \\
inv2: \ \forall ve \cdot (ve \in recorded\_votes \ \Rightarrow (\exists h \cdot (h \mapsto ve \in registred\_votes\_offices))) \\
inv3: \ \forall v1, b1, b2 \cdot \left(\begin{array}{l} (b1 \mapsto v1 \in registred\_votes\_offices \wedge \\ b2 \mapsto v1 \in registred\_votes\_offices \Rightarrow b1 = b2) \end{array} \right)
\end{array}
$$

L'invariant $inv3$ exprime qu'un vote n'est enregistré que dans un unique bureau de vote. L'enregistrement du vote est ainsi restreint par localisation. Cette restriction est une restriction d'autorisations pour les électeurs de voter dans des bureaux auxquels ils ont été affectés. Une liste est préalablement établie pour assigner les électeurs éligibles dans des bureaux, celle-ci est définie par la constante *votershosting* dans le contexte B $C0\_Recording$ ($votershosting \in emargements\_electors\_list \rightarrow PollingStation$). Une telle liste peut être assimilée à l'enregistrement des numéros des cartes de vote des électeurs éligibles dans un vote classique. Ainsi, les événements pour enregistrer les votes sont renforcés par les gardes :

$$
\begin{array}{l}
grd8: \ \exists sig \cdot (sig \in Signatures \wedge loc \in PollingStation \wedge ((voterx \mapsto sig) \mapsto loc) \in votershosting) \\
grd9: \ loc \mapsto (s \mapsto v) \notin registred\_votes\_offices \wedge (s \mapsto v) \notin ran(registred\_votes\_offices)
\end{array}
$$

et l'action suivante est ajoutée afin de mettre à jour la variable *registred_ votes_ offices* :

$$
act7: \ registred\_votes\_offices := registred\_votes\_offices \cup \{loc \mapsto (s \mapsto v)\}
$$

Alors que la suppression de votes est renforcée par les gardes :

$$grd6 : \ voterx \mapsto s \in emargements\_electors\_list$$
$$grd8 : \ (s \mapsto v) \in ran(registred\_votes\_offices) \land b \mapsto (s \mapsto v) \in registred\_votes\_offices$$

et entraîne la suppression du vote enregistré du bureau dans lequel il a été enregistré :

$$act7 : \ registred\_votes\_offices := registred\_votes\_offices \setminus \{b \mapsto (s \mapsto v)\}$$

De cette manière, seules les personnes autorisées à voter le peuvent, cette propriété exprime l'*éligibilité* par bureau de vote ($inv4$). Elle exprime que pour tous les choix correctement enregistrés ($valid\_votes$) dans les bureaux de vote ($registred\_votes\_offices$), il existe un électeur ayant une signature valide ($honest\_voters\_signatures$ introduite dans le troisième raffinement) avec une signature identique, préalablement enregistrée dans le système, qui a envoyé ce choix.

De plus, cette modélisation garantit le vote unique où chaque électeur ne peut exprimer son choix qu'une seule fois dans un seul bureau i.e., *pas de vote double* (voir invariant $inv3$ combiné avec l'invariant $inv3$ introduit au second raffinement 4.3.1.3).

$$inv4 : \forall\, s, v, h \cdot \left(\begin{array}{l} s \mapsto v \in valid\_votes \land h \mapsto (s \mapsto v) \in registred\_votes\_offices \\ \Rightarrow \exists\, elec \cdot \left(\begin{array}{l} elec \in honest\_voters \\ \land elec \mapsto s \in honest\_voters\_signatures \\ \land (elec \mapsto s) \mapsto h \in votershosting \end{array} \right) \end{array} \right)$$

4.3.1.7 Sixième raffinement

Jusqu'à présent, nous avons considéré les choix des électeurs comme étant des éléments de l'ensemble des choix. Cependant, dans le monde réel, un vote correspond à l'expression des choix parmi les options de vote selon le type et le mode de scrutin visé. Une telle expression correspond aux suffrages qui sont concrétisés par les bulletins de vote. Par conséquent, l'enregistrement des votes est précédé du choix que peut effectuer un électeur habilité à voter. Le choix de bulletins doit être anonyme, ce qui peut être assuré par des enveloppes comme c'est le cas dans un vote classique. On introduit dans ce raffinement quatre nouvelles variables qui modélisent en premier lieu le choix ainsi que l'enregistrement de vote via des enveloppes. La variable $valid\_envelopes$ correspond aux enveloppes choisies par les électeurs. Un électeur ayant pris une enveloppe est ajouté dans la variable $voters\_envelopes$. Chaque choix est affecté à une et une seule enveloppe ($inv3$ et $inv4$)

INVARIANTS

$$inv1 : \ valid\_envelopes \subseteq Envelopes$$
$$inv2 : \ voters\_envelopes \in Electors \nrightarrow Envelopes$$
$$\land voters\_envelopes^{-1} \in Envelopes \nrightarrow Electors$$
$$inv3 : \ correct\_choices\_envelopes \in Choices \nrightarrow Envelopes$$
$$inv4 : corrupt\_choices\_envelopes \in Choices \nrightarrow Envelopes$$

Le choix des électeurs est concrétisé par l'événement ***choose***. Pour qu'un électeur puisse faire un choix, ce dernier doit avoir l'autorisation pour cette action. L'activation des actions de cet événement est gardée par l'existence de la personne qui souhaite initier un

processus de vote dans la liste préalablement établie *emargements_ electors_ list* ($grd1$) et qu'aucune signature n'est encore enregistrée pour son vote ($grd2$).

EVENT ***choose***
  ANY
    $elec, e, loc$
  WHERE
    grd1 : $elec \in dom(emargements\_ electors\_ list)$
    grd2 : $\neg(\exists s \cdot (s \in valid\_ signatures \land elec \mapsto s \in honest\_ voters\_ signatures))$
    grd3 : $e \in Envelopes \land e \notin valid\_ envelopes$
    grd4 : $elec \notin dom(voters\_ envelopes) \land e \notin ran(voters\_ envelopes)$
      $\land e \notin ran(correct\_ choices\_ envelopes) \land e \notin ran(corrupt\_ choices\_ envelopes)$
    grd5 : $\exists sig \cdot (sig \in Signatures \land loc \in PollingStation \land ((elec \mapsto sig) \mapsto loc) \in votershosting)$
      ...
  THEN
    act1 : $voters\_ envelopes := voters\_ envelopes \cup \{elec \mapsto e\}$
    act2 : $valid\_ envelopes := valid\_ envelopes \cup \{e\}$
END

L'événement a pour conséquence l'enregistrement de l'enveloppe qui, dans ce modèle abstrait, représente les bulletins (ces derniers sont introduits dans le raffinement suivant). Lors d'un vote correct c'est la variable *correct_ choices_ envelopes* qui est concernée par l'ajout de l'électeur ayant voté correctement avec son enveloppe. Dans le cas d'une intrusion, c'est la variable *corrupt_ choices_ envelopes* qui est modifiée dans les événements intrusifs. À noter que les domaines et les co-domaines des deux variables sont disjoints :

INVARIANTS
  $inv8 :\ dom(correct\_ choices\_ envelopes) \cap dom(corrupt\_ choices\_ envelopes) = \varnothing$
  $inv9 :\ ran(correct\_ choices\_ envelopes) \cap ran(corrupt\_ choices\_ envelopes) = \varnothing$

L'enregistrement des votes corrects entraîne la modification de la variable *correct_ choices_ envelopes* par l'ajout du choix et de l'enveloppe concernés par cette action, alors qu'un vote corrompu est sauvegardé dans la variable *corrupt_ choices_ envelopes*. Des éventuelles modifications de choix sont également prises en compte, en introduisant un événement pour supprimer le choix déjà effectué. Ensuite, l'électeur peut à nouveau choisir son nouveau vote.

Les choix par procuration sont aussi pris en compte dans notre développement. L'événement pour choisir avec une procuration est introduit à ce niveau, alors que l'événement qui concerne l'enregistrement des votes est introduit au troisième raffinement 4.3.1.4.

Plusieurs propriétés sont également introduites à ce niveau de raffinement pour gérer les relations entre les enveloppes ainsi que les choix et les variables des intrus.

4.3.1.8 Septième raffinement

Les modes de scrutin sont un moyen pour concrétiser l'objectif d'un vote. Plus précisemment, ces derniers sont exprimés par le biais de bulletins qui correspondent aux choix i.e., à une représentation particulière. Par exemple, un vote présidentiel majoritaire où un

candidat doit être élu est représenté par des bulletins où chaque candidat est l'option de vote et chaque bulletin correspond à une voix. C'est sur cette base que le dépouillement est effectué. Dans ce mode de scrutin un bulletin est une voix de vote et correspond au candidat inscrit sur ce même bulletin. Un bulletin est assigné à un et un seul candidat. Cette contrainte est représentée en Event-B par une constante définie comme suit :

AXIOMS
$axm1 :\ bulletins\_representatives \in Representatives \rightarrowtail Bulletins$

Dans le cas d'un vote à préférences, les électeurs doivent faire leur choix sur les bulletins où tous les candidats sont inscrits sur tous les bulletins. Ce choix correspond à un ordre de préférences mentionné à coté de chaque candidat sur le même bulletin. Cette contrainte correspond à un produit cartésien représenté comme suit en Event-B :

AXIOMS
$axm1 :\ bulletins\_representatives = Representatives \times Bulletins$

Ces contraintes contextualisent donc le développement et nous les qualifions de ***contexte de contraintes***. Chaque mode de scrutin n'a aucune interprétation sémantiquement équivalente dans l'autre. Par conséquent, chacun de ces modes est défini dans un contexte B événementiel différent. Ces deux contextes B étendent le premier contexte introduit en début de cette section ($C0\_Recording$) et sont notés par $C1\_Recording\_T1$ et $C1\_Recording\_T2$.

Arrivé à ce stade de raffinement la machine introduite dans le raffinement précédent est raffinée en deux machines différentes. Cette décomposition permet à chacune des machines de voir un contexte Event-B différent. Ainsi, la machine $M8\_Recording\_T1$, (respectivement la machine $M8\_Recording\_T2$) voit le contexte $C1\_Recording\_T1$ (respectivement le contexte $C1\_Recording\_T2$). Dans ce qui suit, nous développons le premier type de vote. Le second type de vote est spécifié de manière similaire. Cependant, ce dernier se différencie par rapport au premier type par la structure des ballots ainsi que par l'ajout des mentions qui font référence aux préférences des électeurs ou aux poids (dans le cas d'un vote cumulatif) indiqués sur chaque bulletin de vote une fois le choix effectué.

La machine $M8\_Recording\_T1$ introduit de nouvelles variables, on peut citer par exemple les variables suivantes :

INVARIANTS
$inv1 :\ bulletins\_voters \in Electors \nrightarrow Bulletins$
$\quad \wedge bulletins\_voters^{-1} \in Bulletins \nrightarrow Electors$
$inv2 :\ ballots \in Envelopes \nrightarrow Bulletins$
$inv3 :\ choice\_bulletins \in Choices \leftrightarrow Bulletins$
$inv4 :\ envelopes\_representatives \in Representatives \leftrightarrow valid\_envelopes$
$inv5 :\ ballots\_offices \in ballots \nrightarrow PollingStation$
$inv6 :\ recorded\_bulletins \subseteq Bulletins$
$inv7 : ballots^{-1} \in Bulletins \nrightarrow Envelopes$

Chaque électeur ayant choisi un bulletin est ajouté dans la variable $bulletins\_voters$ avec son bulletin. Cette action est observée dans l'événement ***choose***. Ainsi, un électeur peut prendre un certain nombre de bulletins chacun correspondant à un candidat différent,

puis effectue son choix. Le bulletin choisi ainsi que le votant sont mis dans la variable *bulletins_ voters*. Ce choix définit un ballot stocké dans la variable *ballots*. L'électeur glisse un et un seul nom (candidat) dans l'urne. Par conséquent, un et un seul bulletin est déposé dans une enveloppe, ceci est exprimé par la propriété invariante $inv7$ qui complète l'invariant $inv2$. Au plus un ballot est enregistré par bureau de vote ($inv5$).

La propriété $inv30$ exprime le fait qu'on ne peux avoir qu'un bulletin correct enregistré dans le système et connu par les votants.

$$inv30 : \forall\, v, vv, bullt1, bullt2 \cdot \left(\begin{array}{l} v \in honest\_voters \wedge\ vv \in valid\_choices \wedge vv \in honest\_knowledge(v) \\ \wedge\ v \mapsto bullt1 \in bulletins\_voters \\ \wedge\ v \mapsto bullt2 \in bulletins\_voters \\ \wedge\ vv \mapsto bullt1 \in choice\_bulletins \\ \wedge\ vv \mapsto bullt2 \in choice\_bulletins \Rightarrow bullt1 = bullt2 \end{array} \right)$$

L'invariant $inv35$

$$inv35 : \forall\, v, vv, bullt1 \cdot \left(\begin{array}{l} v \in honest\_voters \wedge vv \in valid\_choices \wedge vv \in honest\_knowledge(v) \\ \wedge v \mapsto bullt1 \in bulletins\_voters \wedge vv \mapsto bullt1 \in choice\_bulletins \\ \Rightarrow \forall\, elec \cdot \left(\begin{array}{l} elec \neq v \Rightarrow (elec \mapsto bullt1 \notin bulletins\_voters \\ \wedge elec \mapsto bullt1 \notin corrupt\_bulletins\_voters) \end{array} \right) \end{array} \right)$$

Ainsi, l'événement ***choose*** est raffiné par deux événement *choose_ correctly* et *choose_ dishonestly* pour illustrer les choix corrects et les choix incorrects. À titre indicatif, l'événement *choose_ correctly* est donné comme suit :

EVENT ***choose_ correctly***
REFINES ***choose***
ANY
$elec, e, b, repr, loc, c, selected\_b$
WHERE
grd1 : $elec \in dom(emargements\_electors\_list)$
grd2 : $\neg(\exists s \cdot (s \in valid\_signatures \land elec \mapsto s \in honest\_voters\_signatures))$
grd3 : $e \in Envelopes \land e \notin valid\_envelopes$
grd4 : $elec \notin dom(voters\_envelopes) \land e \notin ran(voters\_envelopes)$
$\land e \notin ran(correct\_choice\_envelope) \land e \notin ran(corrupt\_choice\_envelope)$
grd5 : $b \subseteq Bulletins \land selected\_b \in b$
grd6 : $finite(b) \land card(b) \geq 2 \land b \subseteq ran(bulletins\_representatives)$
grd7 : $\forall bi \cdot (bi \in b \Rightarrow bi \notin ran(ballots) \land e \mapsto bi \notin ballots)$
grd8 : $elec \notin dom(bulletins\_voters)$
$\land (\forall bi \cdot (bi \in b \Rightarrow bi \notin ran(bulletins\_voters) \land elec \mapsto bi \notin bulletins\_voters))$
grd9 : $repr \in Representatives$
grd10 : $repr \mapsto e \notin envelopes\_representatives$
grd11 : $c \in Choices \setminus dom(choice\_bulletins)$
grd12 : $timer \geq start\_time \land timer < end\_time$
grd13 : $\exists sig \cdot (sig \in Signatures \land loc \in PollingStation \land ((elec \mapsto sig) \mapsto loc) \in votershosting)$
grd14 : $b \not\subseteq recorded\_bulletins$
grd15 : $elec \in electors\_session$
grd16 : $e \notin ran(envelopes\_representatives)$
grd17 : $e \mapsto selected\_b \notin ballots \land (e \mapsto selected\_b) \mapsto loc \notin ballots\_offices$
grd18 : $repr \mapsto selected\_b \in bulletins\_representatives \setminus collected\_bulletins\_representatives$
grd19 : $e \notin corrupt\_envelopes \land e \notin deleted\_envelopes$
grd20 : $\forall elecx, bb \cdot (bb \in b \land elecx \mapsto bb \notin corrupt\_bulletins\_voters \land bb \notin ran(corrupt\_bulletins\_voters))$
grd21 : $\forall bi \cdot bi \in b \land bi \notin deleted\_bulletins$
grd22 : $b \not\subseteq dom(signed\_bulletins)$
grd23 : $\forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow timer \notin i \mathinner{..} j$
THEN
act1 : $voters\_envelopes := voters\_envelopes \cup \{elec \mapsto e\}$
act2 : $valid\_envelopes := valid\_envelopes \cup \{e\}$
act3 : $bulletins\_voters := bulletins\_voters \cup \{elec \mapsto selected\_b\}$
act4 : $ballots := ballots \cup \{e \mapsto selected\_b\}$
act5 : $envelopes\_representatives := envelopes\_representatives \cup \{repr \mapsto e\}$
act6 : $choice\_bulletins := choice\_bulletins \cup \{c \mapsto selected\_b\}$
act7 : $recorded\_bulletins := recorded\_bulletins \cup \{selected\_b\}$
act8 : $collected\_bulletins\_representatives := collected\_bulletins\_representatives \cup \{repr \mapsto selected\_b\}$
END

Notons qu'il est possible aussi de supprimer un choix une fois effectué (avant son enregistrement dans le système), et qu'il est aussi possible de le modifier suite à cette action de suppression.

Les bulletins enregistrés sont un sous-ensemble de l'ensemble des bulletins et qui correspondent aux bulletins émis par l'ensemble des votants *inv*6. Cette variable nous servira dans le contexte B de dépouillement correspondant à ce type de vote que nous détaillerons dans la deuxième phase de notre développement. Nous introduisons également une nouvelle variable à ce niveau pour illustrer le sous-ensemble des valeurs de la constante *bulletins_ representatives* qui a été recueilli durant le vote.

INVARIANTS
$inv: collected\_bulletins\_representatives \subseteq bulletins\_representatives$
$inv: recorded\_bulletins = ran(collected\_bulletins\_representatives)$

C'est sur la base des représentants mentionnés sur les bulletins recueillis via la variable *collected_ bulletins_ representatives* que sera effectuée l'attribution des voix pour ces représentants. Les ballots sont enregistrés par bureau, ainsi la variable *ballots_ offices* permet d'enregistrer les ballots par bureau de vote. Un ballot est enregistré en un et un seul bureau de vote *inv*5.

Le dépôt des votes dans les urnes est concrétisé par la modification de la variable *recorded_ votes* et les actions qui l'accompagnent dans les événements appropriés. Cette action est accompagnée du dépôt réel des ballots dans les urnes affectées par bureau. C'est la variable *ballots_ offices* qui est concernée par cette modification. Cette représentation est suffisante pour notre développement dans cette phase. Cependant, pour les besoins du dépouillement dans la deuxième phase et plus précisément, pour les contraintes qui doivent être respectées dans le contexte B de dépouillement du même type de vote, la variable *choice_ bulletins* est ajoutée dans cette machine. Cette variable devient une constante dans le contexte B de dépouillement pour ce type de vote via le mécanisme de dépendance que nous avons explicité dans la première partie de notre contribution. Ces actions sont exécutées dans les événements ***vote_ correctly***, ***vote_ with_ procuration***, ***corrupt_ choices_ signatures_ simultaneously*** ainsi que ***stuffing_ choices***. La suppression d'un vote élimine également les ballots qui ont été introduits par les votants qui les ont émis. Cette action est ajoutée dans l'événement ***remove_ votes_ correctly***. Une telle suppression est accompagnée de la suppression des choix, des ballots des bureaux ainsi que des enveloppes attachées à ces mêmes ballots.

À titre indicatif, l'événement *vote_ correctly* est raffiné comme suit :

EVENT ***vote_ correctly***
REFINES ***vote_ correctly***
ANY
$s, v, voterx, loc, e, bullt$
WHERE
grd1 : $v \notin ran(corrupt\_votes\_signatures)$
grd2 : $e \in valid\_envelopes$
grd3 : $voterx \mapsto e \in voters\_envelopes \land v \mapsto e \notin correct\_choice\_envelope$
$\land(\forall vv \cdot (vv \neq v \Rightarrow vv \mapsto e \notin correct\_choice\_envelope))$
grd4 : $e \in ran(envelopes\_representatives)$
grd5 : $v \in dom(choice\_bulletins)$
grd6 : $timer \geq start\_time \land timer < end\_time$
grd7 : $\exists sig \cdot (sig \in Signatures \land loc \in PollingStation$
$\land((voterx \mapsto sig) \mapsto loc) \in votershosting \land s = sig)$
grd8 : $voterx \mapsto \{v\} \notin honest\_knowledge$
grd9 : $loc \mapsto (s \mapsto v) \notin registred\_votes\_offices$
$\land(s \mapsto v) \notin ran(registred\_votes\_offices)$
grd10 : $e \mapsto bullt \in ballots \land e \mapsto bullt \notin dom(ballots\_offices)$
$\land((e \mapsto bullt) \mapsto loc) \notin ballots\_offices$
grd11 : $voterx \in electors\_session$
grd12 : $s \mapsto v \notin recorded\_votes$
grd13 : $voterx \notin dom(corrupt\_signatures\_voters)$
$\land(\forall sigg \cdot voterx \mapsto sigg \notin corrupt\_signatures\_voters)$
grd14 : $s \notin ran(corrupt\_signatures\_voters) \land e \notin ran(corrupt\_choice\_envelope)$
$\land(\forall vv \cdot (vv \neq v \Rightarrow vv \mapsto e \notin corrupt\_choice\_envelope))$
grd15 : $bullt \notin ran(corrupt\_bulletins\_voters)$
$\land(\forall elec \cdot (elec \mapsto bullt \notin corrupt\_bulletins\_voters))$
grd16 : $\forall hh \cdot hh \neq loc \Rightarrow (e \mapsto bullt) \mapsto hh \notin ballots\_offices$
grd17 : $v \mapsto bullt \in choice\_bulletins$
$\land(\forall vv \cdot vv \neq v \Rightarrow vv \mapsto bullt \notin choice\_bulletins)$
grd18 : $bullt \notin dom(signed\_bulletins)$
$\land(\forall sig \cdot (sig \neq s \Rightarrow bullt \mapsto sig \notin signed\_bulletins))$
$\land(\forall bul \cdot bul \neq bullt \Rightarrow bul \mapsto s \notin signed\_bulletins)$
grd19 : $\forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow timer \notin i \mathinner{..} j$
grd20 : $v \notin dom(corrupt\_choice\_envelope)$
$\land(\forall elec \cdot (elec \mapsto \{v\} \notin dishonest\_vote\_knowledge))$
grd21 : $voterx \in Electors \setminus honest\_voters \land voterx \notin dom(attorney\_letters)$
grd22 : $voterx \mapsto s \notin honest\_voters\_signatures$
grd23 : $s \in Signatures \setminus valid\_signatures$
$\land(\forall elec \cdot (elec \mapsto \{s\} \notin dishonest\_sig\_knowledge))$
grd24 : $v \in Choices \setminus dom(correct\_choice\_envelope)$
$\land v \notin dom(corrupt\_choice\_envelope) \land v \notin deleted\_votes$

THEN
act1 : $recorded\_votes := recorded\_votes \cup \{s \mapsto v\}$
act2 : $correct\_choice\_envelope := correct\_choice\_envelope \cup \{v \mapsto e\}$
act3 : $registred\_votes\_offices := registred\_votes\_offices \cup \{loc \mapsto (s \mapsto v)\}$
act4 : $honest\_knowledge := honest\_knowledge \cup \{voterx \mapsto \{v\}\}$
act5 : $ballots\_offices := ballots\_offices \cup \{((e \mapsto bullt) \mapsto loc)\}$
act6 : $valid\_choices := valid\_choices \cup \{v\}$
act7 : $signed\_bulletins := signed\_bulletins \cup \{bullt \mapsto s\}$
act8 : $list\_electors\_notyet\_voters := list\_electors\_notyet\_voters \setminus \{voterx\}$
act9 : $honest\_voters := honest\_voters \cup \{voterx\}$
act10 : $honest\_voters\_signatures(voterx) := s$
act11 : $valid\_votes := valid\_votes \cup \{s \mapsto v\}$
act12 : $valid\_signatures := valid\_signatures \cup \{s\}$
END

Les gardes de l'événement responsable des suppressions de vote correctes sont effectuées selon l'événement *remove_votes_correctly* comme suit :

EVENT ***remove_ votes_ correctly***
REFINES ***remove_ votes_ correctly***
ANY
$v, s, voterx, b, e, bullt$
WHERE
grd1 : $voterx \mapsto bullt \in bulletins\_voters$
grd2 : $e \mapsto bullt \in ballots$
grd3 : $e \in ran(envelopes\_representatives)$
grd4 : $bullt \in ran(choice\_bulletins)$
grd5 : $(e \mapsto bullt) \in dom(ballots\_offices) \land ((e \mapsto bullt) \mapsto b) \in ballots\_offices$
grd6 : $voterx \in honest\_voters \land honest\_knowledge(voterx) = \{v\}$
grd7 : $v \mapsto e \in correct\_choice\_envelope$
grd8 : $voterx \mapsto s \in emargements\_electors\_list$
grd9 : $voterx \in electors\_session$
grd10 : $\neg(\exists elecx \cdot (elecx \in dom(voters\_envelopes)$
$\land elecx \neq voterx \land elecx \mapsto e \in voters\_envelopes))$
grd11 : $\neg(\exists voter2 \cdot (voter2 \neq voterx$
$\land voter2 \in dom(honest\_knowledge) \land honest\_knowledge(voter2) = \{v\}))$
grd12 : $\neg(\exists v2 \cdot (v2 \neq v \land \{v2\} \in ran(honest\_knowledge) \land honest\_knowledge(voterx) = \{v2\}))$
grd13 : $\neg(\exists votery \cdot (votery \neq voterx \land votery \mapsto e \in voters\_envelopes))$
grd14 : $\neg(\exists votery \cdot (votery \neq voterx \land votery \mapsto bullt \in bulletins\_voters))$
grd15 : $\neg(\exists votery \cdot (votery \neq voterx \land votery \mapsto e \in voters\_envelopes$
$\land votery \mapsto bullt \in bulletins\_voters \land (e \mapsto bullt) \in dom(ballots\_offices)$
$\land ((e \mapsto bullt) \mapsto b) \in ballots\_offices))$
grd16 : $v \in dom(correct\_choice\_envelope)$
grd17 : $s \in valid\_signatures$
grd18 : $s \mapsto v \in valid\_votes \land s \mapsto v \in recorded\_votes$
grd19 : $timer \geq start\_time \land timer < end\_time$
grd20 : $voterx \in dom(honest\_voters\_signatures) \land honest\_voters\_signatures(voterx) = s$
grd21 : $b \mapsto (s \mapsto v) \in registred\_votes\_offices$
grd22 : $e \in valid\_envelopes \land bullt \in recorded\_bulletins \land voterx \mapsto e \in voters\_envelopes$
grd23 : $\forall sig \cdot (sig \neq s \Rightarrow sig \mapsto v \notin recorded\_votes)$
grd24 : $bullt \mapsto s \in signed\_bulletins$
$\land (\forall sig \cdot (sig \neq s \Rightarrow bullt \mapsto sig \notin signed\_bulletins))$
$\land (\forall bul \cdot bul \neq bullt \Rightarrow bul \mapsto s \notin signed\_bulletins)$
grd25 : $voterx \notin list\_electors\_notyet\_voters$
grd26 : $\forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow timer \notin i \mathinner{..} j$

Les actions de l'événement pour des suppressions de vote correctes sont effectuées comme suit :

THEN
act1 : $ballots\_offices := ballots\_offices \setminus \{((e \mapsto bullt) \mapsto b)\}$
act2 : $valid\_choices := valid\_choices \setminus \{v\}$
act3 : $collected\_bulletins\_representatives := collected\_bulletins\_representatives \rhd \{bullt\}$
act4 : $deleted\_votes := deleted\_votes \cup \{v\}$
act5 : $valid\_votes := valid\_votes \setminus \{s \mapsto v\}$
act6 : $valid\_signatures := valid\_signatures \setminus \{s\}$
act7 : $correct\_choice\_envelope := \{v\} \lhd correct\_choice\_envelope$
act8 : $recorded\_votes := recorded\_votes \setminus \{s \mapsto v\}$
act9 : $registred\_votes\_offices := registred\_votes\_offices \setminus \{b \mapsto (s \mapsto v)\}$
act10 : $honest\_voters := honest\_voters \setminus \{voterx\}$
act11 : $honest\_voters\_signatures := honest\_voters\_signatures \setminus \{voterx \mapsto s\}$
act12 : $honest\_knowledge := honest\_knowledge \setminus \{voterx \mapsto \{v\}\}$
act13 : $valid\_envelopes := valid\_envelopes \setminus \{e\}$
act14 : $voters\_envelopes := voters\_envelopes \setminus \{voterx \mapsto e\}$
act15 : $envelopes\_representatives := envelopes\_representatives \rhd \{e\}$
act16 : $choice\_bulletins := choice\_bulletins \rhd \{bullt\}$
act17 : $bulletins\_voters := bulletins\_voters \setminus \{voterx \mapsto bullt\}$
act18 : $ballots := ballots \setminus \{e \mapsto bullt\}$
act19 : $recorded\_bulletins := recorded\_bulletins \setminus \{bullt\}$
act20 : $deleted\_envelopes := deleted\_envelopes \setminus \{e\}$
act21 : $deleted\_bulletins := deleted\_bulletins \cup \{bullt\}$
act22 : $signed\_bulletins := signed\_bulletins \setminus \{bullt \mapsto s\}$
act23 : $list\_electors\_notyet\_voters := list\_electors\_notyet\_voters \cup \{voterx\}$
END

Les gardes de l'événement de suppression malhonnête de choix sont définies comme suit :

EVENT ***dishonest_ delete_ choices***
REFINES ***dishonest_ delete_ choices***
ANY
$c, elec$
WHERE
grd1 : $c \in valid\_votes \land c \in ran(valid\_votes) \land c \notin dom(corrupt\_choice\_envelope)$
$\land(\exists voterx \cdot voterx \in dom(honest\_voters\_signatures)$
$\land voterx \neq elec \land voterx \mapsto valid\_votes^{-1}(c) \in honest\_voters\_signatures)$
$\land(\exists bullt \cdot bullt \mapsto valid\_votes^{-1}(c) \in signed\_bulletins)$
grd2 : $timer \geq start\_time \land timer < end\_time$
grd3 : $elec \in intruders\_session$
grd4 : $\exists voterx, h \cdot voterx \in honest\_voters \land honest\_knowledge(voterx) = \{c\}$
$\land(voterx \mapsto valid\_votes^{-1}(c)) \mapsto h \in votershosting$
grd5 : $\exists l \cdot l \in dom(registred\_votes\_offices) \land valid\_votes^{-1}(c) \mapsto c \in ran(registred\_votes\_offices)$
$\land valid\_votes^{-1}(c) \mapsto c \in recorded\_votes$
$\land l \mapsto (valid\_votes^{-1}(c) \mapsto c) \in registred\_votes\_offices$
grd6 : $\forall l1, l2 \cdot (l1 \mapsto (valid\_votes^{-1}(c) \mapsto c) \in registred\_votes\_offices$
$\land l2 \mapsto (valid\_votes^{-1}(c) \mapsto c) \in registred\_votes\_offices \Rightarrow l1 = l2)$
grd7 : $\neg(\exists elec1, elec2 \cdot (elec1 \mapsto valid\_votes^{-1}(c) \in honest\_voters\_signatures$
$\land elec2 \mapsto valid\_votes^{-1}(c) \in honest\_voters\_signatures \land elec1 \neq elec2))$
grd8 : $\forall e \cdot (e \in valid\_envelopes \land e \neq correct\_choice\_envelope(c)$
$\Rightarrow \exists bullt \cdot bullt \notin dom(signed\_bulletins \rhd \{valid\_votes^{-1}(c)\}) \land e \mapsto bullt \in ballots$
grd9 : $\forall x, e \cdot (e \in valid\_envelopes \land x \in honest\_voters$
$\land x \mapsto e \in voters\_envelopes \land e \neq correct\_choice\_envelope(c)$
$\Rightarrow (\exists bullt \cdot x \mapsto bullt \in bulletins\_voters$
$\land bullt \notin dom(signed\_bulletins \rhd \{valid\_votes^{-1}(c)\})))$
grd10 : $\forall x, bullt \cdot (x \in honest\_voters \land x \mapsto bullt \in bulletins\_voters$
$\land bullt \notin dom(signed\_bulletins \rhd \{valid\_votes^{-1}(c)\})$
$\Rightarrow (\exists e \cdot (e \in valid\_envelopes \land x \mapsto e \in voters\_envelopes \land e \neq correct\_choice\_envelope(c))))$
grd11 : $\forall i, j \cdot i \mapsto j \in interrupt\_sequences \Rightarrow timer \notin i \mathinner{..} j$
grd12 : $elec \mapsto \{c\} \in dishonest\_vote\_knowledge$

Les actions de l'événement de suppression malhonnête de choix sont définies comme suit :

act1 : $choice\_bulletins := choice\_bulletins \setminus (\{c\} \lhd choice\_bulletins)$
act2 : $valid\_votes := valid\_votes \setminus \{c\}$
act3 : $valid\_votes := valid\_votes \rhd \{c\}$
act4 : $valid\_signatures := valid\_signatures \setminus \{valid\_votes^{-1}(c)\}$
act5 : $recorded\_votes := recorded\_votes \setminus \{valid\_votes^{-1}(c) \mapsto c\}$
act6 : $alone\_corrupt\_signatures := alone\_corrupt\_signatures \cup \{valid\_votes^{-1}(c)\}$
act7 : $honest\_knowledge := honest\_knowledge \setminus \{honest\_knowledge^{-1}(\{c\}) \mapsto \{c\}\}$
act8 : $valid\_envelopes := valid\_envelopes \setminus \{correct\_choice\_envelope(c)\}$
act9 : $correct\_choice\_envelope := correct\_choice\_envelope \setminus \{c \mapsto correct\_choice\_envelope(c)\}$
act10 : $voters\_envelopes := voters\_envelopes \rhd \{correct\_choice\_envelope(c)\}$
act11 : $honest\_voters := honest\_voters \setminus \{honest\_knowledge^{-1}(\{c\})\}$
act12 : $honest\_voters\_signatures := honest\_voters\_signatures \setminus \{honest\_knowledge^{-1}(\{c\}) \mapsto valid\_votes^{-1}(c)\}$
act13 : $registred\_votes\_offices := (registred\_votes\_offices \setminus (registred\_votes\_offices \rhd \{valid\_votes^{-1}(c) \mapsto c\}))$
$\cup\{h \mapsto (s \mapsto cc) \mid h \mapsto (s \mapsto cc) \in registred\_votes\_offices \rhd (\{valid\_votes^{-1}(c) \mapsto c\})$
$\wedge h \in dom(registred\_votes\_offices \rhd (\{valid\_votes^{-1}(c) \mapsto c\}))$
$\wedge (s \mapsto cc \in recorded\_votes \setminus \{valid\_votes^{-1}(c) \mapsto c\})\}$
act14 : $corrupt\_choice\_envelope := corrupt\_choice\_envelope \cup \{c \mapsto correct\_choice\_envelope(c)\}$
act15 : $corrupt\_envelopes := corrupt\_envelopes \cup \{correct\_choice\_envelope(c)\}$
act16 : $signed\_bulletins := signed\_bulletins \setminus (signed\_bulletins \rhd \{valid\_votes^{-1}(c)\})$
act17 : $bulletins\_voters := bulletins\_voters \setminus (bulletins\_voters \rhd (dom(signed\_bulletins \rhd \{valid\_votes^{-1}(c)\})))$
act18 : $ballots\_offices :=$
$(ballots\_offices \setminus ((ballots \rhd dom(signed\_bulletins \rhd \{valid\_votes^{-1}(c)\})) \lhd ballots\_offices))$
$\cup\{(e \mapsto bul) \mapsto l \mid (e \mapsto bul) \mapsto l \in$
$(ballots \rhd dom(signed\_bulletins \rhd \{valid\_votes^{-1}(c)\})) \lhd ballots\_offices$
$\wedge l \in ran((ballots \rhd dom(signed\_bulletins \rhd \{valid\_votes^{-1}(c)\})) \lhd ballots\_offices)$
$\wedge ((e \mapsto bul) \in ballots \rhd dom(signed\_bulletins \rhd \{valid\_votes^{-1}(c)\}))\}$
act19 : $corrupt\_bulletins\_voters := corrupt\_bulletins\_voters$
$\cup\{elecb \mapsto bul \mid elecb = elec \wedge bul \in dom(signed\_bulletins \rhd \{valid\_votes^{-1}(c)\})\}$
act20 : $dishonest\_envelope := dishonest\_envelope \cup \{elec \mapsto correct\_choice\_envelope(c)\}$
act21 : $list\_electors\_notyet\_voters := list\_electors\_notyet\_voters \cup \{honest\_knowledge^{-1}(\{c\})\}$
act22 : $ballots := ballots \setminus (ballots \rhd dom(signed\_bulletins \rhd \{valid\_votes^{-1}(c)\}))$
act23 : $recorded\_bulletins := recorded\_bulletins \setminus (dom(signed\_bulletins \rhd \{valid\_votes^{-1}(c)\}))$
act24 : $collected\_bulletins\_representatives := collected\_bulletins\_representatives \setminus$
$(collected\_bulletins\_representatives \rhd (dom(signed\_bulletins \rhd \{valid\_votes^{-1}(c)\})))$

Le deuxième type de vote se distingue par l'ajout de deux variables qui illustrent les mentions indiquées pour chaque représentant sur les bulletins. Celles-ci peuvent être définies comme suit :

INVARIANTS
$voters\_mentions \in (Electors \times bulletins\_representatives) \pfun \mathbb{N}$
$registred\_mentions \in (ran(dom(voters\_mentions)) \pfun ran(voters\_mentions)) \pfun PollingStation$

Cependant, les preuves ne sont pas toujours faciles et parfois même impossibles à réaliser car les structures (domaine de la variable *registred_ mentions* est défini comme un ensemble de fonctions) deviennent très complexes ce qui empêche les prouveurs de Rodin de réussir les preuves.

Nous ajoutons en plus de ces variables deux nouvelles variables :

$$inv10 : elect\_mentions \subseteq Electors$$
$$inv11 : bullt\_mentions \subseteq bulletins\_representatives$$

Les deux variables ci-dessus sont redéfinies comme suit :

$$inv12 : voters\_mentions \in (elect\_mentions \times bullt\_mentions) \nrightarrow \mathbb{N}$$
$$inv13 : registred\_mentions \in bullt\_mentions \rightarrow \mathbb{N}$$

De plus, il est nécessaire d'avoir les mentions par bureau de vote. La variable *mentions_localities* est prévue à cet effet :

$$inv14 : mentions\_localities \in registred\_mentions \rightarrow PollingStation$$

Les deux propriétés suivantes expriment que les votants qui ont eu le droit pour exprimer un choix (des mentions) sur les bulletins sont ceux qui ont été enregistrés dans la variable *bulletins_voters*. De telles propriétés permettent de préciser les éléments du domaine de la variable *voters_mentions* car celle-ci est une fonction partielle.

$$inv15 : \forall\, elec, bullt, r \cdot (elec \in dom(dom(voters\_mentions)) \wedge bullt \in ran(ran(dom(voters\_mentions))) \wedge (elec \mapsto (r \mapsto bullt)) \in dom(voters\_mentions) \Rightarrow elec \mapsto bullt \in bulletins\_voters$$
$$inv16 : \forall\, elec, bullt \cdot (elec \in dom(bulletins\_voters) \wedge bullt \in ran(bulletins\_voters) \wedge elec \mapsto bullt \notin votant\_bulletins \Rightarrow (\forall r \cdot (elec \mapsto (r \mapsto bullt) \notin dom(voters\_mentions))))$$

Pour résumer (cf.figure 4.2), cette phase décrit le déroulement du vote proprement dit. Un électeur souhaitant exprimer son choix doit établir une connexion à son espace via la donnée des crédentials et mot de passe qui lui ont été fournis préalablement. L'électeur éligible exprime son choix. Une fois le choix enregistré, le votant peut modifier son vote en supprimant d'abord le premier bulletin enregistré, et ainsi enregistrer un nouveau bulletin. A tout moment, l'électeur peut se déconnecter de son espace (i.e. l'événement *close_session* est activable à tout moment suite à l'établissement d'une connexion). Notons que les bulletins et les enveloppes correspondent aux vraies données sur lesquelles les objectifs du vote sont réellement recueillis, les choix ne sont qu'un moyen conceptuel permettant de raisonner rigoureusement sur les comportements dans ce système.

Cette phase achevée (c-à-d : le compteur pour exprimer le temps arrive à sa fin), l'étape du dépouillement peut alors commencer. Celle-ci dépend alors des résultats fournis durant la première phase. Nous détaillons dans ce qui suit cette phase.

4.4 Relation de dépendance entre les phases du vote

À l'issue de la première phase, deux composants se dégagent dans notre modélisation, à savoir, le composant spécifique aux scrutins qui expriment un représentant par un et un seul bulletin et le composant où un bulletin contient tous les représentants du vote visé. Sur cette base le dépouillement des suffrages est différent. En effet, dans le premier cas, un bulletin représente une voix alors que dans le second cas, ce sont les mentions indiquées par les votants qui permettent de faire un calcul des nombres des voix pour chaque représentant. Ainsi un bulletin contient tous les représentants accompagnés des mentions indiquées pour chaque représentant. Nous avons expliqué que ces deux schémas

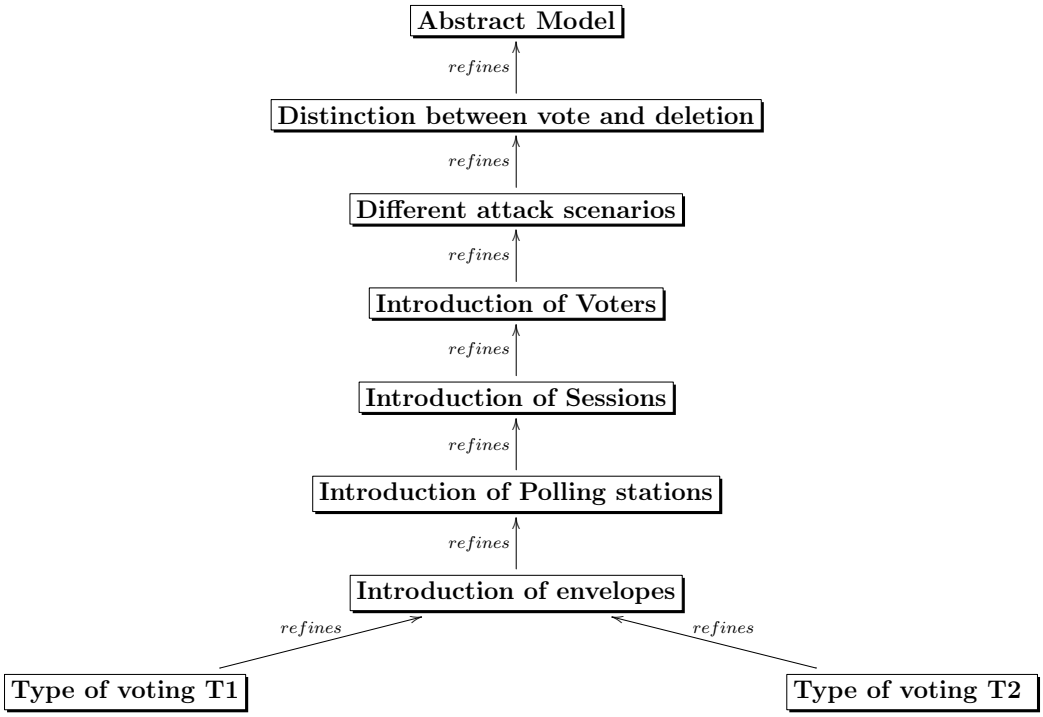

FIGURE 4.2 – Stratégies de raffinement pour la phase du vote

ne trouvent pas de correspondance sémantique entre eux, ce sont par conséquent deux contextes Event-B qui doivent être définis à ce niveau dans notre développement. Ces deux contextes Event-B dépendent des résultats fournis par chaque mode de scrutin lors de la phase d'enregistrement. Les données fournies pour les contextes Event-B de cette phase sont déduites de la première phase ce qui correspond à une validation de ces deux contextes Event-B par les machines issues de la première phase.

Nous détaillons dans ce qui suit le dépouillement du premier type de scrutin qui correspond à la machine $M8\_Recording\_T1$ détaillée dans la partie ci-dessus 4.3.1.8. Le deuxième type est similaire au premier type à l'exception des représentations des bulletins diffèrent avec le premier type ainsi que les mentions indiquées dans ce deuxième type et qui n'existent pas dans le premier type.

4.4.1 Contexte dépendant pour la phase de dépouillement

Dans ce contexte Event-B on retrouve tous les éléments définis dans le contexte Event-B de la première phase, à savoir, le contexte $C1\_Recording\_T1$. Le contexte B $C0\_Tallying\_T1$ étend le contexte B $C1\_Recording\_T1$ et contient, en plus des éléments définis dans $C1\_Recording\_T1$, une partie des variables de la machine $M8\_Recording\_T1$ qui sont définies comme des constantes dans le présent contexte B :

— *valid_signatures* : contient l'ensemble des signatures valides issues de la première phase :

$$axm1 : valid\_signatures \subseteq Signatures \wedge finite(valid\_signatures)$$

— *recorded_votes* : ce sont tous les votes qui ont été enregistrés lors de la phase précédente :

$$axm2 : recorded\_votes \in Signatures \leftrightarrow Choices \land finite(recorded\_votes)$$

— *valid_envelopes* : ce sont les enveloppes enregistrées lors de la phase d'enregistrement :

$$axm3 : valid\_envelopes \subseteq Envelopes \land finite(valid\_envelopes)$$

— *ballots* : correspondent aux bulletins accompagnés de leur enveloppes :

$$axm4 : ballots \in valid\_envelopes \nrightarrow Bulletins$$

— *ballots_offices* : correspondent aux ballots enregistrés par bureau de vote :

$$axm5 : ballots\_offices \in ballots \rightarrow PollingStation$$

— *honest_voters* : correspond à l'ensemble des votants qui ont effectivement voté lors de la première phase :

$$axm6 : honest\_voters \subseteq dom(emargements\_electors\_list) \land honest\_voters \subseteq Electors$$

— *recorded_bulletins* : définit l'ensemble des bulletins qui ont été enregistrés lors de la première phase :

$$axm7 : recorded\_bulletins \subseteq Bulletins$$

— *collected_bulletins_representatives* : contient les représentants indiqués sur chaque bulletin, et choisis par les votants :

$$\begin{aligned} axm8 : {} & collected\_bulletins\_representatives \subseteq bulletins\_representatives \\ & \land finite(collected\_bulletins\_representatives) \\ & \land ran(collected\_bulletins\_representatives) = recorded\_bulletins \end{aligned}$$

— *envelopes_representatives* : contient les enveloppes choisies avec leur représentants :

$$\begin{aligned} axm9 : {} & envelopes\_representatives \in Representatives \leftrightarrow valid\_envelopes \\ & \land dom(collected\_bulletins\_representatives) = dom(envelopes\_representatives) \end{aligned}$$

— *valid_choices* : contient le sous-ensemble des votes (choix) valides :

$$axm10 : valid\_choices \subseteq Choices$$

— *choice_bulletins* : lie chaque vote (choix) à son bulletin :

$$axm11 : choice\_bulletins \in Choices \nrightarrow Bulletins$$

— *registred_votes_offices* : contient l'ensemble des choix effectués par bureau ou localisation de manière légale :

$$axm12 : registred\_votes\_offices \in PollingStation \nleftrightarrow recorded\_votes$$

— *correct_choices_envelopes* : enregistre les choix ainsi que les enveloppes dans lesquelles ces derniers ont été mis :

$$correct\_choices\_envelopes \in valid\_choices \rightarrow valid\_envelopes$$

— *corrupt_ votes_ signatures* : correspond à l'ensemble des votes (choix/signatures) enregistrés de manière illégale :

$$axm13 : corrupt\_votes\_signatures \subseteq recorded\_votes$$

— *alone_ corrupt_ signatures* : contient l'ensemble des signatures corrompues :

$$axm14 : alone\_corrupt\_signatures \subseteq Signatures$$

— *alone_ corrupt_ choices* : contient un sous ensemble de votes (choix) corrompus :

$$axm15 : alone\_corrupt\_choices \subseteq Choices$$

— *corrupt_ choices_ envelopes* : contient l'ensemble des choix effectués par bureau ou localisation de manière illégale :

$$axm16 : corrupt\_choices\_envelopes \in Choices \pfun Envelopes$$

— *timer* : correspond au compteur défini afin de comptabiliser le temps :

$$axm17 : timer \geq start\_time \land timer \leq end\_time \land timer = end\_time$$

— De plus nous avons :

$$axm18 : alone\_corrupt\_signatures \cap valid\_signatures = \varnothing$$

— et :

$$axm19 : dom(correct\_choices\_envelopes) \cap alone\_corrupt\_choices = \varnothing$$

— et :

$$axm20 : (alone\_corrupt\_choices \cup ran(corrupt\_votes\_signatures)) \subseteq dom(corrupt\_choices\_envelopes)$$

— ainsi que :

$$axm21 : valide \in BOOL$$

D'autres constantes sont aussi introduites dans ce contexte Event-B, qui sont définies aussi comme variables dans la machine Event-B dont le présent contexte dépend. Par exemple, les bulletins signés (*signed_ bulletins*) qui définissent les bulletins signés par les votants, etc.

Cette modélisation reflète le fait qu'à la fin de la première phase, aucune modification ne peut être faite sur ces éléments en tant que variables, puisque ces dernières dans la phase d'enregistement du vote concervent leur valeurs à la terminaison. Par conséquent, nous pouvons les définir en tant que constantes dans le présent contexte Event-B, d'où la nécessité d'établir les preuves de stabilité pour la phase précédente qui permettent d'atteindre des états stables dans le système.

Un vote est validé uniquement lorsque toutes les contraintes définies dans ce contexte B sont valides. La validation de telles contraintes repose sur les faits ou les données produits lors la phase d'enregistrement. Ceci implique l'existence d'un ensemble de valeurs d'état dans le modèle $M8\_Recording\_T1$ qui satisfait ces contraintes que nous qualifions

de ***contexte déduit ou combiné de situations et de contraintes***. La satisfaction des axiomes ainsi définis, et en particulier les axiomes **axm23** et **axm24**, exprime la ***"configuration initiale"*** de la présente phase du vote :

$$C0\_Tallying\_T1(s2, c2) \wedge Init_2$$

où s_2 et c_2 sont respectivement, les ensembles et les constantes du contexte Event-B $C0\_Tallying\_T1$. Une telle relation exprime une dépendance entre ces deux composants. En particulier, les deux axiomes **axm23** et **axm24**, donnés ci-dessous, devraient être des théorèmes dans ce contexte Event-B. Leur interprétation logique correspond aux faits fournis par le premier composant de la première phase.

Pour exprimer la validation de ces contraintes, nous introduisons la constante *valide*. Cette constante (*valide*) est elle aussi fonction de l'état de la machine 8 de la phase d'enregistrement du vote.

Conditions de dépendances Les valeurs d'état qui permettent de valider ces contraintes sont des valeurs où, en plus de satisfaire les axiomes **axm1** ... **axm22**, doivent remplir également les conditions définies dans l'axiome **axm24** comme suit :

— l'heure de fermeture des bureaux de vote est arrivée :

$$timer = end\_time$$

— aucune signature corrompue n'a été enregistrée :

$$alone\_corrupt\_signatures = \varnothing$$

— aucun choix corrompu assigné à une enveloppe n'a été enregistré :

$$corrupt\_choices\_envelopes = \varnothing$$

— les votes enregistrés correspondent aux signatures correctes qui sont en bijection avec les votes corrects :

$$recorded\_votes \in valid\_signatures \rightarrowtail\!\!\!\!\!\rightarrow valid\_choices$$

— les choix et les signatures sont enregistrés dans les bureaux de vote uniquement lorsque les électeurs qui ont effectué ces choix et émargé ont été enregistrés dans ces bureaux (c-à-d : uniquement lorsqu'ils sont autorisés à voter dans ces bureaux de vote) :

$$\forall\, s, v, h \cdot (s \mapsto v \in recorded\_votes \wedge h \mapsto (s \mapsto v) \in registred\_votes\_offices \Rightarrow \exists\, elec \cdot ((elec \mapsto s) \mapsto h \in votershosting))$$

— les ballots enregistrés correspondent aux enveloppes enregistrées qui sont en bijection avec l'ensemble des bulletins enregistrés ; ceci exprime que les enveloppes ainsi que les bulletins enregistrés sont en même nombre :

$$ballots \in valid\_envelopes \rightarrowtail\!\!\!\!\!\rightarrow recorded\_bulletins$$

— les votes corrects sont en même nombre que les enveloppes enregistrées :

$$correct\_choices\_envelopes \in valid\_choices \rightarrowtail\!\!\!\!\!\rightarrow valid\_envelopes$$

— les bulletins enregistrés contiennent au plus une représentation :

$$bulletins\_representatives \in Representatives \rightarrowtail recorded\_bulletins$$

— un même vote enregistré (choix et signatures) ne peut appartenir à deux bureaux différents :

$$\forall v1, b1, b2 \cdot (b1 \mapsto v1 \in registred\_votes\_offices \\ \wedge b2 \mapsto v1 \in registred\_votes\_offices \Rightarrow b1 = b2)$$

— les choix effectués correctement sont en même nombre que les bulletins enregistrés :

$$choice\_bulletins \in valid\_choices \rightarrowtail\!\!\!\!\!\!\rightarrow recorded\_bulletins$$

— à la fermeture des bureaux de vote, on a la garantie que tous les votants qui ont émargé et exprimé leur choix ont un bulletin qui sera dépouillé ou comptabilisé :

$$\forall elec, sig \cdot (elec \in honest\_voters \wedge sig \in valid\_signatures \wedge elec \mapsto sig \in emargements\_electors\_list \\ \Rightarrow (\exists bullt, v \cdot (bullt \in recorded\_bulletins \\ \wedge bullt \in ran(collected\_bulletins\_representatives) \\ \wedge v \mapsto bullt \in choice\_bulletins \\ \wedge sig \mapsto v \in recorded\_votes)))...$$

et inversement, tous les bulletins qui seront dépouillés ont été émis par les votants qui ont exprimé leur choix et ont émargé :

$$\forall bullt, v \cdot (bullt \in recorded\_bulletins \wedge bullt \in ran(collected\_bulletins\_representatives) \\ \wedge v \mapsto bullt \in choice\_bulletins \Rightarrow \\ (\exists elec, sig \cdot (elec \in honest\_voters \wedge sig \in valid\_signatures \\ \wedge elec \mapsto sig \in emargements\_electors\_list \\ \wedge sig \mapsto v \in recorded\_votes)))...$$

Cette propriété est illustrée dans l'axiome **axm24** donné ci-dessous. Une telle propriété exprime la vérifiabilité sans altération des bulletins, signatures, ou choix et identités des votants car il s'agit à ce niveau de constantes. Ainsi, via le mécanisme de dépendances, les données fournies par le premier composant du vote ($M8\_Recording\_T1$) devraient satisfaire cette propriété avant le dépouillement proprement dit.

$$
\begin{array}{l}
\textbf{axm24} : valide = TRUE \Leftrightarrow \\
\left(\begin{array}{l}
alone\_corrupt\_signatures = \varnothing \wedge corrupt\_choices\_envelopes = \varnothing \\
\wedge recorded\_votes \in valid\_signatures \rightarrowtail\!\!\!\!\!\!\rightarrow valid\_choices \\
\wedge \left(\begin{array}{l} \forall s, v, h \cdot (s \mapsto v \in recorded\_votes \\ \wedge h \mapsto (s \mapsto v) \in registred\_votes\_offices \\ \Rightarrow \exists elec \cdot ((elec \mapsto s) \mapsto h \in votershosting)) \end{array} \right) \\
\wedge ballots \in valid\_envelopes \rightarrowtail\!\!\!\!\!\!\rightarrow recorded\_bulletins \\
\wedge correct\_choice\_envelope \in valid\_choices \rightarrowtail\!\!\!\!\!\!\rightarrow valid\_envelopes \\
\wedge \left(\begin{array}{l} \forall v1, b1, b2 \cdot (b1 \mapsto v1 \in registred\_votes\_offices \\ \wedge b2 \mapsto v1 \in registred\_votes\_offices \Rightarrow b1 = b2) \end{array} \right) \\
\wedge envelopes\_representatives^{-1} \in valid\_envelopes \rightarrowtail\!\!\!\!\!\!\rightarrow Representatives \\
\wedge envelopes\_representatives = bulletins\_representatives; ballots^{-1} \\
\wedge choice\_bulletins \in valid\_choices \rightarrowtail\!\!\!\!\!\!\rightarrow recorded\_bulletins \\
\wedge choice\_bulletins = correct\_choice\_envelope; ballots \\
\wedge choice\_bulletins = \\
correct\_choice\_envelope; envelopes\_representatives^{-1}; collected\_bulletins\_representatives \\
\wedge ran(ballots\_offices) = PollingStation \\
\wedge signed\_bulletins \in recorded\_bulletins \rightarrowtail\!\!\!\!\!\!\rightarrow valid\_signatures \wedge timer = end\_time \\
\wedge \left(\begin{array}{l} \forall elec, sig, h, v \cdot \left(\begin{array}{l} elec \in honest\_voters \wedge sig \in valid\_signatures \\ \wedge elec \mapsto sig \in emargements\_electors\_list \\ \wedge (elec \mapsto sig) \mapsto h \in votershosting \\ \wedge elec \mapsto sig \in honest\_voters\_signatures \\ \wedge sig \mapsto v \in recorded\_votes \\ \Rightarrow \left(\begin{array}{l} \exists bullt, ev \cdot \left(\begin{array}{l} bullt \in recorded\_bulletins \\ \wedge bullt \in ran(collected\_bulletins\_representatives) \\ \wedge v \mapsto bullt \in choice\_bulletins \\ \wedge (\forall vv \cdot vv \neq v \Rightarrow vv \mapsto bullt \notin choice\_bulletins) \\ \wedge ev \in valid\_envelopes \\ \wedge ev \mapsto bullt \in ballots \\ \wedge elec \mapsto bullt \in bulletins\_voters \\ \wedge elec \mapsto ev \in voters\_envelopes \\ \wedge (ev \mapsto bullt) \mapsto h \in ballots\_offices \\ \wedge v \mapsto ev \in correct\_choice\_envelope \\ \wedge bullt \mapsto sig \in signed\_bulletins \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right) \\
\wedge \left(\begin{array}{l} \forall bullt, v, h, ev \cdot \left(\begin{array}{l} bullt \in recorded\_bulletins \\ \wedge bullt \in ran(collected\_bulletins\_representatives) \\ \wedge v \mapsto bullt \in choice\_bulletins \\ \wedge ev \in valid\_envelopes \wedge v \in valid\_choices \\ \wedge ev \mapsto bullt \in ballots \\ \wedge (ev \mapsto bullt) \mapsto h \in ballots\_offices \\ \wedge v \mapsto ev \in correct\_choice\_envelope \\ \Rightarrow \left(\begin{array}{l} \exists elec, sig \cdot \left(\begin{array}{l} elec \in honest\_voters \\ \wedge sig \in valid\_signatures \\ \wedge elec \mapsto sig \in emargements\_electors\_list \\ \wedge sig \mapsto v \in recorded\_votes \\ \wedge elec \mapsto sig \in honest\_voters\_signatures \\ \wedge bullt \mapsto sig \in signed\_bulletins \\ \wedge (elec \mapsto sig) \mapsto h \in votershosting \\ \wedge elec \mapsto bullt \in bulletins\_voters \\ \wedge v \mapsto bullt \in choice\_bulletins \\ \wedge elec \in honest\_voters \end{array} \right) \end{array} \right) \end{array} \right) \end{array} \right) \\
\end{array} \right)
\end{array}
$$

Notons que la vérifiabilité dans le présent contexte B est exprimée sans faire recours à la variable *bulletins_ voters* du premier composant. Le lien entre les votants et leur bulletins n'est pas explicité dans cette propriété i.e., aucun rapprochement n'est fait à ce niveau entre l'électeur et son vote.

Dans le cas où il y aurait corruption, la constante *valide* n'est pas vraie :

axm23 : $dom(recorded\_votes) \neq valid\_signatures \lor ran(recorded\_votes) \neq valid\_choices$
$\lor alone\_corrupt\_signatures \neq \varnothing \lor corrupt\_choices\_envelopes \neq \varnothing$
$\lor corrupt\_envelopes \neq \varnothing \Rightarrow valide = FALSE$

Ces contraintes laissent entendre qu'aucune intrusion n'est tolérée. Pourtant, dans certains systèmes certaines tolérances peuvent exister. Des affaiblissements de ces contraintes peuvent être envisagés en introduisant des seuils d'acceptation comme étant des constantes dans le contexte B, en restreignant les cardinalités de chaque constante qui correspond à une intrusion (*alone_ corrupt_ signatures*, *corrupt_ choices_ envelopes*, ...) par ces seuils. Une telle approche servirai pour l'ajout les primitives de chiffrement probabilistes. Nous donnerons quelques précisions sur ce point en conclusion.

Le contexte Event-B de dépouillement du second type de vote se distingue par l'ajout des constantes issues de la phase du vote et qui indiquent les mentions pour chaque représentant :

axm23 : $registred\_mentions \in (bulletins\_representatives \rightarrow \mathbb{N}_1)$
$\land recorded\_mentions\_by\_office \in registred\_mentions \rightarrow PollingStation$
$\land (\forall\, b, r, m, loc \cdot ((((r \mapsto b) \mapsto m) \mapsto loc) \in recorded\_mentions\_by\_office$
$\Rightarrow (\exists\, e \cdot ((e \mapsto b) \mapsto loc \in ballots\_offices))))...$

Obligation de preuve de dépendance Dans le cas du premier type de vote :
— $timer = end\_time \in \mathcal{T}_1(x_1)$
— $\mathcal{C}_2(s_2, c_2) = axm23 \land axm24$

$$\left.\begin{array}{l} C1\_Recording\_T1(s_1, c_1), \\ Inv(s_1, c_1, x_1), \\ v_1 \subseteq x_1, \\ c_{21} \subseteq c_2, \\ \mathcal{T}_1(x_1) \end{array}\right\} \vdash \mathcal{C}_2(s_2, c_2)(v_1/c_{21})$$

Dasn la réalité, les axiomes axm23 et axm24 doivent être des théorèmes déduits selon les états de la phase source du vote.

4.4.2 Phase de dépouillement

4.4.2.1 Modèle abstrait : phase du dépouillement

La propriété de terminaison stable souhaitée pour la phase de dépouillement d'un protocole de vote est identique pour tous les types de vote, d'où le premier modèle abstrait $M0\_Tallying$ commun aux deux types de vote introduit dans cette phase. Ce modèle décrit le dépouillement via trois événements définis à ce niveau. L'événement *tally*, *finish*

et *maintain*. La seule variable introduite à ce niveau est un booléen qui vérifie seulement si le dépouillement est effectué ou non. L'événement *tally* ne fait qu'observer la valeur de cette variable qui vaut faux dans sa garde, et n'exécute aucune action dans cette machine. C'est dans les raffinements de chacun des types de vote que seront introduites les actions pour dépouiller, et qui sont différentes selon le type de vote. L'événement *finish* observe la valeur du booléen qui vaut faux et la remet à vrai, alors que l'événement *maintain* ne fait qu'observer la valeur de cette variable qui vaudra vrai suite à l'exécution de l'événement *finish* pour éviter un blocage dans le système.

VARIABLES
checked
INVARIANTS
inv : $checked \in BOOL$
INITIALISATION
act : $checked := FALSE$
END
EVENT tally
STATUS anticipated
WHEN
grd : $checked = FALSE$
THEN
act : $checked := FALSE$
END

EVENT finish
WHEN
grd : $checked = FALSE$
THEN
act : $checked := TRUE$
END
EVENT maintain
WHEN
grd : $checked = TRUE$
THEN
act : *skip*
END

4.4.2.2 Premier raffinement : phase de dépouillement

Le dépouillement dans le premier cas qui correspond aux bulletins qui expriment une voix pour une représentation donnée ne peut débuter que si le vote est terminé (voir l'axiome *axm17* dans le contexte vu par la présente machine 4.4.1).

On ne dépouille que les choix corrects. La modification de la variable *checked* dans cet événement ($act1 :\ checked := TRUE$) permet l'activation de l'événement qui comptabilise les votes. Dans cette machine qui raffine la machine $M0\_Tallying$, le dépouillement se fait d'abord par bureau. La variable $correct\_result\_office$ caractérise les scores des représentants par bureau de vote.

$$inv1 :\ correct\_result\_office \subseteq PollingStation \times (Representatives \times \mathbb{N})$$

Dans chaque bureau de vote, les scores de chaque représentant sont uniques :

$$inv2 :\ \forall h, r, x1, x2 \cdot \left(\begin{array}{l} h \in PollingStation \land r \in Representatives \\ \land h \mapsto (r \mapsto x1) \in correct\_result\_office \\ \land h \mapsto (r \mapsto x2) \in correct\_result\_office \end{array} \right) \Rightarrow x1 = x2$$

À l'initialisation, aucun représentant n'a de voix :

INITIALISATION
$act1 : correct\_result\_office := PollingStation \times (Representatives \times \{0\})$

La comptabilisation des choix des votants nécessite les représentants qui sont dans les enveloppes enregistrées (*destroyed_ envelopes_ representatives*), ainsi que les ballots enregistrés par bureau (*destroyed_ ballots_ office*). De plus, il faut connaître les représentants sur les bulletins enregistrés (*destroyed_ bulletins_ representatives*). Ces variables peuvent être vues comme des ***copies*** des constantes définies dans le contexte B vu par la présente machine.

Pour vérifier que tous les bulletins enregistrés ont été correctement dépouillés, une fois le dépouillement achevé, il faut disposer de moyens qui nous garantissent que tous les votants ayant une signature ont un bulletin qui a été décompté, et inversement, que tous les bulletins décomptés correspondent à des choix de votants qui ont effectivement émargé. Nous introduisons à cet effet la variable *counted_ bulletins_ representatives* qui contiendra les bulletins effectivement dépouillés. Ceci nous garantit la vérifiabilité au niveau de la dynamique du système (*inv*8). Cette propriété n'est vraie que dans le cas où les votes ont été enregistrés correctement sans corruption.

$$
\begin{array}{l}
inv3 : destroyed\_ envelopes\_ representatives \subseteq envelopes\_ representatives \\
inv4 : destroyed\_ bulletins\_ representatives \subseteq \\
\qquad (collected\_ bulletins\_ representatives \rhd (ran(valid\_ envelopes \lhd ballots))) \\
inv5 : destroyed\_ ballots\_ office \subseteq ballots\_ offices \\
inv6 : counted\_ bulletins\_ representatives \subseteq \\
\qquad (collected\_ bulletins\_ representatives \rhd (ran(valid\_ envelopes \lhd ballots))) \\
inv7 : \ counted\_ bulletins\_ representatives \cap destroyed\_ bulletins\_ representatives = \varnothing
\end{array}
$$

À l'initialisation, toutes ces variables sont initialisées aux valeurs des constantes leur correspondant, à l'exception de la variable *counted_ bulletins_ representatives* qui est initialisée à l'ensemble vide. La variable *checked* introduite dans le modèle abstrait de cette même phase sera maintenue dans cette machine et celle qui la raffine.

Le dépouillement comptabilise tous les choix enregistrés correctement dans les urnes (*destroyed_ bulletins_ representatives* qui est une copie des choix recueillis dans *collected_ bulletins_ representatives*).

$$
\begin{array}{l}
inv8 : \\
\left(\begin{array}{l}
destroyed\_envelopes\_representatives = \varnothing \\
\quad \wedge\ destroyed\_bulletins\_representatives = \varnothing \\
\quad \wedge\ destroyed\_ballots\_office = \varnothing \\
\quad \wedge\ counted\_bulletins\_representatives = \\
\qquad (collected\_bulletins\_representatives \rhd (ran(valid\_envelopes \lhd ballots))) \\
\quad \wedge\ checked = TRUE \wedge valide = TRUE \\
\qquad \left(\begin{array}{l}
elec \in honest\_voters \wedge sig \in valid\_signatures \\
\wedge elec \mapsto sig \in emargements\_electors\_list \\
\wedge (elec \mapsto sig) \mapsto h \in votershosting \\
\wedge elec \mapsto sig \in honest\_voters\_signatures \\
\wedge sig \mapsto v \in recorded\_votes
\end{array}\right) \\
\Rightarrow \forall\ elec, sig, h, v \cdot \left(\begin{array}{l}
bullt \in (recorded\_bulletins \cap (ran(valid\_envelopes \lhd ballots))) \\
\wedge bullt \in ran(counted\_bulletins\_representatives) \\
\wedge v \mapsto bullt \in choice\_bulletins \\
\wedge (\forall vv \cdot vv \neq v \Rightarrow vv \mapsto bullt \notin choice\_bulletins) \\
\Rightarrow \exists\ bullt, ev \cdot \wedge ev \mapsto bullt \in ballots \\
\wedge elec \mapsto bullt \in bulletins\_voters \\
\wedge (ev \mapsto bullt) \mapsto h \in ballots\_offices \\
\wedge ev \in valid\_envelopes \wedge elec \mapsto ev \in voters\_envelopes \\
\wedge v \mapsto ev \in correct\_choice\_envelope
\end{array}\right)
\end{array}\right) \\
\wedge \\
\left(\begin{array}{l}
destroyed\_envelopes\_representatives = \varnothing \\
\wedge destroyed\_bulletins\_representatives = \varnothing \\
\wedge destroyed\_ballots\_office = \varnothing \wedge \\
counted\_bulletins\_representatives = \\
\qquad (collected\_bulletins\_representatives \rhd (ran(valid\_envelopes \lhd ballots))) \\
\wedge checked = TRUE \wedge valide = TRUE \\
\qquad \left(\begin{array}{l}
bullt \in (recorded\_bulletins \cap (ran(valid\_envelopes \lhd ballots))) \\
\wedge bullt \in ran(counted\_bulletins\_representatives) \\
\wedge v \mapsto bullt \in choice\_bulletins \\
\wedge ev \in valid\_envelopes \\
\wedge v \in valid\_choices \\
\wedge ev \mapsto bullt \in ballots \\
\wedge (ev \mapsto bullt) \mapsto h \in ballots\_offices \\
\wedge v \mapsto ev \in correct\_choice\_envelope
\end{array}\right) \\
\Rightarrow \forall bullt, v, h, ev \cdot \left(\begin{array}{l}
\exists elec, sig \cdot (elec \in honest\_voters \\
\wedge sig \in valid\_signatures \\
\wedge elec \mapsto sig \in emargements\_electors\_list \\
\wedge sig \mapsto v \in recorded\_votes \\
\Rightarrow \wedge elec \mapsto sig \in honest\_voters\_signatures \\
\wedge (elec \mapsto sig) \mapsto h \in votershosting \\
\wedge elec \mapsto bullt \in bulletins\_voters \\
\wedge v \mapsto bullt \in choice\_bulletins \\
\wedge elec \in honest\_voters)
\end{array}\right)
\end{array}\right)
\end{array}
$$

Le dépouillement se fait par l'observation des actions exécutées de l'événement *tally* qui raffine le même événement introduit dans le modèle abstrait. Pour chaque localisation, le dépouillement consiste à ajouter une voix pour chaque représentant assigné à chaque bulletin enregistré (non détruit) dans les ballots stockés dans cette même localisation ou block. L'événement *tally* est gardé par :

$grd1 : checked = FALSE \wedge score \in \mathbb{N}$
$\quad \wedge repr \in Representatives$
$\quad \wedge bullt \in (recorded\_bulletins \cap (ran(valid\_envelopes \lhd ballots)))$
$grd2 : h \in PollingStation \wedge (repr \mapsto score) \in ran(\{h\} \lhd correct\_result\_office)$
$grd3 : (env \mapsto bullt) \mapsto h \in destroyed\_ballots\_office$
$grd4 : repr \mapsto env \in destroyed\_envelopes\_representatives$
$grd5 : repr \mapsto bullt \in destroyed\_bulletins\_representatives$
$grd6 : repr \mapsto bullt \notin counted\_bulletins\_representatives$

avec *bullt , env, repr, h* et *score* sont les paramétres de l'événement *tally*.

L'action qui consiste à ajouter une voix pour un représentant incrémente le score de ce dernier en utilisant la variable *correct_ result_ office* :

$act1 : \ correct\_result\_office := (correct\_result\_office \setminus \{h \mapsto (repr \mapsto score)\}) \cup \{h \mapsto (repr \mapsto score + 1)\}$

Cet ajout est accompagné de la destruction des ***copies*** de l'enveloppe, du bulletin liés à ce ballot enregistré dans le même bureau. Le bulletin détruit est considéré comme dépouillé (*act5*) :

$act2 : destroyed\_envelopes\_representatives := destroyed\_envelopes\_representatives \setminus \{repr \mapsto env\}$
$act3 : destroyed\_bulletins\_representatives := destroyed\_bulletins\_representatives \setminus \{repr \mapsto bullt\}$
$act4 : destroyed\_ballots\_office := destroyed\_ballots\_office \setminus \{(env \mapsto bullt) \mapsto h\}$
$act5 : counted\_bulletins\_representatives := counted\_bulletins\_representatives \cup \{repr \mapsto bullt\}$

La variable *checked* est un booléen initialisé à faux qui affirme que le dépouillement peut continuer tant qu'il existe des bulletins non encore dépouillés. Cette propriété est traduite par l'expression définie dans le variant de cette machine. Ainsi, la preuve de convergence dans cette machine est effectuée en décrémentant le variant défini comme suit :

VARIANT
$(collected\_bulletins\_representatives \rhd (ran(valid\_envelopes \lhd ballots))) \setminus counted\_bulletins\_representatives$

L'événement *tally* est alors un événement convergent qui lorsque le variant vaudra 0, cardinalité de *collected_ bulletins_ representatives* $\setminus$ *counted_ bulletins_ representatives*, il ne sera plus observable.

Les deux autres événements introduits dans le modèle abstrait sont raffinés en gardant inchangé l'événement *maintain*, et en ajoutant la garde :

$grd : collected\_bulletins\_representatives = counted\_bulletins\_representatives$

à l'événement *finish*. Cet événement *finish* remet la valeur de la variable *checked* à vrai dès que l'expression du variant s'annule.

Dans cette phase, la vérifiabilité est exprimée à la fois au niveau du contexte Event-B dépendant et au niveau de la machine qui voit ce contexte.

D'autres propriétés sont aussi exprimées à ce niveau. Par exemple, pour illustrer le fait que s'il existe des ballots corrompus, alors ces derniers ne sont pas dépouillés. Typiquement, ce type de propriétés est issu du domaine et est déduit selon le contexte, puisqu'il permet de raisonner et de justifier les preuves réalisées. Nous détaillerons ce point en section 4.6.

Preuve de stabilité de la phase du dépouillement

Les preuves de stabilité sont effectuées en suivant la démarche donnée pour la première phase d'enregistrement, avec les mêmes hypothèses d'équité sur l'événement convergent (voir section 4.3.1.1). La propriété de terminaison souhaitée est définie comme suit :

$$\begin{aligned}\mathcal{T}_2 \;\widehat{=}\; & collected\_bulletins\_representatives = counted\_bulletins\_representatives \\ & \wedge checked = TRUE\end{aligned}$$

Nous posons :
$x_2 = \{checked, correct\_result\_office, destroyed\_envelopes\_representatives,$
$destroyed\_bulletins\_representatives, destroyed\_ballots\_office,$
$counted\_bulletins\_representatives, global\_result\}$.

Et nous devons démontrer :

$$\Phi_{\mathcal{M}_2} \widehat{=} INIT_2(x_2) \rightsquigarrow \mathcal{T}_2$$

Qui signifie qu'on atteindra à partir de l'initialisation un état où l'ensemble des bulletins dépouillés sera égal à l'ensemble des bulletins recueillis et où la valeur du booléen vaudra vrai. Les hypothèses d'équité posées sur la présente machine sont :

$$L_{\mathcal{M}_2} \widehat{=} WF_{x_2}(tally) \wedge WF_{x_2}(finish)$$

$$\begin{aligned}NEXT_2 \;\widehat{=}\; & \vee BA(finish)(x_2, x_2') \\ & \vee BA(tally)(x_2, x_2') \\ & \vee BA(maintain)(x_2, x_2') \\ & \vee (x_2 = x_2')\end{aligned}$$

$$\begin{aligned}INIT_2(x_2) \;\widehat{=}\; & br \in collected\_bulletins\_representatives \\ & \wedge\, br \notin counted\_bulletins\_representatives \\ & checked = FALSE\end{aligned}$$

Soient

$$\begin{aligned}P_1 \;\widehat{=}\; & br \in collected\_bulletins\_representatives \\ & \wedge\, br \in counted\_bulletins\_representatives \\ & checked = FALSE\end{aligned}$$

$INIT_2(x_2) \rightsquigarrow P_1$ signifie qu'un bulletin qui n'a pas encore été dépouillé finira fatalement par l'être. Nous posons :
$D(n) = card(ran(collected\_bulletins\_representatives) \backslash ran(counted\_bulletins\_representatives)) = n$. La propriété de terminaison $\Phi_{\mathcal{M}_2}$ peut être exprimée comme suit :

$$\Phi_{\mathcal{M}_2} \widehat{=} \{D(n) \rightsquigarrow D(n-1), (\exists n.D(n) \rightsquigarrow D(0))\}$$

La propriété P_1 exprime qu'à partir d'un état où *checked* est faux on atteindra fatalement un état où la différence entre les bulletins recueillis et les bulletins non dépouillés diminue d'un pas de 1 ($D(n) \rightsquigarrow D(n-1)$). La deuxième partie stipule que cette même différence convergera fatalement vers une valeur zéro, cardinalité d'un ensemble vide ($\varnothing$). Nous avons :

$$P_1 \wedge [NEXT_2]_{x_2} \Rightarrow (P_1')$$

Si la constante *collected_ bulletins_ representatives* n'est pas vide, nous avons le scénario suivant : L'hypothèse d'équité faible sur l'événement *tally* nous permet d'affirmer que $D(n) \rightsquigarrow D(n-1)$, car l'événement *tally* fait décroître le variant. En posant $D_1 \widehat{=} D(n)$ et $D_2 \widehat{=} D(n-1)$, on a :

— $D_1 \wedge [NEXT]_{x_2} \Rightarrow (D_1' \vee D_2')$
— $D_1 \wedge \langle NEXT \wedge tally \rangle_{x_2} \Rightarrow D_2'$
— $D_1 \Rightarrow ENABLED\langle tally \rangle_{x_2}$

La règle WF1 nous donne :
$\Box[NEXT]_{x_2} \wedge WF_{x_2}(tally) \Rightarrow (D_1 \rightsquigarrow D_2)$. En utilisant la règle LATTICE sur l'ensemble bien fondé des entiers naturels $\mathbb{N}$, on démontre que $\exists n.D(n) \rightsquigarrow D(0)$ converge vers 0.

Nous posons **(f)** :
$P_1 \widehat{=} D_0 \wedge checked = FALSE$. L'hypothèse d'équité faible sur l'événement $finish$ nous permet de déduire que de
— $P_1 \wedge [NEXT]_{x_2} \Rightarrow (P_1 \vee \mathcal{T}_2')$
— $P_1 \wedge \langle NEXT \wedge finish \rangle_{x_2} \Rightarrow \mathcal{T}_2'$
— $P_1 \Rightarrow ENABLED\langle finish \rangle_{x_2}$

On a $\Phi_{\mathcal{M}_2} \widehat{=} INIT_2(x_2) \rightsquigarrow \mathcal{T}_2$. Cette propriété reste vraie, car l'événement qui reste activable ne modifie aucunemenet les variables *counted_ bulletins_ representatives* et *checked*, d'où la persistance. Et l'on déduit : $\mathcal{S}pec(\mathcal{M}_2) \vdash \Phi_{\mathcal{M}_2}$.

Dans le cas où la constante *collected_ bulletins_ representatives* est vide alors on reprend le même raisonnement à partir de **(f)**.

Les preuves sont identiques pour le raffinement de la présente machine puisqu'on n'introduit aucun événement nouveau.

Notons que le fait d'exprimer la progression dans le système par le biais d'événement convergent et de variant, permet également d'établir la propriété de *Pas de résultat partiel*.

4.4.2.3 Deuxième raffinement : phase de dépouillement

Finalement, le score de chaque représentant correspond au nombre de voix globales (somme des voix par bureau). Ce raffinement introduit le calcul global des voix de chaque représentant stocké dans la variable *global_ result*.

$inv1 : global\_result \in Representatives \rightarrow \mathbb{N}$
$inv2 : \forall\, h, r, x, x1 \cdot (h \in PollingStation \wedge r \in Representatives \wedge$
$\quad (r \mapsto x) \in global\_result \wedge$
$\quad h \mapsto (r \mapsto x1) \in correct\_result\_office \Rightarrow x1 \leq x)$

À l'initialisation, aucun représentant n'a de voix

INITIALISATION
$act1 : global\_result := Representatives \times \{0\}$

On ajoute ainsi à l'événement qui permet le dépouillement l'action qui consiste à incrémenter le nombre de voix globales de chaque représentant.

$$act1 : global\_result(repr) := global\_result(repr) + 1$$

Le dépouillement dans le second cas de vote est réalisé en enregistrant pour chaque bulletin les scores pour tous les représentants par bureau, puis globalement. m est la mention indiquée sur les bulletins qui correspond à un représentant donné.

$act1 : correct\_result\_office := correct\_result\_office \Leftarrow (\{h\} \times \{repr \mapsto score + m$
$\quad | h \mapsto (repr \mapsto score) \in correct\_result\_office \wedge repr \mapsto bullt \mapsto m \in registred\_mentions\})$
$act5 : global\_result := \{repr \mapsto score + m$
$\quad | repr \mapsto score \in global\_result \wedge repr \mapsto bullt \mapsto m \in registred\_mentions\}$

4.4.2.4 Bilan des preuves

Les modèles conçus ont généré 1309 obligations de preuve, parmi lesquelles 752 ont été déchargées de manière non-automatique dû en grande partie à la présence du quantificateur universel dans les formules et les invariants définis. La plus grande partie des obligations viennent des deux dernières machines pour les deux types de vote.

		Total des POs	POs Automatiques	POs Interactives
Voting Phase	Abstract Model $\mathcal{M}_1$	7	6 (85.7%)	1 (14.3%)
	1st refinement $\mathcal{M}_2$	2	1 (50.0%)	1 (50.0%)
	2nd refinement $\mathcal{M}_3$	62	47 (75.8%)	15 (24.2%)
	3th refinement $\mathcal{M}_4$	78	42 (50.6%)	36 (49.4%)
	4th refinement $\mathcal{M}_5$	62	46 (77.4%)	16 (22.9%)
	5th refinement $\mathcal{M}_6$	57	32 (56.1%)	25 (43.9%)
	6th refinement $\mathcal{M}_7$	196	100 (51.0%)	96 (49.0%)
	7th refinement $\mathcal{M}_8\_T_1$	352	113 (32.3%)	239 (67.7%)
	7th refinement $\mathcal{M}_8\_T_2$	433	124 (28.8%)	309 (71.2%)
Tallying Phase	Abstract Model $\mathcal{M}_1$	0	0	0
	1st refinement $\mathcal{M}_1\_T_1$	19	14 (76.2%)	5 (23.8%)
	2nd refinement $\mathcal{M}_2\_T_1$	7	6 (85.7%)	1 (14.3%)
	1st refinement $\mathcal{M}_1\_T_2$	17	12 (73.7%)	5 (26.3%)
	2nd refinement $\mathcal{M}_2\_T_2$	6	3 (50.0%)	3 (50.0%)
	Total	1309	560 (42.5%)	752 (57.5%)

4.5 Extensions possibles des patrons des protocoles de vote

L'objectif final est l'obtention d'un réel système de vote ayant toutes les caractéristiques souhaitées pour son déploiement. Le développement donné ici est orienté but, où l'action est notre première préoccupation dans le système à modéliser. Cette modélisation est encore loin d'être complète et exhaustive. L'ajout de spécificités revient à introduire d'autres

raffinements. Parmi les exigences les plus importantes pour la conception d'un système de vote sûr et certifiable, nous retrouvons les hypothèses de chaque produit, les contraintes sur le types de vote pour lequel le système sera déployé, l'ajout des actifs, l'ajout des dates pour enregistrer l'heure précise à laquelle un vote a été émis. Le développement de tout système de vote nécessite aussi une phase de préparation du vote, les mécanismes cryptographiques, tels que les signatures en aveulge (*Blind signature* en anglais), les mixes nets, ainsi que les schémas de chiffrement probabilistes sont indispensables pour la construction de ces systèmes. Ce point sera discuté en conclusion.

Autres types de vote : Deux grandes familles de vote se dégagent de notre développement : Le vote qui consiste à comptabiliser un bulletin pour une voix et le vote qui extrait les mentions indiquées sur chaque bulletin de vote afin d'en faire le calcul des voix attribuées à chaque représentation. Pourtant, cette connaissance (vote majoritaire, vote à préférence, ...) reste implicite dans notre développement. Il s'ensuit que l'interprétation des résultats n'est pas prise en compte dans nos modèles (la décision à prendre ressort des autorités chargées d'anoncer les résultats). D'un point de vue sémantique, il est important de préciser le mode de scrutin afin de prendre une décision. Une telle interprétation dépend alors du contexte dans lequel la preuve est faite. Notamment lorsque le vote requiert plusieurs tours à l'issue desquels les résultats fournis durant un premier tour seront l'entrée du tour suivant. Ce traitement correspond à une nouvelle *configuration initiale* de la phase de préparation du vote dans notre modélisation. Ce point répond à la question qui concerne l'impact du contexte sur la signification des résultats fournis par notre développement en terme de gagnants, de classement des candidats, etc.

Il faut noter que notre spécification indique les mentions qui sont des entiers qui peuvent être des préférences ou des poids. Mais, il est possible de définir des mentions spécifiques comme étant un ensemble dans un contexte Event-B, par exemple les mentions : très bien, bien, mauvais, etc. Ces contraintes peuvent être ajoutées dans un contexte Event-B étendant le contexte $C1\_Recording\_T2$. Cette précision engendre le renforcement des événements pour effectuer le choix des votants. Une redéfinition des deux variables *correct_ result_ office* et *global_ result* peut être comme suit :

$$correct\_result\_office \subseteq PollingStation \times (Representatives \times Mentions)$$
$$global\_result \in (Representatives \times Mentions) \rightarrow \mathbb{N}$$

si l'on considère $Mentions$ l'ensemble des mentions utilisées.

Les acteurs : Toute entité susceptible d'effectuer une action dans le système doit être ajoutée. Une analyse plus profonde de la sécurité des systèmes de vote électronique repose aussi sur l'identification des *acteurs* pour répondre aux questions sur ce que le système est conçu pour protéger, et de qui [199]. Cela comprend, par exemple, la spécification des acteurs qui participent au processus et leurs responsabilités, les actifs et les opérations à leur égard, ainsi que les contraintes du modèle. Spécifier les acteurs (et leurs responsabilités) dans le modèle permet non seulement de décrire qui fait quoi lors de l'exécution d'un processus, mais surtout dans le contexte du processus de modélisation dans le domaine l'administration publique (Public Administration (PA) en anglais), qui gère quelles données et avec quels privilèges. À titre indicatif, les parties comme les serveurs, les agents électoraux, autorité d'identification et ayants droit, etc, doivent être prises dans

la spécification. Notre modélisation n'explicite que les électeurs. Pour cibler un système particulier, il serait donc nécessaire d'intégrer les différents acteurs. Considérons le protocole Belenios [83]. Pour la spécification de ce protocole, les parties telles que : le serveur de vote, l'administrateur du serveur, l'autorité d'accréditation et les adminitrateurs légaux de confiance doivent être ajoutées à notre développement. Ce traitement peut être réalisé en définissant un ensemble *Actors* dans le contexte Event-B de la phase d'enregistrement des votes et tous ces acteurs seront des constantes incluses dans cet ensemble. L'ensemble des électeurs devient également une constante incluse dans l'ensemble des acteurs avec la contrainte supplémentaire qui est l'intersection vide entre toutes ces constantes.

Les hypothèses : Chaque système et protocole de vote est basé sur ses propres hypothèses de construction et de preuve. Par exemple, Belenios [83] est un protocole basé sur la version Helios [84] avec *credentials* étendue aux signatures et à la preuve de connaissance à divulgation nulle de connaissance (*zero-knowledge proofs*), Dans ce système, les auteurs supposent que les tableaux d'affichage et les autorités responsables d'enregistrer les signatures ne puevent être malhonnêtes simultanément, et ainsi, prouver la correction du protocole de vote est faite sous ces hypothèses de confiance. Cette spécificité peut s'exprimer dans notre développement en : supprimant l'événement consistant à ajouter des signatures et des bulletins de vote simultanément (*corrupt_ choices_ signatures_ simultaneously*) ; et en renforçant les gardes dans les événements qui portent sur l'ajout ou la suppression de signatures et de bourrages des urnes de vote séparément. Par exemple, l'événement *stuffing_ choices* doit être gardé par $grd : alone\_ corrupt\_ signatures = \varnothing$, alors que l'événement *corrupt_ signatures_ only* doit être gardé par la garde $grd : alone\_ corrupt\_ choices = \varnothing$. L'ajout des deux invariants suivants est nécessaire :

$$inv : alone\_ corrupt\_ signatures \neq \varnothing \Rightarrow alone\_ corrupt\_ choices = \varnothing$$
$$inv : alone\_ corrupt\_ choices \neq \varnothing \Rightarrow alone\_ corrupt\_ signatures = \varnothing$$

Les dates : La précision d'horloge et des dates est aussi une exigence importante à considérer dans les spécifications. Notre développement définit une abstraction de ce que peut être réellement le temps dans un système de vote représenté par un simple compteur qui ne décrit finalement que l'heure du système. Pour préciser cette exigence, les modèles Event-B peuvent être étendus en ajoutant dans un contexte Event-B les constantes définies dans notre exemple sur les réservations des billets d'avion et de location de voiture donné au quatrième raffinement 3.4.4.4 en section 3.4 du chapitre 3. L'ajout de deux autres variables dans les machines Event-B est également indispensable pour représenter les minutes, les secondes pour modifier le temps dynamiquement tel que donné dans [34]. Les séquences d'interruption (*interrupt_ sequences*) décrites au niveau du modèle abstrait 4.3.1.1 doivent être raffinées et modifiées pour faire des comparaisons en utilisant la constante *convertdatetonumber* définie dans le contexte Event-B décrit au quatrième raffinement 3.4.4.4 en section 3.4 du chapitre 3. Ainsi, les actions pour enregistrer les choix des votants sont modifiées pour tenir compte de ces modifications en redéfinissant les variables impliquées par exemple, *valid_ votes valid_ choices* ... , et introduisant les invariants de collage si nécessaire. Notons qu'il faut aussi ajouter des éléments pour décrire les jours, les mois de l'année du déroulement du vote de la même manière que ceux pour l'horloge du temps. Une différence notable ici est que dans ce dernier cas, les valeurs

considérées sont les jours de la semaine, et les noms des mois de l'année et non des entiers comme c'est le cas de l'horloge du temps.

La phase de décision : La preuve d'existence du résultat du vote conforme aux attentes des concepteurs et justifable est aussi de montrer qu'il n'y a pas de perte dans les résultats fournis. Ainsi, l'*intégrité* des résultats est assurée en montrant que ces derniers n'ont pas été modifiés. Nous donnons ici une façon de vérifier, via le mécanisme de dépendances entre la phase du dépouillement et la phase finale qui correspond à la décision qui peut être prise, que les axiomes définis dans un contexte Event-B dépendant de la phase du dépouillement soient validés ou non. Cette étape permet donc d'aider à la décision finale des autorités responsables de confirmer le résultat final d'un vote.

Pour exprimer qu'il n'y a pas de perte dans le système avant et après le dépouillement, ou que les résultats sont conservés, nous introduisons un contexte Event-B *Decision_ T2* pour le deuxième type de vote qui *étend* le contexte *C0_ Tallying_ T2* et *dépend* de la machine *M1_ Tallying_ T2*. Ainsi, les variables définies dans la machine *M1_ Tallying_ T2* deviennent des constantes dans le contexte Event-B *Decision_ T2* (*destroyed_ envelopes_ representatives*, *checked*, *destroyed_ bulletins_ representatives*, *destroyed_ ballots_ office*, *global_ result*, *counted_ bulletins_ representatives*). Pour chaque représentation, il faut avoir une bijection entre le résultat qui correspond à ce représentant, ainsi que les résultats inscrits sur les bulletins dépouillés.

$$
\begin{array}{l}
axm13: \ destroyed\_envelopes\_representatives = \varnothing \wedge \\
\quad destroyed\_bulletins\_representatives = \varnothing \wedge \\
\quad destroyed\_ballots\_office = \varnothing \wedge \\
\quad counted\_bulletins\_representatives = collected\_bulletins\_representatives \wedge \\
\quad collected\_bulletins\_representatives \neq \varnothing \wedge checked = TRUE \\
\Rightarrow \left(\forall repr, n \cdot \left(\begin{array}{l} global\_result(repr) = n \wedge \\ finite(counted\_bulletins\_representatives[\{repr\}]) \\ \quad \Rightarrow \left(\exists f \cdot \left(\begin{array}{l} (f \in 1 \mathinner{..} n \rightarrowtail\!\!\!\!\!\rightarrow counted\_bulletins\_representatives[\{repr\}]) \\ \vee (f \in 0 \mathinner{..} 0 \rightarrowtail\!\!\!\!\!\rightarrow \varnothing \wedge n = 0) \end{array} \right) \right) \end{array} \right) \right)
\end{array}
$$

Une autre façon d'exprimer la conservation du résultat peut être réalisée avec une fonction pour sommer les résultats au lieu des bijections comme suit :

$$
\begin{array}{l}
axm1: \ sum \in \mathbb{P}(Bulletins) \rightarrow \mathbb{N} \\
axm2: \ \forall so, t \cdot (t \in Bulletins \wedge so \in \mathbb{P}(Bulletins) \wedge t \in so \Rightarrow sum(so) = 1 + sum(so \setminus \{t\})) \\
axm3: \ (\varnothing \mapsto 0) \in sum \\
axm12: \ destroyed\_envelopes\_representatives = \varnothing \\
\qquad \wedge \ destroyed\_bulletins\_representatives = \varnothing \\
\qquad \wedge \ destroyed\_ballots\_office = \varnothing \\
\qquad \wedge \ checked = TRUE \wedge \\
\qquad counted\_bulletins\_representatives = collected\_bulletins\_representatives \\
\qquad \Rightarrow \left(\begin{array}{l} \forall repr, n \cdot (global\_result(repr) = n \\ \wedge finite(counted\_bulletins\_representatives[\{repr\}]) \\ \Rightarrow (n = sum(counted\_bulletins\_representatives[\{repr\}]))) \end{array} \right)
\end{array}
$$

Le même traitement peut être appliqué au premier type de vote : définir le contexte Event-B *Decision_ T1* qui étend le contexte *C0_ Tallying_ T1* et qui *dépend* de la machine *M1_ Tallying_ T1*.

Notons enfin qu'il est aussi possible de s'abstraire des enveloppes dans d'ultérieurs raffinements pour les substituer de valeurs binaires qui peuvent indiquer seulement qu'un bulletin est soit crypté ou non, ce qui correspond au fait qu'un bulletin est soit dans une enveloppe ou non.

4.5.1 Composition des modèles des protocoles de vote et le modèle de l'attaquant Dolev-Yao

La sécurité des systèmes de vote repose essentiellement sur des procédés cryptographiques. La cryptographie est par conséquent, une importante exigence à considérer dans la conception d'un système de vote et particulièrement pour l'anonymat des votes. De nombreux mécanismes cryptographiques ont été mis en œuvre dans la littérature pour le chiffrement et l'authentification. Les cryptosystèmes sont toujours définis par quatre ou trois algorithmes, dont l'algorithme de chiffrement est purement probabiliste. Les probabilités traitent à la fois des événements qui se produisent dans des espaces échantillons dénombrables ou continus, ou bien des ensembles finis dénombrables ou continus de l'espace d'échantillons liés à l'ensemble de tous les résultats possibles. Bien qu'il existe des extensions pour prendre les calculs probabilistes dans le formalisme Event-B (citons à titre indicatif [195]), celles-ci sont liées aux probabilités que les événements dans les machines soient observables ou non. Les primitives et les mécanismes de chiffrement probabilistes utilisent le second type des probabilités car elles portent sur les valuations (estimations) des données. Ces traitements nécessitent des théories pour les calculs probabilistes et logarithmiques. Ce plug-in n'étant pas en marche, nous ne traitons donc pas cette problèmatique dans ce manuscrit et nous donnerons quelques pistes prometteuses en perspectives.

Les méthodes de modélisation des protocoles cryptographiques pour construire les primitives cryptographiques (chiffrement, signatures, fonctions de hachage, échange de clefs) sont basées : soit sur le modèle formel de Dolev-Yao [106], qu'est un modèle dédié pour les protocoles en cascade, soit sur les modèles calculatoires, où l'attaquant dans ce cas est une machine de Turing. Dans cette partie, nous montrons comment combiner nos modèles pour les protocole de vote avec le modèle de l'attaquant Dolev-Yao [106] développé précédemment par N. Benaissa dans ses travaux [33]. Ses travaux portent sur les propriétés d'authentification, et d'établissement des clés [35, 36] associés à la connaissance de l'attaquant, ainsi que sur ses comportements conformes au modèle de Dolev-Yao [106] développé dans [33].

Les modèles d'authentication peuvent être réutilisés en tant qu'entrée fournie à partir de la phase de préparation du vote, pour la phase d'enregistrement des votes comme conséquence de l'application du mécanisme de dépendance. Ce traitement nécessite un contexte Event-B étendu par le contexte Event-B $C0\_Recording$ défini dans nos patrons, pour reprendre les modèles de Benaissa.

Bien que défini pour les protocoles cryptographiques en cascade, ce modèle [33] peut également être appliqué à notre phase d'enregistement du vote avec la combinaison de la propriété d'équilibre (*balanced* en anglais) [106] d'un protocole sécurisé exprimée par les relations de *reduction* et de fonctions *normal* appliquées aux données *Messages* dans le modèle de Benaissa [33].

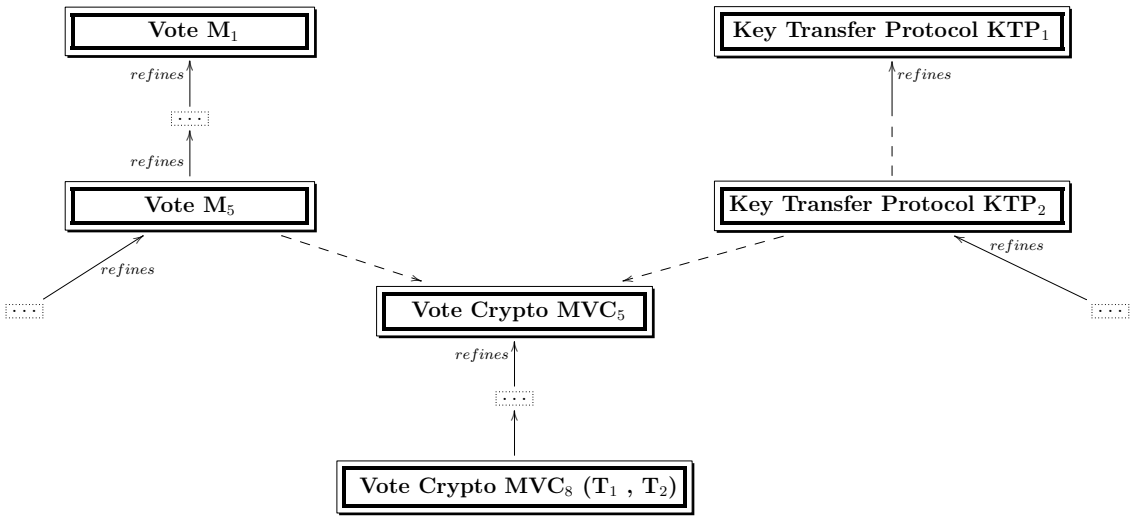

FIGURE 4.3 – Composition des patrons de vote avec le modèle de l'attaquant

La fonction de réduction réduit les séquences de clefs en supprimant de ces séquences celles qui enchaînent une clef de chiffrement d'un agent suivie immédiatement d'une clef de déchiffrement de ce même agent, et inversement, les séquences qui enchaînent une clef de déchiffrement d'un agent suivie immédiatement d'une clef de chiffrement de ce même agent. La séquence des deux clefs (de chiffrement/de déchiffrement, déchiffrement/chiffrement) est retirée de la séquence des clefs. La réduction est aussi appliquée itérativement. Quand la réduction n'est plus applicable, la séquence est dite de *forme normale* (*normal form* en anglais). Quand la composition de toutes les clefs de cryptage et de décryptage contenues dans la séquence est égale à la fonction d'identité cela signifie que l'on peut obtenir le texte M (remplacé par les (*Passwords*) ainsi que les (*Credentials*)) à partir de la séquence des chiffrés des agents impliqués dans le protocole. Cela veut dire que la fonction normale est égale à l'ensemble vide, auquel cas le message devient en clair. Il faut donc montrer que cette séquence n'est jamais égale à l'ensemble vide. Pour qu'un attaquant ne puisse pas avoir les messages en clair, le protocole doit être équilibré (*balanced*). Cette propriété est montrée en exprimant que pour toute séquence et pour tout agent ayant une combinaison de clefs, si la clefs de déchiffrement est dans la séquence de symboles alors la clef de chiffrement de ce même agent est aussi dans la séquence des symboles.

La composition du modèle de Benaissa avec nos patrons s'entreprend en remplaçant et en instanciant à chaque fois ces données *Messages* par les concepts définis dans notre développement, à savoir : les mots de passe, les credentials, les bulletins, les choix, ..., à l'exception des données *Electeurs* qui doivent rester publiques. Cette composition donnerait lieu à un modèle volumineux et des obligations de preuve assez conséquentes. Il suffit de l'appliquer aux données qui concernent les mots de passe (*Passwords*) ainsi que les (*Credentials*) dans notre développement, car c'est à partir de ces connaissances qu'un attaquant peut avoir des accès que ce soit pour établir des connexions, pour voter et même pour connaître les votes des autres. Les messages M dans la description ci-dessus peuvent être remplacés par les deux concepts (*Passwords*) ainsi que (*Credentials*). Le sous-ensemble des intrus est, quant à lui, instancié pour n'avoir qu'un attaquant en ajoutant un axiome dans le premier contexte de la phase d'enregistrement du vote. Ainsi, la composition doit être appliquée au niveau du quatrième raffinement (comme le montre la figure 4.3), où sont introduits les événements pour établir des connexions, et en introduisant tous les événements définis pour l'attaquant dolev-Yao dans le modèle de Benaissa.

Rappelons que le nombre d'agents qui entrent en échange dans ce type de protocoles est de deux, et qu'un attaquant dans ce modèle tente d'intercepter donc les messages ou les données envoyés entre ces deux agents. L'ensemble des électeurs correspond à l'ensemble des agents dans le développement de Benaissa.

4.6 Exploitation des connaissances du domaine pour concevoir des systèmes de vote

L'association avec une ontologie de domaine dans le cas des systèmes de vote est une tâche difficile. De plus il n'existe pas de travaux assez conséquent dans ce domaine, et la construction d'ontologies nécessite un temps assez conséquent par les experts d'un domaine. Toutefois l'expertise acquise nous a permis d'établir quelques bases et confirmer ainsi nos orientations. Nous montrons dans cette section comment il est possible d'exploiter les ontologies d'un domaine, leur conceptualisation ainsi que leur raisonnement pour faire évoluer un système de vote à la fois sur le plan de la vérification et de la validation de modèles Event-B. Nous résumons l'ensemble des transformations entre les deux structures, à savoir, les modèles Event-B et la base de connaissances comme suit :

— Les prédicats unaires : ensembles/constantes/variables se transforment en concepts, sous-concepts ontologiques :

$$T^{-1}(honest\_voters \subseteq Electors) = honest\_voters \sqsubseteq Electors$$

— Les prédicats binaires constantes/variables se transforment en propriétés sémantiques (rôles ontologiques) :

$$T^{-1}(corrupt\_signatures\_voters \in Electors \nrightarrow Signatures) = \begin{cases} \top \sqsubseteq \leq 1\ corrupt\_signatures\_voters \\ \geq 1\ corrupt\_signatures\_voters \sqsubseteq Electors \\ \top \sqsubseteq \forall corrupt\_signatures\_voters.Signatures \end{cases}$$

— Tout prédicat n-aire peut être transformé en prédicats binaires et unaires qui le définissent ;
— Autres transformations :

$$T^{-1}(valid\_choices \cap alone\_corrupt\_choices = \varnothing) =$$
$$(valid\_choices \sqcap alone\_corrupt\_choices = \bot)$$

Sur le plan de la vérification, on prendra deux exemples pour expliquer l'intégration au niveau des contextes Event-B et au niveau des machines Event-B, et cela pour deux raffinements différents pour montrer l'intérêt de la granularité de l'ontologie. Pour la validation, nous donnerons l'exemple sur les contextes Event-B que nous avons pu confirmer avec l'animateur utilisé pour animer nos modèles Event-B.

4.6.1 Extraction pour la vérification

Pour ce qui concerne les concepts et les relations ontologiques, nous pouvons se référer à l'étude de cas définie dans le chapitre précédent, où nous avons montré que les concepts et les relations ontologiques peuvent être extraits soit dans les contextes Event-B pour ce qui concerne les propriétés statiques dans Event-B, soit pour les variables et par conséquent les

états définis dans les machines, ceci en respectant le degré de granularité de la connaissance ontologique. Dans le cas d'une extraction pour la partie statique, il s'agit de propriétés sémantiques intrinsèques, alors que dans le cas d'une extraction pour la partie dynamique, il s'agit de propriétés extrinsèques qui concernent en général les propriétés de *dépendances* d'un concept : dépendent d'autres propriétés pour qu'elles soient vérifiées. Par exemple,

$$deleted\_votes \subseteq Choices$$

pour qu'une propriété du sous-concept *deleted_votes* (choix supprimé) soit vérifiée, il faut que le choix ait été effectué et que le votant le connaisse, ... Ces propriétés permettent de structurer les états dans les machines suivant le *concept intentionnel* défini par l'action.

Pour ce cas d'extraction, où il s'agit seulement de récupérer ces concepts et ces relations ontologiques, il s'agit d'une extraction par induction. À titre indicatif, dans l'étude de cas des systèmes de vote, nous pouvons citer tous les ensembles définis dans le premier contexte de la phase d'enregistement du vote ($C0\_Recording$). Citons à titre indicatif, les ensembles *Signatures*, *Envelopes*, *Representatives*, etc. Rappelons que ces ensembles sont définis comme étant des concepts ontologiques de haut niveau, que l'extraction est définie soit comme ensembles, soit comme constantes incluses dans l'unique ensemble *Thing*. C'est au concepteur de faire son choix.

Dans le cas d'une extraction par déduction, et si l'on admet que les données sont fournies dans la composante assertionnelle de l'ontologie, alors il s'agit de manière générale d'extraire des propriétés qui sont *nécessairement vérifiées par déduction* suivant les données contenues dans l'ontologie. On peut donner dans ce cas l'exemple des propriétés telles que le fait d'avoir des électeurs ayant le droit de voter, alors on déduit nécessairement que leur age est supérieur ou égal à une valeur donnée imposée selon les pays et les lois. Ceci correspond donc à des ajouts de constantes et des axiomes dans le contexte Event-B $C0\_Recording$. Plus précisément, la connexion ou l'association avec l'ontologie fait que les propriétés ainsi que les contraintes sont affirmées. Ceci est particulièrement important dans le cas où les modèles sont destinés pour être certifiés. La déduction consiste donc à ajouter les constantes suivantes selon les transformations données dans les annexes : $votersage$, qui est définie comme attribut dans l'ontologie de l'ensemble (du concept) des électeurs, comme suit :

$$axm:\ votersage \in Electors \rightarrow \mathbb{N}_1$$

de la constante de l'age minimum $minimage$, et la contrainte suivante :

$$thm:\ \forall elec.elec \in Electors \Rightarrow votersage(elec) \geq minimage$$

Cette propriété consiste en une *identification*, une *classification* puis une déduction sur l'ensemble des individus de la composante assertionnelle. Ici nous montrons seulement le principe de manière informelle, pour une formalisation complète, nous donnerons en perspectives de cette thèse comment l'extraction peut être effectuée.

Nous considérons donc le cas où les propriétés exprimées dans notre système comme étant une possibilité parmi toutes celles qu'un domaine de ce système pourrait nous présenter. Nous interrogeons donc la base de connaissances sur ce qu'elle connaît réellement, c-à-d, les faits ou les objets réels qu'elle contient. Cette extraction doit s'accompagner des propriétés qui impliquent strictement notre théorème. Dans ce cas précis, il s'agit de

la constante pour l'age *minimage* et de la fonction qui définit l'age de chaque électeur *votersage*. Ces deux éléments s'appellent *liens*, alors que la description de tous les objets qui doivent remplir le théorème s'appelle *trait caractéristique* (ou features en anglais). Il faut donc que la *description* de tous les objets dans la composante assertionnelle remplisse ce théorème auquel cas, nous pourrions requantifier universellement sur l'ensemble des électeurs pour obtenir ce théorème. Dans le cas où il existe des objets qui ne satisfont pas cette propriété, alors ce théorème n'est pas validé et est donc réfuté. Cette démarche doit s'appuyer sur les définitions des liens et traits caractéristiques de l'ACL [118] que nous donnerons en perspectives de nos travaux. En particulier, puisque les classifications ne sont pas vraies dans l'absolu, il faut donc changer de logique, en utilisant le pré-ordre défini dans la logique contextualisée de l'ACL [118].

Typiquement, dans le cas du vote, et pour cet exemple de l'age ainsi que pour toute propriété intrinsèque aux électeurs, nous pouvons aussi appliquer le principe des dépendances, où finalement ces constantes sont construites durant la phase de préparation du vote avant de débuter le vote proprement dit. Cela signifie que ce théorème a été construit et nous avons la garantie de sa bonne construction par les machines de la phase de préparation dont le contexte $C0\_Recording$ dépend. Ce qui constitue une richesse pour le formalisme Event-B.

En résumé :

1. par preuve d'existence :

$$\begin{array}{rl} \textbf{axm :}\ minimage \in \mathbb{N}_1 & /*\, lien\, */ \\ \textbf{axm :}\ votersage \in Electors \rightarrow \mathbb{N}_1 & /*\, lien\, */ \\ A\text{-}Box & /*\, \text{extension} \neq \varnothing\ (\text{feat})\, */ \\ \Downarrow & //\text{déduction selon contexte, si vraie pour tous les objets électeurs} \end{array}$$

$$\textbf{thm :}\ \forall elec.elec \in Electors \Rightarrow votersage(elec) \geq minimage$$

2. par dépendance : puisque **thm** est déduit par construction selon les états de la machine source (phase de préparation du vote)

$$\Longrightarrow constante :\ \ Electors \subseteq Citizen \subseteq \ldots Thing$$

Notons que feat est l'abréviation de *Features* en anglais qui correspond aux traits caractéristiques.

Dans le cas d'une machine, on reprend l'exemple de la phase de dépouillement du premier type de vote donnée par la machine $M0\_Tallying\_T1$. Nous avons montré que la vérifiabilité exprimée par l'invariant $inv8$ dans cette machine pour ce type de vote comme étant une propriété qui vérifie que pour tous les bulletins enregistrés ont été correctement dépouillés, une fois le dépouillement achevé, en garantissant que tous les votants ayant une signature ont un bulletin qui a été décompté, et inversement, que tous les bulletins décomptés correspondent à des choix de votants qui ont effectivement émargé correctement.

Pour justifier maintenant qu'à la fin du vote, s'il existe des votes corrompus, alors ceux-ci ne sont pas dépouillés. Ceci est exprimé par le théorème $thm8$ ci-dessous. Il s'agit bien d'une déduction ontologique selon le principe donné dans le chapitre précédent. L'implication correspond à la déduction en supposant qu'on a les données et donc le contexte est précisé (les faits sont donnés), et par conséquent l'extraction est dans ce cas

est possible. Ceci est justifié dans le chapitre précédent puisque l'extraction n'est possible qu'en présence d'objets finis.

$$
\begin{array}{l}
thm8 : \ destroyed\_envelopes\_representatives = \varnothing \\
\quad \wedge destroyed\_bulletins\_representatives = \varnothing \\
\quad \wedge destroyed\_ballots\_office = \varnothing \\
\quad \wedge counted\_bulletins\_representatives = \\
\qquad\qquad (collected\_bulletins\_representatives \rhd (ran(valid\_envelopes \lhd ballots))) \\
\quad \wedge checked = TRUE \wedge valide = FALSE \\
\qquad\qquad \left(\begin{array}{l} elec \notin honest\_voters \wedge sig \in alone\_corrupt\_signatures \\ \wedge sig \mapsto v \in recorded\_votes \end{array} \right. \\
\quad \Rightarrow \forall elec, sig, v \cdot \left(\begin{array}{l} \Rightarrow \forall bullt, ev \cdot \left(\begin{array}{l} v \mapsto bullt \in choice\_bulletins \\ \wedge ev \mapsto bullt \in ballots \wedge \\ elec \mapsto bullt \in corrupt\_bulletins\_voters \\ \wedge ev \in corrupt\_envelopes \\ \wedge elec \mapsto ev \in dishonest\_envelope \\ \wedge v \mapsto ev \in corrupt\_choices\_envelopes \\ \wedge (\exists repr \cdot (repr \mapsto bullt \in collected\_bulletins\_representatives)) \\ \Rightarrow bullt \in (recorded\_bulletins \cap (ran(corrupt\_envelopes \lhd ballots))) \\ \wedge bullt \notin ran(counted\_bulletins\_representatives) \end{array} \right) \end{array} \right)
\end{array}
$$

Pour ce qui concerne le cas d'une induction dans la partie dynamique dans Event-B, on peut donner l'exemple au raffinement $M8\_Recording\_T2$. Ce cas concerne les relations n-aires. Nous avons montré qu'il n'était pas possible d'avoir les preuves des variables définies pour les mentions indiquées pour chaque représentant sur les bulletins données par :

INVARIANTS

$voters\_mentions \in (Electors \times bulletins\_representatives) \pfun \mathbb{N}$

$registred\_mentions \in (ran(dom(voters\_mentions)) \pfun ran(voters\_mentions)) \pfun PollingStation$

C'est la raison pour laquelle nous avons opéré des changements afin de réussir des preuves. Il s'agit d'être en mesure d'avoir toutes les relations qui définissent des faits ontologiques. Une extraction par intégration de l'ontologie des éléments qui subsument les relations n-aires est donc utile dans ce cas par exemple. D'où la définition données par les invariants $inv10$ et $inv11$ ainsi que les autres invariants définies au raffinement $M8\_Recording\_T2$.

4.6.2 Extraction pour la validation

Si les extractions données ci-dessus peuvent suffir pour la vérification, elles le sont moins pour la validation. Nous présentons dans ce cas l'exemple que nous avons pu tester avec l'animateur ProB intégré dans la plateforme Rodin. Nous avons donc appliqué une approche holistique des définitions des constantes que ce soit pour les prédicats n-aires car l'animateur ne produit pas de valeurs pour animer des constantes définies par des relations n-aires, ou pour avoir des relations qui lient tous les objets pour définir des faits. Il faut donc appliquer l'intégration des constantes qui entrent dans la définition de chaque prédicat n-aires tel que c'est le cas dans la machine $M8\_Recording\_T2$ en reproduisant

ces mêmes éléments, mais définis comme étant des constantes dans le contexte dépendant de cette machine.

Un autre cas où l'extraction à partir d'une ontologie peut aider pour l'animation et donc la validation est celui des constantes définissant des relations. Par exemple, si l'on prend l'exemple du contexte $C0\_Tallying\_T1$, où nous avons donné les constantes définies au niveau de la relation de dépendances entre le contexte $C0\_Tallying\_T1$ et la machine $M8\_Recording\_T1$. Pour que l'animation soit valide il faut intégrer toutes les relations des domaines dans leur co-domaine. Par exemple, pour la relation *recorded_ votes* définie par l'axiome $axm2$ dans ce contexte par $recorded\_votes \in Signatures \leftrightarrow Choices$, ainsi que la relation *choice_ bulletins* définie par l'axiome $axm11$ $choice\_bulletins \in Choices \nrightarrow Bulletins$. Pour avoir tous les objets en relation pour constituer des faits ontologiquement parlant, il faut alors ajouter la relation définie de l'ensemble des bulletins dans l'ensemble des signatures, puisqu'on a systématiquement des bulletins liés à des signatures. Ceci permettrai de n'avoir que des instances de propriétés qui sont réellement liées par ces trois propriétés ontologiques. L'animateur dans ce cas nous a fourni des valeurs correctes pour lier ces instances de ces trois relations. Nous avons donc injecté ces propriétés au fur et à mesure dans les contextes dépendants et dans les machines sources de ces derniers (machines de la phase du vote).

Notons enfin que cette approche a été appliquée pour l'ensemble des modèles Event-B définis dans ce chapitre. Les séquences d'interruption ne permettent pas d'animer les modèles réalisés, nous avons donc créé deux versions, une avec les séquences d'interruption, et l'autre sans ces dernières pour animer tous les modèles.

4.7 Conclusion

Ce chapitre a présenté une application réelle de nos propositions sur la contextualisation et les dépendances en Event-B. Nous avons montré qu'il était possible d'exprimer de nombreuses propriétés des systèmes de vote qu'elles soient liées à la phase de vote ou de dépouillement, de sûreté ou des propriétés liées au temps. C'est grâce aux mécanismes d'abstraction et de raffinement que présente le formalisme Event-B ainsi qu'à sa capacité expressive qui utilise la logique du premier ordre ainsi que la théorie des ensembles auxquels s'ajoute sa structure en contextes et machines. Les différentes approches de modélisation qui existent dans la littérature sont orientées agent, orientées processus ou orientées connaissances. Nous avons montré comment il est possible de considérer à la fois les aspects comportements et les aspect liés au protocoles qui peuvent être utilisés, et où agent, processus, et connaissances sont représentés par des entités modélisées par des ensembles, constantes et variables dans les modèles Event-B. Mais, l'utilisation de protocoles nécessite des calculs probabilistes non pris en compte par le formalisme Event-B. Nous présentons en conclusion de nos travaux quelques éléments utiles pour intégrer les primitives de chiffrement probabilistes à nos patrons de conception. Par ailleurs, il faut dire que le mécanisme de dépendance facilite considérablement la tâche de modélisation. En particulier lorsqu'il s'agit de modéliser des systèmes complexes où de nombreuses contraintes et notions doivent être prises en compte. La seule condition est que les phases modélisées doivent être caractérisées par la stabilité pour pouvoir exploiter les constantes dans les modèles dépendants.

L'exploitation des ontologies en modélisation suivant le triangle sémiotique nous a per-

mis d'établir des liens entre ses trois dimensions, à savoir, la dimension synatxique définie par le modèle référent Event-B, la dimension sémantique donnée par le domaine ainsi que la dimension pragmatique définie par les aspects fonctionnels des modèles prescrivant ainsi les comportements du système en question. L'application aux système de vote est une étude de cas pertinente qui donne les différentes visions que l'on peut avoir et la manière dont il serait possible de les intégrer (inductivement en fonction du domaine et déductivement selon le contexte). Cette intégration nécessite de combiner les mécanismes d'identification et de classification des raisonnements ontologiques pour une meilleure exploitation partiellement automatisée au niveau des modèles Event-B par raffinement, permettant ainsi de juger la pertinence des niveaux de raffinement qu'elle soit pour la vérification (complètude vs. intérêt défini par les propriétés à vérifier), ou pour la validation (complètude vs. fidélité de la satisfaction par rapport aux données rélles). Cette partie correspond à la définition de vues logiques non définies au niveau des raisonneurs ontologiques basés sur les logiques de description. Ce traitement requiert la définition d'un nouveau formalisme. Nous avons jugé intéressant d'exploiter l'analyse de concepts formelle logique. Ces formalismes sont réputés par leur pertinence pour la certification de programmes, de modèles, etc. Nous présentons dans le chapitre suivant en perspectives de nos travaux les grandes lignes qui expliquent cette intégration.

Chapitre 5

Conclusion et perspectives

« On ne fait jamais attention à ce qui à été fait,
on ne voit que ce qui reste à faire. »
- Marie Curie.

« Vu de trop près, le monde perd ses reliefs et la vie ses perspectives. »
- Antonine Maillet

5.1 Bilan des travaux réalisés

Cette thèse a traité les problématiques liées à l'intégration de connaissances du domaine dans la conception et la modélisation, pour la vérification et la validation des systèmes. Le formalisme de modélisation utilisé dans cette thèse, à savoir le langage Event-B est suffisamment expressif de part sa logique et son mécanisme de raffinement qui permet d'abstraire les détails selon des vues pertinentes à considérer. Il possède aussi l'avantage de prendre en compte la preuve de propriétés liées au temps au moyen des traces d'exécution permettant ainsi d'inclure la dimension temporelle définie en utilisant les opérateurs temporels à la TLA. Mais il est toujours difficile de rendre compte de toute notion liée au temps, au fonctionnement et aux comportements qui peuvent évoluer dans le temps en fonction des besoins et des situations nouvelles.

Un système, selon l'hypothèse sémiotique est toujours lié à son domaine d'application pour se conformer à sa réalité. La vision constructiviste d'un système s'attache toujours à sa réalité décrite dans son domaine. Nous avons montré l'utilité d'exploiter des ontologies de domaine comme formalisme associé aux modèles Event-B par raffinement. Ceci nous a permis de juger de la pertinence des détails intégrés au niveau des modèles Event-B dans le but de les justifier et ainsi de les certifier. Le but est de définir des stratégies standardisées pouvant être exploitées dans les activités de vérification et de validation. La vérification par rapport à un objectif donné doit être aussi complète au regard des comportements et des propriétés souhaités. La validation au regard d'un objectif donné doit être la plus fidèle possible au système qu'elle représente.

Dans cette thèse, nous avons étudié les différentes sources qui peuvent conduire à des conflits et à des ambiguïtés, et par conséquent rendent complexe la tâche de vérification

et de validation lors de la conception d'un système. Il apparaît clairement que le contexte est la première problématique auquel les concepteurs sont confrontés. Nous avons étudié en particulier les différentes visions qui peuvent exister et les aspects pour juger de la pertinence des représentations logiques pour un système pour être fidèle à sa réalité. Les formalismes de représentation logiques sont un moyen puissant et rationnel qui peut être exploité dans différents problèmes liés à différentes contraintes, à différents niveaux dans les phases de conception d'un système.

Nous avons établi différentes stratégies d'intégration des connaissances liées au contexte et au domaine pour être exploitées par raffinement dans la modélisation en Event-B. Un nouveau mécanisme de dépendance inspiré des logiques qui interprètent le contexte comme un ensemble de situations a été défini. Cette dépendance est conditionnée par la stabilité des phases impliquées dans la modélisation. La stabilité d'une phase est une propriété liée à la temporalité et est donc définie sur les traces d'exécution d'un modèle Event-B. L'objectif étant de faciliter la tâche de modélisation, et de mieux structurer les modèles Event-B. Ce mécanisme a nécessité la définition d'une nouvelle obligation de preuve pour montrer que les contraintes statiques qu'on appelle axiomes en Event-B sont paramétrées par les états de la phase source. Nous avons montré l'applicabilité de cette approche sur des applications réelles telles que les systèmes de vote, où de nombreuses exigences et contraintes doivent être respectées. Nous avons pu exprimer différentes propriétés à différents niveaux dans la modélisation telles que la stabilité, la vérifiabilité, le vote unique, l'éligibilité, etc.

La vérification au moyen du formalisme Event-B n'est réalisée que par raffinement. Elle n'est jamais évaluée dans l'absolu, mais toujours par rapport aux comportements et aux propriétés introduits à un niveau de raffinement donné. Par ailleurs, les connaissances d'un domaine sont définies par une relation de généralisation/spécialisation selon la granularité d'observation pour rendre compte de la réalité d'un domaine. De même, la validation n'est jamais évaluée dans l'absolu, mais toujours au regard d'un objectif d'utilisation (des configurations et des scénarios). Sa validité est donc jugée selon sa fidélité à la réalité qu'elle représente. Nous avons donc jugé pertinent que ce soit en fonction de ces paramètres que l'intégration de connaissances soit réalisée. Plusieurs problèmes et défis sont relevés :

— Premièrement, la logique du premier ordre qu'utilise Event-B est plus expressive que les logiques de description ontologiques. Elle intégre les quantificateurs existentiel et universel, etc ;
— Deuxièmement, les raisonneurs ontologiques définis pour les logiques de description sont monotones et appliquent toutes les règles du raisonneur à la fois, ce qui n'est pas le cas en Event-B, où les connaissances sont intégrées par raffinement et les prédicats avant-après permettent d'exprimer le changement d'état dans le système. Pourtant le raisonnement ontologique peut être exploité pour des vérifications de cohérence par exemple, par les mécanismes de classification et d'identification ;
— Troisièmement, il faut être en capacité de distinguer le contexte, par rapport au domaine et au système ;

Nous avons montré que le raffinement est un mécanisme qui contextualise les systèmes par l'intégration de contraintes et des hypothèses a posteriori dans la tâche de modélisation.

Nous avons donc défini des stratégies pour une intégration par raffinement de connaissances issues du domaine :

— inductivement extraites selon le domaine. Ce cas consiste à considérer les concepts et les relations ontologiques subsumant les éléments définis au niveau du modèle

Event-B considéré. Cette approche exploite les mécanismes d'identification et de classification automatiques des raisonneurs ontologiques pour pouvoir construire des propriétés Event-B contenant les quantificateurs existentiel et universel et vérifier ainsi la conformité et la cohérence de la propriété intégrée dans les modèles Event-B ;
— déductivement extraites selon le contexte de preuve au niveau du modèle Event-B considéré ;
— selon la granularité et la hiérarchie des connaissances de l'ontologie pour une extraction nécessaire et suffisante au regard du raffinement Event-B en question.

5.2 Perspectives

5.2.1 Sur le plan des dépendances

Le mécanisme de dépendance défini dans cette thèse présente une approche de construction de modèles Event-B dépendants pour des systèmes où il est vérifié, qu'à la stabilité du composant source, ses situations ou ses états valident les contraintes statiques déjà vérifées du composant dépendant. Nous avons exploré d'autres études de cas où ce mécanisme peut être appliqué. En effet, les dépendances peuvent être exploitées pour modéliser des systèmes régis par la contrainte de stabilité des phases modélisées dans ce système. On peut citer les algorithmes distribués où aucun événement ne peut être déclenché. La stabilité est détectée par des configurations terminales, où la vérification de la terminaison est réalisée par des algorithmes de vagues.

Ce mécanisme définit explicitement les machines dans le composant source. Il s'agit seulement d'un paramétrage des axiomes dépendants par les états de la première phase dans la modélisation. Ce mécanisme répond à la question suivante : Existe-t-il des situations dans le modèle Event-B source qui valident des axiomes dans le modèle dépendant ? Cette problématique ne correspond pas au fait de construire une machine Event-B à partir d'axiomes dans le modèle dépendant. En effet, cette question relève d'un autre problème différent et ne correspond pas au principe défini dans cette thèse. Il s'agit de deux problèmes différents. Dans ce cas, il s'agit de considérer, à partir de contraintes statiques ou d'axiomes, d'être en mesure de construire des machines Event-B qui ont permis de valider ces axiomes. Ce problème est un problème difficile, mais qui peut être intéressant à explorer. Cette étude pourrait apporter des réponses aux problèmes rencontrés en cryptographie [124] par exemple. Ces travaux montrent qu'il était possible d'enfreindre la sécurité du chiffrement ElGamel, où les nombres premiers paramètres du système sont des constantes qui vérifient certaines propriétés, peuvent être dérobés par des personnes malveillantes leur permettant ainsi de casser le logarithme discret (le chiffrement ElGamal). Ces travaux montrent que cette attaque est bien réelle, en fabriquant un nombre premier pour lequel un calcul de logarithme discret a été effectué par le biais d'une trapdoor a priori indétectable. Ce travail a montré l'utilité d'avoir ces paramètres premier avec une *garantie de bonne construction traçable* dans les standards. Il ne suffit plus alors de garantir que ces constantes aient des propriétés vérifiées, mais il s'agit également de garantir leur bonne construction.

Si l'on projette ce problème en modélisation Event-B, il s'agit bien des dépendances définies, où l'on construit une preuve de bonne construction dans des machines Event-B des constantes dans les contextes Event-B dépendants. Mais, si l'on souhaite explo-

rer des pistes pour savoir comment un attaquant pourrait être en mesure de construire ces constantes, ceci revient à exprimer le fait d'être en capacité pour montrer comment construire des machines Event-B qui permettent de construire des constantes d'un autre modèle Event-B. Dans ce cas, il serait alors possible d'avoir un moyen de construire une machine Event-B (un système de transition) pouvant générer des constantes. Ce qui permettrait de comprendre alors comment les attaques peuvent être gérées telles que c'est le cas dans l'exemple ci-dessus. Pouvoir explorer cette piste afin de découvrir dans quel cas on pourrait construire une machine Event-B à partir de contextes, ou axiomes Event-B serait utile pour pouvoir gérer ce type de problèmes. Le cas des dépendances est alors nécessairement inclus dans ce problème.

D'autres exemples pour les schémas de chiffrement peuvent être modélisés en appliquant le mécanisme de dépendance. Citons à titre indicatif, les cryptosystèmes définis par la donnée de trois ou quatre algorithmes : un premier algorithme (`Setup(`λ`)`) prend en entrée le paramètre probabiliste λ pour générer en sortie les paramètres (`params`) publics communs du système ; un deuxième algorithme (`KeyGen(`λ`)`) qui prend les paramètres (`params`) en entrée, puis fournit en sortie des paires clefs publiques/clefs privées ; un troisième algorithme (`Encrypt`) probabiliste qui a comme entrée les clefs publiques généres par le précédent algorithme, ainsi que d'autres paramètres tels que les messages dans l'ensemble des messages, pour fournir en sortie un chiffré associé au message en entrée ; un quatrième algorithme (`Decrypt`) qui déchiffre le chiffré en entrée de cet algorithme et génère le message en clair.

Toutefois, la modélisation de ce type d'algorithmes requiert des propriétés spécifiques pour les calculs probabilistes pour montrer qu'un attaquant ne parvient pas à deviner le bit aléatoire avec une probabilité significative plus grande qu'une valeur donnée. Le formalisme Event-B modélise des systèmes discrets et ne permet d'exprimer ce type de propriétés qu'en définissant des théories au moyen du plugin dédié à cette tâche [5]. En effet, ce plugin permet de définir des opérateurs, de nouvelles définitions de types de données et des définitions axiomatiques. Les définitions d'opérateur peuvent être des opérateurs d'expression tel que card, des opérateurs de prédicats. Les extensions de types de données peuvent être utilisées pour définir des types de données énumérés, ainsi que des types de données inductifs. Les définitions axiomatiques peuvent être utilisées pour définir de nouveaux types de données tels que les nombres réels. Il s'agit donc de développer des bibliothèques de ces définitions afin de les exploiter pour des problèmes similaires. Il serait alors possible d'instancier les opérateurs définis dans des théories pour le calcul probabiliste, dans les contextes et les machines Event-B pour exprimer des propriétés définies pour ce type de données.

5.2.2 Sur le plan de l'extraction par intégration

L'idée développée dans cette thèse pour une extraction partiellement automatisée des propriétés du domaine nous a semblé intéressante pour les raisons qui suivent :

— Au niveau de la conception d'un système, la capacité à suivre le cycle de vie des logiciels afin de garantir leur traçabilité est une étape indispensable, en particulier en ingénierie des exigences dirigées par les modèles [172]. Cette exigence doit garantir des mesures dans lesquelles chaque élément d'un produit de développement logiciel en établissant leur *raison d'être*.

— Par ailleurs, une bonne méthodologie devrait aider à rassembler les fonctionnalités en ligne de produits en identifiant les comportements typiques qu'il serait possible de réunir dans l'objectif de l'adapter en fonction des spécifications. Des transformations en différents niveaux de granularité doivent être appliquées pour une personnalisation du produit selon les attentes des clients. L'assemblage de modèles dans les lignes de produits assure le regroupement des fonctionnalités attribuées aux mêmes versions tout en séparant celles ne possèdant aucune propriété en commun. Aussi, ces méthodes permettent d'identifier les différentes configurations pour aider la prise de décision à des fins de refactorisation, et par conséquent une réutilisation du produit.

Les travaux dans la littérature ont montré qu'en plus d'être adaptées aux contraintes de variabilités des lignes de produits dans l'objectif de procéder au maintien et la ré-ingénierie des systèmes conçus, l'analyse formelle de concepts est aussi le meilleur formalisme pour la réconciliation des mécanismes utilisés pour des fins de recherche et d'extraction de connaissances [118].

— En effet, l'analyse formelle de concepts paraît une bonne méthode pour gérer les variabilités dans les systèmes, car elle peut être automatisée même pour des problèmes de taille considérable à l'instar des protocoles de vote que nous avons étudiés. Ses structures possèdent d'intéressantes propriétés pour identifier, selon la vue considérée, les ensembles qui partagent les *intensions* et *extensions* dans leur totalité. Cette propriété est déterminante pour regrouper des fonctionnalités, puisqu'elle assure qu'aucun produit, ni aucune fonctionnalité n'a été omis. Les *concepts maximum* et *minimum* dans ces structures identifient des éléments communs à tous les produits et au contraire présents dans aucun. Ces formalismes peuvent par conséquent être intéressants à exploiter.
— De plus, l'organisation des *connaissances factuelles* en contexte, qui associe à chaque fait une propriété définie par un attribut à un objet est le principe de structuration dans l'analyse formelle de concepts. La possibilité d'extraire un ensemble de concepts d'un contexte, fait des mécanismes d'extraction de propriétés ou des faits une bonne approche qui s'accorde bien avec les différentes approches d'extraction, à savoir la navigation et l'interrogation, grâce à la dualité qui existe entre extension et intension des concepts qui rassemble les notions d'endroit et de requête.

Néanmoins, les descriptions dans l'analyse formelle de concepts se limitent à des ensembles d'attributs non précisés. Des variantes telles que [118] ont été proposées pour remplacer cet ensemble d'attributs par des formules de logique presque arbitraire dans l'objectif de traiter des applications de nature diverses. Ce cadre peut être utilisé dans le génie logiciel car il possède des critères du monde système tels que : des mécanismes de raisonnement décidables ; des résultats exacts et reproductibles ; et il adopte l'hypothèse du monde clos. Ces variantes sont fondées sur l'analyse formelle de concepts en proposant une extension logique appelée ACL (pour analyse logique de concepts) afin de construire une structure de navigation automatique qu'on appelle *treillis de concepts*, où les concepts jouent à la fois le rôle de répertoire et de requête. La *navigation* et l'*interrogation* sont les principaux paradigmes de recherche d'information dans ces travaux faisant l'usage de logique pour la description des *requêtes* et des *liens* de navigation. La logique dans cette variante est un paramètre qui peut être instancié à n'importe quelle logique (propositionnelle, des prédicats, ...), ce qui fait d'elle un formalisme *générique* au sens de la logique utilisée. L'expression et la découverte de connaissances sont réalisées par des mécanismes développés pour l'interrogation et la navigation via des *liens* et des *traits ca-*

ractéristiques[26], favorisant la restriction de l'espace de navigation, et pouvant restreindre le langage des requêtes en définissant des *vues logiques* qui constituent un dialogue homme-machine fondé sur le treillis de concepts. Un tel dialogue permet de retrouver des objets lorsqu'il s'agit de naviguer dans la structure du treillis, et de découvrir des régularités entre les objets en extrayant des connaissances. Nous préconison l'utilisation de cette extension pour l'assemblage des formules de la logique des prédicats et la logique de descriptions utilisée dans cette thèse pour former un nouveau formalisme. Ceci implique la définition du langage de requêtes pour ce nouveau formalisme logique tout en exploitant les propriétés d'*induction* et de *déduction* dans l'analyse formelle de concepts pour faire de l'extraction de propriétés dans le but d'aider à la fois la vérification et la validation des modèles Event-B. Habituellement, en logique nous utilisons la règle du modus ponens pour ces déductions. Dans l'analyse logique de concepts, le mécanisme de déduction par navigation permet en plus de déduire des propriétés sur les modèles, mais aussi des faits qui peuvent être exploités dans le cas de la validation. Dans le cas de l'interrogation ce sont donc les objets qui seront extraits.

Le raffinement est au coeur du formalisme Event-B qui définit des solutions d'intégration *correct-par-construction* tout en favorisant la réconciliation des différents points de vue des différents acteurs et parties prenantes dans le système lui procurant ainsi une flexibilité de conception. Guider la recherche de propriétés par raffinement lui donne une troisième dimension orthogonale plus avantageuse. La recherche de propriétés dans l'analyse logique de concepts se fait par la définition de *liens* maximaux au sens de la subsomption qui sont restreints par des *features* et éventuellement des *vues* dans les *diagrammes de Hasse* en combinant les approches définies par Linding [180] et Godin [132] pour l'analyse formelle de concepts. Nous avons pu établir des correspondances entre le mécanisme de raffinement de modèles en Event-B et les vues ainsi que les liens de navigations définis dans l'analyse logique de concepts dans le but de restreindre l'espace de recherche et de n'extraire que les propriétés pertinentes et intelligibles situées à chaque niveau de raffinement. Les *liens* et les *features* permettent de restreindre l'extension (ou les faits) courante au raffinement n, alors que les vues déterminent et restreignent l'espace de recherche des *liens* de navigation et le domaine des traits, garantissant ainsi une plus forte traçabilité qui facilite à la fois la compréhension des artefacts, mais aussi en offrant un degré de confiance plus élevé par une automatisation des liens sémantiques définis. Il faut alors définir la sémantique de ce langage et ses répercussions sur les obligations de preuve en Event-B.

Nous avons défini dans cette thèse le contexte de preuve comme étant les connaissances minimales qu'un système de preuve devrait avoir afin de fournir une sémantique aux preuves établies dans un système de preuve pour le formalisme Event-B. Cette sémantique permet donc de fournir à la fois, une interprétation valide dans ce même contexte aux modèles développés, mais aussi de déduire de nouvelles connaissances à partir des preuves effectuées. L'inférence de nouvelles connaissances à partir de ce qui est explicité a son importance dans la mesure où ce procédé engendre de nouvelles propriétés ou des faits qui affirment bien la cohérence des preuves réalisées, puisque de manière générale, le contexte peut être utilisé soit en *agrégation* ou comme *interprétation*. L'agrégation consiste en la composition des connaissances du contexte brut. L'interprétation quant à elle, applique des abstractions de données du contexte à des fins de visibilité.

Plusieurs techniques sont utilisées dans la littérature pour raisonner et faire de l'inférence sur le contexte suivant les formalismes de sa représentation. Nous soutenons le point

26. appelés *features* en Anglais

de vue des auteurs [75, 255] qui postulent que la combinaison des techniques d'apprentissage machine et les ontologies peut être une technique intéressante à appliquer. Les deux paradigmes de recherche et d'extraction de connaissances sur lesquels il faut se baser, à savoir l'*interrogation* et la *navigation* peuvent être considérés comme des techniques de conception qui visent à utiliser les connaissances et les théories existantes pour construire de meilleures solutions aux problèmes liés à la représentation et l'extraction de connaissances pour améliorer certaines situations dans la conception des systèmes, en particulier, pour avoir des preuves plus automatiques. Le raisonnement sur le contexte par *agrégation* est représenté par l'ajout de nouvelles propriétés privilégiant ainsi le mécanisme de *navigation*, alors que le raisonnement par *interprétation* consiste à faire des substitutions par *interrogation* et donc la donnée des faits ou des *extensions* ou des objets satisfaisant les propriétés décrivant ces objets. Il serait alors possible de définir une connexion de Galois en utilisant le domaine concrétisé par une ontologie et les assertions définies dans les modèles Event-B. Ceci est justifié par le fait qu'à chaque raffinement des obligations de preuve sont ajoutées et par conséquent, les traces d'exécution sont restreintes.

Notons enfin que l'association d'une ontologie aux modèles Event-B pour extraire des propriétés n'établit qu'une correspondance et fournit des suggestions pour le concepteur au niveau des modèles Event-B. Ces suggestions sont construites en calculant les éléments (concepts et propriétés sémantiques) qui subsument les éléments définis dans les modèles Event-B au raffinement en question. Une telle approche donne plus de choix qui peut être exploité pertinemment par un concepteur en ne selectionnant que les éléments qui sont susceptibles de l'intéresser. Les requêtes définies dans l'analyse logique de concepts correspondent alors aux obligations de preuve définies au niveau des modèles Event-B. Les vues logiques ACL sont définies par les ensembles, les constantes, les variables au niveau du raffinement considéré.

Nous avons identifié les axes suivants pour construire ce formalisme :

— Premièrement, il est indispensable de définir la structure de l'association entre ontologie et modèles Event-B en se basant sur les stratégies définies dans cette thèse et les correspondances définies dans l'annexe B. La connexion de Galois sera établie par l'ensemble des formules issues des modèles Event-B et les formules définies dans l'ontologie en utilissant les vues logiques à chaque niveau de raffinement. L'ensemble donné des individus dans l'ontologie jouera alors le rôle d'extension définie dans les contextes de l'analyse logique de concepts. La logique de l'analyse logique de concepts correspond dans ce cas à la logique contextualisée définie dans le chapitre de l'état de l'art, où le contenu donné dans la composante assertionnelle d'une ontologie définit le contenu de la structure définie par la logique contextualisée. L'ontologie dans ce cas n'est qu'un paramètre du langage des formules logiques. Lorsqu'elle est utilisée dans l'ontologie, la relation de subsomption $\sqsubseteq_K$ (cf. La subsomption contextualisée utilisée dans la définition 5.2.2 en sous-section 1.9.3 du chapitre 1) correspond à l'inclusion des constantes dans les ensembles, ou bien des variables dans les ensembles ou les constantes, selon les définitions données dans Event-B ; lorsqu'elle est utilisée pour définir la structure d'un contexte au sens de l'analyse logique de concepts, elle correspond à l'implication entre les formules logiques. La construction d'un modèle consistant repose sur la preuve des obligations de preuve qui doivent être déchagrées par les prouveurs intégrés dans Rodin. Il faut alors opérer des traitements sur l'ensemble des obligations de preuve et les traces d'exécution d'un modèle Event-B. On définit alors un modèle comme référent au sens de l'hypothèse sémiotique donnée dans cette thèse. Les obligations de preuve

définissent les intensions des contextes de l'analyse logique de concepts, alors que les traces d'exécution définissent les extensions des contextes au sens de l'analyse logique de concepts. Le raffinement entre les modèles Event-B conserve les propriétés déjà prouvées au niveau des modèles les plus abstraits dans la chaîne de raffinement. Toutes les propriétés définies dans les modèles Event-B sont redéfinies en ajoutant les axiomes terminologiques définis dans une ontologies de domaine $\mathcal{D}$. Par conséquent, les obligations de preuve sont aussi étendues pour tenir compte de ces axiomes terminologiques ;

— Deuxièmement, il faut définir l'extraction ou l'évolution de cette structure en utilisant les techniques d'extraction définies pour l'analyse logique de concepts. L'évolution et les mises-à-jour s'effectuent en appliquant les règles de mise-à-jour données par l'opérateur $\diamond$ défini dans l'analyse logique de concepts. Il s'agit de techniques d'aprentissage machine (*machine learning* en anglais) Ceci reviendrait à un calcul du point fixe. Cette approche a été également exploitée par Benaissa dans ses travaux [34]. La différence est que dans le cas d'une extraction le calcul du point fixe se fait en association avec l'ontologie et par conséquent, ses propriétés ainsi que son contenu. L'extraction peut aussi concerner les propriétés incompatibles et complémentaires ;

Les vues logiques correspondent au raisonnement a posteriori, tel que les vues définies en bases de données, ou les vues définies dans le langage OntoQL [157]. Mais dans ces derniers cas les modèles utilisés sont des modèles relationnels. Puisque les formalismes utilisés dans nos travaux sont des formalismes logiques, les vues que nous cherchons sont donc des vues logiques telles que définies dans l'ACL.

L'intégration des connaissances du domaine pour la conception des systèmes impose des logiques qui ne sont pas forcément classiques, alors que la logique des prédicats utilisée dans le formalisme Event-B est une logique classique. Les connaissances implicites ne sont donc pas systématiquement vraies dans l'absolu. Pour illustrer nos propos, reprenons l'étude donnée par l'auteur [118] dans ses travaux :

Supposons que $A : p \sqsubseteq_{S,\mathcal{D}} q$ soit vérifiée dans le système S et dans le domaine $\mathcal{D}$ et que l'on a : $A : p \sqsubseteq_{\mathcal{D}} q$: qui est non vraie dans le domaine.

$\Longrightarrow$ Cela signifie que $p \sqsubseteq_S q : faux$ n'apparaît pas explicitement dans la logique, mais implicitement.

Comment peut-t-on faire pour résoudre ce type de problèmes ? En particulier, dans le cas des logiques de description qui incluent une couche épistémique, chose que nous n'avons pas dans Event-B. Il faut donc changer de logique. Cette logique doit satisfaire les critères suivants :

— Nous avons besoin d'une *logique booléenne* qui vérifie les conditions suivantes :

$$\begin{array}{rcl} p : S \sqcap \mathcal{D} & \Longrightarrow & p : S \text{ et } p : \mathcal{D} \\ p : S \text{ ou } p : \mathcal{D} & \Longrightarrow & p : S \sqcup \mathcal{D} \end{array}$$

et si les conditions sont vérifiées dans le sens inverse ($\Longleftarrow$), alors la logique est booléenne ;

— Si nous considérons des systèmes fermés : alors il faut disposer d'une *logique extensionnelle* qui vérifie les conditions suivantes :

$$ext(S \sqcup \mathcal{D}) = ext(S) \cup ext(\mathcal{D})$$
$$ext(\neg S) = \mathcal{O} \setminus ext(S)$$

avec *ext* : est l'abréviation de extension (objets). C'est la raison pour laquelle nous optons pour l'utilisation de la logique contextualisée et la subsomption $\sqsubseteq_K$ (cf. La subsomption contextualisée utilisée dans la définition 5.2.2 en sous-section 1.9.3 du chapitre 1) que nous reprenons comme suit :

Définition 5.2.1 (Subsomption contextualisée) *La subsomption contextualisée se définit comme suit : Soient* $K = (\mathcal{O}, \mathcal{L}, d)$ *et* $f, g \in \mathcal{L}$. *On dit que* f *est contextuellement subsumée par* g *dans le contexte* K, *notée* $f \sqsubseteq_K g$, *ssi* $\beta_K(f) \subseteq \beta_K(g)$, *c-à-d, tout objet qui satisfait* f *satisfait aussi* g.

Définition 5.2.2 (Logique contextualisée) *Soit* $K = (\mathcal{O}, \mathcal{L}, d)$ *un contexte ACL. La logique contextualisée est définie par l'ensemble partiellement ordonné* $\mathcal{L}_K = \langle L; \sqsubseteq_K \rangle$ *modulo la relation d'équivalence* $\equiv_K$.

Dans cette association, nous considérons notre système comme une possibilité parmi toutes celles que le domaine peut nous offrir. C'est la raison pour laquelle nous induisons les propriétés selon le domaine, et nous déduisons celles liées au contexte éventuellement par raffinement en utilisant les liens et les traits caractéristiques définis dans ACL.

Nous ne souhaitons pas changer les structures initiales, celles d'Event-B et des ontologies, mais en créer de nouvelles en interaction entre ces deux premières :

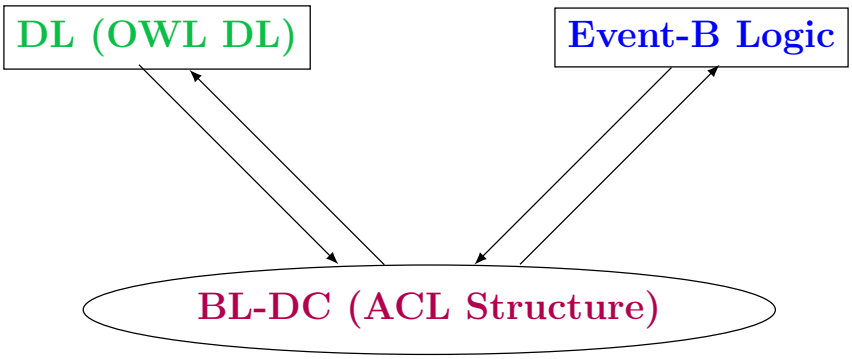

FIGURE 5.1 – Structure ACL pour Event-B et les ontologies

Nous résumons les étapes de l'intégration comme suit :

1. Définition de la structure du contexte ACL K comme suit :
   — les *intensions* $\mathcal{L}_{\mathcal{D} \uplus \mathcal{B}}$ définies sur les formules Event-B : Axm, Thm et Inv union les formules $T\text{-}Box$;
   — et les *extensions* $\mathcal{O}$ définies sur l'ensemble de la composante assertionnelle $A\text{-}Box$;
   — d : fonction associant pour chaque objet de $A\text{-}Box$ son ensemble de formules dans $\mathcal{L}_{\mathcal{D} \uplus \mathcal{B}}$ qui le décrivent ;
2. Définition de la connexion de Galois entre formules et objets ;
3. Reformulation du contexte ACL $K = (\mathcal{T}r, \mathcal{L}_{PO}, d_{\mathcal{B}})$ par raffinement ;
4. Intégration par raffinement $\Longrightarrow$ définition de vues logiques :
   — Une vue logique = $\{set_n, const_n, var_n\}$, avec n : niveau de raffinement considéré ;
   — Les requêtes définies par les obligations de preuve ;
   — Les liens : des restrictions des requêtes les impliquant strictement ;
   — Les traits caractéristiques : propriétés dont la description des objets n'est pas vide (extensions non vides) : invariants, axiomes et théorèmes Event-B ;

5. Les *intensions* $\mathcal{L}_{PO}$ définies sur les obligations de preuve Event-B augmentées des formules de la *T-Box* :

exemple : $T(\text{T-Box}) \wedge \mathcal{T}h(set, cst) \wedge \mathcal{I}nv(set, cst, var) \wedge BA(e)(set, cst, var, var') \Rightarrow \mathcal{I}nv(var')$;

6. Les *extensions* $\mathcal{T}r$ définies sur l'ensemble des traces d'exécution (complètes) ;
7. $d_{\mathcal{B}}$: fonction qui associe à chaque trace son ensemble d'obligations de preuve avec les contraintes du domaine qui décrivent cette trace ;
8. Les deux ensembles $(2^{\mathcal{T}r}, \subseteq)$ et $(\mathcal{L}_{PO}, \sqsubseteq_{K\mathcal{B}}$ [27]) sont ordonnés :
   — La connexion de Galois est définie par :
   $$\alpha_K : 2^{\mathcal{T}r} \rightarrow \mathcal{L}_{PO} \text{ et}$$
   $$\beta_K : \mathcal{L}_{PO} \rightarrow 2^{\mathcal{T}r}$$
9. Un calcul de point fixe sur le résultat obtenu doit être réalisé ;

Plusieurs défis sont identifés :

— Premièrement, les raisonneurs ontologiques sont monotones. Pour résoudre ce problème, il faut alors utiliser des techniques de résolution définies telles que celles proposées par Robinson [223] pour la démonstration automatique et qui sont étendues pour la logique du premier ordre [202].
— Deuxièmement, le raisonnement dans les logiques de description est un raisonnement a priori, où toutes les règles du raisonneur sont appliquées pour toutes les propriétés (relations) sémantiques d'un concept du domaine. Ce problème est résolu en utilisant les vues définies dans l'analyse logique de concepts.
— Troisièmement, l'évolution des connaissances est qualifiée de mesures numériques de confiance et de support, ainsi que des calculs souvent probabilistes sur l'ensemble des extensions défini dans un contexte au sens de l'analyse logique de concepts. Dans ce cas, il est aussi indispensable d'utiliser le plugin pour les théories dans Event-B. Une telle démarche serait alors définies en fonction de trois éléments, à savoir :
  — l'ontologie qui définit le domaine de conceptualisation ;
  — les théories pour les calculs sur les paramètres extraits de l'ontologie ;
  — et les modèles Event-B pour exploiter à la fois, les théories instanciées, mais aussi l'ontologie via les liens définis par la structure de l'analyse logique de concepts.
— Enfin, lorsque le contenu de l'ontologie est assez volumineux, il serait alors possible d'explorer les travaux réalisés en fouille de données (citons à titre indicatif les travaux [222]) pour appliquer des transposées des connexions de Galois au contexte défini au sens de l'analyse logique de concept.

Nous préconisons l'utilisation de la navigation combinée avec l'interrogation pour la vérification, et l'utilisation de l'interrogation pour la validation. De plus, le formalisme ACL définit deux modes d'extraction, à savoir, un mode *hors-ligne* où le contexte est construit une fois pour toute, et le mode *en-ligne* où la construction est réalisée progressivement en interaction avec les obligations de preuve définies comme requêtes dans notre cas. Nous privilégions une extraction en ligne que ce soit pour la vérification ou la validation pour une meilleure interaction avec le raisonneur.

27. relation de raffinement entre modèles Event-B

Annexe A

Les logiques de description et la logique du premier ordre

A.1 Relations des logiques de description avec la logique du premier ordre

Les logiques de description ont été interprétées dans de nombreux formalismes tels que les logiques modales, et la logique du premier [25]. Dans ce travail nous ne détaillerons que celle qui concerne la logique du premier ordre [55].

A.1.1 Logique de description et logique du premier ordre FOL

Les logiques de description peuvent être vues comme un fragment de la logique du premier ordre. En raison de la syntaxe sans variables des logiques de description et du fait que les concepts désignent des ensembles d'individus, la traduction des concepts donne des formules dans une variable libre. Suivant la définition de [55], et en considérant qu'un nom de concept est une relation (prédicat) unaire, et que les noms des rôles sont des relations (prédicats) binaires, deux fonctions de traduction Ψ_x et Ψ_y, qui inductivement, font corrrespondre les expressions conceptuelles $\mathcal{ALC}$ en formules du premier ordre avec les variables libres x et y. Un concept C et sa traduction $\Psi(C)(x)$ sont dits équivalents *ssi*, pour toute interprétation $\mathcal{I} = (\Delta^{\mathcal{I}}, .^{\mathcal{I}})$, et tout $o \in \Delta^{\mathcal{I}}$, nous avons :

$$o \in \Delta^{\mathcal{I}} \; ssi \; \mathcal{I} \models \Psi(C)(o)$$

Chaque concept A est vu comme un prédicat unaire, et chaque rôle R est vu comme un prédicat binaire. Pour une expression conceptuelle, la traduction est définie inductivement comme suit :

$$\begin{array}{rcl@{\qquad}rcl}
\Psi_x(A) & = & A(x), & \Psi_y(A) & = & A(y) \\
\Psi_x(C \sqcap D) & = & \Psi_x(C) \wedge \Psi_x(D), & \Psi_y(C \sqcap D) & = & \Psi_y(C) \wedge \Psi_y(D) \\
\Psi_x(C \sqcup D) & = & \Psi_x(C) \vee \Psi_x(D), & \Psi_y(C \sqcup D) & = & \Psi_y(C) \vee \Psi_y(D) \\
\Psi_x(\exists R.C) & = & \exists y.R(x,y) \wedge \Psi_y(C), & \Psi_y(\exists R.C) & = & \exists x.R(y,x) \wedge \Psi_x(C) \\
\Psi_x(\forall R.C) & = & \forall y.R(x,y) \Rightarrow \Psi_y(C), & \Psi_y(\forall R.C) & = & \forall x.R(y,x) \Rightarrow \Psi_x(C)
\end{array}$$

D'autres constructeurs de concepts et rôles peuvent aussi être traduits en logique du premier ordre, comme l'inverse des rôles, les conjonctions, les disjonctions et la négation des rôles. La traduction des restrictions sur les nombres $(\geq n\ R\ )$ et $(\leq n\ R)$ implique le calcul des quantificateurs $(\exists^{\geq n})$ et $(\exists^{\leq n})$.

$$\begin{array}{rcccl} \Psi_x(\geq n\ R) & = & \exists^{\geq n} y.R(x,y) & = & \exists y_1,...,y_n.\bigwedge_{i\neq j} y_i \neq y_j \wedge \bigwedge_i R(x,y_i) \\ \Psi_x(\leq n\ R) & = & \exists^{\leq n} y.R(x,y) & = & \forall y_1,...,y_{n+1}.\bigwedge_{i\neq j} y_i \neq y_j \Rightarrow \bigvee_i \neg R(x,y_i) \end{array}$$

Pour les restrictions de nombre qualifié, les traductions peuvent être facilement modifiées avec la même conséquence sur le nombre de variables impliquées.

La traduction ne se préoccupe pas seulement de la préservation des traductions de concepts dans des formules logiques. En logique de description, la consistance et la cohérence conceptuelle ainsi que l'implication logique en respect avec la terminologie T-Box, ainsi que la consistance de l'A-Box sont aussi importantes.

Etant donné la traduction Ψ donnée ci-dessus, la traduction de la terminologie T d'une base de connaissances $\Sigma = (T, A)$, est donnée comme suit :

$$\Psi(T) = \bigwedge_{C \sqsubseteq D \in T} \forall x.(\Psi_x(C) \Rightarrow \Psi_x(D))$$

et la base de faits A est donnée comme suit :

$$\Psi(A) = \bigwedge_{o_1:C \in A} \Psi(C)[x/o_1] \wedge \bigwedge_{(o_1,o_2):r \in A} r(o_1, o_2)$$

où la notation $[x/o_1]$ correspond à une substitution des occurences libres du concept x par ses individus dans A-Box.

Notons que cette traduction préserve bien la sémantique des logiques de description i.e. nous pouvons considérer les interprétations des logiques de description comme des interprétations de la logique du premier ordre et vice versa, et il est facile de montrer que la traduction préserve des modèles. Comme une conséquence, on peut dire que le raisonnement dans logiques de description correspond l'inférence dans la logique du premier ordre :

Théorème A.1.1 (***Préservation de la sémantique de la traduction***) *Soient* $\Sigma = (T, A)$ *une base de connaissances,* C *et* D *deux expressions conceptuelles, et* o *un individu. On a alors :*

1. (T, A) *est consistante* *ssi* $\Psi(T) \wedge \Psi(A)$ *est consistante,*
2. $(T, A) \models C \sqsubseteq D$ *ssi* $(\Psi(T) \wedge \Psi(A)) \Rightarrow (\Psi(\{C \sqsubseteq D\}))$ *est valide,*
3. $(T, A) \models o : C$ *ssi* $(\Psi(T) \wedge \Psi(A)) \Rightarrow (\Psi(\{o : C\}))$ *est valide*

Malgré la faisabilité de la traduction directe entre la logique du premier ordre et logiques de description, la garantie du raisonnement complet et final nécessite une transformation différente, telle que la transformation structurelle. Cette transformation est basée sur une forme normale de conjonction, qui remplace les sous-formules FOL par de nouveaux prédicats, pour lesquels elle fournit également des définitions. Un avantage majeur de la transformation structurelle est qu'elle évite la croissance exponentielle de la taille des clauses.

Annexe B

Correspondances entre la logique de description et le formalisme Event-B

Description OWL DL	**DL**	**Sémantique**	**Event-B**
A (refrence URI)	A	$A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$	const/var $A \subseteq Thing$
owl:Thing	$\top$	owl :Thing$^{\mathcal{I}} = \Delta^{\mathcal{I}}$	Set $Thing$
owl:Nothing	$\bot$	owl :Nothing$^{\mathcal{I}} = \{\}$	$\varnothing$
intersectionOf(C_1, C_2 ...)	$C_1 \sqcap C_2$	$(C_1 \sqcap C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$	const/var $C_1 \cap C_2$
unionOf(C_1, C_2 ...)	$C_1 \sqcup C_2$	$(C_1 \sqcup C_2)^{\mathcal{I}} = C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$	const/var $C_1 \cup C_2$
complementOf(C)	$\neg C$	$(\neg C)^{\mathcal{I}} = \Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$	const/var $NC = Thing \setminus C$
oneOf(o_1 ...)	$\{o_1, ...\}$	$\{o_1, ...\}^{\mathcal{I}} = \{o_1^{\mathcal{I}}, ...\}$	const $o_1, ...o_n \wedge$ const/var cb $cb \subseteq Thing \wedge$ partition$(cb, \{o_1\}, ..., \{o_n\})$
restric(R someValuesFrom(C))	$\exists R.C$	$(\exists R.C)^{\mathcal{I}} = \{x \mid \exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$	const/var $C, R \wedge \{\text{dom}(R \rhd C)\}$
restric(R allValuesFrom(C))	$\forall R.C$	$(\forall R.C)^{\mathcal{I}} = \{x \mid \forall y.(x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}}\}$	$\{x \mid x \in \text{dom}(R)$ $\wedge(\forall y.(x \mapsto y) \in R \Rightarrow y \in C)\}$
restri(R hasValue(o))	$R : o$	$(\forall R.o)^{\mathcal{I}} = \{x \mid (x, o^{\mathcal{I}}) \in R^{\mathcal{I}}\}$	$\{\text{dom}(R \rhd \{o\}) \mid \text{const } o \wedge o \in \text{ran}(R)\}$
restri(R minCardinality(n))	$\geq nR$	$(\geq nR)^{\mathcal{I}} = \{x \mid card(\{y.(x,y) \in R^{\mathcal{I}}\}) \geq n\}$	$\{x \mid x \in dom(R) \wedge \text{finite}(R[\{x\}])$ $\wedge \text{card}(R[\{x\}]) \geq n\}$
restri(R maxCardinality(n))	$\leq nR$	$(\leq nR)^{\mathcal{I}} = \{x \mid card(\{y.(x,y) \in R^{\mathcal{I}}\}) \leq n\}$	$\{x \mid x \in dom(R) \wedge \text{finite}(R[\{x\}])$ $\wedge \text{card}(R[\{x\}]) \leq n\}$
	$\leq nR.C$	$(\leq nR.C)^{\mathcal{I}} = \{x \mid (\forall y(x,y) \in R^{\mathcal{I}} \Rightarrow y \in C^{\mathcal{I}})$ $\wedge(card(\{y.(x,y) \in R^{\mathcal{I}}\}) \leq n)\}$	$\{x \mid x \in dom(R) \wedge (\forall y.(x \mapsto y) \in R \Rightarrow y \in C)$ $\wedge \text{finite}(R[\{x\}]) \wedge card(R[\{x\}]) \leq n\}$
	$\geq nR.C$	$\{x \mid (\exists y.(x,y) \in R^{\mathcal{I}} \wedge y \in C^{\mathcal{I}})$ $\wedge(card(\{y.(x,y) \in R^{\mathcal{I}}\})) \geq n\}$	$\{x \mid x \in dom(R) \wedge \text{finite}([\{x\}])$ $\wedge card(R[\{x\}]) \geq n \wedge dom(R \rhd C)\}$
restri(U someValuesFrom(D))	$\exists U.D$	$(\exists U.D)^{\mathcal{I}} = \{x \mid \exists y(x,y) \in U^{\mathcal{I}} \wedge y \in D^{\mathcal{D}}\}$	idem que ci-dessus
restri(U allValuesFrom(D))	$\forall U.D$	$(\forall U.D)^{\mathcal{I}} = \{x \mid \forall y(x,y) \in U^{\mathcal{I}} \Rightarrow y \in D^{\mathcal{D}}\}$	
restri(U hasValue(v))	$U : v$	$(U : v)^{\mathcal{I}} = \{x \mid (x, v^{\mathcal{I}}) \in U^{\mathcal{I}}\}$	
restri(U minCardinality(n))	$\geq nU$	$(\geq nU)^{\mathcal{I}} = \{x \mid card(\{y.(x,y) \in U^{\mathcal{I}}\}) \geq n\}$	
restri(U maxCardinality(n))	$\leq nU$	$(\leq nU)^{\mathcal{I}} = \{x \mid card(\{y.(x,y) \in U^{\mathcal{I}}\}) \leq n\}$	
Data Range (D)			
D (refrence URI)	D	$D^{\mathcal{D}} \subseteq \Delta_{\mathcal{D}}^{\mathcal{I}}$	const/var $A \subseteq D^{\mathcal{D}}$
oneOf(v_1 ...)	$\{v_1, ...\}$	$\{v_1, ...\}^{\mathcal{I}} = \{v_1^{\mathcal{D}}, ...\}$	const $v_1, ...v_n \wedge$ const/var $cb \wedge cb \in \{v_1, ...v_n\}$
Object Properties (R)			
R (refrence URI)	R	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	const/var $R \wedge R \subseteq Thing \times Thing$
	R^{-1}	$(R^{-1})^{\mathcal{I}} = (R^{\mathcal{I}})^{-1}$	const/var $R \wedge R \subseteq Thing \times Thing$ $\wedge R^{-1} \subseteq Thing \times Thing$
Datatype Properties (U)			
U (refrence URI)	U	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta_{\mathcal{D}}^{\mathcal{I}}$	const/var $U \wedge U \subseteq Thing \times SetData$
Individus (o)			
o (refrence URI)	o	$o^{\mathcal{I}} \in \Delta^{\mathcal{I}}$	const/var $o \wedge o \in Thing$
Data Values (v)			
v (Literal RDF)	v	$v^{\mathcal{I}} = v^{\mathcal{D}}$	const $v \wedge v \in SetData$

FIGURE B.1 – Récapitulatif des transformations entre les logiques de description (Plages de données, propriétés, individus et valeurs de données) et Event-B

Description OWL DL	**DL**	**Sémantique**	**Event-B**
Classes			
`Class(`A, `partial` $C_1 ... C_n$`)`	$A \sqsubseteq C_1 \sqcap ... \sqcap C_n$	$A^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \cap ... \cap C_n^{\mathcal{I}}$	const/var $C_1, ...C_n, A \wedge A \subseteq C_1 \cap ... \cap C_n$
`Class(`A, `complete` C_1 ... C_n`)`	$A = C_1 \sqcap ... \sqcap C_n$	$A^{\mathcal{I}} = C_1^{\mathcal{I}} \cap ... \cap C_n^{\mathcal{I}}$	const/var $C_1, ...C_n, A \wedge A = C_1 \cap ... \cap C_n$
`EnumeratedClass(`A o_1 ... o_n`)`	$A = \{o_1, ..., o_n\}$	$A^{\mathcal{I}} = \{o_1^{\mathcal{I}}, ..., o_n^{\mathcal{I}}\}$	const $A, o_1, ...o_n \wedge A = \{o_1, ...o_n\} * *voir\ var$
`SubClassOf(`C_1 C_2`)`	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$	const/var $C_1, C_2 \wedge C_1 \subseteq C_2$
`EquivalentClasses(`C_1 ...C_n`)`	$C_1 = ... = C_n$	$C_1^{\mathcal{I}} = ... = C_n^{\mathcal{I}}$	const/var $C_1, ...C_n \wedge C_1 = ... = C_n$
`DisjointClasses(`C_1 ...C_n`)`	$C_i \sqcap C_j = \bot, i \neq j$	$C_i^{\mathcal{I}} \cap C_j^{\mathcal{I}} = \{\}, i \neq j$	const/var $C_1, ...C_n \wedge C_i \cap C_j = \varnothing$
`Datatype(`D`)`		$D^{\mathcal{I}} \subseteq \Delta_{\mathcal{D}}^{\mathcal{I}}$	const $D \wedge D \subseteq Thing$
Propriétés Datatype			
`DatatypeProperty(`U `super(`U_1`)...super(`U_n`)`	$U \sqsubseteq U_i$	$U^{\mathcal{I}} \subseteq U_i^{\mathcal{I}}$	const/var $U, U_1, ...U_n \wedge U \subseteq Thing$ $\wedge$ partition(U, $U_1, ...U_n$)
`domain(`C_1`) ...domain(`C_n`)`	$\geq 1\ U \sqsubseteq C_i$	$U^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta_{\mathcal{D}}^{\mathcal{I}}$	const/var $U, C_i \wedge C_i \subseteq Thing \wedge$ $U \subseteq C_i \times SetData$
`range(`D_1`) ...range(`D_n`)`	$\top \sqsubseteq \forall U.D_i$	$U^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times D_i^{\mathcal{I}}$	const/var $U, D_i \wedge D_i \subseteq Thing \wedge$ $U \subseteq SetData \times D_i$
`[Functional])`	$\top \sqsubseteq \leq 1\ U$	$U^{\mathcal{I}}$ est fonctionnel	$U \in Thing \nrightarrow Thing$
`SubPropertyOf(`U_1 U_2`)`	$U_1 \sqsubseteq U_2$	$U_1^{\mathcal{I}} \subseteq U_2^{\mathcal{I}}$	const/var $U_1, U_2 \wedge U_1 \subseteq U_2$
`EquivalentProperties(`U_1 ...U_n`)`	$U_1 = ... = U_n$	$U_1^{\mathcal{I}} = ... = U_n^{\mathcal{I}}$	const/var $U_1, .., U_n \wedge U_1 = ... = U_n$
Propriétés des objets			
`ObjectProperty(`R `super(`R_1`)...super(`R_n`))`	$R \sqsubseteq R_i$	$R^{\mathcal{I}} \subseteq R_i^{\mathcal{I}}$	const/var $R, R_i \wedge R \subseteq R_i$
`domain(`C_1`)...domain(`C_m`)`	$\geq 1\ R \sqsubseteq C_i$	$R^{\mathcal{I}} \subseteq C_i^{\mathcal{I}} \times \Delta^{\mathcal{I}}$	const/var $R, C_i \wedge C_i \subseteq Thing \wedge$ $R \subseteq C_i \times Thing$
`range(`C_1`)...range(`C_l`)`	$\top \sqsubseteq \forall R.C_i$	$R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_i^{\mathcal{I}}$	const/var $R, C_i \wedge C_i \subseteq Thing \wedge$ $R \subseteq Thing \times C_i$
`[inverseOf(`R_0`)]`	$R = (R_0^{-1})$	$R^{\mathcal{I}} = (R_0^{\mathcal{I}})^{-1}$	const/var $R_0 \wedge R_0 \subseteq Thing \times Thing$ $R = R_0^{-1} \wedge R = R_0^{-1}$
`[Symmetric]`	$R = (R^{-1})$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{-1}$	$R = R^{-1}$
`[Functional]`	$\top \sqsubseteq \leq 1R$	$R^{\mathcal{I}}$ est une fonction	$R \in Thing \nrightarrow Thing$
`[InverseFunctional]`	$\top \sqsubseteq \leq 1R^{-1}$	$(R^{\mathcal{I}})^{-1}$ est une fonction	$R^{-1} \in Thing \nrightarrow Thing$
`[Transitive]`	$Tr(R)$	$R^{\mathcal{I}} = (R^{\mathcal{I}})^{+}$	$R\,;R \subseteq R$
`SubPropertyOf(`R_1 R_2`)`	$R_1 \sqsubseteq R_2$	$R_1^{\mathcal{I}} \subseteq R_2^{\mathcal{I}}$	const/var $R_1, R_2 \wedge R_1 \subseteq R_2$
`EquivalentProperties(`R_1 R_2`)`	$R_1 = ... = R_n$	$R_1^{\mathcal{I}} = ... = R_n^{\mathcal{I}}$	const/var $R_1, ...R_n \wedge R_1 = ... = R_n$
Individus			
`Individuals(`o `type(`C_1`)...type(`C_n`)`	$o \in C_i$	$o^{\mathcal{I}} \in C_i^{\mathcal{I}}$	const $o \wedge o \in Thing \wedge$ const/var $C_i \subseteq Thing \wedge o \in C_i$
`value(`$R_1 o_1$`)...value(`$R_n o_n$`)`	$(o, o_i) \in R_i$	$(o^{\mathcal{I}}, o_i^{\mathcal{I}}) \in R_i^{\mathcal{I}}$	const $o, o_i \wedge o \in Thing \wedge o_i \in Thing \wedge$ const/var $R \wedge o \mapsto o_i \in R$
`value(`$U_1 v_1$`)...value(`$U_n v_n$`))`	$(o, v_i) \in U_i$	$(o^{\mathcal{I}}, v_i^{\mathcal{I}}) \in U_i^{\mathcal{I}}$	const $o, v_i \wedge o \in Thing \wedge v_i \in D^{\mathcal{D}}$ const/var $R \wedge o \mapsto v_i \in R$
`SameIndividual(`$o_1 ... o_n$`)`	$o_1 = ... = o_n$	$o_1^{\mathcal{I}} = ... = o_n^{\mathcal{I}}$	non utilisée dans notre transformation
`DifferentIndividuals(`$o_1 ... o_n$`)`	$o_i \neq o_j, i \neq j$	$o_i^{\mathcal{I}} \neq o_j^{\mathcal{I}}, i \neq j$	Identification unique des individus

FIGURE B.2 – Récapitulatif des transformations entre les logiques de description (Axiomes et Faits) et Event-B

DL	Sémantique	Event-B
$E_1 \sqsubseteq \top \sqcap E_2 \sqsubseteq \top \sqcap R$ $\sqcap (o_1, o_2) \in R$	$E_1^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \wedge E_2^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \wedge$ $R^{\mathcal{I}} \subseteq E_1^{\mathcal{I}} \times E_2^{\mathcal{I}} \wedge (o_1^{\mathcal{I}}, o_2^{\mathcal{I}}) \in R^{\mathcal{I}}$	$o_1 \in E_1 \wedge o_2 \in E_2 \wedge o_1 \mapsto o_2 \in E_1 \times E_2 = R$
$E_2^{\mathcal{I}} \sqsubseteq E_1^{\mathcal{I}}$	$E_2^{\mathcal{I}} \subseteq E_1^{\mathcal{I}}$	$E_2 \in \mathbb{P}_1(E_1) \Leftrightarrow \forall x.x \in E_2 \Rightarrow x \in E_1$
$S \sqsubseteq \top \sqcap e \in S \sqcap$ $1 \geq P \sqsubseteq S \sqcap \forall P^{-1} : e \not\sqsubseteq \bot$	$S^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \wedge P^{\mathcal{I}} \subseteq S^{\mathcal{I}} \times \Delta^{\mathcal{I}} \wedge$ $\{y \mid (y, e^{\mathcal{I}}) \in (P^{\mathcal{I}})^{-1}\} \neq \varnothing$	$e \in \{x \mid x \in S \wedge P(x)\} \Leftrightarrow e \in S \wedge P(e)$
$E_1 \sqsubseteq E_2 \sqcap E_2 \sqsubseteq E_1$	$E_1^{\mathcal{I}} \subseteq E_2^{\mathcal{I}} \wedge E_2^{\mathcal{I}} \subseteq E_1^{\mathcal{I}}$	$E_1 = E_2 \Leftrightarrow E_1 \subseteq E_2 \wedge E_2 \subseteq E_1$
$e \in E_1 \sqcup e \in E_2 \equiv e \in E_1 \sqcup E_2$	$e^{\mathcal{I}} \in E_1^{\mathcal{I}} \vee e^{\mathcal{I}} \in E_2^{\mathcal{I}}$	$e \in E_1 \cup E_2 \Leftrightarrow e \in E_1 \vee e \in E_2$
$e \in E_1 \sqcap e \in E_2 \equiv e \in E_1 \sqcap E_2$	$e^{\mathcal{I}} \in E_1^{\mathcal{I}} \wedge e^{\mathcal{I}} \in E_2^{\mathcal{I}}$	$e \in E_1 \cap E_2 \Leftrightarrow e \in E_1 \wedge e \in E_2$
$e \in E_1 \sqcap \neg(e \in E_2)$	$e^{\mathcal{I}} \in E_1^{\mathcal{I}} \wedge e \notin E_2^{\mathcal{I}}$	$e \in E_1 \setminus E_2 \Leftrightarrow e \in E_1 \wedge e \notin E_2$
$e \in \bot$	$e^{\mathcal{I}} \in \{\}$	$e \in \varnothing \Leftrightarrow \bot$
$E \sqsubseteq \top \sqcap E_1 \sqsubseteq E \sqcap e \in E_1$	$E^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \wedge E_1^{\mathcal{I}} \subseteq E^{\mathcal{I}} \wedge e^{\mathcal{I}} \in E_1^{\mathcal{I}}$	$e \in \mathrm{union}(E) \Leftrightarrow \exists E_1.E_1 \in E \wedge e \in E_1$
$E \sqsubseteq \top \sqcap \forall i\ E_i \sqsubseteq E \sqsubseteq e \in E_i$ à revoir $\exists i\ E_i \not\sqsubseteq E \ \sqcup\ e \in E_i$	$E^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \wedge \ \forall i\ E_i^{\mathcal{I}} \subseteq E^{\mathcal{I}} \Rightarrow e^{\mathcal{I}} \subseteq E_i^{\mathcal{I}}$	$e \in \mathrm{inter}(E) \Leftrightarrow \forall E_1.E_1 \in E \Rightarrow e \in E_1$
$S \sqsubseteq \top \sqcap \geq 1P \sqsubseteq S \sqcap \geq 1T \sqsubseteq S \sqcap$ $P^{-1} : x \not\sqsubseteq \bot \sqcap (x, e) \in T$	$S^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \wedge P^{\mathcal{I}} \subseteq S^{\mathcal{I}} \times \Delta^{\mathcal{I}} \wedge x^{\mathcal{I}} \in S^{\mathcal{I}} \wedge$ $T^{\mathcal{I}} \subseteq S^{\mathcal{I}} \times \Delta^{\mathcal{I}} \wedge$ $\{y \mid (y, x^{\mathcal{I}}) \in (P^{\mathcal{I}})^{-1}\} \neq \varnothing \wedge (x, e^{\mathcal{I}}) \in T^{\mathcal{I}}$	$e \in \bigcup x.x \in S \wedge P(x) \vert T(x) \Leftrightarrow$ $\exists x.x \in S \wedge P(x) \wedge e \in T(x)$
		$\forall x.x \in S \wedge P(x) \Rightarrow e \in T(x) \Leftrightarrow$ $e \in \bigcap x.x \in S \wedge P(x) \vert T(x)$
$R \sqcap (x, y) \in R$ $\sqcap (\forall R^{-1} : x \not\sqsubseteq \bot)$	$(x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}} \wedge$ $\{y \mid (y, x^{\mathcal{I}}) \in (R^{-1})^{\mathcal{I}}\} \neq \varnothing$	$x \in \mathrm{dom}(R) \Leftrightarrow \exists y.x \mapsto y \in R$
$R \sqcap (x, y) \in R \sqcap$ $(\forall R : y \not\sqsubseteq \bot)$	$(x^{\mathcal{I}}, y^{\mathcal{I}}) \in R^{\mathcal{I}} \wedge$ $\{y \mid (x, y^{\mathcal{I}}) \in R^{\mathcal{I}}\} \neq \varnothing$	$y \in \mathrm{ran}(R) \Leftrightarrow \exists x.x \mapsto y \in R$
$(y, x) \in R \sqcap (x, y) \in R^{-1}$	$(x^{\mathcal{I}}, y^{\mathcal{I}}) \in (R^{-1})^{\mathcal{I}} \Leftrightarrow (y^{\mathcal{I}}, x^{\mathcal{I}}) \in R^{\mathcal{I}}$	$x \mapsto y \in (R^{-1}) \Leftrightarrow y \mapsto x \in R$
$(\geq 1\ r \sqsubseteq S) \sqcap (x, y) \in r \sqcap$ $(\forall r^{-1} : x = T)$	$r^{\mathcal{I}} \subseteq S^{\mathcal{I}} \times \Delta^{\mathcal{I}} \wedge (x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}} \wedge$ $\{y \mid (y^{\mathcal{I}}, x^{\mathcal{I}}) \in (r^{-1})^{\mathcal{I}}\} = T^{\mathcal{I}}$	$r \in S \leftrightarrow T \Leftrightarrow r \in S \leftrightarrow T \wedge \mathrm{ran}(r) = T$ $(\forall y \in T \Rightarrow \exists x. \in S \wedge x \mapsto y \in r)$
$(\top \sqsubseteq \forall r.T) \ \sqcap\ (x, y) \in r \sqcap$ $(\forall r : y = S)$	$r^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times T^{\mathcal{I}} \ \wedge\ (x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}} \wedge$ $\{x \mid (x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}}\} = S^{\mathcal{I}}$	$r \in S \leftarrow\!\!\!\!\!\rightarrow T \Leftrightarrow r \in S \leftrightarrow T \wedge \mathrm{dom}(r) = S$
Voir les deux traductions précédentes		$r \in S \leftarrow\!\!\!\!\!\rightarrow\!\!\!\!\!\rightarrow T \Leftrightarrow r \in S \leftrightarrow\!\!\!\!\!\rightarrow T \wedge r \in S \leftarrow\!\!\!\!\!\rightarrow T$
$x \in S \ \sqcap\ (x, y) \in r$	$x^{\mathcal{I}} \in S^{\mathcal{I}} \wedge (x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}}$	$x \mapsto y \in S \lhd r \Leftrightarrow x \in S \wedge x \mapsto y \in r$
$y \in T \ \sqcap\ (x, y) \in r$	$y^{\mathcal{I}} \in T^{\mathcal{I}} \wedge (x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}}$	$x \mapsto y \in r \rhd T \Leftrightarrow x \mapsto y \in r \wedge y \in T$
$x \in \neg S \ \sqcap\ (x, y) \in r$	$x^{\mathcal{I}} \in \Delta^{\mathcal{I}} \setminus S^{\mathcal{I}} \wedge (x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}}$	$x \mapsto y \in S \lhd r \Leftrightarrow x \notin S \wedge x \mapsto y \in r$
$y \in \neg T \ \sqcap\ (x, y) \in r$	$y^{\mathcal{I}} \in \Delta^{\mathcal{I}} \setminus T^{\mathcal{I}} \wedge (x^{\mathcal{I}}, y^{\mathcal{I}}) \in r^{\mathcal{I}}$	$x \mapsto y \in r \rhd T \Leftrightarrow x \mapsto y \in r \wedge y \notin T$
$w \ \sqsubseteq\ \top \ \sqcap\ x \in w \ \sqcap\ (x, y) \in r$	$w^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \wedge x^{\mathcal{I}} \in w^{\mathcal{I}} \wedge (x^{\mathcal{I}}, y^{\mathcal{I}}) \in r$	$y \in r[w] \Leftrightarrow \exists x.x \in w \wedge x \mapsto y \in r$
$z \in \top \ \sqcap\ (x, z) \in p \ \sqcap\ (z, y) \in q$	$z^{\mathcal{I}} \in \Delta^{\mathcal{I}} \wedge (x^{\mathcal{I}}, z^{\mathcal{I}}) \in p^{\mathcal{I}} \wedge (z^{\mathcal{I}}, y^{\mathcal{I}}) \in q^{\mathcal{I}}$	$x \mapsto y \in (p\,;q) \Leftrightarrow \exists z.x \mapsto z \in p \wedge z \mapsto y \in q$
		$(\mathrm{dom}(q) \lhd p) \cup q$
$\geq 1\ idr \sqsubseteq S \sqcap \geq 1\ idr^{-1} \sqsubseteq S \sqcap$ $x \in S \sqcap y = x$	$idr^{\mathcal{I}} \subseteq S^{\mathcal{I}} \times \Delta^{\mathcal{I}} \wedge (idr^{-1})^{\mathcal{I}} \subseteq S^{\mathcal{I}} \times \Delta^{\mathcal{I}} \wedge$ $x^{\mathcal{I}} \in S^{\mathcal{I}} \wedge (x^{\mathcal{I}}, y^{\mathcal{I}}) \in idr^{\mathcal{I}} \wedge y^{\mathcal{I}} = x^{\mathcal{I}}$	$x \mapsto y \in \mathrm{id}(S) \Leftrightarrow x \in S \wedge y = x$
$(x, y) \in p \ \sqcap\ (x, z) \in q$	$(x^{\mathcal{I}}, y^{\mathcal{I}}) \in p^{\mathcal{I}} \wedge (x^{\mathcal{I}}, z^{\mathcal{I}}) \in q^{\mathcal{I}}$	$x \mapsto (y \mapsto z) \in p \otimes q \Leftrightarrow x \mapsto y \in p \wedge x \mapsto z \in q$
$x \in S \ \sqcap\ y \in T \ \sqcap\ z \in \top \ \sqcap\ z = x$	$x^{\mathcal{I}} \in S^{\mathcal{I}} \wedge y^{\mathcal{I}} \in T^{\mathcal{I}} \wedge z^{\mathcal{I}} = x^{\mathcal{I}} \wedge z^{\mathcal{I}} \in \Delta^{\mathcal{I}}$	$(x \mapsto y) \mapsto z \in \mathrm{prj}_1(S, T) \Leftrightarrow$ $x \in S \wedge y \in T \wedge z = x$
$x \in S \ \sqcap\ y \in T \ \sqcap\ z = y$	$x^{\mathcal{I}} \in S^{\mathcal{I}} \wedge y^{\mathcal{I}} \in T^{\mathcal{I}} \wedge z^{\mathcal{I}} = y^{\mathcal{I}}$	$(x \mapsto y) \mapsto z \in \mathrm{prj}_2(S, T) \Leftrightarrow$ $x \in S \wedge y \in T \wedge z = y$
$(x, z) \in p \ \sqcap\ (y, h) \in q$	$(x^{\mathcal{I}}, z^{\mathcal{I}}) \in p^{\mathcal{I}} \wedge (y^{\mathcal{I}}, h^{\mathcal{I}}) \in q^{\mathcal{I}}$	$(x \mapsto y) \mapsto (z \mapsto h) \in p \parallel q \Leftrightarrow$ $x \mapsto z \in p \wedge y \mapsto h \in q$
Voir le tableau précédent		$f \in S \nrightarrow T$
$\top \sqsubseteq \leq 1f \ \sqcap\ \forall f : y = S$	f est fonction partielle $\wedge \{x \mid (x^{\mathcal{I}}, y^{\mathcal{I}}) \in f^{\mathcal{I}}\} = S^{\mathcal{I}}$	$f \in S \rightarrow T \Leftrightarrow f \in S \nrightarrow T \wedge S = \mathrm{dom}(f)$
$\top \sqsubseteq \leq 1f \ \sqcap\ \top \sqsubseteq \leq 1f^{-1}$	$f^{\mathcal{I}} \ \wedge\ (f^{-1})^{\mathcal{I}}$ fonctions partielles	$f \in S \rightarrowtail T \Leftrightarrow f \in S \nrightarrow T \wedge f^{-1} \in T \nrightarrow S$
Voir fonctions partielles et totales		$f \in S \rightarrowtail T \Leftrightarrow f \in S \rightarrow T \wedge f^{-1} \in T \nrightarrow S$
$\top \sqsubseteq \leq 1f \ \sqcap\ \forall f^{-1} : x = T$	$f^{\mathcal{I}}$ partielle $\wedge \{y \mid (y^{\mathcal{I}}, x^{\mathcal{I}}) \in (f^{-1})^{\mathcal{I}}\} = T^{\mathcal{I}}$	$f \in S \twoheadrightarrow T \Leftrightarrow f \in S \nrightarrow T \wedge T = \mathrm{ran}(T)$
f totale $\sqcap\ \forall f^{-1} : x = T$	f totale $\wedge \{y \mid (y^{\mathcal{I}}, x^{\mathcal{I}}) \in (f^{-1})^{\mathcal{I}}\} = T^{\mathcal{I}}$	$f \in S \rightarrow T \Leftrightarrow f \in S \rightarrow T \wedge T = \mathrm{ran}(f)$
Voir transformations injection et surjection totales		$f \in S \rightarrowtail\!\!\!\!\!\rightarrow T \Leftrightarrow f \in S \rightarrowtail T \wedge f \in S \twoheadrightarrow T$

FIGURE B.3 – Récapitulatif des transformations entre Event-B et logique de description

Bibliographie

[1] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In Chris Hankin and Dave Schmidt, editors, *Conference Record of POPL 2001 : The 28th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, London, UK, January 17-19, 2001*, pages 104–115. ACM, 2001.

[2] J.-R. Abrial. *The B-book : Assigning Programs to Meanings*. Cambridge University Press, New York, NY, USA, 1996.

[3] Jean-Raymond Abrial. *Modeling in Event-B : System and Software Engineering*. Cambridge University Press, 2010.

[4] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin : an open toolset for modelling and reasoning in Event-B. *International journal on software tools for technology transfer*, 12(6) :447–466, 2010.

[5] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Michael Leuschel, Matthias Schmalz, and Laurent Voisin. Proposals for mathematical extensions for Event-B, 2009. http ://deploy-eprints.ecs.soton.ac.uk/80.

[6] Jean-Raymond Abrial, Michael J. Butler, Stefan Hallerstede, and Laurent Voisin. A roadmap for the rodin toolset. In *ABZ*, volume 5238 of *Lecture Notes in Computer Science*, page 347. Springer, 2008.

[7] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, decomposition, and instantiation of discrete models : Application to Event-B. *Fundamenta Informaticae*, 77(1-2) :1–28, 2007.

[8] Jean-Raymond Abrial and Thai Son Hoang. Using design patterns in formal methods : An Event-B approach. In John S. Fitzgerald, Anne Elisabeth Haxthausen, and Hüsnü Yenigün, editors, *ICTAC*, volume 5160 of *Lecture Notes in Computer Science*, pages 1–2. Springer, 2008.

[9] Yamine Ait Ameur and Dominique Méry. Making explicit domain knowledge in formal system development. *Science of Computer Programming*, 121(100–127), March 2016.

[10] Art Akerman and Jeff Tyree. Using ontology to support development of software architectures. *IBM Systems Journal*, 45(4) :813–826, 2006.

[11] Varol Akman and Mehmet Surav. The use of situation theory in context modeling. *Computational Intelligence*, 13(3) :427–438, 1997.

[12] Eman H Alkhammash. Derivation of Event-B Models from OWL Ontologies. In *MATEC Web of Conferences*, volume 76, page 04008. EDP Sciences, 2016.

[13] Paul Alpar and Sebastian Olbrich. Legal requirements and modelling of processes in e-government. *Electronic journal of e-government*, 3(3) :107–116, 2005.

[14] Yamine Aït Ameur, J. Paul Gibson, and Dominique Méry. On Implicit and Explicit Semantics : Integration Issues in Proof-Based Development of Systems. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Specialized Techniques and Applications - 6th International Symposium,*, volume 8803 of *Lectures Notes in Computer Science*, pages 604–618, Corfu, Greece, October 2014. Tiziana Margaria and Bernhard Steffen, Springer.

[15] Rachid Anane, Richard Freeland, and Georgios K. Theodoropoulos. e-voting requirements and implementation. In *9th IEEE International Conference on E-Commerce Technology (CEC 2007) / 4th IEEE International Conference on Enterprise Computing, E-Commerce and E-Services (EEE 2007), 23-26 July 2007, National Center of Sciences, Tokyo, Japan*, pages 382–392. IEEE Computer Society, 2007.

[16] Manamiary Bruno Andriamiarina. *Développement d'algorithmes répartis corrects par construction. (Developing correct-by-construction distributed algorithms)*. Thèse de doctorat, Université de Lorraine, Nancy, France, 2015.

[17] Manamiary Bruno Andriamiarina, Dominique Méry, and Neeraj Kumar Singh. Integrating proved state-based models for constructing correct distributed algorithms. In Johnsen and Petre [158], pages 268–284.

[18] Aristote. *Oeuvres complètes*. Flammarion, 2014.

[19] André-Jean Arnaud. Recension de Jean-Louis Le Moigne, La modélisation des systèmes complexes, 1990. *Droit et Société*, 19(1) :424–424, 1991.

[20] A. Arnold. *Systèmes de transitions finis et sémantique des processus communicants*. Etudes et recherches en informatique. MASSON, 1992.

[21] Alessandro Artale and Enrico Franconi. A temporal description logic for reasoning about actions and plans. *J. Artif. Intell. Res.*, 9 :463–506, 1998.

[22] Cyrille Artho, Doron Drusinksy, Allen Goldberg, Klaus Havelund, Mike Lowry, Corina Pasareanu, Grigore Rosu, and Willem Visser. Experiments with test case generation and runtime analysis. In *Proceedings of the Abstract State Machines 10th International Conference on Advances in Theory and Practice*, ASM'03, pages 87–108, Berlin, Heidelberg, 2003. Springer-Verlag.

[23] Nathalie Aussenac-Gilles. *Bottom-up methods for Knowledge Engineering*. Habilitation à diriger des recherches, Université Paul Sabatier - Toulouse III, December 2005.

[24] Franz Baader. *The description logic handbook : theory, implementation, and applications*. Cambridge university press, 2003.

[25] Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors. *The Description Logic Handbook : Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[26] Ralph-Johan Back and Michael Butler. Fusion and simultaneous execution in the refinement calculus. *Acta Informatica*, 35(11) :921–949, 1998.

[27] Michael Backes, Catalin Hritcu, and Matteo Maffei. Automated verification of remote electronic voting protocols in the applied pi-calculus. In *Proceedings of the 21st IEEE Computer Security Foundations Symposium, CSF 2008, Pittsburgh, Pennsylvania, 23-25 June 2008*, pages 195–209. IEEE Computer Society, 2008.

[28] Patrick Barlatier. *Conception et implantation d'un modèle de raisonnement sur les contextes basé sur une théorie des types et utilisant une ontologie de domaine*. Thèse de doctorat, Université de Savoie, 2009.

[29] Jon Barwise. Conditionals and conditional information. In Elizabeth Traugott, Alice ter Meulen, Judy Reilly, and Charles Ferguson, editors, *On Conditionals*, pages 21–54. Cambridge University Press, Cambridge, England, 1986.

[30] Jon Barwise. *The situation in logic*. Number 17. Center for the Study of Language (CSLI), 1989.

[31] Jon Barwise and John Perry. *Situations and Attitudes*. MIT Press, Cambridge, MA, 1983.

[32] A. Baskar, R. Ramanujam, and S. P. Suresh. Knowledge-based modelling of voting protocols. In *Proceedings of the 11th Conference on Theoretical Aspects of Rationality and Knowledge*, TARK '07, pages 62–71, New York, NY, USA, 2007. ACM.

[33] Nazim Benaïssa. Modelling Attacker's Knowledge for Cascade Cryptographic Protocols. In Egon Börger, Michael Butler, Jonathan P. Bowen, and Paul Boca, editors, *First International Conference on Abstract State Machines, B and Z - ABZ 2008*, volume 5238, pages 251–264, London, United Kingdom, September 2008. Springer.

[34] Nazim Benaïssa. *La composition des protocoles de sécurité avec la méthode B événementielle. (Security protocols composition using Event B)*. Thèse de doctorat, Université Henri Poincaré , Nancy, France, 2010.

[35] Nazim Benaïssa and Dominique Méry. Cryptographic protocols analysis in event B. In Amir Pnueli, Irina Virbitskaite, and Andrei Voronkov, editors, *Perspectives of Systems Informatics, 7th International Andrei Ershov Memorial Conference, PSI 2009, Novosibirsk, Russia, June 15-19, 2009. Revised Papers*, volume 5947 of *Lecture Notes in Computer Science*, pages 282–293. Springer, 2009.

[36] Nazim Benaïssa and Dominique Méry. Proof-based design of security protocols. In Farid M. Ablayev and Ernst W. Mayr, editors, *Computer Science - Theory and Applications, 5th International Computer Science Symposium in Russia, CSR 2010, Kazan, Russia, June 16-20, 2010. Proceedings*, volume 6072 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2010.

[37] Josh Benaloh and Dwight Tuinstra. Receipt-free secret-ballot elections (extended abstract). In *Proceedings of the Twenty-sixth Annual ACM Symposium on Theory of Computing*, STOC '94, pages 544–553, New York, NY, USA, 1994. ACM.

[38] Jens Bendisposto and Michael Leuschel. Proof assisted model checking for B. In Karin Breitman and Ana Cavalcanti, editors, *Proceedings of ICFEM 2009*, volume 5885 of *Lecture Notes in Computer Science*, pages 504–520. Springer, 2009.

[39] Béatrice Bérard, Michel Bidoit, Alain Finkel, François Laroussinie, Antoine Petit, Laure Petrucci, and Philippe Schnoebelen. *Systems and software verification : model-checking techniques and tools*. Springer Science & Business Media, 2013.

[40] Yolanta Beresnevichiene. A role and context based security model. Technical Report UCAM-CL-TR-558, University of Cambridge, Computer Laboratory, January 2003.

[41] Matthew A. Bishop. *The Art and Science of Computer Security*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[42] Dines Bjørner. Rôle of domain engineering in software development - why current requirements engineering is flawed! In *Ershov Memorial Conference*, volume 5947 of *Lecture Notes in Computer Science*, pages 2–34. Springer, 2009.

[43] Dines Bjørner. From Domains to Requirements : A Gentle Introduction to Domain Engineering. Lecture Notes, http://www2.imm.dtu.dk/~dibj/2013book.pdf, August 2012.

[44] Dines Bjørner. Manifest Domains : Analysis and Description. Lecture Notes, http://www2.compute.dtu.dk/~dibj/domains/daad-p.pdf, April 2015.

[45] Dines Bjørner. Domain analysis & description - the implicit and explicit semantics problem. In Régine Laleau, Dominique Méry, Shin Nakajima, and Elena Troubitsyna, editors, *Proceedings Joint Workshop on Handling IMPlicit and EXplicit knowledge in formal system development (IMPEX) and Formal and Model-Driven Techniques for Developing Trustworthy Systems (FM&MDD), IMPEX/FM&MDD 2017, and Formal and Model-Driven Techniques for Developing Trustworthy Systems (FM&MDD) Xi'An, China, 16th November 2017.*, volume 271 of *EPTCS*, pages 1–23, 2017.

[46] Dines Bjørner and Asger Eir. Compositionality : Ontology and mereology of domains. In Dennis Dams, Ulrich Hannemann, and Martin Steffen, editors, *Concurrency, Compositionality, and Correctness, Essays in Honor of Willem-Paul de Roever*, volume 5930 of *Lecture Notes in Computer Science*, pages 22–59. Springer, 2010.

[47] Bruno Blanchet. An efficient cryptographic protocol verifier based on prolog rules. In *Proceedings of the 14th IEEE Workshop on Computer Security Foundations*, CSFW '01, pages 82–96, Washington, DC, USA, 2001. IEEE Computer Society.

[48] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. In *LICS*, pages 331–340. IEEE Computer Society, 2005.

[49] Jean Paul Bodeveix, Mamoun Filali, and César A. Muñoz. A Formalization of the B-Method in Coq and PVS. *LNCS*, 1709 :33–49, 1999.

[50] B. W. Boehm. Verifying and validating software requirements and design specifications. *IEEE Softw.*, 1(1) :75–88, January 1984.

[51] Barry W. Boehm. A spiral model of software development and enhancement. *Computer*, 21(5) :61–72, 1988.

[52] W. Bohlken, P. Koopmann, and B. Neumann. *SCENIOR : Ontology-based Interpretation of Aircraft Service Activities*. Bericht. Bibliothek des FB Informatik, 2011.

[53] Pascal Boldini. Formalizing context in intuitionistic type theory. *Fundam. Inform.*, 42(2) :105–127, 2000.

[54] Elena Paslaru Bontas, David Schlangen, and Thomas Schrader. Creating ontologies for content representation-the ontoseed suite. In Robert Meersman, Zahir Tari, Mohand-Said Hacid, John Mylopoulos, Barbara Pernici, Özalp Babaoglu, Hans-Arno Jacobsen, Joseph P. Loyall, Michael Kifer, and Stefano Spaccapietra, editors, *On the Move to Meaningful Internet Systems 2005 : CoopIS, DOA, and ODBASE, OTM Confederated International Conferences, CoopIS, DOA, and ODBASE 2005, Agia Napa, Cyprus, October 31 - November 4, 2005, Proceedings, Part II*, volume 3761 of *Lecture Notes in Computer Science*, pages 1296–1313. Springer, 2005.

[55] Alexander Borgida. On the relative expressiveness of description logics and predicate logics. *Artif. Intell.*, 82(1-2) :353–367, 1996.

[56] Thouraya (épouse. Tebibel) Bouabana. *Une approche multi-formalismes pour la spécification et la vérification des systèmes communicants*. Thèse de doctorat, Université des Sciences et de la Technologie Houari Boumediene- USTHB-Alger Algérie, 2007.

[57] Jonathan P. Bowen and Michael G. Hinchey. Ten commandments of formal methods. *Computer*, 28(4) :56–63, April 1995.

[58] Ronald J. Brachman and Hector J. Levesque. The tractability of subsumption in frame-based description languages. In *AAAI*, pages 34–37. AAAI Press, 1984.

[59] Dirk Brade. *A generalized process for the verification and validation of models and simulation results*. PhD thesis, Universität der Bundeswehr München, Universitätsbibliothek, 2004.

[60] Patrick Brézillon. Context in problem solving : a survey. *Knowledge Eng. Review*, 14(1) :47–80, 1999.

[61] Patrick Brézillon. Hors du contexte, point de salut. *Séminaire" Objets Communicants*, 48 :74–90, 2002.

[62] Patrick Brézillon and Marcos Cavalcanti. Modeling and using context. *Knowledge Eng. Review*, 13(2) :185–194, 1998.

[63] Patrick Brézillon and Charles Tijus. Une representation basee sur le contexte des utilisateurs a travers leurs pratiques. *Extraction et Gestion de Connaissances Paris*, page 12, 2005.

[64] Jérémy Briffaut. *Formalisation et garantie de propriétés de sécurité système : application à la détection d'intrusions. (Formalization and guaranty of system security properties : application to the detection of intrusions)*. Thèse de doctorat, Université d'Orléans, France, 2007.

[65] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3) :560–599, June 1984.

[66] Michael Butler. Decomposition structures for Event-B. In *Integrated Formal Methods*, pages 20–38. Springer, 2009.

[67] Michael J. Butler. csp2b : A practical approach to combining CSP and B. *Formal Asp. Comput.*, 12(3) :182–198, 2000.

[68] Sasa Buvac. Quantificational logic of context. In William J. Clancey and Daniel S. Weld, editors, *Proceedings of the Thirteenth National Conference on Artificial Intelligence and Eighth Innovative Applications of Artificial Intelligence Conference, AAAI 96, IAAI 96, Portland, Oregon, August 4-8, 1996, Volume 1.*, pages 600–606. AAAI Press / The MIT Press, 1996.

[69] Sasa Buvac, Vanja Buvac, and Ian A. Mason. Metamathematics of contexts. *Fundam. Inform.*, 23(2/3/4) :263–301, 1995.

[70] Coral Calero, Francisco Ruiz, and Mario Piattini, editors. *Ontologies for Software Engineering and Software Technology*. Springer, 2006.

[71] Verónica Castañeda, Luciana Ballejos, Ma. Laura Caliusco, and Ma. Rosa Galli. The use of ontologies in requirements engineering. *Global Journal of Research In Engineering*, 10(6), 2010.

[72] Samuel W. K. Chan and James Franklin. Dynamic context generation for natural language understanding : a multifaceted knowledge approach. *IEEE Trans. Systems, Man, and Cybernetics, Part A*, 33(1) :23–41, 2003.

[73] K. Mani Chandy and Jayadev Misra. *Parallel program design - a foundation*. Addison-Wesley, 1989.

[74] Vincent Chapurlat. *Vérification et validation de modèles de systèmes complexes : application à la Modélisation d'Entreprise*. Habilitation à diriger des recherches, Université Montpellier II - Sciences et Techniques du Languedoc, March 2007.

[75] Bachir Chihani, Emmanuel Bertin, Fabrice Jeanne, and Noël Crespi. Context-aware systems : A case study. In Hocine Cherifi, Jasni Mohamad Zain, and Eyas El-Qawasmeh, editors, *Digital Information and Communication Technology and Its Applications - International Conference, DICTAP 2011, Dijon, France, June 21-23, 2011, Proceedings, Part II*, volume 167 of *Communications in Computer and Information Science*, pages 718–732. Springer, 2011.

[76] Alessandro Cimatti, Edmund M. Clarke, Enrico Giunchiglia, Fausto Giunchiglia, Marco Pistore, Marco Roveri, Roberto Sebastiani, and Armando Tacchella. Nusmv 2 : An opensource tool for symbolic model checking. In Ed Brinksma and Kim Guldstrand Larsen, editors, *Computer Aided Verification, 14th International Conference, CAV 2002,Copenhagen, Denmark, July 27-31, 2002, Proceedings*, volume 2404 of *Lecture Notes in Computer Science*, pages 359–364. Springer, 2002.

[77] E. M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Program. Lang. Syst.*, 8(2) :244–263, April 1986.

[78] Edmund M. Clarke and E. Allen Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Logic of Programs, Workshop*, pages 52–71, Berlin, Heidelberg, 1982. Springer-Verlag.

[79] M. Clavel, F. Dur án, S. Eker, P. Lincoln, N. Mart í Oliet, J. Meseguer, and J.F. Quesada. Maude : specification and programming in rewriting logic. *Theoretical Computer Science*, 285(2) :187 – 243, 2002. Rewriting Logic and its Applications.

[80] Election Assistance Commission. Voluntary Voting System Guidelines. Volume I : Version 1.0, United States, 2005.

[81] Venice Commission et al. European commission for democracy through law. *Report on the Democratic Oversight of the Securiy Services. Adopted by the Venice Commission at its*, 71, 2004.

[82] Robin Cooper. Records and record types in semantic theory. *Journal of Logic and Computation*, 15(2) :99–112, 2005.

[83] Véronique Cortier, Georg Fuchsbauer, and David Galindo. BeleniosRF : A strongly receipt-free electronic voting scheme. *IACR Cryptology ePrint Archive*, 2015 :629, 2015.

[84] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachène. A generic construction for voting correctness at minimum cost - application to helios. *IACR Cryptology ePrint Archive*, 2013 :177, 2013.

[85] Véronique Cortier, David Galindo, Stéphane Glondu, and Malika Izabachene. Election verifiability for helios under weaker trust assumptions. In *Proceedings of the 19th European Symposium on Research in Computer Security (ESORICS'14)*, volume 8713 of *LNCS*, pages 327–344, Wroclaw, Poland, September 2014. Springer.

[86] Lorrie Faith Cranor. Electronic voting : computerized polls may save money, protect privacy. *Crossroads*, 2(4) :12–16, 1996.

[87] Lorrie Faith Cranor and Ron K Cytron. *Design and implementation of a practical security-conscious electronic polling system*. Washington University, Department of Computer Science, 1996.

[88] Judith Crow and Ben Di Vito. Formalizing space shuttle software requirements : Four case studies. *ACM Trans. Softw. Eng. Methodol.*, 7(3) :296–332, July 1998.

[89] James L. Crowley, Joëlle Coutaz, Gaëtan Rey, and Patrick Reignier. Perceptual components for context aware computing. In Gaetano Borriello and Lars Erik Holmquist, editors, *UbiComp 2002 : Ubiquitous Computing, 4th International Conference, Göteborg, Sweden, September 29 - October 1, 2002, Proceedings*, volume 2498 of *Lecture Notes in Computer Science*, pages 117–134. Springer, 2002.

[90] Aba-Sah Dadzie, Ravish Bhagdev, Ajay Chakravarthy, Sam Chapman, José Iria, Vitaveska Lanfranchi, João Magalhães, Daniela Petrelli, and Fabio Ciravegna. Applying semantic web technologies to knowledge sharing in aerospace engineering. *J. Intelligent Manufacturing*, 20(5) :611–623, 2009.

[91] Kriangsak Damchoom and Michael Butler. Applying event and machine decomposition to a flash-based filestore in Event-B. In *Formal Methods : Foundations and Applications*, pages 134–152. Springer, 2009.

[92] Richard Dapoigny and Patrick Barlatier. Modeling contexts with dependent types. *Fundam. Inform.*, 104(4) :293–327, 2010.

[93] Randall Davis, Howard E. Shrobe, and Peter Szolovits. What is a knowledge representation ? *AI Magazine*, 14(1) :17–33, 1993.

[94] David Déharbe, Pascal Fontaine, Yoann Guyot, and Laurent Voisin. SMT solvers for rodin. In John Derrick, John S. Fitzgerald, Stefania Gnesi, Sarfraz Khurshid, Michael Leuschel, Steve Reeves, and Elvinia Riccobene, editors, *Abstract State Machines, Alloy, B, VDM, and Z - Third International Conference, ABZ 2012, Pisa, Italy, June 18-21, 2012. Proceedings*, volume 7316 of *Lecture Notes in Computer Science*, pages 194–207. Springer, 2012.

[95] Stéphanie Delaune and Steve Kremer. Spécificités des protocoles de vote électronique. Deliverable AVOTE 1.1 (ANR-07-SESU-002), January 2009. 8 pages.

[96] Stéphanie Delaune, Steve Kremer, and Mark Ryan. Verifying privacy-type properties of electronic voting protocols. *J. Comput. Secur.*, 17(4) :435–487, December 2009.

[97] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Coercion-resistance and receipt-freeness in electronic voting. In *Proceedings of the 19th IEEE Computer Security Foundations Workshop (CSFW'06)*, pages 28–39, Venice, Italy, July 2006. IEEE Computer Society Press.

[98] Guy Van den Broeck. *Lifted Inference and Learning in Statistical Relational Models (Eerste-orde inferentie en leren in statistische relationele modellen)*. PhD thesis, Katholieke Universiteit Leuven, Belgium, 2013.

[99] The Coq development team. *The Coq proof assistant reference manual*. LogiCal Project, 2004. Version 8.0.

[100] Keith Devlin. *Logic and information*. Cambridge University Press, 1991.

[101] Anind K Dey. *Providing architectural support for building context-aware applications*. PhD thesis, Georgia Institute of Technology, 2000.

[102] Anind K. Dey. Understanding and using context. *Personal Ubiquitous Comput.*, 5(1) :4–7, January 2001.

[103] Philippe Dhaussy and Frédéric Boniol. Mise en œuvre de composants MDA pour la validation formelle de modèles de systèmes d'information embarqués. *Ingénierie des Systèmes d'Information*, 12(5) :133–157, 2007.

[104] Philippe Dhaussy, Frédéric Boniol, Jean-Charles Roger, and Luka Leroux. Improving model checking with context modelling. *Adv. Software Engineering*, 2012 :547157 :1–547157 :13, 2012.

[105] P. Dockhorn Costa, J.P. Andrade Almeida, L. Ferreira Pires, G. Guizzardi, and M.J. van Sinderen. Towards conceptual foundations for context-aware applications. In T.R. Roth-Berghofer, S. Schulz, and D.B. Leake, editors, *AAAI Workshop on Modeling and Retrieval of Context 2006*, volume WS-06- of *AAAI Technical Report*, pages 54–58, Menlo Park, CA, USA, 2006. AAAI Press.

[106] Danny Dolev and Andrew Chi-Chih Yao. On the security of public key protocols. *IEEE Trans. Information Theory*, 29(2) :198–207, 1983.

[107] Méry Dominique. Modèles et algorithmes, 2015. http://www.loria.fr/~mery/malg/partie1-20142015.pdf.

[108] Francesco M. Donini, Maurizio Lenzerini, Daniele Nardi, and Andrea Schaerf. Deduction in concept languages : From subsumption to instance checking. *J. Log. Comput.*, 4(4) :423–452, 1994.

[109] Paul Dourish. Seeking a foundation for context-aware computing. *Human-Computer Interaction*, 16(2-4) :229–241, 2001.

[110] Doron Drusinsky. The temporal rover and the atg rover. In *Proceedings of the 7th International SPIN Workshop on SPIN Model Checking and Software Verification*, pages 323–330, London, UK, UK, 2000. Springer-Verlag.

[111] Bruno Borlini Duarte, André Luiz de Castro Leal, Ricardo de Almeida Falbo, Giancarlo Guizzardi, Renata S. S. Guizzardi, and Vítor E. Silva Souza. Ontological foundations for software requirements with a focus on requirements at runtime. *Applied Ontology*, 13(2) :73–105, 2018.

[112] Steve Easterbrook. Verification and validation of requirements for mission critical systems. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 673–674, New York, NY, USA, 1999. ACM.

[113] Egidio Astesiano and Michel Bidoit and Hélène Kirchner and Bernd Krieg-Brckner and Peter D. Mosses and Donald Sannella and Andrzej Tarlecki. Casl : the common algebraic specification language. *Theoretical Computer Science*, 286(2) :153 – 196, 2002. Current trends in Algebraic Development Techniques.

[114] Hartmut Ehrig and Bernd Mahr. *Fundamentals of Algebraic Specification 1 : Equations und Initial Semantics*, volume 6 of *EATCS Monographs on Theoretical Computer Science*. Springer, 1985.

[115] Dejene Ejigu, Vasile-Marian Scuturici, and Lionel Brunie. An ontology-based approach to context modeling and reasoning in pervasive computing. In *Fifth Annual IEEE International Conference on Pervasive Computing and Communications - Workshops (PerCom Workshops 2007), 19-23 March 2007, White Plains, New York, USA*, pages 14–19. IEEE Computer Society, 2007.

[116] Jérôme Euzenat, Jérôme Pierson, and Fano Ramparany. Dynamic context management for pervasive applications. *Knowledge Engineering Review*, 23(1) :21–49, 2008.

[117] Dieter Fensel. Formal specification languages in knowledge and software engineering. *Knowledge Eng. Review*, 10(4) :361–404, 1995.

[118] Sébastien Ferré. *Systèmes d'information logiques : un paradigme logico-contextuel pour interroger, naviguer et apprendre*. Thèse de doctorat, Université de Rennes1, oct 2002.

[119] Anthony CW Finkelstein, Dov Gabbay, Anthony Hunter, Jeff Kramer, and Bashar Nuseibeh. Inconsistency handling in multiperspective specifications. *Software Engineering, IEEE Transactions on*, 20(8) :569–578, 1994.

[120] R. W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Proc. Symp. Appl. Math. 19, Mathematical Aspects of Computer Science*, volume 19, pages 19 – 32. American Mathematical Society, 1967.

[121] Pierre-Alain Fouque. *Le partage de clés cryptographiques : Théorie et Pratique*. Thèse de doctorat, Université Paris 7, France, 2001.

[122] Igor Nai Fovino and Marcelo Masera. Through the description of attacks : A multidimensional view. In Janusz Górski, editor, *Computer Safety, Reliability, and Security, 25th International Conference, SAFECOMP 2006, Gdansk, Poland, September 27-29, 2006, Proceedings*, volume 4166 of *Lecture Notes in Computer Science*, pages 15–28. Springer, 2006.

[123] Frederick K. Frantz. A taxonomy of model abstraction techniques. In *Proceedings of the 27th Conference on Winter Simulation*, WSC '95, pages 1413–1420, Washington, DC, USA, 1995. IEEE Computer Society.

[124] Joshua Fried, Pierrick Gaudry, Nadia Heninger, and Emmanuel Thomé. A kilobit hidden SNFS discrete logarithm computation. In *EUROCRYPT (1)*, volume 10210 of *Lecture Notes in Computer Science*, pages 202–231, 2017.

[125] Atsushi Fujioka, Tatsuaki Okamoto, and Kazuo Ohta. A practical secret voting scheme for large scale elections. In Jennifer Seberry and Yuliang Zheng, editors, *Advances in Cryptology - AUSCRYPT '92, Workshop on the Theory and Application of Cryptographic Techniques, Gold Coast, Queensland, Australia, December 13-16, 1992, Proceedings*, volume 718 of *Lecture Notes in Computer Science*, pages 244–251. Springer, 1992.

[126] Bernhard Ganter and Rudolf Wille. *Formal concept analysis - mathematical foundations*. Springer, 1999.

[127] Marie-Claude Gaudel. Structuring and modularizing algebraic specifications : The PLUSS specification language, evolutions and perspectives. In *STACS*, volume 577 of *Lecture Notes in Computer Science*, pages 3–18. Springer, 1992.

[128] J. Paul Gibson, Souad Kherroubi, and Dominique Méry. Applying a Dependency Mechanism for Voting Protocol Models Using Event-B. In Ahmed Bouajjani and Alexandra Silva, editors, *Formal Techniques for Distributed Objects, Components, and Systems - 37th IFIP WG 6.1 International Conference, FORTE 2017, Held as Part of the 12th International Federated Conference on Distributed Computing Techniques, DisCoTec 2017, Neuchâtel, Switzerland, June 19-22, 2017, Proceedings*, volume 10321 of *Lecture Notes in Computer Science*, pages 124–138. Springer, 2017.

[129] J. Paul Gibson and Jean-Luc Raffy. Vote types : Generic modelling of functional requirements in impex projet (spl - e-voting), 2015.

[130] Fausto Giunchiglia and Chiara Ghidini. Local models semantics, or contextual reasoning = locality + compatibility. In Anthony G. Cohn, Lenhart K. Schubert, and Stuart C. Shapiro, editors, *Proceedings of the Sixth International Conference on Principles of Knowledge Representation and Reasoning (KR'98), Trento, Italy, June 2-5, 1998.*, pages 282–291. Morgan Kaufmann, 1998.

[131] Martin Glas. *Ontology-based Model Integration for the Conceptual Design of Aircraft*. PhD thesis, Technical University Munich, 2013.

[132] Robert Godin, Rokia Missaoui, and Alain April. Experimental comparison of navigation in a galois lattice with conventional information retrieval methods. *International Journal of Man-Machine Studies*, 38(5) :747–767, 1993.

[133] Carla P. Gomes, Ashish Sabharwal, and Bart Selman. Model counting. In Armin Biere, Marijn Heule, Hans van Maaren, and Toby Walsh, editors, *Handbook of Satisfiability*, volume 185 of *Frontiers in Artificial Intelligence and Applications*, pages 633–654. IOS Press, 2009.

[134] A. Gómez-Pérez. Ontological engineering : A state of the art. *Expert Update : Knowledge Based Systems and Applied Artificial Intelligence*, 2(3) :33–43, 1999. Ontology Engineering Group ? OEG.

[135] Ali Gondal, Michael Poppleton, and Michael Butler. Composing Event-B specifications-case-study experience. In *Software Composition*, pages 100–115. Springer, 2011.

[136] M. J. C. Gordon and T. F. Melham, editors. *Introduction to HOL : A Theorem Proving Environment for Higher Order Logic*. Cambridge University Press, New York, NY, USA, 1993.

[137] Zeineb Graja. *Vérification formelle des systèmes multi-agents auto-adaptatifs. (Formal verification of self-adaptive multi-agent systems)*. Thèse de doctorat, Paul Sabatier University, Toulouse, France, 2015.

[138] RAISE Language Group and Anne Elisabeth Haxthausen. *The RAISE Specification Language*. Prentice Hall Int., 1992.

[139] Thomas R Gruber. Toward principles for the design of ontologies used for knowledge sharing ? *International journal of human-computer studies*, 43(5) :907–928, 1995.

[140] Nicola Guarino and Christopher Welty. Identity, unity, and individuality : Towards a formal toolkit for ontological analysis. In *Proceedings of the 14th European Conference on Artificial Intelligence*, ECAI'00, pages 219–223, Amsterdam, The Netherlands, The Netherlands, 2000. IOS Press.

[141] Nicola Guarino and Christopher Welty. Identity and subsumption. In *The Semantics of Relationships*, pages 111–126. Springer, 2002.

[142] Nicola Guarino and Christopher A. Welty. A formal ontology of properties. In *Proceedings of the 12th European Workshop on Knowledge Acquisition, Modeling and Management*, EKAW '00, pages 97–112, London, UK, UK, 2000. Springer-Verlag.

[143] G. Guizzardi and Giancarlo Guizzardi. *Ontological foundations for structural conceptual models*. PhD thesis, University of Twente, 10 2005.

[144] Giancarlo Guizzardi and Gerd Wagner. Towards an ontological foundation of discrete event simulation. In *Proceedings of the 2010 Winter Simulation Conference, WSC 2010, Baltimore, Maryland, USA, 5-8 December 2010*, pages 652–664. WSC, 2010.

[145] Cecilia Haskins, Kevin Forsberg, Michael Krueger, D Walden, and D Hamelin. Systems engineering handbook. In *INCOSE,*, 2006.

[146] Klaus Havelund, Mike Lowry, SeungJoon Park, Charles Pecheur, John Penix, Willem Visser, Jonathan White, et al. Formal analysis of the remote agent before and after flight. In *Proceedings of the 5th NASA Langley Formal Methods Workshop*, volume 134, 2000.

[147] Thai Son Hoang and Jean-Raymond Abrial. Reasoning about liveness properties in Event-B. In Shengchao Qin and Zongyan Qiu, editors, *Formal Methods and Software Engineering - 13th International Conference on Formal Engineering Methods, ICFEM 2011, Durham, UK, October 26-28, 2011. Proceedings*, volume 6991 of *Lecture Notes in Computer Science*, pages 456–471. Springer, 2011.

[148] Thai Son Hoang, Andreas Fürst, and Jean-Raymond Abrial. Event-B patterns and their tool support. In *Proceedings of the 2009 Seventh IEEE International Conference on Software Engineering and Formal Methods*, SEFM '09, pages 210–219, Washington, DC, USA, 2009. IEEE Computer Society.

[149] Thai Son Hoang, Andreas Fürst, and Jean-Raymond Abrial. Event-B patterns and their tool support. *Software and System Modeling*, 12(2) :229–244, 2013.

[150] ThaiSon Hoang and Jean-Raymond Abrial. Event-b decomposition for parallel programs. In Marc Frappier, Uwe Glässer, Sarfraz Khurshid, Régine Laleau, and Steve Reeves, editors, *Abstract State Machines, Alloy, B and Z*, volume 5977 of *Lecture Notes in Computer Science*, pages 319–333. Springer Berlin Heidelberg, 2010.

[151] C. A. R. Hoare. An axiomatic basis for computer programming. *Communications of the ACM*, 12(10) :576–580, October 1969.

[152] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1985.

[153] Harri Hursti. EVEREST : Evaluation and Validation of Election-Related Equipment, Standards and Testing. Final Report December 7. Technical report, 2007.

[154] Michael Jackson. *Software Requirements &Amp ; Specifications : A Lexicon of Practice, Principles and Prejudices*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995.

[155] William James. *The will to believe and other essays*. Longmans, Green, and Co, 1897.

[156] Matti Jarvisalo, Daniel Le Berre, Olivier Roussel, and Laurent Simon. The International SAT Solver Competitions. *Artificial Intelligence Magazine (AI Magazine)*, 1(33) :89–94, 2012.

[157] Stéphane Jean. *OntoQL, un langage d'exploitation des bases de données à base ontologique. (OntoQL, an exploitation language for ontology-based databases)*. Thèse de doctorat, Université de Poitiers, France, 2007.

[158] Einar Broch Johnsen and Luigia Petre, editors. *Integrated Formal Methods, 10th International Conference, IFM 2013, Turku, Finland, June 10-14, 2013. Proceedings*, volume 7940 of *Lecture Notes in Computer Science*. Springer, 2013.

[159] Cliff B. Jones. *Systematic Software Development Using VDM (2Nd Ed.)*. Prentice-Hall, Inc., Upper Saddle River, New Jersey, USA, 1990.

[160] H. L. Jonker and W. Pieters. Receipt-freeness as a special case of anonymity in epistemic logic. In *Proc. Workshop On Trustworthy Elections (WOTE'06)*, June 2006.

[161] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In *Proceedings of the 2005 ACM Workshop on Privacy in the Electronic Society*, WPES '05, pages 61–70, New York, NY, USA, 2005. ACM.

[162] Haruhiko Kaiya and Motoshi Saeki. Using domain ontology as domain knowledge for requirements elicitation. In *Proceedings of the 14th IEEE International Requirements Engineering Conference*, RE '06, pages 186–195, Washington, DC, USA, 2006. IEEE Computer Society.

[163] Daniel Kayser. *La représentation des connaissances*. Hermès, 1997.

[164] Souad Kherroubi. La contextualisation en Event-B. Technical report, LORIA-Université de Lorraine, (2016).

[165] Souad Kherroubi and Dominique Méry. Contextualization and dependency in state-based modelling - application to Event-B. In Ouhammou et al. [207], pages 137–152.

[166] Michael Kifer, Georg Lausen, and James Wu. Logical foundations of object-oriented and frame-based languages. *J. ACM*, 42(4) :741–843, July 1995.

[167] Ralf Klischewski. Semantic web for e-government. In Roland Traunmüller, editor, *Electronic Government, Second International Conference, EGOV 2003, Prague, Czech Republic, September 1-5, 2003, Proceedings*, volume 2739 of *Lecture Notes in Computer Science*, pages 288–295. Springer, 2003.

[168] Anders Kofod-Petersen and Jörg Cassens. Using activity theory to model context awareness. In Thomas Roth-Berghofer, Stefan Schulz, and David B. Leake, editors, *Modeling and Retrieval of Context, Second International Workshop, MRC 2005, Edinburgh, UK, July 31 - August 1, 2005, Revised Selected Papers*, volume 3946 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 2005.

[169] Gerald Kotonya and Ian Sommerville. *Requirements Engineering : Processes and Techniques*. Wiley Publishing, 1st edition, 1998.

[170] Steve Kremer and Mark Ryan. Analysis of an electronic voting protocol in the applied pi calculus. In Shmuel Sagiv, editor, *Programming Languages and Systems, 14th European Symposium on Programming,ESOP 2005, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2005, Edinburgh, UK, April 4-8, 2005, Proceedings*, volume 3444 of *Lecture Notes in Computer Science*, pages 186–200. Springer, 2005.

[171] Ernest Jackson Kuofie. Radex : A rationale-based ontology for aerospace design explanation. Master's thesis, University of Twente, 2010.

[172] Patricia Lago, Henry Muccini, and Hans van Vliet. A scoped approach to traceability management. *Journal of Systems and Software*, 82(1) :168–182, 2009.

[173] Leslie Lamport. "sometime" is sometimes "not never" : On the temporal logic of programs. In *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '80, pages 174–185, New York, NY, USA, 1980. ACM.

[174] Leslie Lamport. The temporal logic of actions. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3) :872–923, 1994.

[175] Jean-Claude Laprie, Jean Arlat, JP Blanquart, A Costes, Y Crouzet, Y Deswarte, JC Fabre, H Guillermain, M Kaâniche, K Kanoun, et al. *Guide de la sûreté de fonctionnement*. Cépaduès éditions, 1995.

[176] Kim Guldstrand Larsen, Paul Pettersson, and Wang Yi. UPPAAL : status & developments. In *CAV*, volume 1254 of *Lecture Notes in Computer Science*, pages 456–459. Springer, 1997.

[177] Douglas B. Lenat, Ramanathan V. Guha, Karen Pittman, Dexter Pratt, and Mary Shepherd. CYC : toward programs with common sense. *Commun. ACM*, 33(8) :30–49, 1990.

[178] Tong Li and John Mylopoulos. Modeling and applying security patterns using contextual goal models. In Fabiano Dalpiaz and Jennifer Horkoff, editors, *Proceedings of the Seventh International i* Workshop co-located with the 26th International Conference on Advanced Information Systems Engineering (CAiSE 2014), Thessaloniki, Greece, June 16-17, 2014.*, volume 1157 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2014.

[179] David Lightfoot. *Formal specification using Z.* Macmillan Press, 1991.

[180] Christian Lindig et al. Concept-based component retrieval. In *Working notes of the IJCAI-95 workshop : Formal Approaches to the Reuse of Plans, Proofs, and Programs*, pages 21–25, 1995.

[181] Zhaohui Luo. A unifying theory of dependent types : The schematic approach. In Anil Nerode and Michael A. Taitslin, editors, *Logical Foundations of Computer Science - Tver '92, Second International Symposium, Tver, Russia, July 20-24, 1992, Proceedings*, volume 620 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 1992.

[182] Simon Malenfant-Corriveau. *Proposition d'une méthode de développement d'ontologie pour un système expert en sécurité.* Thèse de doctorat, École Polytechnique de Montréal, 2017.

[183] John McCarthy. Notes on formalizing context. In *Proceedings of the 13th International Joint Conference on Artifical Intelligence - Volume 1*, IJCAI'93, pages 555–560, San Francisco, CA, USA, 1993. Morgan Kaufmann Publishers Inc.

[184] Annabelle McIver and Carroll Morgan. *Abstraction, Refinement and Proof for Probabilistic Systems.* Monographs in Computer Science. Springer, 2005.

[185] Carlo Meghini, Fabrizio Sebastiani, Umberto Straccia, and Costantino Thanos. A model of information retrieval based on a terminological logic. In Robert Korfhage, Edie M. Rasmussen, and Peter Willett, editors, *Proceedings of the 16th Annual International ACM-SIGIR Conference on Research and Development in Information Retrieval. Pittsburgh, PA, USA, June 27 - July 1, 1993*, pages 298–307. ACM, 1993.

[186] Jean-Pierre Meinadier. *Ingénierie et intégration des systèmes.* Hermes Paris, 1998.

[187] Tim Menzies and Charles Pecheur. Verification and validation and artificial intelligence. *Advances in Computers*, 65 :154–203, 2005.

[188] Dominique Méry and Michael Poppleton. Formal modelling and verification of population protocols. In Johnsen and Petre [158], pages 208–222.

[189] Dominique Méry and Michael Poppleton. Towards an integrated formal method for verification of liveness properties in distributed systems : with application to population protocols. *Software and System Modeling*, 16(4) :1083–1115, 2017.

[190] Dominique Méry, Sawant Rushikesh, and Anton Tarasyuk. Integrating Domain-Based Features into Event-B : a Nose Gear Velocity Case Study. In Ladjel Bellatreche and Yannis Manolopoulos, editors, *Model and Data Engineering - 5th International Conference, MEDI 2015*, volume lncs 9344 of *Model and Data Engineering - 5th International Conference, MEDI 2015*, pages 89–102, Rhodes, Greece, September 2015. springer.

[191] Pedro Meseguer. Verification of multi-level rule-based expert systems. In *Proceedings of the Ninth National Conference on Artificial Intelligence - Volume 1*, AAAI'91, pages 323–328. AAAI Press, 1991.

[192] R. Milner. *Communication and Concurrency.* Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[193] Andreas Mitschke et al. Crystal public aerospace use case development. Report — version 3, EADS IW G - Airbus Group Innovations Germany, 2016. EADS IW G — Airbus Group Innovations Germany, Deliverable D208.903.

[194] Linda Mohand-Oussaïd and Idir Aït-Sadoune. Formal modelling of domain constraints in Event-B. In Ouhammou et al. [207], pages 153–166.

[195] Carroll Morgan, Thai Son Hoang, and Jean-Raymond Abrial. The challenge of probabilistic *Event B* - extended abstract. In Helen Treharne, Steve King, Martin C. Henson, and Steve A. Schneider, editors, *ZB 2005 : Formal Specification and Development in Z and B, 4th International Conference of B and Z Users, Guildford, UK, April 13-15, 2005, Proceedings*, volume 3455 of *Lecture Notes in Computer Science*, pages 162–171. Springer, 2005.

[196] Ghita Kouadri Mostéfaoui and Patrick Brézillon. Modeling context-based security policies with contextual graphs. In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, PERCOMW '04, pages 28–, Washington, DC, USA, 2004. IEEE Computer Society.

[197] Michael Mrissa. *Médiation Sémantique Orientée Contexte pour la Composition de Services Web.* Thèse de doctorat, University of Claude Bernard Lyon I, November 2007.

[198] Geoff P Mullery. Core-a method for controlled requirement specification. In *Proceedings of the 4th international conference on Software engineering*, pages 126–135. IEEE Press, 1979.

[199] Suvda Myagmar, Adam J Lee, and William Yurcik. Threat modeling as a basis for security requirements. In *Symposium on requirements engineering for information security (SREIS)*, volume 2005, pages 1–8. Citeseer, 2005.

[200] John Mylopoulos. Information modeling in the time of the revolution. *Information systems*, 23(3) :127–155, 1998.

[201] Arnold Neumaier and Flaviu Adrian Mărginean. Logic in context, 2011.

[202] Robert Nieuwenhuis and Albert Rubio. Paramodulation-based theorem proving. In *Handbook of Automated Reasoning*, pages 371–443. Elsevier and MIT Press, 2001.

[203] Tobias Nipkow and Lawrence C. Paulson. Isabelle-91. In Deepak Kapur, editor, *Automated Deduction - CADE-11, 11th International Conference on Automated Deduction, Saratoga Springs, NY, USA, June 15-18, 1992, Proceedings*, volume 607 of *Lecture Notes in Computer Science*, pages 673–676. Springer, 1992.

[204] Explanatory Note. European commission for democracy through law (venice commission).

[205] Bashar Nuseibeh, Jeff Kramer, and Anthony Finkelstein. A framework for expressing the relationships between multiple views in requirements specification. *Software Engineering, IEEE Transactions on*, 20(10) :760–773, 1994.

[206] Standards Coordinating Committee of the IEEE Computer Society. Ieee standard glossary of software engineering terminology. Technical report, IEEE, 1990. International Electronic and Electrical Engineers Standard.

[207] Yassine Ouhammou, Mirjana Ivanovic, Alberto Abelló, and Ladjel Bellatreche, editors. *Model and Data Engineering - 7th International Conference, MEDI 2017, Barcelona, Spain, October 4-6, 2017, Proceedings*, volume 10563 of *Lecture Notes in Computer Science*. Springer, 2017.

[208] S. Owre, S. Rajan, J.M. Rushby, N. Shankar, and M.K. Srivas. PVS : combining specification, proof checking, and model checking. In Rajeev Alur and Thomas A. Henzinger, editors, *Computer-Aided Verification, CAV '96*, number 1102 in Lecture Notes in Computer Science, pages 411–414, New Brunswick, NJ, July/August 1996. Springer-Verlag.

[209] Sam Owre, S. Rajan, John M. Rushby, Natarajan Shankar, and Mandayam K. Srivas. Pvs : Combining specification, proof checking, and model checking. In *Proceedings of the 8th International Conference on Computer Aided Verification*, CAV '96, pages 411–414, London, UK, UK, 1996. Springer-Verlag.

[210] Georgios Petasis, Ralf Möller, and Vangelis Karkaletsis. BOEMIE : reasoning-based information extraction. In *NLPAR@LPNMR*, volume 1044 of *CEUR Workshop Proceedings*, pages 60–75. CEUR-WS.org, 2013.

[211] Amir Pnueli. The temporal semantics of concurrent programs. *Theoretical Computer Science*, 13(1) :45 – 60, 1981. Special Issue Semantics of Concurrent Computation.

[212] Robert Pollack. Dependently typed records for representing mathematical structure. In Mark Aagaard and John Harrison, editors, *Theorem Proving in Higher Order Logics, 13th International Conference, TPHOLs 2000, Portland, Oregon, USA, August 14-18, 2000, Proceedings*, volume 1869 of *Lecture Notes in Computer Science*, pages 462–479. Springer, 2000.

[213] Michael Poppleton. The composition of Event-B models. In *Abstract State Machines, B and Z*, pages 209–222. Springer, 2008.

[214] Susanne Prediger. Simple concept graphs : A logic approach. In Marie-Laure Mugnier and Michel Chein, editors, *Conceptual Structures : Theory, Tools and Applications, 6th International Conference on Conceptual Structures, ICCS '98, Montpellier, France, August 10-12, 1998, Proceedings*, volume 1453 of *Lecture Notes in Computer Science*, pages 225–239. Springer, 1998.

[215] Alun D. Preece. Verification, validation, and test of knowledge-based systems. *AI Magazine*, 13(4) :77, 1992.

[216] Alun D. Preece, Rajjan Shinghal, and Aïda Batarekh. Principles and practice in verifying rule-based systems. *Knowledge Eng. Review*, 7(2) :115–141, 1992.

[217] Ghassan Z. Qadah. Requirements, design and implementation of an e-voting system. In *IADIS AC*, pages 405–409. IADIS, 2005.

[218] Jean-Pierre Queille. The CESAR system : An aided design and certification system. In *Proceedings of the 2nd International Conference on Distributed Computing Systems, Paris, France, 1981*, pages 149–161. IEEE Computer Society, 1981.

[219] Q W. Van Orman Quine. *Du Point de Vue Logique*. Vrin., Paris, 1953/2003.

[220] Amine Raji. *Intégration des activités de preuve dans le processus de développement de logiciels pour les systèmes embarqués*. Thèse de doctorat, Télécom Bretagne, Université de Bretagne-Sud, 2012.

[221] REVVA. *VV&A methodological guidelines Reference Manual.* . Common Validation, Verification and Accreditatation Framework for Simulation, 2004.

[222] François Rioult. *Extraction de connaissances dans les bases de donn'ees comportant des valeurs manquantes ou un grand nombre d'attributs*. Thèse de doctorat, University of Caen Normandy, France, 2005.

[223] John Alan Robinson. A machine-oriented logic based on the resolution principle. *J. ACM*, 12(1) :23–41, 1965.

[224] Jean-Charles Roger. *Exploitation de contextes et d'observateurs pour la validation formelle de modèles*. Thèse de doctorat, ENST Bretagne, Université de Rennes I, 2006. Thèse de doctorat dirigée par Terrier, François.

[225] Winston W Royce. Managing the development of large software systems : concepts and techniques. In *Proceedings of the 9th international conference on Software Engineering*, pages 328–338. IEEE Computer Society Press, 1987.

[226] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[227] Driss Sadoun. *Des spécifications en langage naturel aux spécifications formelles via une ontologie comme modèle pivot. (From natural language specifications to formal specifications via an ontology as a pivot model)*. Thèse de doctorat, Université Paris-Sud, Orsay, France, 2014.

[228] Gwen Salaün, Michel Allemand, and Christian Attiogbé. Specification of an access control system with a formalism combining CCS and CASL. In *IPDPS*. IEEE Computer Society, 2002.

[229] Sigrid Schefer-Wenzl and Mark Strembeck. Modeling context-aware RBAC models for business processes in ubiquitous computing environments. In *Third FTRA International Conference on Mobile, Ubiquitous, and Intelligent Computing, MUSIC 2012, Vancouver, Canada, June 26-28, 2012*, pages 126–131. IEEE, 2012.

[230] Steve Schneider and Abraham Sidiropoulos. Csp and anonymity. In *Proceedings of the 4th European Symposium on Research in Computer Security : Computer Security*, ESORICS '96, pages 198–218, London, UK, UK, 1996. Springer-Verlag.

[231] Steve A. Schneider, Helen Treharne, Heike Wehrheim, and David M. Williams. Managing LTL properties in Event-B refinement. In *Integrated Formal Methods*, volume 8739 of *Lecture Notes in Computer Science*, pages 221–237. Springer, 2014.

[232] A Th Schreiber, Guus Schreiber, Hans Akkermans, Anjo Anjewierden, Nigel Shadbolt, Robert de Hoog, Walter Van de Velde, and Bob Wielinga. *Knowledge engineering and management : the CommonKADS methodology*. MIT press, 2000.

[233] Sally Shlaer and Stephen J. Mellor. *Object Lifecycles : Modeling the World in States*. Yourdon Press, Upper Saddle River, NJ, USA, 1992.

[234] Neeraj Kumar Singh. *Reliability and Safety of Critical Device Software Systems. (Fiabilité et sûreté des systèmes informatiques critiques)*. Thèse de doctorat, Université Henri Poincaré, Nancy, France, 2011.

[235] Gregor Snelting. Concept analysis - A new framework for program understanding. In Thomas Ball, Frank Tip, and A. Michael Berman, editors, *Proceedings of the SIGPLAN/SIGSOFT Workshop on Program Analysis For Software Tools and Engineering, PASTE '98, Montreal, Canada, June 16, 1998*, pages 1–10. ACM, 1998.

[236] Amina Souag. Vers une nouvelle génération de définition des exigences de sécurité fondée sur l'utilisation des ontologies. In *Actes du XXXème Congrès INFORSID, Montpellier, France, 29 - 31 mai 2012*, pages 583–590, 2012.

[237] John F. Sowa. *Knowledge Representation : Logical, Philosophical and Computational Foundations*. Brooks/Cole Publishing Co., Pacific Grove, CA, USA, 2000.

[238] John F. Sowa. *The Role of Logic and Ontology in Language and Reasoning*, pages 231–263. Springer Netherlands, Dordrecht, 2010.

[239] Ljiljana Stojanovic, Nenad Stojanovic, and Dimitris Apostolou. Change management in e-government : Ontogov case study. *EG*, 3(1) :74–92, 2006.

[240] Rudi Studer, V. Richard Benjamins, and Dieter Fensel. Knowledge engineering : Principles and methods. *Data and knowledge engineering*, 25(1-2) :161–197, 1998.

[241] Alistair G. Sutcliffe, Stephen Fickas, and McKay Moore Sohlberg. PC-RE : a method for personal and contextual requirements engineering with some experience. *Requir. Eng.*, 11(3) :157–173, 2006.

[242] Mehdi Talbi. *Spécification et vérification automatique des propriétés des protocoles de vote électronique en utilisant la logique ADM*. Thèse de doctorat, Université Rennes 1, 2010.

[243] Ben H Thacker, Scott W Doebling, Francois M Hemez, Mark C Anderson, Jason E Pepin, and Edward A Rodriguez. Concepts of model verification and validation. Technical report, Los Alamos National Lab., 2004.

[244] Amie L Thomasson et al. *Fiction and metaphysics*. Cambridge University Press, 1999.

[245] Roberto Tiella, Adolfo Villafiorita, and Silvia Tomasi. Specification of the control logic of an evoting system in UML : the ProVotE experience. In *Proceedings of the 5th International Workshop on Critical Systems Development Using Modeling Languages*. Citeseer, 2006.

[246] Roberto Tiella, Adolfo Villafiorita, and Silvia Tomasi. Fsmc+, a tool for the generation of java code from statecharts. In *PPPJ*, volume 272 of *ACM International Conference Proceeding Series*, pages 93–102. ACM, 2007.

[247] Yannick Toussaint. *Text Mining : Symbolic methods to build ontologies and to semantically annotate texts*. Habilitation à diriger des recherches, Université Henri Poincaré - Nancy I, November 2011.

[248] Eleni Tsekmezoglou and John Iliadis. A critical view on internet voting technology, 2005.

[249] Jonathan Tutcher. Ontology-driven data integration for railway asset monitoring applications. In *BigData Conference*, pages 85–95. IEEE, 2014.

[250] Paolo Umiliacchi, David Lane, Felice Romano, and A SpA. Predictive maintenance of railway subsystems using an ontology based modelling approach. In *Proceedings of 9th world conference on railway research, May*, pages 22–26, 2011.

[251] Université catholique de Louvain, Université libre de Bruxelles, et al. Bevoting — étude des systèmes de vote électronique. Livrable version 1.1, Avril 2007. Partie I de l'étude vote automatisé version déf. 18122006.

[252] Université catholique de Louvain, Université libre de Bruxelles, et al. Bevoting — étude des systèmes de vote électronique. Livrable version 1.02, Décembre 2007. Deuxième partie de l'étude vote automatisé version déf Vs. 18122006.

[253] Jan van Eijck and Simona Orzan. Epistemic verification of anonymity. *Electronic Notes in Theoretical Computer Science*, 168 :159–174, 2007.

[254] Axel Van Lamsweerde and Laurent Willemet. Inferring declarative requirements specifications from operational scenarios. *IEEE Transactions on Software Engineering*, 24(12) :1089–1114, 1998.

[255] Marten J. van Sinderen, Aart Tijmen van Halteren, Maarten Wegdam, Hendrik B. Meeuwissen, and Henk Eertink. Supporting context-aware mobile applications : an infrastructure approach. *IEEE Communications Magazine*, 44(9) :96–104, 2006.

[256] Marie-Madeleine Varet-Pietri. *L'ingénierie de la connaissance : la nouvelle épistémologie appliquée*, volume 696. Presses Univ. Franche-Comté, 2000.

[257] Adolfo Villafiorita, Komminist Weldemariam, and Roberto Tiella. Development, formal verification, and evaluation of an e-voting system with VVPAT. *IEEE Trans. Information Forensics and Security*, 4(4) :651–661, 2009.

[258] Hai Wang, Yuan Fang Li, Jing Sun, Hongyu Zhang, and Jeff Pan. A semantic web approach to feature modeling and verification. In *Workshop on Semantic Web Enabled Software Engineering (SWESE 05)*, page 46, 2005.

[259] Xiao Hang Wang, Daqing Zhang, Tao Gu, and Hung Keng Pung. Ontology based context modeling and reasoning using OWL. In *2nd IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2004 Workshops), 14-17 March 2004, Orlando, FL, USA*, pages 18–22. IEEE Computer Society, 2004.

[260] Komminist Weldemariam. *Using Formal Methods for Building More Reliable and Secure e-Voting Systems*. PhD thesis, University of Trento, March 2010.

[261] Rudolf Wille. Why can concept lattices support knowledge discovery in databases ? *Journal of Experimental and Theoretical Artificial Intelligence*, 14(2-3) :81–92, 2002.

[262] Jeannette M. Wing. A specifier's introduction to formal methods. *Computer*, 23(9) :8–23, September 1990.

[263] Martin Wirsing. Algebraic specification. In Jan van Leeuwen, editor, *Handbook of Theoretical Computer Science (Vol. B)*, pages 675–788. MIT Press, Cambridge, MA, USA, 1990.

[264] René Witte, Yonggang Zhang, and Juergen Rilling. Empowering software maintainers with semantic web technologies. In Enrico Franconi, Michael Kifer, and Wolfgang May, editors, *The Semantic Web : Research and Applications, 4th European Semantic Web Conference, ESWC 2007, Innsbruck, Austria, June 3-7, 2007, Proceedings*, volume 4519 of *Lecture Notes in Computer Science*, pages 37–52. Springer, 2007.

[265] Jean-Claude Zarka. *Systèmes électoraux*. Ellipses, coll. Mise au point , 1998. page 3.

[266] Yajing Zhao, Jing Dong, and Tu Peng. Ontology classification for semantic-web-based software engineering. *IEEE Trans. Services Computing*, 2(4) :303–317, 2009.