Vérification de modèles de processus d'entreprise : une approche formelle

Bernard Kamsu-Foguem, Vincent Chapurlat, François Prunet

Vérification de modèles de processus d'entreprise : une approche formelle

B.Kamsu-Foguem [1] — V.Chapurlat [1] — F.Prunet [2]

[1]LGI2P - Laboratoire de Génie Informatique et d'Ingénierie de Production, Site EERIE de l'Ecole des Mines d'Alès - Parc Scientifique George Besse 30035 Nîmes cedex 1 - Tel. (+33) 4 66 38 70 65 - Fax. (+33) 4 66 38 70 74 {Bernard.Kamsu-Foguem, Vincent.Chapurlat}@ema.fr

[2]LIRMM – Laboratoire d'informatique, de Robotique et de Microélectronique de Montpellier – 165, rue ada, 34065 Montpellier cedex 5 – Tel. (+33) 4 67 41 85 14 Prunet@lirmm.fr

RÉSUMÉ. Dans le cycle de vie de développement d'un système industriel, les phases de vérification et, si possible de validation, impactent notablement sur la qualité du système final c'est à dire son adéquation et sa pertinence vis-à-vis de ses objectifs cibles. Dans le domaine de la Modélisation d'Entreprise, il existe peu d'outils de vérification et/ou de validation basés sur d'autres approches que la simulation. Cet article présente une approche formelle pour la vérification des modèles de processus d'entreprise par preuve de propriétés. Cette démarche s'appuie tout d'abord sur la construction d'une ontologie du domaine concerné visant à définir rigoureusement le vocabulaire qui sera utilisé pendant les phases de vérification. Elle met ensuite en œuvre des mécanismes de traduction et de raisonnement utilisant les graphes conceptuels qui s'avèrent aisés à manipuler dans un milieu industriel. Une mise en œuvre de la démarche de vérification proprement dite est enfin proposée afin de démontrer l'intérêt de ce type d'approche dans le domaine.

MOTS-CLÉS : Modélisation d'entreprise, Vérification, Validation, Approche Formelle, Graphes conceptuels, Ontologie.

ABSTRACT. During the development life cycle of a production system, the quality analysis phase affects the quality of the resulting system. The work presented in this document is about the analysis of operational processes and the formal verification of the process models in order to show that the specified system correctly reflects the requirements with closely reasoned arguments. First, we developed a general framework for properties specification and detection based on the causality of a property model. Secondly, in order to help people have a common understanding of their current enterprise process, we propose a generic method of building formal domain ontologies. Moreover, we use the reasoning mechanisms of conceptual graphs allowing to verify or analyze enterprise process properties and to improve process models on the basis of analysis results.

KEYWORDS: Enterprise Modeling, Verification, Validation, Formal Methods, Conceptual Graphs, Ontology.

1. Introduction

La **vérification** consiste, selon (ISO 8402), à *apporter la confirmation par examen et apport de preuves tangibles (informations dont la véracité peut être démontrée, fondée sur des faits et obtenues par observation, mesure, essai ou autres moyens) que les exigences spécifiées ont été satisfaites*. Cette activité est indissociable de toute activité de modélisation et devient ainsi incontournable dans l'ingénierie de systèmes complexes (Meinadier 1998, Feliot 1997). Les modèles résultant, outre le fait évident d'être syntaxiquement corrects, doivent en effet répondre de manière adéquate aux besoins et aux objectifs du modélisateur puis être exempts d'erreurs et d'ambiguïtés. Quand elle est mise en œuvre, la vérification peut être menée à bien au travers :

– D'une série d'interprétations plus ou moins subjectives car liées et dépendantes d'une certaine expérience, de certaines hypothèses ou d'un point de vue particulier. C'est en effet le modélisateur lui-même ou un expert qui va mener à bien cette interprétation soit à partir du modèle directement soit à partir des résultats d'exécution de celui-ci, c'est à dire des résultats d'une simulation ou d'une émulation. La simulation s'appuie généralement sur une exécution symbolique. Bien que relativement outillée (Kelton *et al.,* 2001) et assez aisée à mettre en œuvre maintenant, cette simulation n'est cependant possible que si une sémantique opérationnelle (des règles d'initialisation, d'exécution et des hypothèses temporelles) est définie sans ambiguïté pour le langage de modélisation utilisé. De même, l'émulation d'un modèle nécessite des règles de traduction non ambiguës et formelles permettant effectivement d'exécuter un modèle équivalent à celui en cours d'étude. Les techniques d'analyse basées sur la simulation, l'émulation ou les tests sont largement utilisées pour vérifier expérimentalement que l'on obtient des résultats corrects à partir d'un échantillon de données initiales. Cela rassure quelque peu l'utilisateur mais n'exclut pas des résultats erronés en raison de la non-exhaustivité des cas envisagés.

– De l'étude des résultats fournis par des outils de preuves formelles de propriétés. Cela est possible, bien que restreint seulement à certains domaines d'application, avec des outils tels qu'un *démonstrateur de théorèmes* (AtelierB 2004) ou un *vérificateur de modèles* (Holzmann 2003). Bien que fournissant une preuve irréfutable, ce type de vérification n'est envisageable que si le langage de modélisation utilisé pour bâtir le modèle est basé sur un formalisme autorisant l'expression et la vérification de propriétés. C'est par exemple le cas de la logique temporelle (comme CTL, Computational Tree Logic proposée par (Clarke *et al.,* 2000)) ou encore de la méthode Z (Spivey 1997). Cependant, l'usage de tels langages formels nécessite un compromis entre puissance, pertinence, rigueur, lisibilité du modèle et l'expression des propriétés

(Lamboley 2001). En particulier, la manipulation des outils de preuve actuellement disponibles nécessite une compétence et des délais de modélisation dont l'acteur au sein d'une entreprise ne pourra facilement disposer.

Pour ces raisons, la vérification formelle est souvent très marginalement abordée, et peu outillée dans le domaine de la modélisation des processus opérationnels en Entreprise (Vernadat 1999). Cet article propose une alternative intéressante aux deux modus operandi cités ci-dessus. Il s'agit d'une approche de vérification qui tente de concilier les aspects formels (avec les qualités que l'on lui prête habituellement de certitude et d'exhaustivité des résultats) et les aspects liés à la facilité d'application comme le temps de mise en œuvre (permis habituellement par la simulation par exemple) qui sont absolument nécessaires pour être utilisable en milieu industriel.

2. Problématique

Le cycle de vie d'un système comporte des phases incontournables d'analyse par vérification (« *ai-je bien construit le modèle ?* ») puis par validation (« *ai-je construit le bon modèle ? celui qu'il me faut ?* ») qui peut se schématiser dans la Figure 1. La phase de modélisation proprement dite, symbolisée par une simple flèche dans cette figure, est soumise à de nombreux problèmes susceptibles de remettre en cause la valeur de l'analyse future du modèle :

- le **point de vue** adopté par le modélisateur peut être plus ou moins clair et stable tout au long du processus de modélisation,
- les **hypothèses** retenues limitent l'expressivité et la précision du modèle qu'elles soient propres au type de système modélisé ou induites par le langage de modélisation utilisé (plus ou moins pertinent ou adapté à un domaine d'application),
- la **perspective d'analyse** future qui sera utilisée par le modélisateur peut être difficile à percevoir,
- la **méconnaissance** voire l'oubli simple de certaines caractéristiques importantes de l'environnement d'évolution du système offrent alors une vision réduite des possibilités d'évolutions du système.

Il résulte de cela qu'une partie de la connaissance pourtant nécessaire pour pouvoir ensuite vérifier, ce qui est le but de ce travail de recherche, ce modèle reste méconnue, absente ou ambiguë. Cette vérification permettra en effet de chercher à s'assurer de la qualité et de la pertinence du modèle pour pouvoir fonder une idée, un jugement ou encore tester une situation particulière du système.

L'absence de cette connaissance peut se justifier objectivement car elle peut apparaître comme dénuée d'intérêt pour le modélisateur, ou subjectivement par une

simple méconnaissance des situations complexes (Le Moigne 1999) dans lesquelles le système peut être amené à évoluer. Il serait alors souhaitable de faire émerger, tôt ou tard, cette connaissance pour mieux comprendre certains effets non prévus à l'origine et pour améliorer le modèle du système voire se poser des questions sur le système lui-même. Chaque phase de l'analyse, comme le montre la Figure 1, met en œuvre des outils de vérification syntaxique (non traités ici), sémantique et comportementale comme la simulation, l'émulation ou la preuve de propriétés comportementales. Elle peut avoir un résultat plus ou moins discutable selon le niveau de formalisation du modèle analysé, des règles et des mécanismes de preuve et / ou d'exécution mis en œuvre.

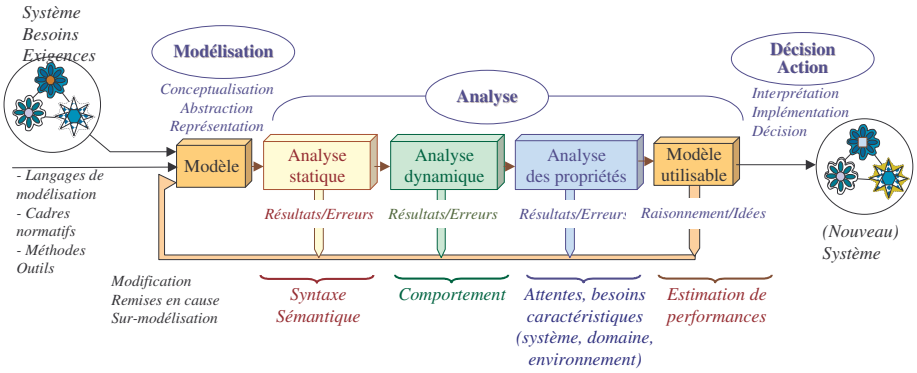

Figure 1. *Modélisation / Vérification / Validation / Décision et action*

L'intérêt accordé à ces phases d'analyse va croissant dans divers domaines, dont notamment la sûreté de fonctionnement (Moncelet 1998) et l'ingénierie des besoins (Rolland 2001). Si l'on considère tout particulièrement le domaine de la modélisation d'entreprise, de nombreux langages, méthodes et cadres de référence ont été développés : PSL (Process Specification Language) (ISO 18629), GERAM (Generalized Enterprise Reference Architecture and Methodology) (Bernus *et al.,* 1996), (ISO 19439), (ISO 19440), CIMOSA (AMICE 1993), GRAI-GIM (GRAI Integrated Methodology) (Doumeingts *et al.,* 1998), ACNOS (El Mahmedi 1997), Olympios (Haurat *et al.,* 1994). Cette liste n'est évidemment pas exhaustive. La plupart de ces approches restent encore dépourvues de mécanismes autorisant des phases de vérification à proprement parlé. On peut cependant citer par exemple le langage PSL (Bock *et al.,* 2004) qui intègre des mécanismes de vérification des spécifications. De même, d'autres travaux (Hilger *et al.,* 1990), (Dubois *et al.,* 1994) ou (Zoetekouw 1992) ont abordé le problème de formalisation des processus d'entreprise. Ainsi, le langage Albert II (Dubois *et al.,* 1994) permet une description formelle des besoins afin de faciliter leurs validations grâce à la génération de scénarios. Ces travaux se rapprochent notamment des besoins de validation liés à l'approche CIMOSA.

L'approche proposée dans cet article est basée sur un ensemble de concepts et d'outils permettant simultanément :

– De représenter de la connaissance nécessaire à l'expression précise d'un besoin servant de support à la spécification du système et à la description de son modèle.
– D'assurer de manière formelle l'exploitation de cette nouvelle connaissance pour vérifier le modèle.

3. Démarche

La démarche décrite ci-dessous (Kamsu 2004) vise à fournir une aide durant les phases d'analyse des modèles de systèmes complexes, tels qu'un processus dans une entreprise, par la spécification et la vérification formelle de propriétés.

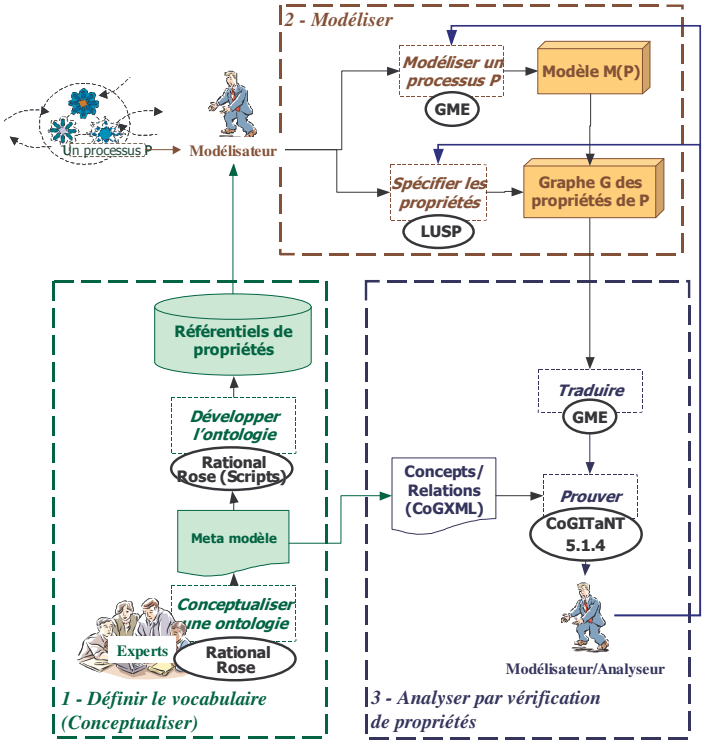

Figure 2. *Démarche schématisée et outils support associés*

Cette démarche, schématisée avec ses outils informatiques supports dans la Figure 2, se décompose en trois étapes essentielles :

– **Définition du vocabulaire** : une ontologie du domaine de la modélisation d'entreprise est formalisée. Cette formalisation est en soi une activité de clarification terminologique au cours de laquelle des concepts plus ou moins imprécis doivent être amenés à une expression dénuée de toute ambiguïté. Les entités et les relations entre ces entités nécessaires à la modélisation de processus d'entreprise dans un univers donné (production, tertiaire, etc.) sont ainsi représentées rigoureusement. Ces entités et ces relations s'inspirent évidemment des concepts imposés par le langage de modélisation choisi ou proposé par d'autres méthodes ou cadres de modélisation de référence du domaine concerné, ici la modélisation d'entreprise. Cette première étape ne sera pas abordée en détail dans cet article du fait de sa complexité. Le lecteur pourra se référencer à (Kamsu 2004, Chapurlat *et al.,* 2005).
– **Modélisation** : le processus d'entreprise est modélisé en utilisant le langage choisi. En même temps, les propritétés devant être prouvées sur le modèle sont choisies et spécifiées par le modélisateur. Ce sont ces propriétés qui fournissent ainsi la connaissance non incluse dans le modèle, connaissance pourtant nécessaire pour la vérification.
– **Analyse des modèles par vérification de propriétés** : la vérification nécessite de traduire les modèles obtenus dans la phase précédente dans des modèles formels exprimés dans le formalisme des graphes conceptuels. Ce formalisme offre en effet divers mécanismes de raisonnement dont l'utilisation est détaillée dans la suite. Cette approche permet de s'assurer que ce modèle respecte un ensemble de propriétés ou de détecter au contraire des manques ou des défauts nécessitant de modifier le modèle ou de spécifier de nouvelles propriétés.

La démarche proposée permet ainsi de faire référence aux même concepts (homogénéité du vocabulaire) tant pour décrire les modèles de processus d'entreprise que les graphes de propriétés et les graphes conceptuels qui permettront d'analyser ces modèles. Ensuite, la représentation graphique des connaissances permet à des utilisateurs de comprendre, créer ou modifier directement cette connaissance de façon beaucoup plus simple qu'avec une représentation sous forme de formules logiques par exemple. Enfin, le cadre formel fournit par les graphes conceptuels permet de lever toute ambiguïté par une unique interprétation, interprétation basée sur des mécanismes fournissant des traces du raisonnement, traces éminemment intéressantes pour faciliter les modifications et améliorations ultérieures du modèle.

Afin d'aider le lecteur à comprendre et à appliquer concrètement les différents concepts proposés dans cette approche, un exemple de mise en œuvre des étapes de modélisation et d'analyse est détaillé dans la suite du document. Cet exemple part d'un processus d'entreprise devant être modélisé puis vérifié avant de pouvoir prendre un certain nombre de décisions d'amélioration. Ce processus décrit les étapes nécessaires à la production puis à l'expédition d'un article à partir d'une commande client. Il décompose en trois processus plus élémentaires :

– La prise en compte de la commande client via un centre d'appels téléphoniques. Une fois cette commande enregistrée, il faut vérifier sa faisabilité technique et contractuelle puis la valider afin de lancer les ordres de fabrication nécessaires. Une fois la production terminée, cette activité prendra en charge la facturation, ainsi que l'émission du bordereau d'expédition.
– La production de l'article demandé qui comprend la gestion des ordres de fabrication, la fabrication elle-même, l'emballage, l'étiquetage et la mise à disposition de la marchandise.

La gestion des exceptions relatives à des cas particuliers qui comprennent la modification des commandes, la planification des commandes répétitives et le traitement des ordres de retour, par exemple en cas de rupture de stock temporaire, de même que le traitement d'un article inconnu et le fonctionnement en marche dégradée.

4. Phase de Modélisation

4.1. *Modélisation du processus*

La modélisation du processus nécessite dans un premier temps de choisir le langage de modélisation de processus à utiliser. Celui-ci est inspiré de UEML 1.0. (Unified Enterprise Modelling Language) (UEML 2003) dont une version du méta modèle est donnée dans la Figure 3. UEML définit un ensemble de concepts et de relations communs à certains langages de modélisation d'entreprise existants. Ces concepts et relations sont décrits dans le méta modèle sous forme de classes et de relations en utilisant le langage de classe de UML (Booch *et al.,* 1999). Chaque classe et chaque relation sont définies en détail en annexe de (UEML 2003).

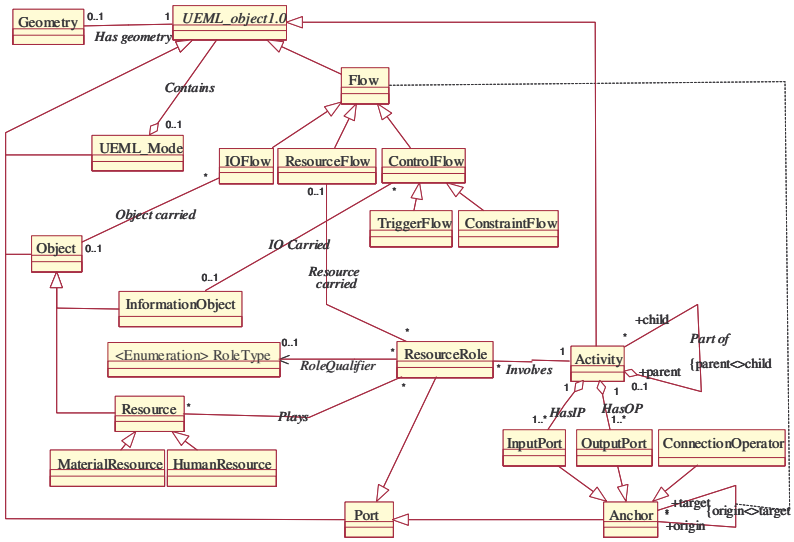

Figure 3. *Méta modèle (hors attributs de classes) de UEML (UEML 2003)*

Il s'agit donc d'un langage d'échange facilitant la communication entre différents langages plutôt qu'un nouveau langage de modélisation en tant que tel. Cependant, il paraît intéressant de démontrer l'intérêt de la démarche proposée plus loin sur des modèles décrits en UEML comme si ces modèles avaient été élaborés à l'origine avec d'autres langages puis traduits sans perte sémantique vers le formalisme de UEML.

Afin de pouvoir manipuler ces concepts, un outil de méta modélisation baptisé GME (Global Modelling Environment) (GME 2004) a été utilisé. Il s'agit d'un outil générique permettant tout d'abord de décrire un méta modèle comme celui d'UEML au moyen d'un méta langage formel. GME permet ensuite de générer automatiquement un outil de modélisation permettant alors de manipuler les concepts et les relations du méta modèle décrit. Dans notre cas, nous obtenons ainsi un outil de modélisation dédié UEML. La Figure 4 et la Figure 5 montre deux exemples de modèles obtenus à partir de GME.

La Figure 4 illustre le modèle obtenu du processus étudié. Sur cette figure, chaque boîte représente un sous processus dont l'objectif est fixé par le modélisateur. Chacun des sous processus est décomposé à un niveau de détail supplémentaire en activités comme le montre la Figure 5. Chaque activité transforme des flux d'objets d'entreprise présents en entrée en flux d'objets d'entreprise en sortie. Elle requiert pour cela des objets d'entreprise pour son contrôle et des ressources pour la durée complète de son exécution.

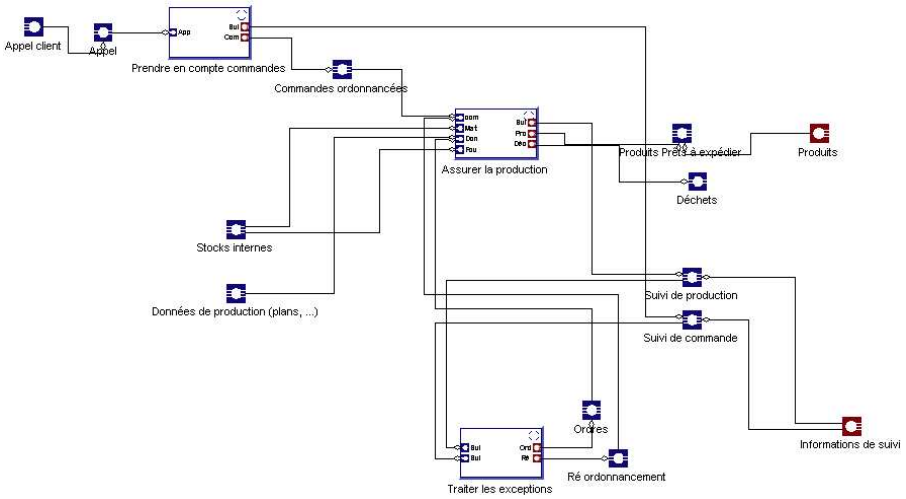

Figure 4. *Le processus de traitement des commandes décrit avec GME (GME 2004)*

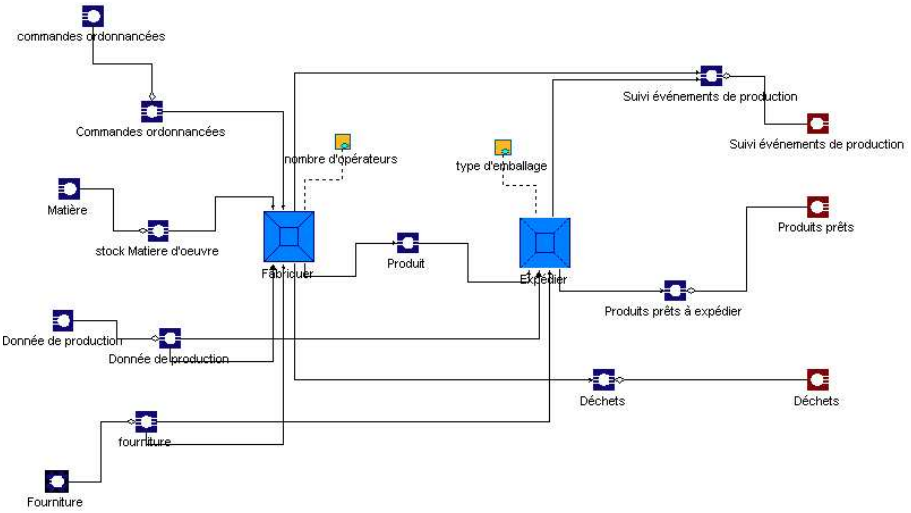

Figure 5. *La description détaillée du sous processus "Assurer la production"*

La construction d'un modèle de processus vise généralement à fournir une aide à la conception, à l'analyse ou à la réorganisation, mais que faire de ce modèle si on ne peut s'assurer de sa complétude, de sa cohérence, de sa validité au regard de l'utilisation qui en sera faite ? La démarche consiste donc maintenant à modéliser la

connaissance manquante ou devant être vérifiée au travers de propriétés avant de procéder à la vérification au moyen de mécanismes décrits plus loin

4.2. *Modélisation de la connaissance sous forme de propriétés*

L'idée de base, développée à partir des travaux de (Lamine 2001), est de fournir un modèle unique et formel de représentation de connaissances, baptisé classiquement propriétés, qui permet de compléter ce que le modélisateur exprime déjà dans son modèle (de manière plus ou moins adaptée à tout mécanisme de vérification par la suite) sur ce qu'il sait explicitement ou implicitement du système. En effet, (Lamine 2001) précise que '*Toute propriété traduit une attente, une exigence, une finalité que l'entité doit satisfaire*'. Elle doit donc soit être prouvée sur le modèle du système lorsque cela est possible, soit amener une connaissance supplémentaire qui peut s'avérer bénéfique à terme pour le modélisateur.

A titre d'exemple, modéliser un système de convoyage de bidons remplis d'acide présentant un très grand risque pour d'éventuels opérateurs à proximité nécessite de disposer d'informations relatives à la mécanique du système de convoyage, de la puissance électrique disponible, des produits transportés, du type de risque encouru, etc. Toutes ces informations peuvent ne pas être représentables en utilisant le seul langage de modélisation proposé.

Ainsi, un langage de modélisation de propriétés baptisé CRED (Causes, Relation, Effets, Degré) a été développé. Dans ce langage, une propriété est définie par une relation de causalité R entre l'ensemble C des données et des informations appelées causes et l'ensemble E des données et des informations appelées effets. Cette relation R peut être typée comme suit :

- Logique (Implication ou Equivalence sans notion de temps ou de durée d'application),
- Temporelle (les notions d'instants ou d'intervalles de temps permettent de définir, par exemple, la précédence : les causes précèdent toujours les effets),
- Influence (la connaissance que l'on a de la cause modifie, avec un sens de variation donné qui peut être interprété comme bon ou mauvais, l'opinion que l'on a sur l'effet),
- Emergence (l'effet de la propriété est alors caractérisé par l'apparition ou la disparition d'une autre propriété appartenant à un autre niveau de détail).

Enfin, D est le degré de détail de la propriété. Ces degrés sont ordonnés sur une échelle de référence appelée Granularité G choisie par le modélisateur. Il peut s'agir d'une granularité basée sur les niveaux de décomposition utilisés dans un modèle (respectant par exemple des règles de décomposition comportementale ou fonctionnelle) ou basée sur une structuration arbitraire du système que l'on souhaite décrire.

La notation formelle des propriétés (non présentée ici) permet d'accéder, via une traduction vers d'autres langages formels, à des techniques de preuve existantes sur le marché et largement outillées. Des prouveurs de théorèmes (theorem prover) comme Z-eves (Saaltink 1999) ou des vérifieurs de modèles (model checker) comme SMV (MacMillan 2000) peuvent ainsi permettre de vérifier certaines propriétés. Un theorem prover permet une vérification interactive qui repose sur la construction progressive d'axiomes ou de théorèmes à partir d'une théorie de base. A l'opposé, la vérification par un model checker est basée sur l'exécution complète, éventuellement symbolique, d'un modèle du système étudié. Tous les comportements possibles du modèle étant explorés systématiquement, la vérification des propriétés exprimées dans une logique particulière, souvent temporelle, devient alors possible. Il s'agit alors de prouver certaines propriétés de :

- **Sûreté :** Quelque chose de mauvais ne doit pas se produire.
- **Vivacité :** Quelque chose de bon doit fatalement arriver.
- **Sécurité :** Quelque chose doit toujours arriver ou Son contraire ne doit jamais arriver.
- **Atteignabilité :** Une situation ou un état donné du modèle sera toujours atteint tôt ou tard.

Considérant le processus décrit Figure 4, le modélisateur peut décrire les propriétés suivantes et ainsi compléter la connaissance qu'il possède sur ce processus.

Propriété 1

La confirmation de l'enregistrement d'une commande est toujours précédée de l'établissement de la faisabilité technique de cette commande.		
Cause	***Relation de causalité***	***Effet***
- établissement de la faisabilité technique d'une commande Ci	- précédence temporelle	- confirmation de l'enregistrement de Ci

Propriété 2

L'écart entre réalisations effectives d'une activité de fabrication et prévisions initiales influence les actions correctives auxquelles cette activité sera soumise.		
Cause	***Relation de causalité***	***Effet***
- Ai activité de fabrication - Sorties (Ai) ≠ Prévisions (Ai) - Ecart =Prévisions (Ai) – Sorties (Ai)	- influence	- actions correctives sur l'activité Ai

Propriété 3

Si on utilise un type de procédé de fabrication alors le matériau utilisé et la géométrie des objets produits ont des caractéristiques indissociables de ce procédé.

Cause	*Relation de causalité*	*Effet*
- Procédé de fabrication de type Fi	- implication logique	- le matériau dépend de Fi - la géométrie des sorties a des caractéristiques liées à Fi

Propriété 4

Un produit est expédié à destination d'un client, si et seulement si il a été contrôlé (qualité, sécurité, etc.) et seulement quand il existe une facture correspondant au bon de commande du client.

Cause	*Relation de causalité*	*Effet*
- Produit Pi a subit un contrôle - La facture liée au BC du client Ci pour un produit Pi existe	- équivalence logique - antécédence	- Produit Pi expédié à un client - Produit Pi respecte les normes en vigueur

Propriété 5

Soient deux activités A_i et A_j consécutives (A_j suit immédiatement A_i), si A_i passe en Arrêt propre alors A_j va passer en Arrêt induit.

Cause	*Relation de causalité*	*Effet*
- A_j suit immédiatement A_i - A_i dans l'état *arrêt propre*	- antécédence	- A_j dans l'état *arrêt induit*

Propriété 6

La coopération et la coordination des ressources humaines au cours d'une activité donnée font apparaître de nécessaires compétences complémentaires entre ces ressources.

Cause	*Relation de causalité*	*Effet*
- Coopération et coordination des ressources humaines de l'activité Ai	- émergence	- compétences complémentaires pour soutenir l'activité Ai

Comme on le voit, le concept de propriété, bien que souvent employé, se voit ici décrit rigoureusement dans un modèle formel. Celui-ci reste cependant encore abstrait par le contenu que l'on peut y mettre. Un grand nombre d'études internationales, dont celle du Standish Group (Sgrop 2001) ont montré que plusieurs échecs dans la mise en œuvre et l'utilisation des systèmes industriels sont dus à une mauvaise compréhension des besoins auxquels ces systèmes tentent de répondre. Cette mauvaise compréhension étant régulièrement engendrée par des spécifications déficientes. Le travail effectué a donc consisté à développer une base de connaissances génériques, c'est à dire de propriétés applicables de manière générique à un type de modèle ou de système. Cette connaissance, validée par un groupe d'experts, aide alors le modélisateur à choisir puis à spécifier les propriétés

qui paraissent pertinentes vis-à-vis de son système et qui devront ensuite être vérifiées.

4.2.1. *Typologie des propriétés*

Le référentiel de propriétés définit dans (Kamsu 2004, Chapurlat *et al.,* 2002) est destiné à améliorer de manière significative la phase de spécification des modèles d'entreprise par la proposition d'un ensemble de propriétés types. En substance, l'objectif de ce référentiel de propriétés est donc de :

- Fournir une modélisation systémique et explicite des propriétés usuelles mais déterminantes d'une entreprise et de son environnement.
- Faciliter leur accès, partage et réutilisation par les membres de l'entreprise dans leurs tâches individuelles et collectives.
- Constituer une valeur ajoutée pour des processus organisationnels et de production, notamment en favorisant la mise à jour et la création de nouvelles propriétés pertinentes pour l'entreprise.
- Garantir un ensemble d'éléments concourant à la mise en œuvre des spécifications appropriées et exhaustives pour un problème particulier.

L'architecture du référentiel fournit une vision structurée des propriétés et prenant en compte les divers aspects du système visé. Elle contient trois types de propriétés :

- **Propriétés Système** qui caractérisent les exigences et objectifs, l'aspect structurel, fonctionnel ou comportemental du système lui-même. Elles traduisent (entre autres) des contraintes de déploiement, de sûreté de fonctionnement, de performances, de volumétrie, de confidentialité, d'ergonomie, de maintenabilité, de robustesse (Meinadier 1998), de stabilité ou d'intégrité (Penalva 1999).
- **Propriétés Modèle** qui sont spécifiques à un langage de modélisation. A titre d'exemple, l'on retrouve souvent les propriétés telles que la vivacité, la réinitialisation, le parallélisme ou le blocage dans un langage utilisant le formalisme des automates pour décrire le comportement d'un système. Dans le cas des Réseaux de Petri (David *et al.,* 1992), un certain nombre de propriétés générales (réseau vivant, borné, propre, etc.) garantissent que la structure du réseau étudié est correcte.
- **Propriétés Axiomatiques** qui décrivent des connaissances et contraintes irréfutables d'un domaine (lois, normes, standards). Elles sont inspirées par la systémique, l'ingénierie des systèmes ou les cadres de modélisation employés dans certains domaines d'application. C'est ainsi que des propriétés dites de 'temps réel', qu'un domaine impose sur un système, ont été analysées en détail dans (Jackson 1994). Dans le même ordre d'idée nous pouvons citer les travaux

de (Feliot 1997) qui ont formalisé des propriétés liées aux attributs de temps, d'espace et de forme pour caractériser les processus de transformation de matières et/ou d'énergie relativement à leurs flux entrants et sortants.

Toutes ces connaissances formalisées offrent l'opportunité aux utilisateurs du référentiel, acteurs de l'entreprise, d'y accéder et de les interpréter ensuite selon leur préoccupation. Ainsi, le modélisateur pourra interpréter et instancier les propriétés à sa disposition puis les paramétrer en tenant compte des éléments constitutifs, de la structure et du comportement du modèle à vérifier et de la finalité de son étude. Ce référentiel constitue donc un véritable guide de spécification des propriétés des systèmes de l'entreprise.

4.2.2. *Identification des propriétés*

Au niveau méthodologique, l'identification des différentes propriétés est faite par les moyens suivants : analyse du cahier de charges d'un système, investigation du domaine d'application du système (consultation de documents techniques, interaction avec les experts, etc.), étude des similarités entre problèmes, ou encore par la conception d'une liste de questions structurées pouvant guider le modélisateur. Ainsi ce dernier pourra être amené à s'interroger sur les liens existants entre les composants du système et rendre compte de la logique de composition et des contraintes inhérentes, pour retrouver certaines propriétés. Par exemple, la modélisation d'une opération d'assemblage doit faire nécessairement apparaître deux flux d'entrée de produits semi-finis et un flux de produits élaborés au moins. Le modélisateur devra aussi prendre en compte les possibilités de changement (comme le renouvellement de ressources) ou d'évolution comportementales des éléments du système. Enfin la découverte d'autres propriétés peut s'effectuer en recherchant les éléments justifiant la présence d'un besoin sur le système, en voulant définir le degré de complétude des besoins ou encore en exprimant les exceptions aux propriétés déjà existantes.

L'utilisation d'un unique langage de modélisation, CRED (Causes, Relation, Effets, Degré), permet ainsi de représenter toutes les propriétés d'un modèle de manière homogène et formellement établie. Chaque propriété est ainsi un grain de connaissance qui partage certaines causes ou certains effets avec d'autres propriétés. On voit alors apparaître une structure de graphe de propriétés (Kamsu *et al.* 2003) dont les nœuds sont des causes et des effets, éventuellement commun à plusieurs propriétés, et dont les arcs représentent les différents types de relations possibles entre ces causes et ces effets. Des informations pertinentes sur le modèle vont découler de l'analyse de ce graphe de propriétés. Par exemple, il peut s'avérer que certaines propriétés sont insuffisamment décrites et nécessitent des compléments d'information (valeur d'un attribut ou toute caractéristique propre à un système spécifique) afin d'être utilisables dans une démarche de vérification formelle telle que celle présentée maintenant.

5. Phase d'analyse par vérification de propriétés

On peut distinguer deux approches principales à la vérification formelle (Monin 2000):

- Celle qui met en œuvre un système d'inférence, c'est-à-dire un ensemble bien défini et formalisé de règles de raisonnement issues de la logique (modus ponens, récurrence, substitution de valeurs égales, etc.).
- Celle qui se fonde sur la construction d'un modèle généralement fini qui représente le système d'état/transition décrivant le comportement de l'application modélisée. La vérification des propriétés s'effectue en parcourant exhaustivement ce modèle.

Toutefois, certains facteurs limitatifs (indécidabilité, explosion combinatoire, temps d'apprentissage, etc.) font que la mise en œuvre de ces techniques de vérification formelle n'est pas triviale et par conséquent souvent restreinte à certains systèmes dits critiques (Rushby 2002). Nous soulignons alors l'importance de «démocratiser » ces approches formelles par l'utilisation de formalismes souples (*lightweight formals methods* (Jackson 2001)), de type conceptuel par exemple, à même de transcrire le vocabulaire du domaine de modélisation enfin de promouvoir des actions de vérification plus efficace et mieux accessible.

5.1. *Vers des mécanismes d'Intelligence Artificielle*

On peut raisonnablement estimer que plus les notions manipulées au travers d'un langage formel sont concrètes – proches de l'expérience des utilisateurs – plus ce langage est facilement accessible. Notre choix pour la représentation des modèles et des propriétés à vérifier s'est alors orienté sur le formalisme des graphes conceptuels (Sowa 1984). En effet ce formalisme nous semble parfaitement adapté à la formalisation du vocabulaire d'un domaine de modélisation, car sa syntaxe précise permet de décrire les différentes constructions autorisées et sa sémantique rigoureuse permet d'exprimer sans ambiguïté le sens des constructions syntaxiques. De plus, il est doté de mécanismes de raisonnements permettant d'établir des propriétés sur les constructions réalisées et ceci de manière lisible, y compris par les non-spécialistes. D'autres langages (RDF, DAM-OIL, OWL) liés au Web sont parfois utilisés pour la représentation et l'échange de connaissances. Cependant, comparée aux graphes conceptuels, ils ont des mécanismes d'inférences limitées (Delteil *et al.,* 2003).

La section suivante expose, une description du formalisme, inspirée de celle définie dans (Mugnier 2002).

5.1.1. *Présentation du formalisme*

Les graphes conceptuels sont un langage de représentation des connaissances, proposé par John Sowa, permettant à la fois de définir rigoureusement un vocabulaire (i.e ontologie) et d'utiliser ce vocabulaire pour décrire et raisonner sur des situations. D'un point de vue représentation de connaissances, les graphes de Sowa apportent une structuration des différents types de connaissance qui n'est pas explicite dans d'autres formalismes comme les réseaux sémantiques ou UML. En effet, ce formalisme marque une séparation nette entre les connaissances ontologiques (encodées dans une structure particulière appelée *support*) et les connaissances factuelles qui sont représentées par les graphes. De plus, le formalisme distingue clairement les entités (représentées dans le graphe par des rectangles) des relations entre ces entités (représentées par des ovales).

> **Définition**: *Un graphe conceptuel simple est un graphe biparti, étiqueté, orienté et fini. Les deux composantes formant les nœuds du graphe sont: les concepts et les relations.*

Un *concept* est composé d'un type et d'un référent : [<type> :<marqueur>], par exemple [Atelier : AtelierDeFabrication23].

- Le type de concept représente l'occurrence d'une classe d'objets. Ils sont regroupés dans une structure hiérarchique (muni d'un ordre partiel) appelée treillis des types de concepts.
- Le marqueur précise le sens d'un concept en spécifiant une occurrence du type de concept. Il peut être de différente nature, notamment individuel ou générique.

Une *relation conceptuelle* lie enfin deux ou plusieurs concepts suivant le schéma suivant : [C1]→(relation)→[C2]. La définition de sous relations est parfois nécessaire pour apporter plus de finesse dans la représentation sémantique, on établit alors un treillis de relations. Chaque relation possède une signature qui fixe son arité et donne les types maximaux de concepts qu'une relation de type peut lier.

L'ontologie formelle du domaine de modélisation est alors constituée par la construction d'un support formé des treillis de concepts et de relations. Toutefois, les travaux de Sowa (Sowa 2000) ne précisent pas la manière d'obtenir cette ontologie à partir d'un corpus informel. Cela nous a conduit à définir une méthode de construction progressive d'ontologie formelle.

5.1.2. *Représentation du domaine : Ontologie de la Modélisation d'Entreprise*

La modélisation en entreprise, par sa nature pluridisciplinaire, contient beaucoup de termes différents ayant la même sémantique. Il faut utiliser donc un moyen de conception qui peut normaliser tous les termes employés afin de lever toute ambiguïté. C'est dans le concept d'ontologie que ce problème trouve sa solution. L'ontologie a pour fonction en effet de spécifier explicitement concepts et relations

(Gruber 1993), c'est à dire de les rendre compréhensibles et utilisables par plusieurs agents (humains ou logiciels).

Nous nous sommes inspirés des travaux de (Guarino 1997) pour définir une démarche générique de construction d'ontologie formelle qui est appliquée ici à modélisation d'entreprise. Cette démarche de formalisation comporte deux étapes. En s'appuyant sur des travaux existants (PSL (Schlenoff *et al.,* 2000), UEML (Vernadat 2002, UEML 2003), SAGACE (Penalva 1999), TOVE (Gruninger *et al.,* 2000)), la première étape consiste à bâtir une représentation intermédiaire de l'ontologie avec la notation UML (OMG 2003) (on parle alors de *conceptualisation*). Quant à la deuxième étape, elle permet de dériver une description formelle et structurée en treillis de l'ontologie initiale (transformation appelée *opérationalisation*).

Schématiquement parlant, le processus général de construction d'ontologie est scindé en deux phases comme le montre la Figure 6.

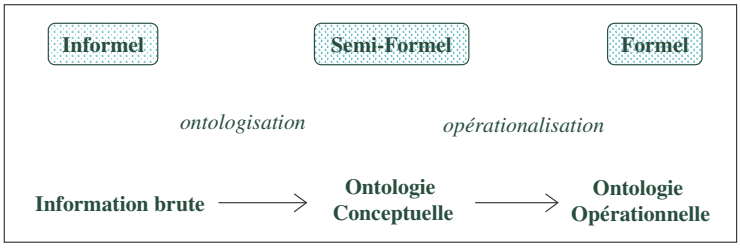

Figure 6. *Construction d'une ontologie formelle*

L'obtention d'une ontologie formelle constitue un élément qui pour nous est primordial, car ceci sert divers aspects du développement de notre approche d'analyse des modèles d'entreprises. Tout d'abord cette ontologie formelle assiste le processus de construction de spécification de modèle, en rendant les documents de spécification mieux compréhensibles et plus précis. Cette précision favorise l'interopérabilité entre des paradigmes, des langages ou des outils différents. En outre, elle soutient la démarche de raisonnement pour la vérification des propriétés de modèle.

5.1.3. *Outils de raisonnements*

Parmi les travaux s'attachant à utiliser le formalisme des graphes conceptuels comme un langage opérationnel deux grands courants se distinguent :

– Le premier consiste à considérer le langage comme une simple notation graphique de la logique des prédicats du premier ordre (Amati *et al.,* 2000). Les raisonnements sont alors effectués, après traduction des graphes en formules

logiques, en utilisant des algorithmes de résolution logique. Le système Prolog+CG (Kabbaj 2000) fournit une vision représentative d'une telle approche ;

– Le second vise à concevoir le langage comme une notation formelle disposant d'opérations de raisonnement (basées sur le morphisme de graphes) internes à la notation, mais conservant les propriétés de consistance et complétude vis à vis de la logique des prédicats du premier ordre, ce qui permet de doter ce formalisme d'une "sémantique" formelle. Le système *CoGITaNT* (Genest 2003) relève de cette approche.

Nous plaçons nos travaux dans ce deuxième courant qui, grâce à son aspect graphique tant du point de vue de la représentation des connaissances que des mécanismes de raisonnement, nous semble être un candidat idéal pour notre problématique. En effet, *Prolog* ayant l'inconvénient de ne pas disposer de sémantique déclarative, il en résulte que le résultat d'une inférence peut dépendre de la forme de la base de connaissances (ordre des clauses ou des conjonctions), conduisant ainsi à des résultats différents pour deux bases de connaissances sémantiquement équivalentes.

5.2. *Preuves de propriétés avec les opérations de graphes*

En accord avec (Mugnier 2000), nous pensons que la projection est l'opération fondamentale de raisonnement dans le formalisme des graphes conceptuels. Cette projection est une sorte d'appariement de graphes, qui peut être facilement visualisé et interprété. Par exemple si nous souhaitons répondre à la question « est ce que la propriété représentée par un graphe (requête) *H* est présente dans un référent également décrit par un graphe *G* ? ». Cet *opérateur de recherche d'informations* nous permet d'y répondre et la vérification de la propriété va consister à rechercher les projections possibles de *H* dans *G*.

Dans la Figure 7 par exemple, à partir du graphe d'une situation décrivant partiellement deux activités précises et en posant une requête sous forme d'un graphe question, nous parvenons à prouver la présence d'une commande en sortie d'une activité. Cette preuve par projection du graphe *requête* dans le graphe *situation*, est fondée sur un calcul de spécialisation entre les deux graphes.

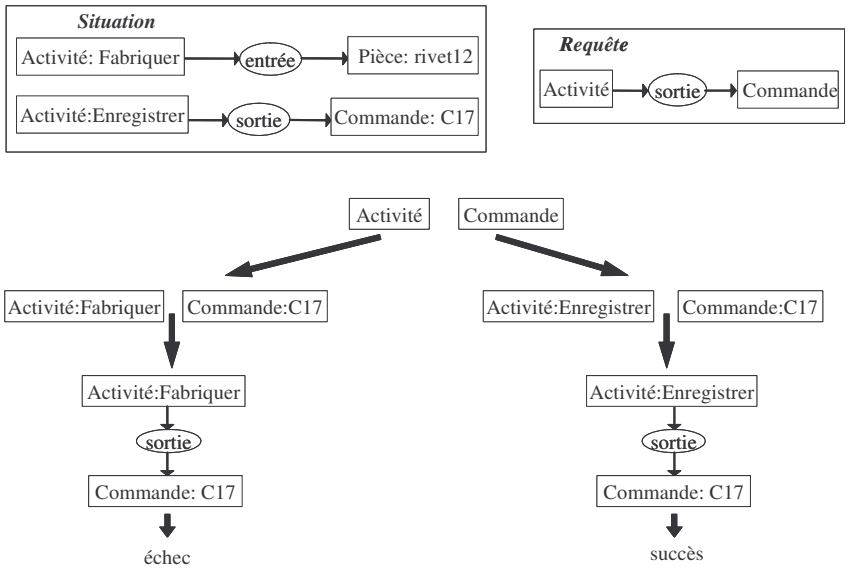

Figure 7. *Un exemple de vérification par projection*

Pour réaliser des raisonnements plus poussés s'appuyant sur cette projection, nous allons mettre en œuvre d'autres types des connaissances de différente nature : règles (d'inférence, d'évolution) et contraintes (Baget *et al.,* 2002).

5.2.1. *Compléter les représentations avec les règles de graphes*

Les outils de modélisation d'entreprise sont souvent descriptifs et font appel à une collection d'heuristiques qui ne sont pas applicables en toutes circonstances. Il est donc nécessaire de recueillir et de formaliser les principes de modélisation d'entreprise qui sont implicitement présents dans ces heuristiques. C'est dans cette optique qu'interviennent les règles de graphes conceptuels, pour décrire ces connaissances implicites sous forme de propriétés dont la prise en compte favorisera l'enrichissement ou l'évolution de modèles. Par exemple la règle « si une entreprise a des flux de fabrication discontinus, alors elle utilise des ateliers avec stockage » peut permettre de déterminer, par inférence de nouvelles connaissances, les en-cours de production d'une entreprise.

De manière générale, nous définissons deux types de règles :

– les règles statiques pour exprimer certaines lois immuables du domaine (Salvat 1997). Par exemple une règle statique pourra être utilisée pour exprimer la symétrie d'une relation de proximité géographique ou encore la transitivité d'une relation binaire, notamment le parallélisme (Figure 8).

– les règles dynamiques pour modéliser les évolutions du monde avec les conditions de changement d'états (BOS 1998). En particulier, une règle dynamique permettra de modéliser des propriétés du type : « si deux activités Ai et Aj se suivent et que l'activité Ai traite des produits (pièces, données, etc.) alors une voie de communication doit assurer la transmission de ces produits vers l'activité Aj » (Figure 9).

Figure 8. *Un exemple de règle statique*

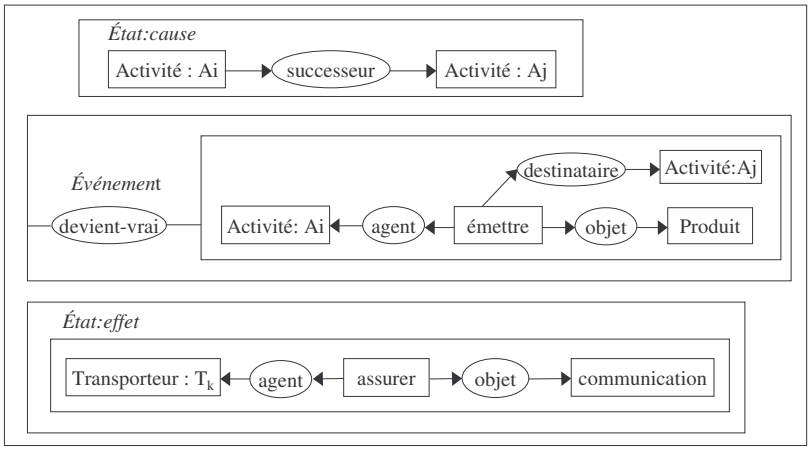

Figure 9. *Un exemple de règle dynamique*

5.2.2. *Valider avec les contraintes*

Dans le modèle de base des graphes conceptuels, le raisonnement est limité à la démonstration de la vérité de propriétés positives (i.e. équivalentes à des formules conjonctives positives). Cependant, nous avons parfois besoin de démontrer la vérité ou la fausseté de graphes ou de morceaux de graphes pour prouver l'exécutabilité ou la non exécutabilité des actions. C'est ainsi qu'apparaissent les contraintes comme graphes qui servent à effectuer la vérification de la validité d'un modèle représenté par un ou plusieurs autres graphes.

Une contrainte est composée d'une partie conditionnelle et d'une partie obligatoire. La partie conditionnelle est un graphe simple qui peut être vide (Baget 2001). Nous considérons deux types de contraintes : les positives et les négatives. Intuitivement, une contrainte positive exprime une propriété du genre: 'Si

l'information A est présente, alors l'information B doit être aussi présente'. Une contrainte négative exprime une propriété du genre: 'Si l'information A est présente, alors l'information B doit être absente'. En d'autres termes, ces contraintes de graphes permettent de bien étudier à la fois la cohérence d'un modèle (il ne doit pas contenir de connaissances décrites dans les contraintes négatives) et la complétude du modèle (il doit contenir les connaissances décrites dans les contraintes positives).

La Figure 10 représente deux contraintes, une contrainte négative C^- qui peut s'exprimer par "toute activité qui déclenche une alarme ne doit pas rester en état d'exécution" et une contrainte positive C^+, qui peut s'exprimer par "tout atelier de fabrication doit être doté au moins d'une machine".

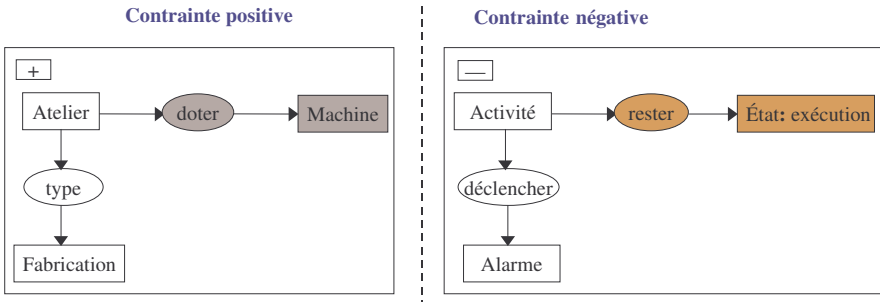

Figure 10. *Exemples de contraintes*

Munis de l'ensemble de ces opérations de graphes (projections, règles et contraintes) comme outils de raisonnements, nous pourrons concrétiser la vérification de propriétés, car nous serons en mesure de garantir qu'un référent (modèle de processus) se comporte correctement relativement à un ensemble de propriétés, qui peuvent être des connaissances expertes ou exigées par des utilisateurs.

Au niveau applicatif, pour qu'un processus soit véritablement intégré dans les politiques organisationnelles d'une entreprise, il est habituellement nécessaire d'une part que les lois du domaine soient satisfaites et d'autre part que l'exécution de ce processus soit conforme à la description de sa spécification (UEML 2002). Ce constat nous impose de pouvoir représenter explicitement, via des procédures de vérifications par preuves de propriétés, la différence entre le comportement requis et le comportement réel des processus. Par exemple, on peut vouloir s'assurer que "*le service des ventes doit répondre à la commande d'un client dans un délai de 48 heures*". Ainsi, le problème précédent revient à savoir si des politiques (managériales, organisationnelles, etc.) sont transgressées durant l'exécution des processus. De cette manière, une activité de notification peut être déclenchée par le non-respect d'une propriété spécifiée afin de prévenir un responsable de l'entreprise.

5.3. *Exemple*

Dans cette section, nous allons prouver deux des six propriétés vues plus haut en utilisant des mécanismes différents afin de bien illustrer le potentiel de l'approche proposée.

Les éléments algorithmiques de la traduction d'un modèle de processus en graphe conceptuel étant présentés dans (Kamsu 2004). On retiendra essentiellement les deux traitements opérés sur les activités et leurs enchaînements. Les activités {fabriquer, expédier,...} sont traduites dans le formalisme des graphes conceptuels suivant leur marqueur respectif (individuel, générique ou variable). Quant aux enchaînements {successeur, parallèle,…} des activités, ils sont traduits en relations conceptuelles liant les graphes des activités.

5.3.1. *Preuve directe de la propriété P_1*

La démarche de preuve directe consiste à prouver des propriétés par la mise en œuvre d'un ensemble de raisonnements déductifs. Dans cette section nous allons appliquer ce principe de preuve pour démontrer la propriété P_1 suivante :

P_1 " La confirmation de l'enregistrement d'une commande est toujours précédée de l'établissement de la faisabilité technique de cette commande ".

Pour la réalisation de cette preuve directe nous allons utiliser la règle R_1 et la contrainte positive C_1, représentées ci-dessous :

R_1 " Si l'enregistrement d'une commande est confirmé, alors cette commande est expédiée ".

C_1 " Pour qu'une commande puisse être expédiée, il faut qu'elle est un agrément de faisabilité technique ".

La démonstration de la propriété P_1 se fait en deux étapes : elle consiste à appliquer la règle R_1 au graphe décrivant une activité du processus, puis à prendre en compte une contrainte du domaine afin d'établir la consistance du graphe résultant. Finalement, la preuve est établie, puisqu'il existe une projection de la propriété dans le dernier graphe construit. La Figure 11 illustre les étapes de raisonnement associées.

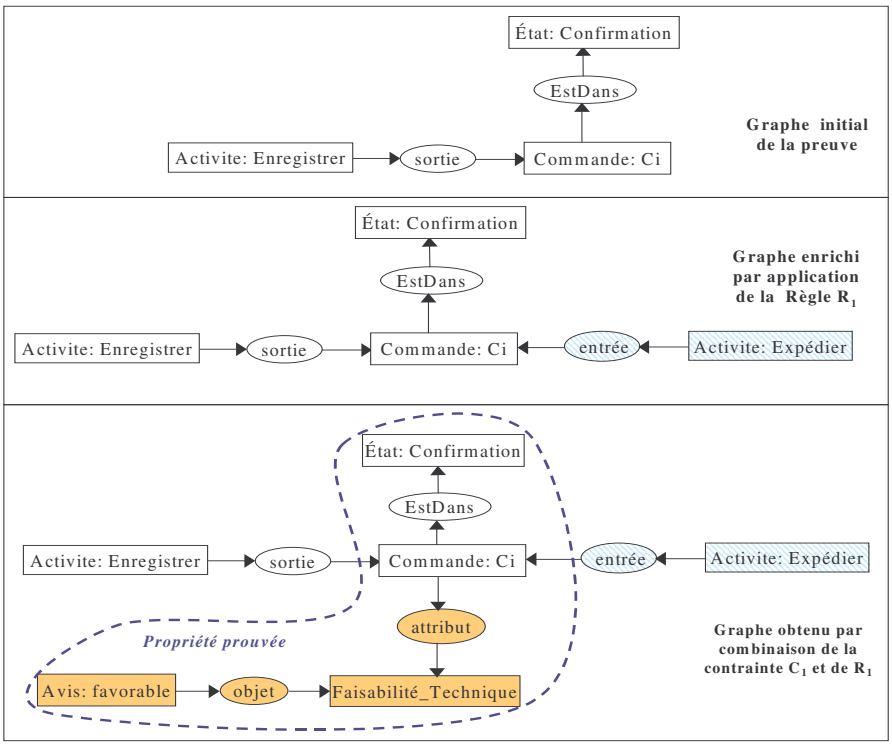

Figure 11. *Eléments de preuve de la propriété P_1*

5.3.2. *Preuve par l'absurde de la propriété P_3*

Nous rappelons que la preuve par l'absurde est une technique de démonstration qui vérifie la vérité d'une proposition en établissant la fausseté des conséquences de sa contradictoire. Nous voulons démontrer par l'absurde la propriété P_3 à l'aide des graphes conceptuels. La propriété P_3 est la suivante :

P_3 "Si on utilise un type de procédé de fabrication alors le matériau utilisé et la géométrie des objets produits ont des caractéristiques indissociables de ce procédé".

Pour la réalisation de cette preuve par l'absurde nous allons utiliser la règle R_3 et la contrainte négative C_3, représentées ci-dessous :

R_3 " Si deux produits donnés ont des attributs géométriques différents, Alors ils sont différents ".

C_3 " Deux activités de fabrication équivalentes doivent fournir en sorties des produits identiques ".

Concrètement nous essayons de démontrer l'équation suivante :

$$Règle\ R_3 \wedge\ Contrainte\ C_3 \Rightarrow Propriété\ P_3\ ?$$

Ce qui équivaut à écrire :

$$Propriété\ P_3\ \text{fausse} \Rightarrow Règle\ R_3\ \text{fausse} \vee Contrainte\ C_3\ \text{fausse}\ ?$$

La démonstration par l'absurde est formalisée en graphes conceptuels comme suit:

– L'hypothèse de départ de la démonstration "supposons qu'il existe deux produits ayant des attributs géométriques différents et qui sont issus de deux activités de fabrication équivalentes A1 et A2" est représentée par le graphe Gh de la Figure 12.

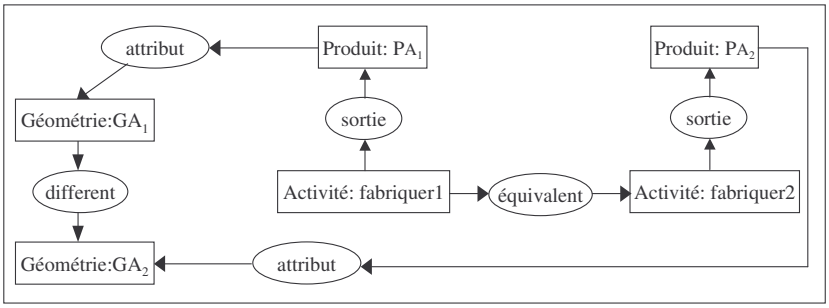

Figure 12. *Graphe Gh de l'hypothèse de la démonstration*

– Le résultat de l'application de la règle R_3 au graphe Gh est représenté par le graphe Gc de la Figure 13.

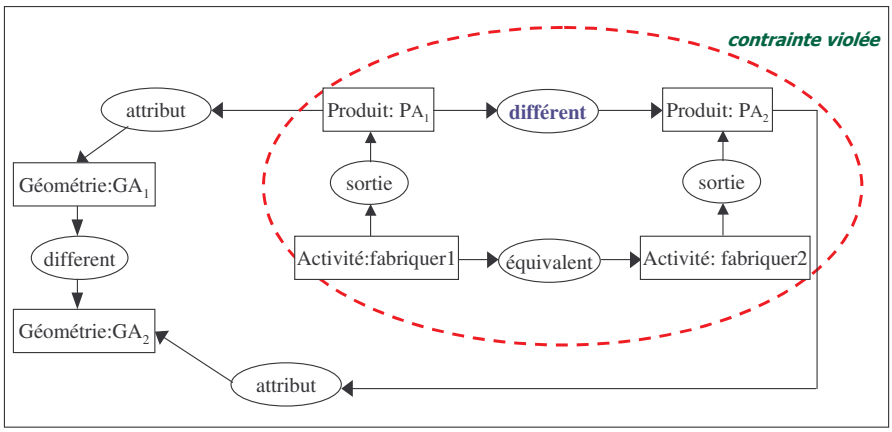

Figure 13. *Le graphe Gc résultant de l'application de la règle R_3 à Gh*

– La violation de la contrainte C_3 sur le graphe de Figure 13, met en évidence une contradiction sur ce graphe. Ceci conduit au rejet de l'hypothèse supposant que la propriété P_3 est fausse. On conclut donc que la propriété P_3 est vraie.

L'intérêt pratique de cette propriété est que l'utilisateur désirant produire des pièces avec des caractéristiques spécifiques, saura dans son contexte recourir à l'atelier adéquat pour exécuter sa tâche.

6. Conclusion

La vérification de propriétés de modèles de systèmes complexes, tels que des processus au sein d'une entreprise, est une activité délicate qui peut grandement influencer le fonctionnement et les performances de cette entreprise ou du réseau de partenaires dans lequel elle est impliquée. Les outils mis alors à disposition des ingénieurs et des décideurs doivent donc non seulement être aptes à représenter convenablement la réalité d'une situation, mais aussi permettre son interprétation de manière précise pour faciliter la compréhension de la complexité et améliorer la confiance dans les représentations à des fins d'argumentation de choix en situation de décision et/ou d'action.

L'approche proposée comprend une ontologie formelle du domaine et s'appuie sur les opérations des graphes conceptuels pour vérifier les propriétés des modèles d'entreprise. Ces opérations de graphes permettent la mise en œuvre de raisonnements directement visualisables. Cela offre aux différents acteurs la possibilité d'améliorer les modèles considérés en fonction des résultats de preuve et de leur réflexion. Les perspectives de ce travail sont nombreuses. On peut citer,

entre autres, la formalisation d'autres types de propriétés, l'utilisation de techniques formelles pour la réparation (Dibie 2000) d'une base de graphes conceptuels et l'application dans le domaine du risque industriel (Chapurlat *et al.,* 2004).

7. Bibliographie

Amati G., Ounis I., « Conceptual Graphs and First Order Logic », *The Computer Journal*, Vol. 43, n° 1, 2000, p. 1-12.

Amice Consortium, *CIMOSA : Open Architecture for CIM*, Berlin, Springer Verlag, 1993.

AtelierB *Manuel de référence du langage B (version 1.8.5)*, Document électronique disponible sur http://www.atelierb.societe.com/, 2004.

Baget J-F., Représenter des connaissances et raisonner avec des hypergraphes : de la projection à la dérivation sous contraintes, Thèse de doctorat, Université de Montpellier 2, Novembre 2001.

Baget J-F., Mugnier M-L., « Extensions of Simple Conceptual Graphs: the Complexity of Rules and Constraints », *Journal of Artificial Intelligence Research (JAIR*), vol. 16, 2002, p. 425-465.

Bernus P., Nemes L., « A Framework to Define a Generic Enterprise Reference Architecture and Methodology », *Computer Integrated Manufacturing Systems*, Vol. 9, n° 3, Juillet 1996, p 179-191.

Bock C., Gruninger M., « PSL: A Semantic Domain for Flow Models », *Software and Systems Modeling Journal*, 2004.

Bos C., Modéliser Pour Simuler Pour Modéliser : Contribution à l'Acquisition de Connaissances Comportementales, Thèse de doctorat, Université des Sciences et Technologies de Lille, Janvier 1998.

Booch G., Rumbaugh J., Jacobson I., *The Unified Modelling Language User Guide*, Addison-Wesley, 1999.

Chapurlat V., Kamsu-Foguem B., Prunet F., « A property reference model and associated tools for system life-cycle management », *World Congress IFAC B'02*, Barcelone, Juillet 2002.

Chapurlat V., Kamsu-Foguem B., Prunet F., « Enterprise model verification and validation: an approach », *Annual Review in Control, IFAC Journal*, vol. 27, no. 2. Elsevier Science, 2003, p. 185-197.

Chapurlat V, Montmain J., Gharbit D., « A proposition for risk analysis in manufacturing and enterprise modeling domain », *International Conference on Enterprise Information Management Techniques, ICEIMT 2004*, Toronto, Canada, Septembre 2004.

Chapurlat V., Kamsu-Foguem B., Prunet F, « A Formal Verification Framework and Associated Tools for Enterprise Modeling: Application to UEML », *Computers in Industry*, Elsevier, à paraitre troisième trismestre 2005.

Clarke E., Grumberg O., Peled D., *Model Checking*, The MIT Press, 2000.

David R., Alla H., *Du Grafcet aux Réseaux de Petri*, Deuxième édition revue et augmentée, Editions Hermès, Paris, novembre 1992.

Delteil A., Faron C., Dieng R., « Le modèle des graphes conceptuels pour le web sémantique Extensions de RDF et RDFS basées sur le modèle des graphes conceptuels », *XML et les objets : RSTI série l'Objet* Vol.9 n° 3, 2003, pages 95-122.

Dibie-Barthélémy J., Validation et Réparation des Graphes Conceptuels, Thèse de Doctorat, Université de Paris 9 Dauphine, Juillet 2000.

Doumeingts G., Vallespir B., Chen D., « GRAI Grid Decisional Modelling », *Handbook on Architecture of Information Systems*, édité par P.Bernus, K.Mertins, G.Schmith, Springer Verlag, 1998, p 313-337.

Dubois E., Dubois P., Dubru F., Petit M., « Agent-Oriented Requirements Engineering: A case Study using the ALBERT Language », *4ème Conférence internationale Dynamic Modelling and Information System, DYNMOD-IV*, Edité par A.Verbraeck, H.G.Sol et P.W.G.Bots, Noordwijkerhoud, The Netherlands, septembre 28-30, 1994.

El Mahmedi A, Lerch C., Sonntag M., « Modélisation des Activités et des Processus de Production : Une Approche Interdisciplinaire », *Journal Européen des Systèmes Automatisés (JESA)*, Vol. 31, n°4, 1997, p. 669-693.

Feliot C., Modélisation de systèmes complexes : intégration et formalisation de modèles, Thèse de doctorat, Université de Lille I, 1997.

GME, *Generic Modeling Environment (GME) version 4: User's Manual 2004 (Release 4-2-3)*, Institute for Software Integrated Systems (ISIS) Vanderbilt University, 2004.

Genest D., *CoGITaNT Version-5.1 - Manuel de référence*, Document électronique accessible sur le site http://cogitant.sourceforge.net, 2003.

Gruber T.R, « A translation approach to portable ontology specifications », *Knowledge Acquisition*, Vol. 5, n°2, 1993, p. 199-220.

Gruninger M., Atefi K., Fox M-S., « Ontologies to Support Process Integration in Enterprise Engineering », *Computational and Mathematical Organization Theory*, Vol. 6, n° 4, 2000, p. 381-394.

Guarino N., « Understanding, Building, and Using Ontologies: A Commentary to Using Explicit Ontologies in KBS Development », *International Journal of Human and Computer Studies*, Vol. 46, 1997, p. 293-310.

Haurat A., Piard F., Rotival H., Schweyer B., « Le modèle OLYMPIOS, un outil d'analyse et de conception du système d'informations des entreprises manufacturières », *Xième congrès INFORSID*, Aix-en-Provence, 1994.

Hilger J., Proth J-M., *Production Management Language*, IFAC'90, Tallinn, Russie, Août 1990.

Holzmann G. J., *The Spin Model Checker*, Addison-Wesley, 2003.

ISO 8402, *Quality management and quality assurance – Vocabulary, Second edition 1994-04-01*, International Standard Organisation, 1994.

ISO 10314, *Reference Model for Shop Floor Production Standards*, Technical report 10314, Part 1, ISO TC 184/SC5/WG1 N126 and Part 2, ISO TC 184/SC5/WG1 N160. 1990.

ISO 19439, *Enterprise Integration - Framework for Enterprise Modelling*, ISO TC 184/SC5/WG1 - CEN TC 310/WG1, 2003.

ISO 19440, *Enterprise Integration - Constructs for Enterprise Modelling*, ISO TC 184/SC5/WG1 - CEN TC 310/WG1, 2003.

ISO 18629, *PSL (Process Specification Language): Principles and Overview*, Document accessible sur http://www.mel.nist.gov/psl/, Janvier 2004.

Jackson D., « Lightweight Formal Methods », *International Symposium of Formal Methods Europe*, Berlin, Allemagne, 2001, p. 12-16.

Jackson M., « Problems, Methods and Specialisation », *Software Engineering Journal*, Vol. 9, n°6, Novembre 1994, p. 249-255 (édité et abrégé dans IEEE Software, Vol. 11, n°6, p.57-62, Novembre 1994).

Kabbaj M.L, « From PROLOG++ to PROLOG+CG : A CG Object-Oriented Logic Programming Language », *Proceedings of 8th International Conference on Conceptual Structures (ICCS'00)*, LNAI 1867, Springer, 2000, p.540-554.

Kamsu-Foguem B., Chapurlat V., Prunet F., « Enterprise Model Verification : a Graph-based Approach », *International IEEE/SMC multiconference on Computational Engineering in Systems Applications, CESA 2003*, Lille, France, Juillet 2003.

Kamsu-Foguem B, Modélisation et Vérification des propriétés de systèmes complexes : Application aux processus d'entreprise, Thèse de Doctorat, Université Montpellier 2, Juillet 2004.

Kelton D., Sadowski D.A., Sadowski R.P., *Simulation With Arena*, McGraw-Hill Science/Engineering/Math, Juillet 2001.

Lamboley P., Proposition d'une méthode formelle d'automatisation de systèmes de production à l'aide de la méthode B, Thèse de Doctorat, Université Henri Poincaré, Nancy I, 2001.

Lamine E., Définition d'un modèle de propriété et proposition d'un langage de spécification associé : LUSP, Thèse de Doctorat, Université Montpellier 2, décembre 2001.

Le Moigne J.L, *La modélisation des systèmes complexes*, Ed. Dunod, 178 pages, 1999.

MacMillan K.L., *The SMV System for SMV Version 2.5.4: SMV Manual*, rapport interne Cadence Berkeley Labs, 2000.

Mayer F., Contribution au Génie Productique : Application à l'ingénierie pédagogique en Atelier Inter établissements de Productique Lorrain, Thèse de doctorat en production automatisée, Université Henri Poincaré de Nancy I, 1995.

Meinadier J-P., *Ingénierie et intégration des systèmes*, Editions Hermès, 253 pages, 1998.

Monin J-F, *Introduction aux méthodes formelles*, Hermès Science Publication, Paris, 2000.

Moncelet G., Application des réseaux de Petri à l'évaluation de la sûreté de fonctionnement des systèmes mécatroniques du monde automobile, Thèse de doctorat, Université Paul Sabatier, octobre 1998.

Mugnier M-L., « Knowledge Representation and Reasonings Based on Graph Homomorphism », 8th International Conference on Conceptual Structures (ICCS'00), Darmstadt, Germany, Août 2000 (édité dans Lecture Notes in AI, vol. 1867, p. 172-192. Springer, 2000).

Mugnier M-L., A Graph-based Approach to Knowledge Representation, Habilitation à Diriger les Recherches, Université de Montpellier 2, Novembre 2002.

OMG *Unified Modeling Language Spécification version 1.5*, Document électronique disponible sur http://www.omg.org/technology/documents/formal/uml.htm, mars 2003.

Penalva J.M., *SAGACE : le systémographe*, Edition interne au CEA, Version 1.0, 1999.

Rolland C., Grosz G., « De la Modélisation Conceptuelle à l'Ingénierie des Besoins », Encyclopédie d'Informatique, Chapitre 4, Hermès, Paris, 2001.

Rushby J., « Using Model Checking to Help Discover Mode Confusions and Other Automation Surprises », *Journal of Reliability Engineering and System Safety*, Vol. 75, n°2, Février 2002, p. 167-177.

Saaltink M., *The Z/EVES 2.0 User's Guide*, TR-99-5493-06a (accessible via http://www.ora.on.ca/z-eves/), 1999.

Salvat E., Raisonner avec des opérations de graphes: graphes conceptuels et règles d'inférence, Thèse de doctorat, Université de Montpellier 2, Décembre 1997.

Schlenoff C., Gruninger M., Lee J. « The Essence of the Process Specification Language », *Transaction of the Society for Computer Simulation International*, Vol. 16, n°4, 2000.

Sgrop, Rapport de recherche interne du standishgroup, document électronique accessible sur le web, http://www.standishgroup.com, 2001.

Sowa J.F., *Knowledge Representation: Logical, Philosophical, and computational Foundations*, Brooks Cole Publishing Co., 2000.

Sowa J.F., *Conceptual Structures : Information Processing in Mind and Machine*, Addison-Wesley Publishing Company, Reading, MA, 1984.

Spivey J.M, *The Z Notation*, Collection Méthodologies du logiciel, traduit par M. Lemoine, Paris, Masson, Prentice Hall, 1997.

UEML. Petit M, et al., *Delivrable D1.1: Enterprise Modelling State of the Art*, Unified Enterprise Modeling Language UEML Thematic Network, IST-2001-34229 (document accessible sur www.ueml.org), 2002.

UEML. Berio G. et al., *Delivrable D3.1: Requirements analysis: initial core constructs and architecture*, Unified Enterprise Modeling Language UEML Thematic Network, IST-2001-34229 (document accessible sur www.ueml.org), 2003.

Vernadat F., « UEML : Toward an Unified Enterprise Modelling Language », *International Journal of Production Research*, vol. 40, n° 17, 2002, p. 4309 – 4321.

Vernadat F., *Techniques de Modélisation en Entreprise : Application aux Processus Opérationnels*, Paris, Economica, Collection Gestion, 1999.

Zoetekouw D., *An overview of formal semantics specification methods,* Enterprise Integration Modelling, Charles J. Petrie, MIT Press, Cambridge, MA, 1992.