



**HAL**  
open science

# Construction incrémentale vs. construction par raffinement de modèles comportementaux UML: sémantique et vérification des relations de spécialisation et d'implantation

Thomas Lambolais, Anne-Lise Courbis, Hong-Viet Luong

## ► To cite this version:

Thomas Lambolais, Anne-Lise Courbis, Hong-Viet Luong. Construction incrémentale vs. construction par raffinement de modèles comportementaux UML: sémantique et vérification des relations de spécialisation et d'implantation. 2008. hal-00498019

**HAL Id: hal-00498019**

**<https://hal.science/hal-00498019v1>**

Submitted on 6 Jul 2010

**HAL** is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

# Construction incrémentale vs. construction par raffinement de modèles comportementaux UML: sémantique et vérification des relations de spécialisation et d'implantation

Thomas Lambolais\*, Anne-Lise Courbis\*  
Hong-Viet Luong\*

\*EMA-LGI2P Parc Scientifique G. Besse 30035 Nîmes Cedex 1 France  
lambolais,luong,courbis@ema.fr

**Résumé.** Nous nous intéressons aux démarches de construction de modèles comportementaux UML (machines d'états) intégrant des procédures de vérification. Le raffinement, consistant à définir des spécifications de plus en plus précises ou complètes, n'a pu être jusqu'à présent formalisé sur les machines d'états. Dans les modèles orientés objets, la relation de raffinement peut se représenter par la relation de spécialisation. Nous proposons une sémantique de la relation de spécialisation entre machines d'états UML. D'autre part, les techniques de construction incrémentale visent à définir les réalisations correspondant aux spécifications élaborées tout au long du processus de conception. Cette correspondance se décrit par la relation d'implantation. Nous proposons également une sémantique de cette relation entre machines d'états UML. Ainsi, nous disposons d'outils permettant de mettre en œuvre des démarches incrémentales et de raffinement en UML.

## 1 Introduction

Pour certains langages comme les modèles ensemblistes (Z ou B), l'intérêt des démarches de construction et de vérification de modèles par raffinements successifs est reconnu (Abrial, 1996). La technique de raffinement présente de nombreux avantages ayant conduit à des succès industriels, notamment dans le domaine ferroviaire. Néanmoins, celle-ci a un coût qui ne la rend pas souvent industriellement acceptable. D'une part, développer plusieurs spécifications formelles successives est exigeant. D'autre part, attendre d'avoir une spécification suffisamment précise pour déployer une réalisation présente un coût que peu de compagnies ne peuvent se permettre.

Dans ce papier, nous proposons de formaliser une démarche de construction *incrémentale* de modèles comportementaux UML (machines d'états). La différence avec une pure approche de raffinements est que, à chaque raffinement de la spécification, une réalisation, ou plutôt un modèle de la réalisation est également proposée. Celle-ci peut-être vue comme une version intermédiaire du produit, fonctionnellement et qualitativement non achevée, mais opérationnelle.

## Raffinement et réalisation incrémentale

L'avantage d'une telle démarche est multiple : elle permet de mieux tenir les coûts et temps de développement des systèmes ; elle contribue à réduire le « mauvais stress » des équipes de réalisation, en le transformant au contraire en source de motivation.

Cette démarche est présentée ici dans le cadre du développement de modèles de machines d'états UML. Ces dernières, à haut niveau d'abstraction, peuvent être définies comme des spécifications, alors qu'à un niveau de détails plus fin, elles peuvent correspondre à des modèles de la réalisation.

Un cadre théorique est développé pour cela. Il consiste à fournir une sémantique opérationnelle des machines d'états UML ainsi qu'une sémantique des relations de spécialisation et d'implantation entre machines d'états. Les résultats de calculs sont exécutés sur les modèles sémantiques.

L'ensemble de ces travaux est illustré sur l'exemple de l'atelier de Milner (1989), appelé par la suite *JobShop*. Milner prouvait le *JobShop* correct grâce à des spécifications et modèles CCS (*Calculus of Communicating Systems*). Ici, nous comparons les résultats que nous obtenons sur les modèles de classes UML et modèles de machines d'états UML. Nous montrons non seulement que la modélisation de machines d'états UML peut-être formelle elle-aussi, mais que la construction incrémentale est un guide et un gain de productivité important par rapport à des approches plus traditionnelles.

Ce papier est organisé de la manière suivante. Au paragraphe 2, nous reprenons un cas d'étude développé par Robin Milner. Nous montrons qu'il est possible de suivre une approche formelle similaire en UML, intégrant des étapes de construction et de vérification intermédiaires. Le paragraphe 3 permet de situer nos travaux par rapport aux travaux existants et présente les définitions fondamentales nécessaires à la compréhension de la sémantique donnée ensuite aux machines d'états. Au paragraphe 4, nous présentons la traduction des machines d'états en systèmes de transitions étiquetées, ainsi que la traduction de compositions de machines. Les relations de spécialisation et d'implantation UML sont également traduites. Les conclusions et perspectives sont présentées au paragraphe 5.

## 2 Raffinement et construction incrémentale

Ci-dessous, nous examinons les notions générales de raffinement de modèles, puis de construction incrémentale.

### 2.1 Raffinement et cohérence de modèles UML

Les travaux relatifs au raffinement de modèles UML se trouvent dans le cadre plus général des études de cohérence entre modèles UML. Dans OMG (2003), le raffinement est vu généralement comme la relation existant entre un modèle spécifié à un certain niveau de détail et une spécification enrichie, comme par exemple entre un modèle d'analyse et un modèle de conception. Huzar et al. (2005) et Hnatkowska et al. (2004) donnent une vision plus précise, qui peut s'exprimer sur les diagrammes de collaboration en termes de propriétés relatives aux classes et opérations raffinées.

Deux types de raffinement sont envisagés. Le premier que l'on appellera « *transformationnel* » consiste à ajouter des éléments de modélisation afin d'obtenir dans l'étape finale un modèle de la réalisation du système. Le second point de vue, que l'on appellera « *additionnel* »,

consiste à étendre les fonctionnalités d'un système existant afin de définir les spécifications d'un nouveau système. Notons que ces deux points de vue ne sont pas exclusifs, et qu'une démarche de raffinement peut les combiner.

Ainsi, Van Der Straeten (2004) propose une définition du raffinement sur les machines d'états, en le rapprochant du problème de *refactoring* de modèles. Le processus de raffinement consiste à ajouter à chaque étape des détails plus concrets. La préservation des propriétés est ici étudiée à l'aide de logique de description.

Nous retiendrons que la technique de raffinement cherchée consiste à ajouter des détails concrets et à réduire l'indéterminisme. Un modèle raffiné est plus précis (ajout de détails ou de fonctionnalités) et plus concret (l'abstraction est réduite, les fonctionnalités nécessaires sont préservées). Ceci est en accord avec les définitions précédentes, mais à l'inverse de Van Der Straeten (2004) nous ne souhaitons pas exprimer explicitement les propriétés à préserver. Nous préférons nous orienter vers une démarche de vérification en intensification des propriétés liées à l'ajout de détails et à la réduction de l'indéterminisme.

## 2.2 Relations existantes : vers une relation de raffinement sur des modèles comportementaux

Dans cette partie, nous donnons une présentation informelle et une comparaison des principaux préordres définis sur les LTS, voir Milner (1999), susceptibles de convenir en tant que relation de raffinement. Dans un premier temps nous considérerons les systèmes de transitions étiquetées et ensuite les machines d'états UML. Nous cherchons une relation asymétrique et transitive.

### 2.2.1 Relations de raffinement sur les LTS.

Afin de donner une sémantique des machines d'états UML, les systèmes de transitions étiquetées seront présentés formellement dans la suite, la compréhension intuitive utile pour le moment est qu'il s'agit de machines interagissant avec leur environnement au moyen d'*actions*. Ce sont des descriptions simples et abstraites, ne précisant ni si les actions sont en entrée ou en sortie (il n'y a pas de distinction événement / action), ni dans quelles circonstances une action est préférable à une autre (il n'y a pas de garde), ni combien de temps leur exécution demande ou après combien de temps leur exécution sera déclenchée. Elles permettent de décrire aisément des comportements indéterministes, ainsi que la réalisation de traitements « internes » au moyen d'une action spéciale.

**Inclusion de traces et relations de simulation.** Dans le cas des LTS, ajouter des détails consiste à définir de nouvelles traces et éventuellement de nouvelles actions. Une trace est une suite d'actions observables s'exécutant à partir de l'état initial. L'ajout de traces répond au premier critère souhaité sur le raffinement. Cependant, la description étendue peut être moins déterministe que l'originale, dans le sens où elle pourra refuser des fonctionnalités définies comme nécessaires dans la spécification. De même, les relations de simulation proposées par ? ne sont pas adéquates, puisqu'un modèle simule une spécification dès qu'il *peut* faire ce que la spécification définit. Nous voulons définir un raffinement qui *doit* accepter ce que la spécification doit accepter.

**Relations de conformité.** La relation **conf** a été proposée comme une relation d'implantation pour vérifier si un modèle d'une implantation respecte sa spécification (Leduc (1991a); Tretmans (1999)). C'est une formalisation de la relation de conformité définie pour le test. Une implantation  $I$  est conforme à sa spécification  $S$  si elle doit faire tout ce que  $S$  doit faire. La relation **conf** capture ainsi spécifiquement la réduction de l'indéterminisme. Mais elle ne garantit par l'ajout de traces. De plus, elle n'est pas transitive.

Les relations d'extension et de réduction (cf. Leduc (1991a); Tretmans (1999)) sont des relations de conformité combinées à des extensions ou des réductions de traces. Les deux sont transitives. Ainsi, la relation d'extension combine la réduction de l'indéterminisme et l'extension de traces. Cependant, ces relations (**conf**, *red* and *ext*) s'avèrent difficiles à vérifier en pratique, cf. Laroussinie et Schnoebelen (2000). Des résultats intéressants ont été trouvés sur les relations similaires que sont les préordres de possibilité et de nécessité définis par Hennessy et implantés par Cleaveland et Hennessy (1992).

La relation d'extension est ainsi la seule qui présente les deux critères recherchés. Examinons plus en détails si elle répond à la définition d'une relation de raffinement. Leduc (1991a) a établi que **conf** et *ext* sont telles que :

$$R \text{ ext } A \Rightarrow \forall P. (P \text{ conf } R \Rightarrow P \text{ conf } A). \quad (1)$$

En considérant que **conf** est une relation d'implantation, cette dernière propriété établit que *ext* est une relation de raffinement, mais que ce n'est pas la plus large. Elle est plus forte que la relation de raffinement  $\sqsubseteq$  caractérisée par :

$$R \sqsubseteq A \Leftrightarrow \forall P.(P \text{ imp } R \Rightarrow P \text{ imp } A). \quad (2)$$

où  $R \sqsubseteq A$  signifie que  $R$  raffine  $A$  et  $P \text{ imp } R$  signifie que  $P$  est une implantation de  $R$ . Ceci rejoint les formulations données par exemple dans Boiten et Bujorianu (2003), en accord avec la définition de raffinement en langage  $B$  ou  $Z$  de Abrial (1996).

La relation *ext* est par conséquent utile dans le cas où elle est satisfaite (puisqu'elle garantit alors que le raffinement est correct), et peut être utilisée comme un avertissement dans le cas où elle ne l'est pas.

### 2.2.2 Relations de raffinement sur les machines d'états UML.

Nous considérons qu'une machine d'états UML décrit les comportements attendus des objets d'une classe. Si une classe  $C_R$  est une spécialisation d'une classe  $C_A$ , ce que l'on peut écrire  $C_R \text{ extends } C_A$ , cela signifie que toute instance de  $C_R$  peut se comporter comme les instances  $C_A$  : toute instance de  $C_R$  est aussi une instance de  $C_A$ . Plus formellement, nous pouvons écrire :

$$C_R \text{ extends } C_A \Rightarrow \text{Instances}(C_R) \subseteq \text{Instances}(C_A). \quad (3)$$

Une définition plus précise peut se trouver dans Ducournau (2002). Les propriétés (1) et (3) sont similaires. Ceci signifie que la relation de spécialisation sur les classes est plus forte qu'une relation de raffinement.

Afin de considérer la signification de cette relation de spécialisation sur les machines d'états associées, notre point de vue est de considérer les LTS comme des interprétations abstraites des machines d'états. Nous ne définissons pas pour l'instant de relation détaillée sur les

machines d'états (comme nous l'avons fait pour la conformité dans Gout (2006)), mais nous élaborons une traduction des machines d'états UML vers les LTS. L'interprétation abstraite des résultats obtenus nous permet de dire que si la relation n'est pas satisfaite sur les LTS, elle ne l'est pas sur les machines d'états.

### 2.3 Construction incrémentale

Nous appelons construction incrémentale la démarche qui consiste à mener conjointement le développement des réalisations avec celui des spécifications successives. Plutôt que d'attendre d'avoir une spécification suffisamment aboutie pour en dériver une implantation, la construction incrémentale fait le choix d'exploiter comme version délibérément partielle ou provisoire une première version de réalisation. Cette démarche s'est révélée très productive industriellement. Elle rejoint les techniques de *extreme programming* et prototypage rapide. Une présentation de cette approche a été formulée par Batory (2008).

Notre objectif ici est d'outiller formellement la démarche de construction incrémentale sur la construction de modèles de machines d'états UML. Les spécifications et implantations considérées seront donc des diagrammes de classes et machines d'états UML.

En donnant une sémantique des machines d'états sur des systèmes de transitions étiquetées, comme nous l'avons évoqué ci-dessus pour la relation de raffinement, nous proposons une sémantique de la relation d'implantation entre machines d'états UML grâce à la relation de conformité entre LTS.

### 2.4 Définitions formelles

Un LTS est un graphe d'états reliés par des transitions étiquetées. Il peut modéliser la spécification ou l'implantation de comportements.

**Définition 1 (Systèmes de transitions étiquetées)** *Un LTS  $P = (S, Act, \rightarrow, s_0)$  est un quadruplet formé d'un ensemble non vide  $S$  d'états, d'un ensemble  $Act$  de noms d'actions, d'une relation de transitions  $\rightarrow \subseteq S \times Act \times S$ , et d'un état initial  $s_0 \in S$ .*

On a  $Act = L \cup \{\tau\}$  où  $L$  est l'ensemble des actions observables et  $\tau$  représente toute action interne, non observable. Soient  $P$  et  $Q$  deux LTS,  $a, a_i$  des actions de  $Act$  et  $\sigma \in L^*$

## Raffinement et réalisation incrémentale

une suite d'actions observables. Nous définissons :

$$\begin{aligned}
s &\xrightarrow{a} s' =_{def} (s, a, s') \in \rightarrow \\
s &\xrightarrow{a_1 \dots a_2} s' =_{def} \exists s_0, \dots, s_n. s = s_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_n = s' \\
s &\xrightarrow{a_1 \dots a_2} s' =_{def} \exists s'. s \xrightarrow{a_1 \dots a_n} s' \\
s &\xrightarrow{\varepsilon} s' =_{def} s = s' \text{ or } s \xrightarrow{\tau \dots \tau} s' \\
s &\xrightarrow{a} s' =_{def} \exists s_1, s_2. s \xrightarrow{\varepsilon} s_1 \xrightarrow{a} s_2 \xrightarrow{\varepsilon} s' \\
s &\xrightarrow{a_1 \dots a_2} s' =_{def} \exists s_0, \dots, s_n. s = s_0 \xrightarrow{a_1} \dots \xrightarrow{a_n} s_n = s' \\
s &\xrightarrow{\sigma} s' =_{def} \exists s'. s \xrightarrow{\sigma} s' \\
s \text{ after } \sigma &=_{def} \{s' \mid s \xrightarrow{\sigma} s'\} \\
P \text{ after } \sigma &=_{def} s_0 \text{ after } \sigma \\
\text{Traces : } Tr(P) &=_{def} \{\sigma \in L^* \mid s_0 \xrightarrow{\sigma}\} \\
Out(p) &=_{def} \{a \in L \mid p \xrightarrow{a}\} \\
Out(p, \sigma) &=_{def} \cup_{p' \in p \text{ after } \sigma} Out(p') \\
Out(P, \sigma) &=_{def} Out(s_0, \sigma)
\end{aligned}$$

L'ensemble d'acceptance de  $P$  après une trace  $\sigma$  se définit par :

**Définition 2 (Ensemble d'acceptance)**  $Acc(P, \sigma) = \{X \mid \exists p' \in P \text{ after } \sigma. X = Out(p', \varepsilon)\}$

Cette notion sera utilisée au paragraphe suivant pour construire des graphes d'acceptance. Un ensemble d'acceptance représente les différents ensembles possibles d'actions nécessaires après une trace. Intuitivement, l'inclusion d'ensembles d'acceptance nous permet de savoir si un processus est plus déterministe qu'un autre.

**Définition 3 (Inclusion d'ensemble d'ensembles)** Soient  $A, B \subseteq 2^{Act}$ .  $A \subset\subset B$  si  $\forall S \in A. \exists S' \in B. S' \subseteq S$ .

La conformité d'un processus à sa spécification traduit le fait qu'un processus *doit* accepter tout ce que sa spécification doit accepter.

**Définition 4 (Conformité)**  $P \text{ conf } Q$  si  $\forall \sigma \in Tr(Q). Acc(P, \sigma) \subset\subset Acc(Q, \sigma)$ .

L'extension et la réduction sont définies comme des extension ou réduction de traces préservant la conformité.

**Définition 5 (Réduction)**  $Q \text{ red } P$  si  $Tr(Q) \subseteq Tr(P)$  et  $Q \text{ conf } P$ .

**Définition 6 (Extension)**  $Q \text{ ext } P$  si  $Tr(P) \subseteq Tr(Q)$  et  $Q \text{ conf } P$ .

La relation **conf** n'est pas transitive. Les relations *red* et *ext* sont réflexives et transitives.

### 3 Étude de cas

Ce paragraphe reprend l'exemple de l'atelier traité par Milner (1989) afin de montrer les avantages que présentent les démarches de construction de modèles comportementaux, qu'elles soient par raffinement ou incrémentale.

#### 3.1 Travaux de Milner

##### Énoncé

« Nous montrons comment modéliser une ligne de production simple. Nous supposons que deux personnes partagent l'utilisation de deux outils, un marteau (*Hammer*) et un maillet (*Mallet*), pour fabriquer des objets à partir de composants simples. Chaque objet est constitué en assemblant un manche dans un socle. On appellera *travail (job)* une paire constituée d'un socle et d'un manche. Les *jobs* arrivent séquentiellement sur un convoyeur et les objets assemblés repartent sur un convoyeur. L'atelier ("*JobShop*") peut impliquer un nombre quelconque de personnes, que nous appellerons "*jobbers*", partageant un ou plusieurs outils. Les travaux peuvent être de trois types : faciles à assembler (*EasyJob*) qui ne nécessitent aucun outil, difficiles (*HardJob*) qui nécessitent un marteau et moyennement difficiles (*MedJob*) nécessitant un maillet ou éventuellement, un marteau. »

##### Démarche suivie et résultats montrés par Robin Milner.

- définition de l'architecture globale finale et spécification des composants ("*agents* ou *processus* CCS) impliqués ;
- vérification de l'assemblage détaillé en le comparant formellement à sa spécification par la relation d'équivalence observationnelle :

$$Jobshop \approx StrongJobber \mid StrongJobber$$

Où l'opérateur  $\mid$  désigne la composition parallèle entre processus CCS. Il montre ainsi que l'atelier peut traiter deux travaux simultanément, et qu'il n'existe pas de blocages définitifs, bien que les ouvriers puissent parfois attendre un outil (blocage provisoire).

#### 3.2 Modélisation incrémentale en UML de l'atelier

Notre approche diffère de celle de Milner dans le sens où on ne propose pas immédiatement une solution d'architecture globale. Nous réalisons la modélisation de l'atelier en deux étapes (cf. figure 1). Dans la première étape, nous nous intéressons à une spécification simplifiée ( $S_0$ ) qui répond de façon incomplète à l'énoncé (un seul travail peut être réalisé à la fois) et nous en proposons une réalisation ( $R_0$ ). Dans la seconde étape, nous nous intéressons à une spécification ( $S_1$ ) répondant précisément à l'énoncé et nous en proposons une réalisation ( $R_1$ )

**1ère étape :  $S_0$  et  $R_0$ .** La spécification comportementale la plus abstraite et la plus simple ( $S_0$ ) que l'on peut donner de l'atelier est constituée d'une classe *StrongJobber* et d'une classe *Job*. La machine d'états associée à *StrongJobber* est donnée dans la figure 2.



## Raffinement et réalisation incrémentale

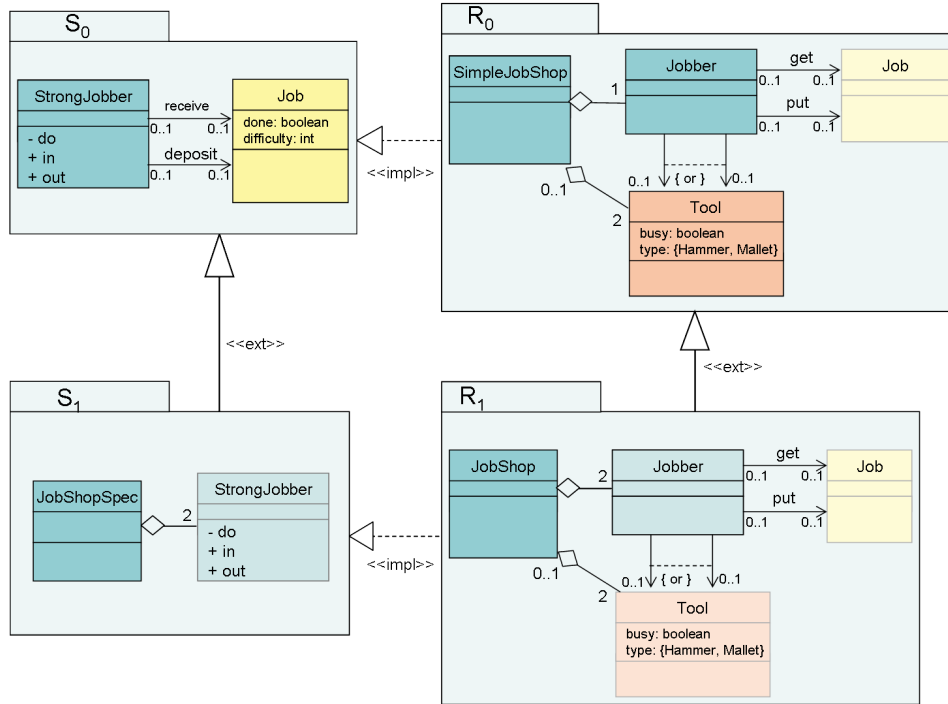


FIG. 1 – Diagramme de Classes.

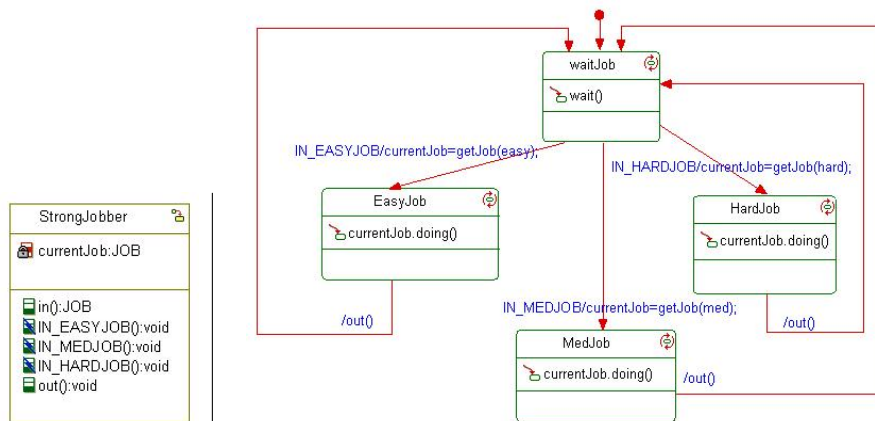


FIG. 2 – Classe StrongJobber et sa machine d'états.

Une réalisation possible ( $R_0$ ) de cette spécification consiste à avoir une instance de la classe *Jobber* qui utilisera un outil de type *Hammer* ou *Mallet* pour faire l'assemblage, outil choisi en fonction de la difficulté (attribut *level*) du *Job* entrant. L'outil est reposé lorsque l'assemblage est terminé. Si l'outil requis est déjà pris, *Jobber* attend qu'il se libère. La machine d'états de la classe *Jobber* est donnée dans la figure 3.

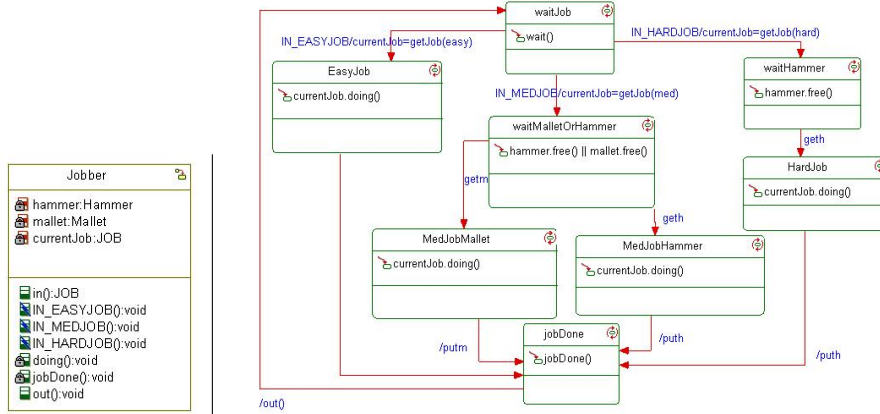


FIG. 3 – Classe *Jobber* et sa machine d'états.

**2ème étape :  $S_1$  et  $R_1$ .** La spécification plus précise que nous proposons ( $S_1$ ) consiste à assembler deux *StrongJobber* afin de pouvoir traiter deux *Jobs* en parallèle. Une réalisation de cette classe ( $R_1$ ) consiste à mettre en parallèle deux instances de la classe *Jobber*, les requêtes d'outils se faisant sur deux instances de la classe *Tool* (*Hammer* et *Mallet*). Les outils sont donc partagés entre les deux *Jobbers*.

Grâce aux travaux sémantiques et résultats que nous avons obtenus précédemment (Luong et al., 2008b,a; Lambolais et al., 2008), nous pouvons montrer qu'après traduction en LTS, nous avons les résultats suivants :

- Les relations de raffinement entre spécifications sont satisfaites :

$$LTS(S_1) \text{ ext } LTS(S_0)$$

- Les relations de raffinement entre réalisations sont satisfaites :

$$LTS(R_1) \text{ ext } LTS(R_0)$$

- Les relations d'implantation entre spécifications et réalisation sont satisfaites :

$$LTS(R_0) \text{ conf } LTS(S_0)$$

$$LTS(R_1) \text{ conf } LTS(S_1)$$

Nous montrons au paragraphe suivant comment définir  $LTS(SM)$ , où  $SM$  est une machine d'états UML.

## 4 Interprétation des machines d'états UML et des relations de spécialisation et d'implantation

### 4.1 Traduction de machines séquentielles

Nous traduisons les machines d'états UML en LTS. Les transitions distinguent l'événement déclencheur, une éventuelle garde et l'action possible associée, alors que la distinction événement et action n'existe pas en LTS. De plus, les états peuvent comporter des activités éventuellement décrites à leur tour par des machines d'états. Ce mécanisme de hiérarchie est également absent dans les LTS. Les événements et actions UML peuvent se traduire systématiquement par des actions LTS, nous distinguons cependant les événements explicites (appels de méthode et réceptions de signal) des événements implicites (événement de terminaison à la fin d'une activité). Les premiers peuvent se traduire par des actions visibles LTS et les seconds par l'action interne. Les événements de changement et temporisés seront également traduits par l'action interne. La correspondance globale des concepts est donnée dans le tableau 1. Il est ainsi nécessaire d'identifier en UML les actions observables ou non. Pour cela, on peut avoir recours à l'utilisation de *tags* UML, tel que nous l'avons manipulé dans Gout (2006).

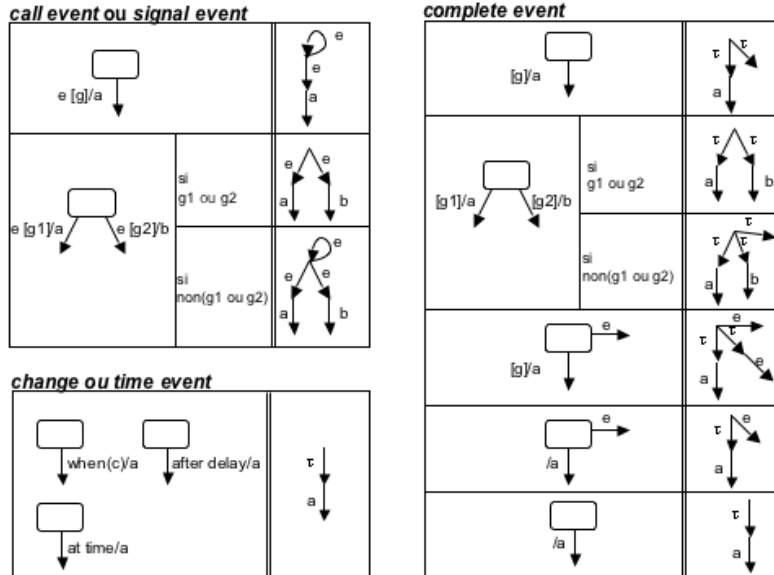
Machines d'états	LTS	Machines d'états	LTS
action observable	action	action non observable	$\tau$
<i>call ou change event</i>	action	<i>time, change, complete event</i>	$\tau$
activité visible	partie du LTS	activité interne	$\tau$
hiérarchie	LTS à plat	entry	$\tau$
gardes	plusieurs schémas. Indéterminisme. Voir tab. 2	exit	$\tau$
historique	non traité	pseudo-état de choix	non traité
spécialisation	<i>ext</i>	implantation	<b>conf</b>

TAB. 1 – Correspondance de concepts

Les gardes UML ne trouvent pas de correspondance exacte. Néanmoins, elles induisent des possibilités de blocages. Une transition UML «  $\frac{e[g]/a}{\rightarrow}$  » se traduit par le schéma  $S =_{\text{def}} e; a; S' + e; S$ . Après l'événement, la machine peut refuser d'exécuter l'action. Il n'y a pas de correspondance bijective entre états. Les schémas de traduction des transitions UML sont donnés au tableau 2.

La satisfaction des relations de conformité sur les LTS garantit l'absence de refus définitif. D'autres propriétés plus précises des machines d'états (telles qu'un refus provisoire ou conditionnel) ne sont pas détectées. Si les relations ne sont pas vérifiées sur les LTS, alors elles ne le sont pas sur les machines d'états, mais l'inverse n'est pas garanti.

Nous avons prouvé et programmé le calcul des relations **conf** et *ext* dans (Luong et al., 2008b,a; Lambolais et al., 2008).



TAB. 2 – Règles de transformation des machines d'états en LTS

## 4.2 Traduction d'une composition de machines d'états

Les "architectures UML" sont traduites comme des compositions parallèles de LTS, en utilisant les différents opérateurs du langage LOTOS (ISO, IS 8807, 1988) :

- composition parallèle synchronisée sur les dépendances :  $P|[...]|Q$
- composition parallèle indépendante :  $P|||Q$

Grâce à la sémantique de l'entrelacement proposée en LOTOS, nous obtenons ainsi un seul LTS pour une composition de machines d'états UML.

## 5 Conclusion

Dans ce papier, nous avons étudié la construction de machines d'états UML selon deux aspects : la construction par raffinement visant à obtenir une implantation à la fin d'une chaîne de spécification et la construction incrémentale visant à proposer différentes versions d'implantation en accord avec leur spécification. En UML, la notion de raffinement peut s'assimiler à la spécialisation. Nous en proposons une sémantique sur les systèmes de transitions étiquetées grâce à la relation d'extension. L'inconvénient de cette démarche est de ne pas proposer de versions intermédiaires de l'implantation. Les démarches de construction incrémentale présentent l'avantage de mener conjointement le développement de spécifications successives avec des modèles de leur réalisation. En proposant une sémantique de la relation d'implantation UML grâce à la relation de conformité définie sur les LTS, différentes étapes de la démarche incrémentale peuvent être vérifiées. Ces travaux correspondent à des démarches de construction formelles non encore transposées à UML.

## Références

- Abrial, J.-R. (1996). *The B-Book : Assigning Programs to Meanings*. Cambridge University Press.
- Batory, D. (2008). The objects and arrows of computational design.
- Boiten, E. et M. Bujorianu (2003). Exploring UML refinement through unification. In J. Jürjens, B. Rumpe, R. France, et E. Fernandez (Eds.), *Critical Systems Development with UML - Proceedings of the UML'03 workshop*, Number TUM-I0323, pp. 47–62. Technische Universität München.
- Cleaveland, R. et M. Hennessy (1992). Testing equivalence as a bisimulation equivalence. *Formal Aspects of Computing* 3.
- Ducournau, R. (2002). "Real world" as an argument for covariant specialization in programming and modeling. In *Proc. of the Workshops on Advances in OOIS*, Volume 2426 of LNCS, London, UK, pp. 3–12. Springer.
- Gout, O. (2006). *Développement incrémental de spécifications orientées objets*. Ph. D. thesis, École des mines d'Alès, université de Montpellier 2.
- Hnatkowska, B., Z. Huzar, et L. Tuzinkiewicz (2004). On Understanding of Refinement Relationship. *Third International Workshop, Consistency Problems in UML-based Software Development III—Understanding and Usage of Dependency Relationships*.
- Huzar, Z., L. Kuzniarz, G. Reggio, et J. Sourrouille (2005). Consistency Problems in UML-Based Software Development. *UML 2004 Satellite Activities*.
- ISO, IS 8807 (1988). *Information Processing Systems, Open Systems Interconnection, LOTOS—A Formal Description Technique Based On the Temporal Ordering of Observational Behaviour*.
- Lambolais, T., A.-L. Courbis, et H.-V. Luong (2008). Raffinement de modèles comportementaux uml, vérification des relations d'implantation et d'extension sur les machines d'états. *Research Report RR08/062, LGI2P*.
- Laroussinie, F. et P. Schnoebelen (2000). The State Explosion Problem from Trace to Bisimulation Equivalence. *LNCS 1784*, 192–207.
- Leduc, G. (1991a). Conformance relation, associated equivalence, and minimum canonical tester in lotos. *PSTV XI. North-Holland*.
- Leduc, G. (1991b). *On the role of implementation relations in the design of distributed systems using LOTOS*. Ph. D. thesis, Université de Liège, Faculté des sciences appliquées.
- Luong, H.-V., T. Lambolais, et A.-L. Courbis (2008a). Implementation of extension and reduction relations for incremental development of behavioural models. *Research Report RR08/057, LGI2P*.
- Luong, H.-V., T. Lambolais, et A.-L. Courbis (2008b). Implementation of the conformance relation for incremental development of behavioural models. *LNCS 5301*, 356–370.
- Milner, R. (1989). *Communication and Concurrency*. HOARE, C.A.R., Prentice Hall International. Series in Computer Science.
- Milner, R. (1999). *Communicating and Mobile Systems : The  $\pi$  Calculus*. Cambridge University Press.

- OMG (2003). Uml 2.0 superstructure specification. OMG Adopted Specification ptc/03-08-02.
- Tretmans, J. (1999). Testing concurrent systems : A formal approach. *CONCUR 99, LNCS* (1664).
- Van Der Straeten, R. (2004). Formalizing Behaviour Preserving Dependencies in UML. *Third International Workshop, Consistency Problems in UML-based Software Development III—Understanding and Usage of Dependency Relationships*.

## Summary

We are interesting in modelling approaches to set up behavioural UML models (state machines) integrating verification techniques. Refinement, consisting in defining more precise or complete specifications has not been yet formalised on state machines. In object oriented models, refinement relation can be represented as a specialisation relation. We propose a semantics of this relation over UML state machines. Moreover, incremental modeling techniques aims at associating implementations to specifications defined all over the design process. This association can be defined as an implementation relation. We also propose a semantics of this relation over UML state machines. By this way, we have developped tools allowing refinement and incremental approaches to be applied.