Intégration de données formelles dans les diagrammes d'états d'UML

Christian Attiogbé*, Pascal Poizat**, Gwen Salaün*

*IRIN, Université de Nantes
2 rue de la Houssinière, B.P. 92208, 44322 Nantes Cedex 3, France
email: {attiogbe,salaun}@irin.univ-nantes.fr

**LaMI - UMR 8042, Université d'Évry Val d'Essonne
Tour Évry 2, 523 Place des terrasses de l'Agora, 91000 Évry, France
email: poizat@lami.univ-evry.fr

Résumé : Dans cet article, nous présentons une approche générique pour intégrer des données exprimées dans des langages de spécification formelle à l'intérieur de diagrammes d'états d'UML. Les motivations principales sont d'une part de pouvoir modéliser les aspects dynamiques des systèmes complexes avec un langage convivial et graphique tel que les diagrammes d'états d'UML ; d'autre part de pouvoir spécifier de façon formelle et potentiellement à un haut niveau d'abstraction les données mises en jeu dans ces systèmes à l'aide de spécifications algébriques ou de spécifications orientées état (telles que celles écrites en Z ou en B). Cette approche introduit une utilisation flexible et générique des données. Elle permet aussi de prendre en compte différentes sémantiques associées aux diagrammes d'états. Nous présentons d'abord les fondements formels de notre approche puis une étude de cas vient illustrer le pragmatisme de la proposition.

Mots-clés : intégration de spécifications formelles et semi-formelles, diagrammes d'états, UML, spécifications algébriques, Z, B.

1 Introduction

Depuis quelques années, la notation UML [37] est devenue omniprésente en conception orientée-objet et ce, principalement grâce à sa convivialité (notations graphiques) et sa relative souplesse d'emploi. Elle permet de décrire les différents aspects des systèmes complexes (aspects statiques – incluant les types de données et les aspects fonctionnels –, aspects dynamiques – incluant la concurrence et les communications – et aspects architecturaux) grâce à ses différents types de diagrammes. Cependant, UML souffre d'une critique incessante concernant son manque de sémantique formelle [39,9], ce qui pose problème pour s'assurer de la cohérence des systèmes décrits à l'aide de plusieurs diagrammes et plus généralement pour les étapes de vérification qui suivent la modélisation de ces systèmes. La formalisation d'UML fait d'ailleurs aujourd'hui l'objet de nombreux travaux dont les principaux sont issus du Precise UML Group [25].

D'un autre côté, les méthodes formelles existent maintenant depuis plusieurs décennies et ont pour finalité la description de systèmes logiciels mais de façon mathématique et non ambiguë. Leur principal intérêt est que la sémantique des langages de spécification est bien définie et permet donc de vérifier la correction des systèmes spécifiés. Les spécifications formelles rendent aussi possible la description des systèmes à un haut niveau d'abstraction. En revanche, leur difficulté d'apprentissage et d'utilisation leur a souvent été reprochée.

L'utilisation conjointe de spécifications formelles et semi-formelles semble donc être un domaine prometteur avec pour objectif de profiter des avantages de ces deux types d'approches : notations graphiques et lisibilité des approches semi-formelles, haut niveau d'abstraction, expressivité, cohérence et moyens de vérification des approches formelles. Dans cet article, nous proposons une approche qui répond à cette motivation. Elle permet de spécifier à l'aide d'un langage intégré les différents aspects des systèmes complexes.

Au niveau des aspects dynamiques, notre travail est partiellement générique. Différentes sémantiques de l'aspect dynamique peuvent être prises en compte, et notre approche est potentiellement

applicable aux Statecharts [26], aux différentes sémantiques des diagrammes d'états d'UML (comme celles de [32,30,43,44]), et plus généralement à tout système à base d'états et de transitions. Dans cet article nous nous intéressons plus particulièrement aux diagrammes d'états d'UML. Les aspects statiques et fonctionnels sont spécifiés à l'aide de langages de spécification formelle (tels que les spécifications algébriques [10], Z [42] ou B [7]). Ces spécifications formelles sont utilisées à la place des spécifications semi-formelles des données qui sont habituellement faites en utilisant les diagrammes de classes. Ceci permet de vérifier les spécifications statiques mais aussi potentiellement de décrire les données à un haut niveau d'abstraction. La flexibilité que nous proposons quand au choix du langage de spécification des aspects statiques permet au spécifieur de choisir le langage formel qui lui parait le plus adapté à cette tâche : celui qu'il connaît le mieux, celui qui est le mieux outillé ou bien encore celui qui lui permet de réutiliser des spécifications statiques déjà écrites.

Dans notre approche, la spécification est dirigée par le contrôle : le comportement dynamique est le comportement principal de la spécification et il décrit comment les données de l'aspect statique sont manipulées. Ce travail s'intègre complètement dans la thématique plus générale de l'*intégration* ou *combinaison* de méthodes formelles où nous avons déjà eu plusieurs résultats [8]. Au niveau global de la spécification, notre approche permet d'aborder les problèmes de cohérence entre parties statiques et dynamiques de descriptions de systèmes.

Le reste de l'article se décompose comme suit. La section 2 présente les fondements de notre approche : les liens syntaxiques entre les diagrammes d'états et les données ainsi que la sémantique de l'intégration. La section 3 illustre ensuite de façon pragmatique comment notre proposition peut être appliquée pour spécifier un système concret (une station essence). Dans la section 4, nous présentons les travaux connexes à notre approche. Enfin, dans la section 5, nous concluons sur ce travail et discutons des perspectives possibles.

2 Fondements formels de la combinaison

Dans cette section, nous formalisons la combinaison des diagrammes d'états d'UML avec des données formelles. Nous étudions successivement les aspects syntaxiques et sémantiques.

2.1 Aspects syntaxiques

Nous détaillons tout d'abord les interactions entre les diagrammes d'états et les données qui s'y intègrent. Nous suivons une approche dirigée par le contrôle. Le comportement principal de la spécification est par conséquent donné par la modélisation de la partie dynamique (comportements et communications). Les données apparaissant dans les diagrammes d'états correspondent à des termes exprimés à partir de définitions formelles de types de données.

Nous étendons tout d'abord les diagrammes d'états avec l'importation de modules et les déclarations de variables locales. Ces déclarations sont effectuées en utilisant les *notes* d'UML qui permettent habituellement de décrire des informations de tout type sous forme textuelle. Une notation (`IMPORT`, Fig. 1) est introduite à ce niveau pour indiquer quels modules (spécifications algébriques, schémas Z, machines B, etc) sont importés ainsi que le langage utilisé pour écrire leurs contenus. Un tel language est appelé *cadre* dans notre approche et est utilisé pour définir les fonctions d'évaluation qui servent ensuite à évaluer les données encapsulées dans les diagrammes d'états. Des variables peuvent aussi être déclarées et typées dans les notes. Puisque les modules contiennent souvent plusieurs définitions de type, et puisque les types de même nom peuvent être définis dans deux modules différents, le type d'une variable peut être préfixé avec le nom d'un module afin d'éviter d'éventuels conflits.

La forme générale d'une transition dans les diagrammes d'états est rappelée dans la Figure 2.

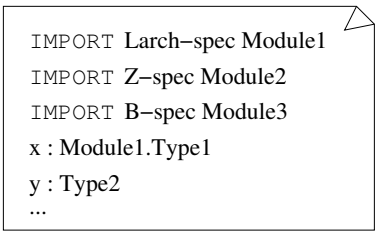

FIG. 1 – Déclaration locale de données dans les diagrammes d'états

FIG. 2 – Transition dans les diagrammes d'états

Les interactions entre les parties dynamiques et statiques se situent à différents niveaux : à l'intérieur des états et sur les transitions (Tableau 1 pour des exemples illustratifs et Tableau 2 pour la grammaire). Au niveau des transitions, nos extensions permettent de (i) recevoir des valeurs portées par des événements et conserver ces valeurs dans des variables locales, (ii) exprimer les gardes des transitions en utilisant des données (prédicats), (iii) envoyer des événements avec des données à l'intérieur, (iv) effectuer des modifications locales de données. Les deux dernières prennent place dans la partie ACTION des transitions. La possibilité d'activités étendues à l'intérieur des états n'est pas directement prise en considération dans notre approche. Cependant, de telles activités peuvent être vues comme des cas particuliers de transitions entre états.

Partie de l'étiquette	Sorte d'interaction	Exemple avec données
EVENEMENT	réception	*nom-evenement*(x_1 :T_1,...,x_n :T_n)
GARDE	garde	*predicat*(t_1,...,t_n)
ACTION	émission	*nom-evenement*(t_1,...,t_n)
ACTION	modification locale	$x := t$

TAB. 1 – Interactions entre diagrammes d'états et données

Les données sont soit des variables, soit des termes algébriques, soit des applications d'opérations pour les langages orientés état (*e.g.* Z, B). En ce qui concerne le langage formel pour les aspects statiques, la seule contrainte est d'avoir un mécanisme d'évaluation bien défini. Notre approche permet d'utiliser conjointement plusieurs langages statiques en même temps. Cependant, un mélange fort de spécifications écrites avec des langages différents n'est pas autorisé afin d'éviter de possibles incompatibilités sémantiques. Afin de vérifier ces contraintes au sein des diagrammes, nous proposons un concept de *méta-type* basé sur des règles de méta-typage (voir la sémantique ci-dessous). Les méta-termes mal méta-typés ne peuvent pas être évalués suivant notre approche.

2.2 Aspects sémantiques

Notre but est de proposer une sémantique formelle des interactions entre comportements et données au sein des diagrammes d'états. Il ne s'agit donc pas de formaliser complètement les diagrammes

```
TRANSITION  ::= [EVENEMENT] [[GARDE]] [/ ACTION+]
EVENEMENT   ::= nom-evenement(DECL-VAR+)
DECL-VAR    ::= VAR : TYPE
GARDE       ::= PREDICAT
ACTION      ::= EMISSION | AFFECTATION
EMISSION    ::= nom-evenement(TERME-DONNEE+)
AFFECTATION ::= VAR := TERME-DONNEE
```

TAB. 2 – Grammaire BNF des interactions

d'états d'UML tel que cela a été fait par exemple dans [32,30,43,44]. Nous conservons une description très générale de ces diagrammes ; de cette façon, par extension, tous les diagrammes d'états et leurs différentes sémantiques sous-jacentes pourront être considérées. Par contre, nous définissons précisément le sens de l'intégration des données dans les diagrammes dynamiques au travers d'une sémantique opérationnelle, et particulièrement par des enrichissements d'environnement et des fonctions d'évaluation.

Pour représenter la sémantique des opérations sur les données, nous nous inspirons du travail de Bruns [15]. Dans la suite, nous posons $\mathcal{D}$ comme étant l'ensemble des diagrammes d'états avec leur sémantique associée (donnée en terme d'états et de transitions). Dans les règles qui suivent, nous considérons un diagramme D qui appartient à $\mathcal{D}$. Nous posons également $\mathcal{T}_D$ comme étant l'ensemble des transitions de D, $\mathcal{S}_D$ l'ensemble de ses états et $\mathcal{S}_{0D}$ l'ensemble de ses états initiaux. Nous notons les éléments de $\mathcal{T}_D$ sous la forme $S \xrightarrow{label} S'$, où *label* a la forme *événement[garde]/action*.

Un environnement de liaison de variables E est un ensemble fini de paires (x, v) qui indiquent que la variable x est liée à la valeur v dans l'environnement E. La notation $E \vdash e \rhd_X v$ signifie qu'en utilisant l'évaluation définie dans le cadre X (*e.g.* Larch, CASL, Z, B) alors v est une évaluation possible de e dans le contexte de E. Nous restons ici volontairement succincts, des détails concernant la sémantique de $\rhd_X$ sont donnés dans la suite. Les éventuelles variables libres présentes dans e sont valuées en utilisant les liaisons définies dans les environnements. De plus, si E et E' sont des environnements alors EE' est l'environnement dans lequel les variables de E et E' sont définies, les liaisons de E' écrasant celles de E. Pour définir les règles d'inférence, nous utilisons la notation $E \vdash decl \Rightarrow E'$ qui dénote que la déclaration *decl* transforme l'environnement E en E'. $TRUE_X$ désigne la valeur de vérité *vrai* dans le cadre X. Pour connaître le cadre X auquel correspond une construction e donnée (*i.e.* une variable, un terme ou une application d'opération), nous définissons des règles de typage particulières que nous appelons règles de *méta-typage* en ce sens qu'elles ne donnent pas un type mais un méta-type (Larch ou Z par exemple) à une construction. Nous utiliserons T pour les types usuels et X pour les méta-types.

Règles de méta-typage. Dans la suite, les fonctions $DeclImp(D)$ et $DeclVar^!(D)$ dénotent respectivement les importations et les déclarations de variables décrites dans les notes d'un diagramme D. $DeclVar^?(D)$ représente l'ensemble des variables (typées) reçues par les événements. *DeclVar* est l'union de $DeclVar^?(D)$ et $DeclVar^!(D)$. La fonction *def* teste si une déclaration d'un type T ou d'une opération *op* est présente dans un module M. La notation $t ::_D X$ signifie que t a X comme méta-type à l'intérieur du diagramme D. Les règles de méta-typage sont données dans la Figure 3. La première permet de méta-typer les variables en se servant des déclarations locales de données. La seconde règle donne par induction le méta-type d'une construction à partir des méta-types des éléments qui la composent.

Les règles de la sémantique se décomposent en trois groupes. Le premier décrit les modifications d'environnement découlant de l'exécution des transitions des diagrammes d'états contenant des don-

$$\frac{D \in \mathcal{D} \quad \texttt{IMPORT}\ X\text{–}SPEC\ M \in DeclImp(D) \quad def(T, M) \quad x : T \in DeclVar(D)}{x ::_D X}$$

$$\frac{D \in \mathcal{D} \quad \texttt{IMPORT}\ X\text{–}SPEC\ M \in DeclImp(D) \quad def(op, M) \quad \forall\, i \in 1..n\ .\ t_i ::_D X}{op(t_1, ..., t_n) ::_D X}$$

FIG. 3 – Règles de méta-typage

nées. Le second groupe de règles présente l'évolution des aspects dynamiques et leur influence sur les modifications et/ou mises à jour des environnements. En ce qui concerne ces deux groupes de règles, nous nous plaçons dans un cadre abstrait qui nous permet de considérer n'importe quelle sémantique des diagrammes d'états. Ces diversités sémantiques s'expliquent principalement par des divergences de choix dans le modèle de communication. Nous montrons comment ces règles peuvent être complétées (troisième groupe) afin de prendre en compte un type précis de communication. Ici, nous nous intéressons aux diagrammes d'états d'UML et particulièrement à la communication de type *broadcast* et complètons notre sémantique dans ce sens.

Règles d'évolution des environnements. Nous détaillons à présent le premier groupe de règles en traitant chaque construction introduite dans le Tableau 1. Avant de donner les règles d'inférence, nous rappelons en encadré la construction grammaticale concernée (Tab. 2).

`AFFECTATION ::= VAR := TERME-DONNEE`

$$\frac{D \in \mathcal{D} \quad \exists X\ .\ y ::_D X\ .\ t ::_D X\ .\ E \vdash t \rhd_X w}{E \vdash y := t \Rightarrow E\{(y, w)\}}$$

Les affectations ou les applications d'opérations mettent à jour les environnements. Concernant les spécifications algébriques, la compréhension de cette règle semble assez naturelle (le terme t se réécrit en w). Pour les langages orientés état, l'interprétation de la règle est quelque peu différente. La variable y désigne l'état du système. La valeur w dénote le nouvel état obtenu à partir de l'environnement et par évaluation de t. La notation utilisée dans cette règle est propre à notre approche et légèrement distincte des notations usuelles en spécification orientée état (*i.e.* pointées ou à effet de bord). Elle se justifie par notre volonté de disposer d'une syntaxe commune aux différents langages de données considérés.

`EVENEMENT ::= nom-evenement (DECL-VAR+)`

$$\frac{\forall\, i \in 1..n\ .\ \exists\, v_i : T_i}{E \vdash evt(x_1 : T_1, \ldots, x_n : T_n) \Rightarrow E\{(x_1, v_1), \ldots, (x_n, v_n)\}}$$

Une réception lie les variables en entrée avec des valeurs du bon type. Enfin, les gardes et les émissions (règles ci-dessous) ne modifient pas localement l'environnement.

`GARDE ::= PREDICAT`

$$\frac{}{E \vdash G \Rightarrow E}$$

`EMISSION ::= nom-evenement (TERME-DONNEE+)`

$$\frac{}{E \vdash evt(e_1, \ldots, e_n) \Rightarrow E}$$

Règles dynamiques. Au premier ensemble de règles viennent s'ajouter des règles dynamiques (Fig. 4) qui symbolisent l'influence du comportement du diagramme sur les environnements décrits précédemment. Deux règles décrivent respectivement l'initialisation et l'évolution individuelle d'un diagramme. À ce niveau, nous faisons complète abstraction d'une quelconque sémantique particulière de la communication dans les diagrammes d'états.

$$\frac{\begin{array}{c} D \in \mathcal{D} \\ S \in \mathcal{S}_{0D} \\ DeclVar^{!}(D) = \{x_1 : T_1, \ldots, x_n : T_n\} \\ \forall\, i \in 1..n \;.\; \exists\, v_i : T_i \end{array}}{\vdash S \Rightarrow \{(x_1, v_1), \ldots, (x_n, v_n)\}}$$

$$\frac{\begin{array}{c} D \in \mathcal{D} \\ S \in \mathcal{S}_D \;\wedge\; S' \in \mathcal{S}_D \\ S \xrightarrow{EVT\ G\ /\ ACT} S' \in \mathcal{T}_D \\ \exists X \;.\; G ::_D X \\ E \vdash EVT \Rightarrow E' \\ E' \vdash G \rhd_X TRUE_X \\ E' \vdash ACT \Rightarrow E'' \end{array}}{E \vdash S \xrightarrow{EVT\ G\ /\ ACT} S' \Rightarrow E''}$$

FIG. 4 – Règles dynamiques

Règles de communication. Nous avons jusqu'ici décrit comment les environnements et les transitions évoluent. Lorsque nous nous intéressons à une sémantique particulière de la communication alors nous ajoutons des règles pour la gérer. Pour illustrer cette dernière étape, nous choisissons le modèle de communication des diagrammes d'états d'UML, et plus particulièrement le cas des états concurrents (communication synchrone en *broadcast*). Nous rappelons que plusieurs états concurrents se synchronisent (implicitement) sur des événements de même nom. La règle est donnée dans la Figure 5. Nous avons n receveurs et un émetteur, ce que nous notons $concur(D_{emet}, \{D_{rec_1}, \ldots, D_{rec_n}\})$.

$$\frac{\begin{array}{c} D_{emet} \in \mathcal{D} \;\wedge\; \forall\, i \in 1..n \;.\; D_{rec_i} \in \mathcal{D} \\ concur(D_{emet}, \{D_{rec_1}, \ldots, D_{rec_n}\}) \\ S_{emet} \in \mathcal{S}_{D_{emet}} \;\wedge\; S'_{emet} \in \mathcal{S}_{D_{emet}} \\ S_{emet} \xrightarrow{EVT_{emet}\ G_{emet}\ /\ evt(e_1,\ldots,e_m)} S'_{emet} \in \mathcal{T}_{D_{emet}} \\ E_{emet} \vdash EVT_{emet} \Rightarrow E'_{emet} \\ \exists X_{emet} \;.\; G_{emet} ::_{D_{emet}} X_{emet} \\ E'_{emet} \vdash G_{emet} \rhd_{X_{emet}} TRUE_{X_{emet}} \\ \forall\, j \in 1..m \;.\; e_j ::_{D_{emet}} X_j \;.\; E'_{emet} \vdash e_j \rhd_{X_j} v_j \\ \forall\, i \in 1..n \;.\; S_{rec_i} \in \mathcal{S}_{D_{rec_i}} \;\wedge\; \forall\, i \in 1..n \;.\; S'_{rec_1} \in \mathcal{S}_{D_{rec_i}} \\ \forall\, i \in 1..n \;.\; S_{rec_i} \xrightarrow{evt(x_1:T_1,\ldots,x_m:T_m)\ G_{rec_i}\ /\ ACT_{rec_i}} S'_{rec_i} \in \mathcal{T}_{D_{rec_i}} \\ \forall\, i \in 1..n \;.\; E'_{rec_i} = E_{rec_i}\{(x_1, v_1), \ldots, (x_m, v_m)\} \\ \forall\, i \in 1..n \;.\; \exists\, X_{rec_i} \;.\; G_{rec_i} ::_{D_{rec_i}} X_{rec_i} \\ \forall\, i \in 1..n \;.\; E'_{rec_i} \vdash G_{rec_i} \rhd_{X_{rec_i}} TRUE_{X_{rec_i}} \\ \forall\, i \in 1..n \;.\; E'_{rec_i} \vdash ACT_{rec_i} \Rightarrow E''_{rec_i} \end{array}}{\begin{array}{c} E_{emet} \vdash S_{emet} \xrightarrow{EVT_{emet}\ G_{emet}\ /\ evt(e_1,\ldots,e_m)} S'_{emet} \Rightarrow E'_{emet} \\ \forall\, i \in 1..n \;.\; E_{rec_i} \vdash S_{rec_i} \xrightarrow{evt(x_1:T_1,\ldots,x_m:T_m)\ G_{rec_i}\ /\ ACT_{rec_i}} S'_{rec_i} \Rightarrow E''_{rec_i} \end{array}}$$

FIG. 5 – Règle de communication

L'idée inhérente à cette règle est que si un état D_{emet} évolue par une émission *evt* et qu'un certain nombre n d'états $(D_{rec_i})_{i \in 1..n}$ sont en attente de réception sur ce même événement, alors la synchronisation a lieu entre les $n + 1$ états concurrents et les différents environnements sont mis à jour. La

présence de gardes et d'actions induit successivement des évaluations de prédicats et des modifications d'environnement.

Nous avons donné la forme de règle la plus générale concernant le cas où les étiquettes des transitions sont complètes (*i.e.* ont la forme *événement[garde]/action*). Les autres règles sont obtenues simplement en supprimant les prémisses qui n'ont plus raison d'être présentes (*e.g.* absence d'évaluation d'une certaine garde à partir d'un environnement donné si la transition ne porte pas de garde).

Sémantique de $\triangleright_X$. Il nous reste à préciser la sémantique de $\triangleright_X$ par rapport aux divers cas de manipulation de données, *i.e.* à quoi correspondent les fonctions permettant l'évaluation. Nous avons à notre disposition plusieurs types de fonctions d'évaluation qui dépendent du langage de spécification de données : Larch, CASL, Z, B, etc. Dans le cas de Z et de B, les fonctions $\triangleright_Z$ et $\triangleright_B$ correspondent à la modification d'état induite par l'opération. L'idée pour définir la fonction d'évaluation de ces langages est de considérer des systèmes de transitions étiquetées (LTS, Labelled Transition Systems) associés aux spécifications. L'emploi des LTS est naturel car ces langages suivent une approche orientée état. En ce qui concerne les spécifications algébriques, la fonction d'évaluation correspond à une fonction de réécriture de termes. Le choix de la réécriture est justifié puisqu'il est adapté à une sémantique opérationnelle et permet ainsi de rester dans un contexte pragmatique et exécutable. De plus amples détails concernant la définition de ces fonctions sont regroupés dans [13].

Application. À présent, illustrons l'application de nos règles sémantiques sur un cas précis de spécification. Il nous faut pour cela instancier le (ou les) environnement(s) de liaison de variables, la règle dynamique montrant l'évolution du diagramme d'états, et la fonction d'évaluation $\triangleright_X$ employée (ainsi que la valeur de vérité $TRUE_X$). Pour rester dans un cas simple et compréhensible, nous utilisons un type de données représentant des entiers naturels (`Nat`) défini usuellement avec les constructeurs `0` et `succ` [12]. Cette sorte est directement spécifiée avec le langage d'entrée du prouveur de théorèmes LP [24], qui est un sous-ensemble de Larch.

Les règles de réécriture sont obtenues à partir des axiomes de la spécification algébrique par application d'une procédure d'ordonnancement (avec la commande **noeq-dsmpos** disponible dans LP). Nous avons donc l'équivalence suivante pour l'opérateur d'évaluation : $\triangleright_{Larch} \equiv \rightsquigarrow_R^*$ avec R l'ensemble des règles de réécriture. $TRUE_{Larch}$ correspond à la constante booléenne *true* de LP. Considérons le diagramme d'états de la Figure 6 qui représente trois comportements concurrents qui interagissent sur l'événement `tick`, ainsi que la règle de la Figure 5 qui décrit la synchronisation entre plusieurs diagrammes concurrents. L'instanciation de cette règle est donnée dans la Figure 7.

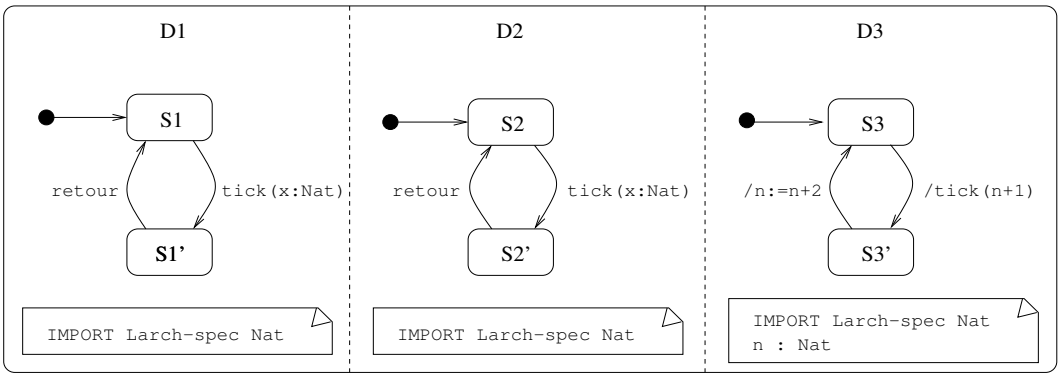

FIG. 6 – Exemple de diagrammes d'états avec synchronisation

$$\frac{\begin{array}{c} concur(D3, \{D1, D2\}) \\ S1 \xrightarrow{\texttt{tick(x:Nat)}} S1' \\ S2 \xrightarrow{\texttt{tick(x:Nat)}} S2' \\ S3 \xrightarrow{\texttt{/tick(n+1)}} S3' \\ \{(\texttt{n}, \texttt{v})\} \vdash \texttt{/tick(n + 1)} \Rightarrow \{(\texttt{n}, \texttt{v})\} \\ \{(\texttt{n}, \texttt{v})\} \vdash \texttt{n + 1} \rightsquigarrow_R^* \texttt{m} \\ \{\} \vdash \texttt{tick(x : Nat)} \Rightarrow \{(\texttt{x}, \texttt{m})\} \end{array}}{\begin{array}{c} \{\} \vdash S1 \xrightarrow{\texttt{tick(x:Nat)}} S1' \Rightarrow \{(\texttt{x}, \texttt{m})\} \\ \{\} \vdash S2 \xrightarrow{\texttt{tick(x:Nat)}} S2' \Rightarrow \{(\texttt{x}, \texttt{m})\} \\ \{(\texttt{n}, \texttt{v})\} \vdash S3 \xrightarrow{\texttt{/tick(n+1)}} S3' \Rightarrow \{(\texttt{n}, \texttt{v})\} \end{array}}$$

FIG. 7 – Instanciation de la règle de communication

L'évolution du diagramme (*i.e.* passage de S1 à S1' par exemple) dénote le passage d'un état à un autre par une transition. Certaines prémisses sont omises dûes au fait que certaines transitions ne sont pas complètes (*i.e.* n'ont pas la forme *événement [garde]/action*) mais ont des formes simplifiées. Nous nous plaçons dans le cas de la première évolution ce qui explique que certains environnements (ceux des états S1 et S2) soient vides au départ. Les variables locales doivent aussi être initialisées lors de l'animation de la spécification, ainsi nous supposons que la variable `n` contient la valeur `v` après application de la règle d'initialisation (Fig. 4). L'avant-dernière prémisse de la règle ci-dessus exprime que le terme `n+1` se réécrit en `m` en substituant la variable `n` par la valeur `v`. Les autres prémisses et conclusions expriment une évolution dynamique possible des diagrammes et les modifications des environnements qui en découlent.

3 Une étude de cas : la station essence

Cette section illustre l'utilisation de notre approche sur un exemple concret, simplifié mais réaliste. Il s'agit du fonctionnement d'une station essence équipée de plusieurs pompes. Nous donnons le cahier des charges, le résultat de son analyse et les principales idées de la modélisation. Par manque de place, nous ne détaillons pas ces différentes parties. L'étude complète avec les détails de chacune de ces parties est présentée dans [12].

3.1 Cahier des charges et analyse

Cahier des charges. La station propose uniquement le service avec paiement par carte bancaire à la pompe. Les transactions nécessaires pour l'authentification de la carte et le paiement effectif ne font pas l'objet de l'étude. La station comprend trois pompes à carburant correspondant au super, au sans plomb et au gas-oil. Elle est équipée d'un pupitre muni d'un lecteur de carte pour les paiements, de boutons pour la sélection de diverses options, d'un pavé numérique pour la saisie, d'un écran d'affichage et d'une imprimante de tickets. La station gère tous ces équipements et a pour fonctionnalité principale de fournir aux utilisateurs, à leur demande, le carburant sélectionné en quantité voulue. La station permet aux utilisateurs de saisir le numéro de carte via le pavé numérique, de sélectionner le type de carburant et de choisir éventuellement l'impression d'un ticket.

Résultat de l'analyse. A partir de cette description succincte du cahier des charges, nous avons éclairci les fonctionnalités attendues de la station considérée ici comme un système : servir du carburant, assurer le paiement par carte, gérer les pompes. Nous avons explicité les conditions nécessaires

pour assurer ses fonctionnalités et les interactions impliquées avec l'environnement. Nous avons distingué une partie statique concernant les données intervenant dans le système (booléens, entiers, réels, carburants, pompes, cuves) et une partie dynamique concernant l'évolution du système à travers trois principaux constituants : un *gestionnaire de carte, un gestionnaire de pompes* et *un gestionnaire de cuves*. D'autres sous-systèmes sont considérés comme externes : un sous-système modélisant les utilisateurs et un sous-système qui fournit, continuellement pendant un service, la quantité de carburant servie et le montant correspondant. Nous percevons le fonctionnement global de la station comme le fonctionnement en parallèle (*i.e.* des états concurrents) de ces trois gestionnaires et de leurs interactions (Fig. 8).

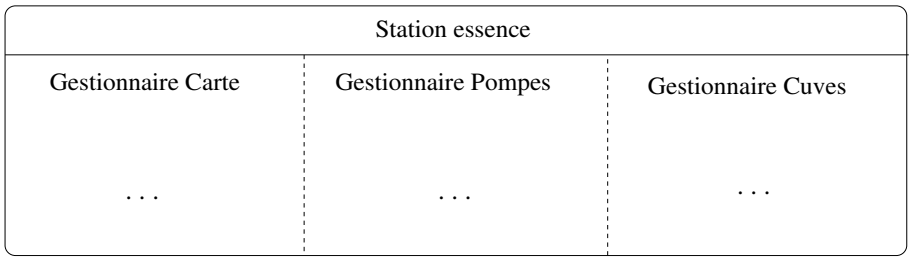

FIG. 8 – Les trois gestionnaires

3.2 Modélisation

Nous avons formalisé les données avec le langage d'entrée du prouveur de théorèmes LP et le langage Z. En LP nous avons décrit les spécifications `Nat` pour les entiers naturels et `Reel` pour les réels. En Z nous avons spécifié les données concernant les pompes et la gestion des cuves. Nous avons modélisé à l'aide de diagrammes d'états d'UML le comportement des gestionnaires de carte, de pompes et de cuves. Les données sont intégrées et manipulées dans les diagrammes comme décrit dans la section 2. Elles apparaissent en argument des événements, dans les gardes, dans les actions et dans les notes comme variables locales avec des types importés (`IMPORT`) à partir des modules spécifiés dans la partie statique.

Aspects statiques. Une partie des données de notre système est modélisée en Z. La spécification complète est analysée avec Z/EVES [4] pour sa correction syntaxique (analyse syntaxique et contrôle de type) et la sémantique statique (contrôle des domaines). Nous ne montrons ici que quelques extraits de la spécification Z. Nous avons $POMPE == \mathbb{N}$, $BoolZ ::= vrai \mid faux$ et $CarburantZ ::= super \mid sans\_plomb \mid gas\_oil$.

Cuve

$capaciteMax : \mathbb{Z}$
$capaciteMin : \mathbb{Z}$
$quantite : \mathbb{Z}$

$quantite < capaciteMax \land quantite \geq capaciteMin$

Une cuve est caractérisée par sa capacité minimale, sa capacité maximale et la quantité courante de carburant.

GCuves

$pompes : \mathbb{F}\,\mathbb{N}$
$cuve : POMPE \pfun Cuve$
$carburant : POMPE \rightarrowtail CarburantZ$
$qtPompe : POMPE \pfun \mathbb{Z}$

$\mathrm{dom}\, cuve = pompes$
$\mathrm{dom}\, carburant = pompes$
$\mathrm{dom}\, qtPompe = pompes$

InitGCuves

$GCuves'$

$pompes' = \{1, 2, 3\}$
$carburant' =$
$\quad \{(1, super), (2, sans\_plomb), (3, gas\_oil)\}$
$qtPompe' = \{(1, 0), (2, 0), (3, 0)\}$
let $cvide == (\mu\, Cuve \mid capaciteMax = 999 \land$
$\quad capaciteMin = 0 \land quantite = 0) \bullet$
$cuve' = \{(1, cvide), (2, cvide), (3, cvide)\}$

A chaque pompe nous associons une cuve, un carburant et la quantité de carburant restant dans la pompe (*qtPompe*). Nous avons écrit un schéma d'initialisation (*InitGCuves*) qui permet de considérer exactement trois pompes et de leur associer des carburants. Concernant les fondements formels, la première règle de la Figure 4 permet de prendre en compte une telle initialisation.

MajQt

$\Delta GCuves$
$pp? : \mathbb{N}$
$qtte? : \mathbb{Z}$

$pp? \in pompes$
$pompes' = pompes$
$cuve' = cuve$
$carburant' = carburant$
$qtPompe'(pp?) = qtPompe(pp?) + qtte?$

L'opération *MajQt* effectue la mise à jour (ajout et retrait) de la quantité de carburant dans une pompe donnée. La quantité peut être négative pour traiter les diminutions ou positive pour les augmentations. Les variables d'entrée *pp*? et *qtte*? peuvent être remplacées par celles utilisées dans les diagrammes en employant la notation Z `MajQt[pp/pp?; nqt/qtte?]`. Le résultat de l'opération est recupéré de la même façon dans une variable d'un diagramme d'états donné. Un autre schéma (*OkSeuil*) permet de tester la quantité de carburant par rapport à un seuil donné. Nous utilisons les définitions suivantes dans les figures des diagrammes : `NatZ` $==$ $\mathbb{N}$ et `IntZ` $==$ $\mathbb{Z}$. Toute la spécification Z, y compris les types de base, est rassemblée dans un module nommé `Z-donneeSE`. Ce module est importé au besoin dans les diagrammes.

Aspects dynamiques. Le gestionnaire de cuves (Fig. 9) utilise comme données locales la variable `etatp` et le module `Z-donneeSE` qui lui permet d'accéder aux données formelles. Le gestionnaire de cuves interagit avec le gestionnaire de pompes ; à la fin d'un service, ces deux gestionnaires se synchronisent pour la mise à jour de la quantité de carburant (`diminuerQt`). Le gestionnaire de cuves interagit aussi avec l'environnement pour l'approvisionnement en carburant (`augmenterQt`). Il réalise l'opération de mise à jour de données sur les cuves (`MajQt`) et l'opération de test des quantités (`OkSeuil`). La variable `etatp` est ainsi mise à jour et transmise au gestionnaire de pompe (`/etatPompe(etatp)`).

Nous avons montré dans cette étude la façon d'utiliser concrètement l'intégration des données formelles dans les diagrammes d'états sur la base des fondements présentés dans la section 2. Les données sont formalisées et analysées au préalable en Z et en LP. Nous obtenons ainsi un modèle intégrant les diagrammes d'états et des données formelles. Ceci ouvre la voix aux expériences d'étude formelle des propriétés du modèle.

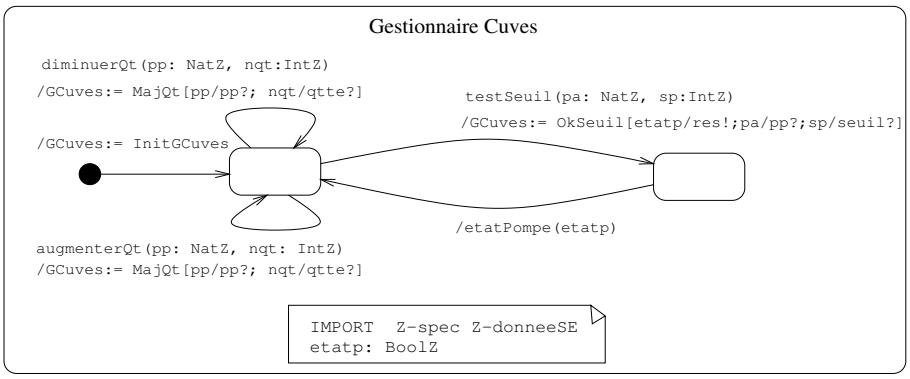

FIG. 9 – Diagramme du gestionnaire de cuves

4 Travaux connexes

Dans cette section, nous comparons brièvement notre approche avec les travaux existants qui visent à intégrer ou combiner les diagrammes d'états d'UML (ou plus généralement les Statecharts) avec des langages de spécification formelle concernant la spécification des aspects statiques. Une comparaison et un état de l'art plus détaillés pourront être trouvés dans [12].

Diagrammes d'états et OCL. L'extension la plus usuelle des diagrammes d'états en UML consiste à leur associer des contraintes OCL (Object Constraint Language). OCL n'est cependant pas à proprement parler un moyen de décrire des données abstraites ou formelles, même si des travaux s'intéressent à sa formalisation [18,20].

Diagrammes d'états et spécifications orientées états. Il existe de nombreux travaux combinant les diagrammes d'états soit avec Z [16,22,14], soit avec B [41,29,31,34,35]. Dans les deux cas, les auteurs procèdent généralement par traduction vers le formalisme statique (Z ou B) qui offre ensuite un cadre de preuve. Laleau et Polack se sont intéressées au problème d'avoir un système de traduction fonctionnant dans les deux sens [28]. Récemment, l'approche de [22] a été combinée avec des diagrammes d'états pour aborder le problème de la cohérence entre statique et dynamique [33]. Il s'agit d'une approche complémentaire à la notre au sens où nous nous préoccupons aussi de ces problèmes de cohérence. Notre approche propose cependant une démarche plus dynamique (sémantique opérationnelle) et cherche à généraliser au niveau de la statique (*i.e.* Z et B mais aussi les spécifications algébriques par exemple).

Dans le cas des approches utilisant Z, comme dans celles utilisant B, la différence principale par rapport à nos travaux est que nous cherchons à utiliser directement les sémantiques des parties dynamiques et statiques sans traduction de l'une d'entre elle. Nous pensons que cette approche permet plus facilement une réutilisation des preuves qui pourraient être associées aux spécifications (selon une approche « composants de confiance » [5]). En effet, cela permet au spécifieur de réutiliser une preuve concernant l'un des aspects (et donc faite dans le cadre qu'il connaît et avec les outils dédiés à cet aspect) dans le cadre d'une preuve concernant le système global (compositionnalité). Dans le cas d'une intégration complète dans l'un des formalismes (*i.e.* concrètement en Z ou en B), le spécifieur devrait entièrement faire sa preuve dans ce cadre, sans pouvoir réutiliser celle qui aurait déjà été associée aux composants réutilisés. La compositionnalité existe dans de nombreux langages non mixtes. Nous espérons qu'à terme notre travail permette une forme de compositionnalité multi-langages de spécification.

Diagrammes d'états et spécifications algébriques. Le langage Casl-Chart [38] combine le langage de spécifications algébriques CASL avec des Statecharts pouvant utiliser des types de données CASL dans les événements, les gardes et les actions. La sémantique du langage combiné est définie à partir des sémantiques originales des deux langages de base. Nos travaux sont donc très proches de Casl-Chart. Notre approche est plus flexible au niveau de la dynamique puisque nous proposons un cadre réutilisable de sémantique opérationnelle à base d'environnements pour l'intégration de données formelles dans d'autres formalismes dynamiques.

L'ensemble des travaux existants, à notre connaissance, impose l'utilisation d'un langage de définition de type de données unique et est donc moins flexible que notre approche. Le niveau de réutilisabilité des composants de spécification statique est donc plus grand dans notre approche : un choix plus vaste de langages statiques induit un plus grand nombre de spécifications statiques réutilisables.

5 Conclusion et perspectives

Le but de notre travail est de proposer un langage convivial, et plus généralement une méthode d'intégration, adaptés à la spécification de systèmes complexes. Nous avons choisi de combiner les diagrammes d'états d'UML avec des techniques de spécification formelle de types de données abstraits (comme par exemple les spécifications algébriques, Z ou B). Cette utilisation conjointe d'une notation semi-formelle pour la modélisation visuelle des aspects dynamiques avec des langages formels pour spécifier les aspects statiques permet de profiter des avantages des deux approches. Contrairement aux approches existantes, notre approche se place à un haut niveau d'abstraction. Elle permet en effet l'utilisation des différentes sémantiques des diagrammes d'états d'UML et plus généralement de celles des Statecharts ou d'autres langages basés sur des états et des transitions comme les travaux récents autour des systèmes de transitions symboliques [17,19]. Notre approche est aussi plus souple et plus flexible puisqu'elle permet l'utilisation de différents langages de description des données, améliorant par là-même le niveau de réutilisabilité des spécifications. Des resultats complémentaires sont présentés dans [13] où nous généralisons notre approche à l'extension de n'importe quel type de diagramme d'états dans le cadre d'une sémantique de communication asynchrone. Nous y donnons aussi une solution pour la prise en compte des activités et des états hiérarchiques des diagrammes d'états d'UML, non traités ici.

Les spécifications obtenues en suivant la démarche présentée peuvent être vérifiées partiellement soit en ce qui concerne la partie dynamique par vérification de modèle ou animation des spécifications (utilisation directe de STATEMATE en oubliant les données, traduction de la sémantique dynamique dans le format d'entrée d'outils tels que CWB-NC [3], soit en ce qui concerne la partie données par utilisation des prouveurs de théorèmes ou des environnements dédiés aux langages utilisés dans l'intégration (tels que CATS [6] pour CASL, LP [24] pour Larch, AtelierB [1] ou B-Toolkit [2] pour B, Z/EVES [4] pour Z). Cette vérification partielle peut avoir un sens. Au niveau des données elle permet par exemple de vérifier la confluence ou la terminaison du système de réécriture utilisé dans l'évaluation des termes algébriques. Elle peut aussi être utilisée pour vérifier le résultat de l'évaluation d'une opération Z. Au niveau de la dynamique, il est possible de faire des hypothèses optimistes ou pessimistes quant à l'évaluation des gardes des transitions pour prouver l'absence de blocage [40]. Cependant, dans le cadre de systèmes mixtes il reste indispensable de pouvoir vérifier la spécification dans sa globalité.

Une première perspective concerne l'approfondissement des aspects de vérification dans cet objectif. Une piste en cours est la traduction de notre langage et de sa sémantique dans le format d'entrée

d'outils puissants basés sur les logiques d'ordre supérieur tel que PVS [21] ou Isabelle [36]. Des détails pourront être trouvés dans [12].

Nous disposons déjà d'un outil permettant d'animer des spécifications écrites en CCS avec passage de valeur. Cet outil, ISA [11], permet de gérer des données potentiellement écrites dans n'importe quel langage de spécification et pour lequel il existe une fonction d'évaluation. Nous pensons qu'il est possible d'étendre cet outil de façon modulaire et générique pour prendre en compte différents langages de spécification concernant la partie dynamique. Dans un second temps, sa version actuelle (CCS et données) mais aussi les diagrammes d'états d'UML (en suivant les principes décrits ici) pourraient donner lieu à des instanciations particulières de cet outil.

Une dernière perspective de ces travaux concerne la généralisation de l'approche dans le but de combiner différents formalismes basés sur l'intégration de données formelles dans des systèmes de transitions (LOTOS [27], SDL [23], Statecharts, Korrigan [19]). En effet, notre approche est relativement générale et flexible en ce qui concerne la dynamique et la statique pour prendre en compte une famille relativement grande de langages mixtes.

Références

1. `http://www.atelierb.societe.com/`.
2. `http://www.b-core.com/btoolkit.html`.
3. `http://www.cs.sunysb.edu/~cwb/`.
4. `http://www.ora.on.ca/z-eves/`.
5. `http://www.trusted-components.org/`.
6. `http://www.tzi.de/cofi/Tools/CATS.html`.
7. J. R. Abrial. *The B-Book*. Cambridge University Press, 1996.
8. M. Allemand, C. Attiogbé, P. Poizat, J.-C. Royer, and G. Salaün. SHE'S Project: a Report of Joint Works on the Integration of Formal Specification Techniques. In *Proc. of the Workshop on Integration of Specification Techniques with Applications in Engineering (INT'02)*, pages 29–36, France, 2002.
9. E. Astesiano. UML as Heterogeneous Multiview Notation. Strategies for a Formal Foundation. In L. Andrade, A. Moreira, A. Deshpande, and S. Kent, editors, *Proc. of the Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'98) – Workshop on Formalizing UML. Why ? How ?*, Canada, 1998.
10. E. Astesiano, H.-J. Kreowski, and B. Krieg-Brückner, editors. *Algebraic Foundations of System Specification*. Springer-Verlag, 1999.
11. C. Attiogbé, A. Francheteau, J. Limousin, and G. Salaün. ISA, a Tool for Integrated Specifications Animation. Disponible à `http://www.sciences.univ-nantes.fr/info/perso/permanents/salaun/ISA/isa.html`, 2002.
12. C. Attiogbé, P. Poizat, and G. Salaün. Intégration de données formelles dans les diagrammes d'états d'UML. Technical Report 02.03, Institut de Recherche en Informatique de Nantes, 2002. Disponible à `http://www.sciences.univ-nantes.fr/info/perso/permanents/salaun/papers/rr023.ps`.
13. C. Attiogbé, P. Poizat, and G. Salaün. Integration of Formal Datatypes within State Diagrams. Technical Report 83-2002, LaMI, University of Evry, October 2002. Disponible à `http://www.sciences.univ-nantes.fr/info /perso/permanents/salaun/papers/APS8302.ps`. Soumis.
14. J.-M. Bruel and R. B. France. Transforming UML models to Formal Specifications. In P.-A. Muller and J. Bézivin, editors, *Proc. of the International Conference on the Unified Modelling Language: Beyond the Notation (UML'98)*, number 1618 in LNCS, France, 1998. Springer-Verlag.
15. G. Bruns. A Language for Value-Passing CCS. LFCS Report Series ECS-LFCS-91-175, University of Edinburgh, 1995.
16. R. Büssow and M. Weber. A Steam-Boiler Control Specification with Statecharts and Z. In J.-R. Abrial, E. Börger, and H. Langmaack, editors, *Formal Methods for Industrial Applications: Specifying and Programming the Steam Boiler*, volume 1165 of *Lecture Notes in Computer Science*, pages 109–128. Springer-Verlag, 1996.

17. M. Calder, S. Maharaj, and C. Shankland. A Modal Logic for Full LOTOS Based on Symbolic Transition Systems. *The Computer Journal*, 45(1) :55–61, 2002.
18. M. V. Cengarle and A. Knap. A Formal Semantics for OCL 1.4. In M. Gogolla and C. Kobryn, editors, *Proc. of the 4th International Conference on the Unified Modeling Language (UML'01)*, volume 2185 of *Lecture Notes in Computer Science*, pages 118–133, Canada, 2001. Springer-Verlag.
19. C. Choppy, P. Poizat, and J.-C. Royer. A Global Semantics for Views. In T. Rus, editor, *Proc. of the 8th International Conference on Algebraic Methodology And Software Technology (AMAST'00)*, volume 1816 of *Lecture Notes in Computer Science*, pages 165–180, USA, 2000. Springer-Verlag.
20. T. Clark and J. Warmer, editors. *Object Modeling with the OCL, The Rationale behind the Object Constraint Language*, volume 2263 of *Lecture Notes in Computer Science*. Springer-Verlag, 2002.
21. J. Crow, S. Owre, J. Rushby, N. Shankar, and M. Srivas. A Tutorial Introduction to PVS. In *Proc. of the Workshop on Industrial-Strength Formal Specification Techniques (WIFT'95)*, USA, 1995. Computer Science Laboratory, SRI International.
22. S. Dupuy, Y. Ledru, and M. Chabre-Peccoud. An Overview of RoZ: A Tool for Integrating UML and Z Specifications. In B. Wangler and L. Bergman, editors, *Proc. of the Advanced Information Systems Engineering Conference (CAiSE'00)*, volume 1789 of *Lecture Notes in Computer Science*, pages 417–430, Sweden, 2000. Springer-Verlag.
23. J. Ellsberger, D. Hogrefe, and A. Sarma. *SDL: Formal Object-oriented Language for Communicating Systems*. Prentice-Hall, 1997.
24. S. J. Garland and J. V. Guttag. A Guide to LP, the Larch Prover. Technical Report, Palo Alto, California, 1991.
25. The Precise UML Group. `http://www.cs.york.ac.uk/puml/`.
26. D. Harel and A. Naamad. The STATEMATE Semantics of Statecharts. *ACM Transactions on Software Engineering and Methodology*, 5(4) :293–333, 1996.
27. ISO/IEC. LOTOS: A Formal Description Technique based on the Temporal Ordering of Observational Behaviour. ISO/IEC 8807, International Organization for Standardization, 1989.
28. R. Laleau and F. Polack. Coming and Going from UML to B: A Proposal to Support Traceability in Rigorous IS Development. In D. Bert, J. P. Bowen, M. C. Henson, and K. Robinson, editors, *Proc. of the 2nd International Z and B Conference (ZB'02)*, volume 2272 of *Lecture Notes in Computer Science*, pages 517–534, France, 2002. Springer-Verlag.
29. K. Lano, K. Androutsopoulos, and P. Kan. Structuring Reactive Systems in B AMN. In *Proc. of the 3rd IEEE International Conference on Formal Engineering Methods (ICFEM'00)*, pages 25–34, UK, 2000. IEEE Computer Society Press.
30. D. Latella, I. Majzik, and M. Massink. Towards a Formal Operational Semantics of UML Statechart Diagrams. In P. Ciancarini and R. Gorrieri, editors, *Proc. of the IFIP TC6/WG6.1 Third International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'99)*, pages 331–347, Italy, 1999. Kluwer Academic Publishers.
31. H. Ledang and J. Souquières. Contributions for Modelling UML State-Charts in B. In M. Butler, L. Petre, and K. Sere, editors, *Proc. of the 3rd International Conference on Integrated Formal Methods (IFM'02)*, volume 2335 of *Lecture Notes in Computer Science*, pages 109–127, Finland, 2002. Springer-Verlag.
32. J. Lilius and I. P. Paltor. The Semantics of UML State Machines. Technical Report 273, Turku Centre for Computer Science, 1999. Disponible à `http://www.tucs.fi/Publications/techreports/TR273.php`.
33. O. Maury, C. Oriat, and Y. Ledru. Invariants de liaison pour la cohérence de vues statiques et dynamiques en UML. In *Actes de la conférence sur les Approches Formelles dans l'Assistance au Développement de Logiciels (AFADL'01)*, pages 23–37, Nancy, France, 2001.
34. E. Meyer. *Développements formels par objets : utilisation conjointe de B et d'UML*. PhD thesis, LORIA – Université Nancy 2, 2001.
35. H. P. Nguyen. *Dérivation de spécifications formelles B à partir de spécifications semi-formelles*. PhD thesis, CEDRIC, CNAM, 1998.
36. T. Nipkow and L. C. Paulson. Isabelle-91. In D. Kapur, editor, *Proc. of the 11th International Conference on Automated Deduction (CADE'92)*, volume 607 of *Lecture Notes in Computer Science*, pages 673–676, USA, 1992. Springer-Verlag.
37. Object Modeling Group. *Unified Modelling Language Specification, version 1.4*, September 2001.
38. G. Reggio and L. Repetto. Casl-Chart: A Combination of Statecharts and of the Algebraic Specification Language Casl. In T. Rus, editor, *Proc. of the 8th International Conference on Algebraic Methodology and Software Technology (AMAST'00)*, volume 1816 of *Lecture Notes in Computer Science*, pages 243–257, USA, 2000. Springer-Verlag.

39. G. Reggio and R. Wieringa. Thirty one Problems in the Semantics of UML 1.3 Dynamics. In *Conference on Object-Oriented Programming, Systems, Languages and Applications (OOPSLA'99) – Workshop "Rigorous Modelling and Analysis of the UML: Challenges and Limitations"*, 1999.

40. J.-C. Royer. Formal Specification and Temporal Proof Techniques for Mixed Systems. In *Proc. of the 15th IPDPS 2001 Symposium, FMPPTA*, USA, 2001. IEEE Computer Society Press.

41. E. Sekerinski and R. Zurob. Translating Statecharts to B. In M. Butler, L. Petre, and K. Sere, editors, *Proc. of the 3rd International Conference on Integrated Formal Methods (IFM'02)*, volume 2335 of *Lecture Notes in Computer Science*, pages 128–144, Finland, 2002. Springer-Verlag.

42. J. M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall International Series in Computer Science, 2nd edition, 1992.

43. M. van der Beeck. Formalization of UML-Statecharts. In M. Gogolla and C. Kobryn, editors, *Proc. of the 4th International Conference on the Unified Modeling Language (UML'01)*, volume 2185 of *Lecture Notes in Computer Science*, pages 406–421, Canada, 2001. Springer-Verlag.

44. D. Varró. A Formal Semantics of UML Statecharts by Model Transition Systems. In A. Corradini and H.J. Kreowski, editors, *Proc. of the 1st International Conference on Graph Transformation (ICGT'2002)*, volume 2505 of *Lecture Notes in Computer Science*, Spain, 2002. Springer-Verlag.