Un assistant méthodologique UML

Modélisation et vérification formelle de protocoles guidées par des patrons

Ludovic Apvrille* — **Pierre de Saqui-Sannes****

* *Institut Télécom, Télécom ParisTech, LTCI CNRS*
2229 route des Crêtes – B.P. 193, F-06904 Sophia Antipolis Cedex
ludovic.apvrille@telecom-paristech.fr

** *CNRS ; LAAS ;*
7 avenue du colonel Roche, F-31077 Toulouse, France
Université de Toulouse ; UPS, INSA, INP, ISAE ; LAAS ;
F-31077 Toulouse, France
pdss@isae.fr

RÉSUMÉ. *La modélisation de services et protocoles est la clé de voute de la validation d'une architecture de communication. L'article propose de mener cette activité complexe dans le TURTLE Toolkit (TTool), environnement UML temps réel doté de capacités de vérification formelle. Les principes d'un assistant méthodologique pour l'analyse d'architecture de communication sont définis dans un cadre général, puis transposés à l'approche TTool sous la forme de patrons basés sur des cas d'utilisation et des scénarios paramétrables. L'utilisation de ces patrons est illustrée sur un protocole de communication point à multipoint par satellite.*

ABSTRACT. *Modeling services and protocols is a key point when the validation of a communication architecture is at stake. The paper proposes to address such a validation with the TURTLE Toolkit (TTool), a UML-based environment with real-time modeling and formal verification capabilities. A pattern-based methodological assistant relying on UML analysis diagrams is defined and applied to TTool. The patterns support parametrized use cases and scenarios. A space-based multicast communication system serves as example.*

MOTS-CLÉS : *Patrons, protocoles, UML, cas d'utilisation, scénarios, vérification formelle.*

KEYWORDS: *Patterns, Protocols, UML, Use cases, Scenarios, Formal verification.*

1. Introduction

La complexité des systèmes répartis justifie le recours à des langages de modélisation supportés par des outils de vérification formelle. Les langages de type SDL et les techniques de modélisation de type réseaux de Petri couvrent essentiellement la phase de conception d'un protocole par la définition d'une architecture et de comportements (en particulier des machines de protocole). Le besoin de couvrir les phases plus en amont et en particulier la phase d'analyse a amené les promoteurs de SDL à lui adjoindre les MSCs qui sont des ancêtres des diagrammes de séquence d'UML. L'arrivée d'UML a permis d'associer service et protocole dans un même modèle, fût-ce au prix de plusieurs diagrammes il est vrai. La définition de profils UML temps réel adossés à des méthodes formelles (Gherbi *et al.*, 2006) apporte une forme de réponse à ce besoin en dotant la notation UML de l'OMG (OMG, 2009) d'une sémantique formelle et d'outils permettant de valider une architecture de communication. Un de ces profils UML temps réel est TURTLE (Timed UML and RT-LOTOS Environment) introduit en (Apvrille *et al.*, 2004) et étendu en (Apvrille *et al.*, 2005). TURTLE profite de sa base « algèbre de processus » pour apporter une réponse au problème de la composition de comportements. L'outil open-source TTool (Apvrille, 2010) supporte - entre autres profils UML 2 - le profil UML temps réel TURTLE. TTool se caractérise par une interface avenante à des outils de vérification applicables aussi bien à une analyse à base de scénarios qu'à une conception objet, par un générateur automatique d'une architecture et de comportements à partir de cas d'utilisation et scénarios, et par des générateurs de code Java et SystemC (The SystemC Community, 2010) pour le prototypage rapide.

Si le profil TURTLE est maintenant stabilisé en termes de syntaxe et de sémantique, *a contrario*, le volet méthodologique mérite d'être approfondi. En particulier, le lien entre une méthodologie exposée *in extenso* (Fontan, 2008) ou sur des exemples d'une part, et l'outil TTool qui supporte le profil d'autre part, reste à établir. L'article propose d'introduire dans TTool un assistant méthodologique et de démarrer l'assistance méthodologique en phase d'analyse. Cela passe par une meilleure compréhension des trois diagrammes UML 2 que sont les diagrammes de cas d'utilisation (UCD), les diagrammes globaux d'interactions (IOD) et les diagrammes de séquences (SD). En effet, si les UCD et SD sont relativement bien connus et utilisés pour la modélisation de protocole, il en va différemment des IOD, en dépit de l'intérêt de ces derniers pour structurer (sous forme d'organigrammes) les scénarios qui documentent les cas d'utilisation. Une première contribution de l'article réside dans la proposition de patrons (*patterns* en anglais). Ceux-ci sont paramétrés et associent UCD, IOD et SD sur une base formelle. La deuxième contribution se situe dans l'instanciation de ces patrons à la conception de protocoles. Enfin, la troisième contribution se place sur le terrain de l'outillage : un assistant méthodologique est (partiellement) implanté dans TTool ; l'article décrit sa mise en œuvre sur un protocole issu du projet européen *Maestro* (Thalès-Alenia-Space, 2006).

L'article est structuré de la manière suivante. La section 2 présente le profil UML TURTLE en ciblant les diagrammes d'analyse utilisés par la suite. La section 3 pro-

pose des patrons paramétrés accouplant UCD et IOD. Le concept d'UCD et IOD paramétrable est introduit à cette occasion. La section 4 spécialise la proposition vers les protocoles. La section 5 présente une étude de cas réalisée avec la première version de l'outil TTool étendu par un assistant méthodologique. La section 6 positionne les contributions de l'article par rapport aux travaux du domaine. La section 7 conclut l'article et ouvre des perspectives.

2. Le profil UML TURTLE

Le concept de « prof l » permet de personnaliser le langage UML normalisé à l'OMG pour répondre aux besoins de modélisation d'une famille de systèmes. Dédiés aux systèmes temps réel, les prof ls UML temps réel s'appliquent à la modélisation de protocoles et à la validation d'architectures de communication.

2.1. *Vue d'ensemble du profil*

2.1.1. *Historique*

Le prof l UML temps réel TURTLE (*Timed UML and RT-LOTOS Environment*) (Apvrille *et al.*, 2004) entre dans la famille des langages de modélisation qui trouvent leur sémantique formelle dans une traduction vers un langage qui disposait déjà d'une sémantique formellement déf nie. Dans le cas de TURTLE, nous nous sommes appuyés sur l'algèbre de processus temporisée RT-LOTOS (Courtiat *et al.*, 2000). L'outil TTool (*TURTLE Toolkit*) supporte le prof l (Apvrille, 2010). Son association à l'outil de vérif cation formelle *RTL* (supportant RT-LOTOS (Courtiat *et al.*, 2000)) fait de TURTLE un langage de modélisation tout particulièrement adapté à la vérif cation d'exigences temporelles. Un générateur de code Java/JMI ouvre des perspectives vers le prototypage d'applications réparties, comme la modélisation du protocole HTTP (Apvrille *et al.*, 2006a) l'a montré.

2.1.2. *LOTOS et RT-LOTOS*

LOTOS (ISO-LOTOS, 1987) est une technique de description formelle normalisée par l'ISO et destinée à la modélisation des systèmes distribués communicants. Une spécif cation LOTOS est composée d'une déf nition de type et d'un processus principal, lui-même structuré en sous-processus. Un processus LOTOS est une boîte noire qui communique avec son environnement *via* des portes de synchronisation. Un échange de données multidirectionnel peut avoir lieu lors d'une synchronisation. Les relations entre processus - y compris les synchronisations - sont décrites à l'aide d'opérateurs de composition (le choix, la séquence, etc., voir le tableau 1). RT-LOTOS étend LOTOS avec deux opérateurs temporels : le délai non déterministe et l'offre de synchronisation limitée dans le temps (Courtiat *et al.*, 2000).

Opérateur	Description	Exemple
$[]$	Choix	$P[a,b,c,d] = P1[a,b][]P2[c,d]$
$\|\|\|$	Parallèle	$P[a,b,c,d] = P1[a,b]\|\|\|P2[c,d]$
$\|[b,c,d]\|$	Synchronisation sur plusieurs portes (b,c,d)	$P[a,b,c,d,e] = P1[a,b,c,d]\|[b,c,d]\|P2[b,c,d,e]$
$>>$	Sequence	$P[a,b,c,d] = P1[a,b] >> P2[c,d]$
$[>]$	Preemption	$P[a,b,c,d] = P1[a,b][> P2[c,d]$
;	Processus préf xé par une action a	$P[a] = a; P1[a]$
$a\{T\}$	Offre limitée dans le temps	$P[a] = a\{T\}; P1[a]$
$delay(t_1,t_2)$	Délai non déterministe	$P[a] = delay(t_1,t_2)a; P1[a]$

Tableau 1. *Opérateurs RT-LOTOS*

2.1.3. *Méthodologie*

En amont du cycle de développement d'un système, TTool supporte les diagrammes d'exigences de SysML, étendus par des chronogrammes aptes à représenter des exigences temporelles (Fontan, 2008). TTool permet ensuite d'entamer la phase d'analyse par un diagramme de cas d'utilisation qui délimite le périmètre du système à modéliser, isole les acteurs et identif e les grandes fonctionnalités que le système doit offrir. Les cas d'utilisation représentatifs de ces fonctionnalités sont documentés par des scénarios représentés à l'aide de SD. Là interviennent les IOD par leur capacité à structurer les SD sous forme d'organigramme. Les SD TURTLE permettent de représenter deux types de communication entre objets : non seulement une communication de type rendez-vous à la LOTOS mais aussi une communication par f le FIFO non bloquante.

En termes de méthode, il est courant de déf nir une première famille de scénarios qui met en évidence les interactions acteurs-système et d'entamer ensuite une itération méthodologique qui permet d'éclater les premiers scénarios et de faire apparaître la structure interne du système. Cette analyse fonctionnelle assistée au besoin d'une recherche des objets - par exemple par la méthode des *mots dans le texte* - débouche ensuite sur une conception objet. Cette conception objet peut aussi être générée de façon automatique par un algorithme de synthèse implanté dans TTool (Apvrille *et al.*, 2005).

La conception consiste à déf nir l'architecture du système et les comportements des objets qui le constituent. La déf nition de l'architecture statique du système repose sur un diagramme de classes/objets TURTLE qui étend ceux d'UML 2 en formalisant le type de relation entre objets (parallélisme, synchronisation, préemption, etc.) et en autorisant ces mêmes objets à communiquer par rendez-vous. Ce mécanisme abstrait

permet de réaliser des modèles de haut niveau en préjugeant le moins possible des mécanismes d'implantation (f les, appel de procédure) ; cela n'empêche pas d'ajouter au modèle, des classes qui implantent des modes de communication plus proches d'une implantation (f le FIFO non bloquante à la SDL, f le FIFO bloquante, appel distant de procédure). Ainsi, pour représenter en conception le mode de communication asynchrone des SD, une classe intémediaire est utilisée pour modéliser la f le d'attente.

Les comportements des objets sont exprimés par des diagrammes d'activités ; ceux-ci expriment les actions de synchronisation par rendez-vous et offrent des opérateurs temporels (délais f xes, intervalles temporels et limite temporelle des offres de rendez-vous). La déf nition de diagrammes d'activités rend le modèle TURTLE exécutable et prêt pour une simulation et/ou une analyse d'accessibilité. Le déploiement du système sur une cible réelle nécessite ensuite l'utilisation de diagrammes de déploiement TURTLE.

Chacun des diagrammes TURTLE - à l'exception de l'UCD - est doté d'une sémantique formelle. Ainsi, une analyse TURTLE (*i.e.* un IOD principal et les diagrammes référencés depuis cet IOD), une conception TURTLE (*i.e.* un diagramme de classes/objets et un ensemble de diagrammes d'activités) ou, enf n, un déploiement TURTLE (*i.e.*, un diagramme de déploiement et un ensemble de composants constitués de classes TURTLE), possèdent une sémantique formelle en (RT-)LOTOS et UPPAAL. Ceci explique pourquoi tous ces diagrammes peuvent se voir appliquer des techniques de vérif cation formelle.

2.1.4. *Outillage support (TTool)*

L'outil TTool, *open-source*, supporte plusieurs prof ls UML 2 ou SysML. Citons par exemple TURTLE (Apvrille *et al.*, 2004), DIPLODOCUS (Apvrille *et al.*, 2006b) et CTTool (Ahumada *et al.*, 2007) (cf. f gure 1). L'apport principal de TTool réside dans la possibilité d'appliquer de la vérif cation formelle sur toute modélisation réalisée dans un des prof ls précités. Ainsi, TTool intègre :

– des facilités d'édition pour les prof ls supportés ;

– des vérif cateurs syntaxiques ;

– des générateurs de code :

- **langages pour la vérification formelle** : RT-LOTOS, LOTOS, UPPAAL ;

- **langages pour la simulation** : C++/SystemC (pour le prof l DIPLODOCUS uniquement) ;

- **langage pour le prototypage** : Java.

– des liens avec des outils de vérif cation externes (RTL (Courtiat *et al.*, 2000), CADP (INRIA-VASY, 2008), UPPAAL ; (Behrmann *et al.*, 2004))

– enf n, des analyseurs internes de traces de vérif cation permettant de présenter les résultats de vérif cation formelle de façon conviviale.

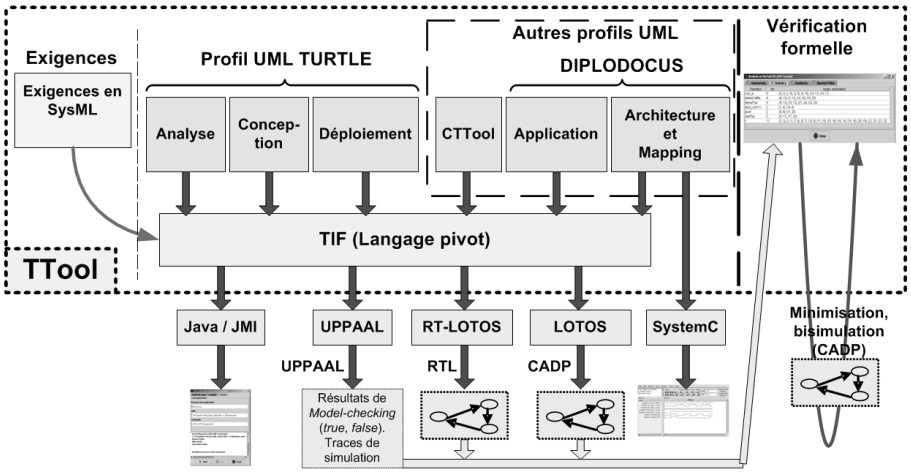

Figure 1. *TTool*

Le site (Apvrille, 2010) précise les restrictions d'utilisation des différents générateurs de code, entre autres. Notons que l'objectif de l'outil est de cacher autant que faire se peut les techniques sous-jacentes de vérif cation formelle ou les simulations, aux personnes en charge des modélisations ; ainsi, aucune connaissance spécif que des algèbres de processus ou des automates temporisés n'est nécessaire pour utiliser TTool. Plus généralement, l'interface rend l'outil accessible à des personnes qui ont besoin de confronter un modèle de système à des exigences temporelles sans pour autant être ferrues de langages formels, de langages d'expression de propriétés (par exemple, Computational Temporal Logic) et d'outils de vérif cation.

La suite de cet article se focalise sur les diagrammes d'analyse TURTLE, à savoir les UCD, IOD et SD. En effet, la pratique courante dans l'ingénierie des protocoles est de réaliser des modélisations sous la forme de cas d'utilisation et de scénarios.

2.2. *Diagrammes de cas d'utilisation*

Un UCD est construit comme suit (f gure 2) :

– une frontière représentée sous la forme d'un rectangle délimite clairement le système et sépare les acteurs des fonctions ;

– les fonctions précisent les cas d'utilisation du système. Elle peuvent être mises en relation d'inclusion $<< include >>$, d'extension $<< extend >>$ (l'extension est en général déf nie par rapport à un point d'extension), ou encore de spécialisation (connecteur UML utilisé pour modéliser l'héritage). L'inclusion permet de modéliser des sous-fonctions d'une fonction. L'extension permet d'ajouter à des fonctions, des fonctions optionnelles et supplémentaires.

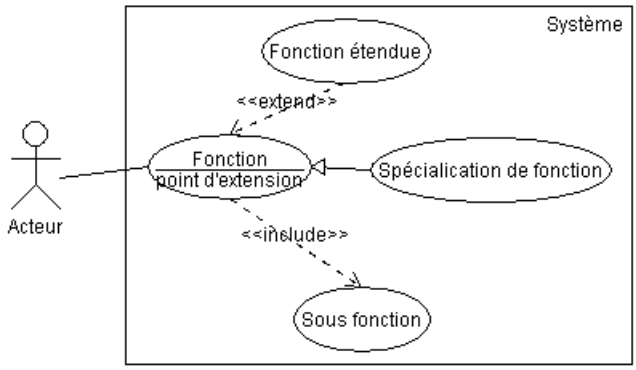

Figure 2. *Exemple d'un diagramme de cas d'utilisation*

– les acteurs peuvent être mis en relation avec des fonctions avec lesquelles ils inter-agissent.

2.3. *IOD*

La norme UML de l'OMG propose d'utiliser une instance de diagramme d'activités appelée "Interaction Overview Diagram" (IOD) pour structurer les scénarios. Plus précisément, un IOD s'apparente à un organigramme dont les actions seraient remplacées par des références à des SD ou à d'autres IOD. Notons que depuis un diagramme de séquences, il n'est pas possible de référencer un IOD. Ces organigrammes sont de plus constitués de nœuds structurants (choix, séquence, parallélisme). TURTLE y ajoute un nœud de préemption af n d'autoriser la préemption d'une branche d'un IOD par une branche d'un autre IOD. L'exemple d'IOD donné à la f gure 9 met en évidence des références vers d'autres IOD (par exemple, *Connection setup* est une référence vers un IOD). Cette même f gure montre des choix entre IOD (le losange) et un opérateur de préemption (la barre horizontale avec le symbole $[>$ à son extrémité droite) qui permet de spécif er que les IOD *Connection release* ou *Connection broken* peuvent à tout moment interrompre la branche partant de la gauche de l'opérateur de préemption.

Finalement, les IOD, bien que peu utilisés actuellement en ingénierie des protocoles, ont la bonne propriété de pouvoir structurer des scénarios et de réduire par là même la complexité de ces scénarios. En effet, ces derniers ont comme avantage d'être un guide visuel d'échanges d'informations entre instances. Les IOD ajoutent la facilité visuelle d'une structuration des scénarios et évitent de se placer directement au niveau de détail des échanges d'information entre instances. De plus, ce confort visuel encourage à donner davantage d'informations au niveau des diagrammes d'analyse, et donc à se rapprocher d'une exhaustivité des comportements à l'aide de scénarios, approche

particulièrement prisée en ingénierie des protocoles. D'autres domaines du génie logiciel, par exemple, la conception de systèmes embarqués, utilisent moins couramment les SD : ainsi, ces domaines sont sans doute moins enclins à rechercher une exhaustivité des comportements en phase d'analyse.

Malheureusement, en UML 2, certains opérateurs des IOD et des SD ont la même sémantique, ce qui implique que la personne qui modélise a la possibilité de faire figurer certaines informations soit au niveau de l'IOD, soit au niveau d'un scénario. Ceci va à l'encontre d'un modèle dual structure/comportement. En TURTLE, nous avons limité certaines de ces possibilités duales en n'offrant, pour une sémantique donnée, qu'une possibilité d'exprimer cette sémantique, soit au niveau de l'IOD, soit au niveau du SD. Par exemple, les références à des scénarios ont été supprimées des SD, mais sont conservées au niveau des IOD. Le même sort est réservé aux opérateurs de parallélisme et de séquence. Nous avons par contre gardé l'opérateur d'alternative aux deux niveaux de modélisation (IOD et SD). En effet, il est apparu, lors d'études de cas, que certains choix relevaient d'alternatives de haut niveau entre scénarios (structure générale du comportement) alors que d'autres choix étaient utilisés pour des décisions locales à des instances.

3. Une méthodologie basée sur les patrons

3.1. *Definition des patrons*

L'approche que nous proposons dans cet article se base sur la notion de patrons UML formellement définis. Chaque patron P est constitué de deux diagrammes qui servent de canevas à l'utilisateur du patron. Un patron est constitué d'un UCD et d'un IOD (appelé IODh pour *IOD de haut niveau*) : P = <UCD, IODh>, et possède les règles suivantes :

– à toute fonction de haut niveau de l'UCD - c'est-à-dire à toute fonction non incluse dans une autre - et à toute fonction optionnelle, l'IODh associe au moins une référence vers un IOD. Cette référence porte le nom de la fonction à laquelle elle correspond ;

– un utilisateur de l'UCD a le droit d'ajouter des sous-fonctions à des fonctions déjà présentes. A partir de ces nouvelles fonctions, il peut spécialiser ces fonctions (héritage) ou ajouter des fonctions optionnelles. Comme l'IODh n'est pas modifiable par l'utilisateur, il n'est pas possible de donner une correspondance à ces nouvelles fonctions dans l'IODh, que ce soit sous forme de référence à un IOD, ou sous forme de référence à un diagramme de séquences. Ainsi, ces ajouts de fonctions sont représentés uniquement au travers de références vers des sous-IOD (ou SD) de IODh ;

– les acteurs de l'UCD sont paramétrés, c'est-à-dire qu'un nombre d'instances de ces acteurs est précisé au niveau de l'UCD. L'utilisateur d'un patron peut utiliser ces données de paramétrage au niveau des scénarios de son analyse afin de modéliser des émissions/réceptions multiples de messages, par exemple, lors de l'analyse d'un sys-

tème client/serveur à clients multiples. La sémantique de ce paramétrage est expliquée plus en détail dans la suite de l'article ;

– dans l'IODh, les références vers des IOD correspondant à des fonctions de haut niveau doivent être obligatoirement complétées ; en d'autres termes, un diagramme IOD doit être fourni. En ce qui concerne les autres références à des IOD de l'IODh, cette procédure est facultative, c'est-à-dire qu'ils peuvent être laissés vides ;

– l'IODh ne peut être modif é par l'utilisateur, sauf au niveau des nœuds de type choix dont les gardes peuvent, si elles sont laissées non renseignées par le patron - c'est-à-dire que le patron possède des choix non déterministes - être renseignées selon ce qu'autorise la syntaxe TURTLE.

3.2. *Méthodologie*

Les patrons ont vocation à être utilisés comme suit (cf. f gure 3) :

1) L'utilisateur choisit un patron P en fonction du type de protocole qu'il doit implémenter (mode non connecté, mode connecté, diffusion de données dans un groupe, etc.). Ces patrons peuvent soit faire partie de TTool, soit avoir été constitués progressivement par une entreprise ou une équipe qui travaille habituellement sur un type de protocole particulier.

2) L'utilisateur peut éventuellement modif er le diagramme de cas d'utilisation selon les règles énumérées précédemment. Pour cela, il est autorisé à utiliser l'opérateur UML $<< include >>$ et des nouveaux cas d'utilisation liés à ces $<< include >>$. Il ne peut en aucun cas ajouter des fonctions de haut niveau c'est-à-dire non liées par un $<< include >>$ aux fonctions déjà fournies dans le diagramme de cas d'utilisation. Dans le cas contraire, les garanties données par notre approche - et expliquées plus loin - ne sont plus valables.

3) Pour chaque IOD référencé dans l'IODh, deux possibilités s'offrent à l'utilisateur : compléter cet IOD ou bien le laisser vide. Le fait de compléter un IOD revient à lui donner une sémantique à l'aide d'un organigramme qui met en exergue des enchaînements vers d'autres IOD ou scénarios. Les fonctions ajoutées au diagramme de cas d'utilisation doivent se retrouver au niveau des IOD référencées par l'IODh, soit sous la forme d'un scénario, soit sous la forme d'un IOD. Notons enf n que la structure générale de l'IODh n'est ainsi pas modif able par l'utilisateur. Ainsi, à la f n de l'étape 3, la modélisation est constituée notamment de l'IODh (lOD sur la f gure 3) ainsi que d'un ensemble d'IOD (IODs sur la f gure 3).

4) Une fois que l'utilisateur a terminé de remplir les références qu'il souhaite dans l'IODh, un algorithme - dit *algorithme de filtrage* - est appliqué af n notamment de débarrasser l'IODh de références vers des IOD vides, et un algorithme d'instanciation est chargé d'éliminer un éventuel paramétrage du modèle.

5) Une vérif cation syntaxique des diagrammes IOD et SD est réalisée au regard du métamodèle d'analyse TURTLE. Les UCD sont ignorés à cette étape car ils ne sont pas pris en compte pour générer une spécif cation formelle.

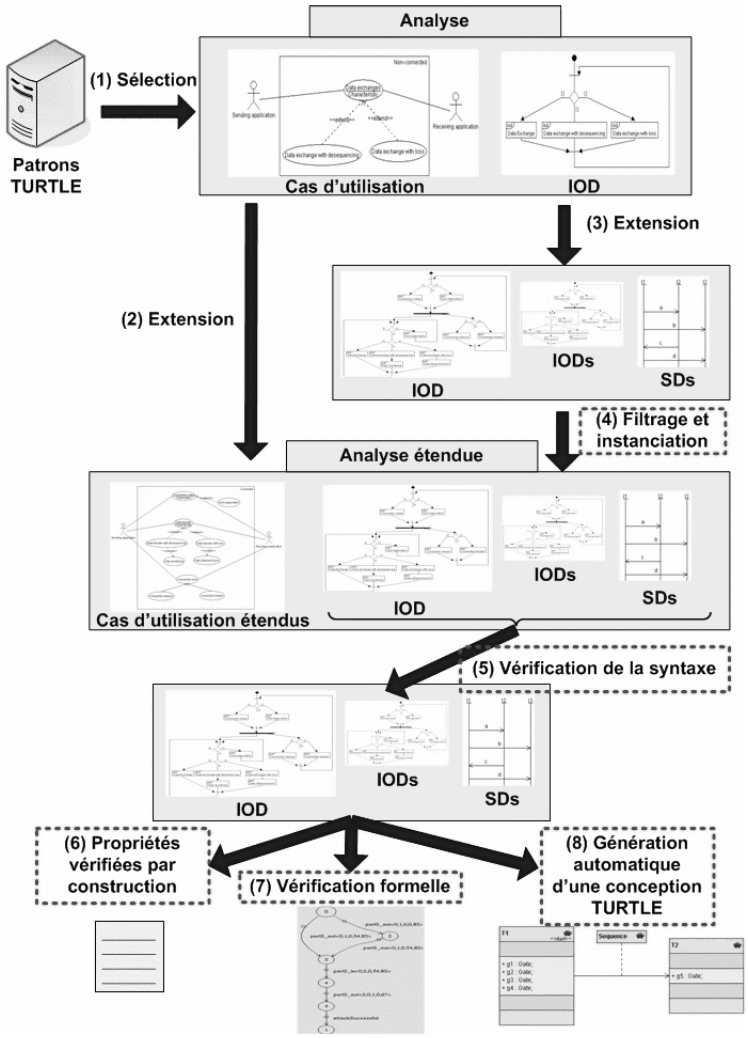

Figure 3. *Méthodologie générale d'utilisation des patrons*

6) Nos patrons garantissent *par construction* un certain nombre de propriétés examinées par la suite, et illustrées sur des exemples.

7) La vérif cation formelle peut s'effectuer avec RTL, CADP ou UPPAAL si et seulement si les diagrammes TURTLE considérés sont implémentables (Apvrille *et al.*, 2005). L'idée est alors de prouver des propriétés qui ne sont pas assurées par construction par le patron considéré. Le résultat ce cette vérif cation est un graphe d'accessibilité ou bien le résultat du *model-checking* d'une formule CTL/LTL.

8) La dernière étape est la génération automatique d'une conception à partir de l'analyse effectuée, c'est-à-dire la construction d'une conception orientée objet sémantiquement équivalente à l'analyse (Apvrille *et al.*, 2005). Rappelons que cette génération automatique préserve les propriétés du modèle d'analyse si et seulement si ce dernier est *implémentable* (Apvrille *et al.*, 2005). Cette dernière étape méthodologique est discutée plus en détail dans la section 5.5.

3.3. *Sémantique*

Les patrons que nous proposons possèdent une sémantique formellement définie par traduction en (RT-)LOTOS, au travers de quatre étapes explicitées ci-après.

3.3.1. *Filtrage*

L'IODh est "filtré" selon la procédure suivante. Tout d'abord, les références vers des IOD qui ont été laissées vides sont éliminées de l'IODh, c'est-à-dire que les connecteurs menant et partant de ces références sont supprimés ; les références elles-mêmes sont supprimées de l'IODh. L'UCD est utilisé par cet algorithme de filtrage pour identifier les références correspondant à une fonction de haut niveau. En effet, si une référence correspondant à une fonction de haut niveau a été laissée vide, alors la traduction ne peut avoir lieu.

3.3.2. *Instanciation*

L'étape de filtrage est suivie d'une procédure d'instanciation des paramètres du modèle.

– le paramétrage se définit au niveau des UCD. Seuls les acteurs peuvent être paramétrés, mais notre approche pourrait être appliquée à d'autres instances des scénarios que les acteurs. Notons que les paramètres doivent être distincts dans leur nommage (n, m, etc.) ;

– l'utilisateur, au moment du filtrage, est amené à saisir pour chaque paramètre (n, m, etc.) une valeur d'instanciation. En effet, nos méthodes de vérification formelle ne peuvent pas prendre en compte des paramètres non instanciés. Ces paramètres doivent bien entendu être choisis par l'utilisateur pour trouver un bon compromis entre réalisme du modèle instancié et explosion combinatoire lors de la vérification formelle ;

– chaque paramètre est remplacé par sa valeur d'instanciation selon les trois schémas suivants (dans les exemples cités ci-après, on suppose que le système comporte un acteur nommé *Actor* et est paramétré avec le paramètre *n*, cf. figure 4) que nous instancions dans nos exemples à la valeur 3.

– **Schéma n° 1 :** Une instance de scénario portant le nom d'un acteur paramétré est remplacée par n instances de cet acteur. Tous les messages arrivant ou partant d'une instance paramétrée sont dupliqués autant de fois que nécessaire. Une option permet

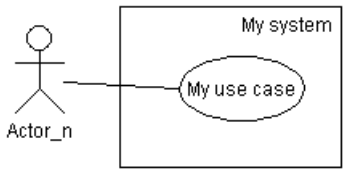

Figure 4. *UCD considéré pour montrer le principe du paramétrage*

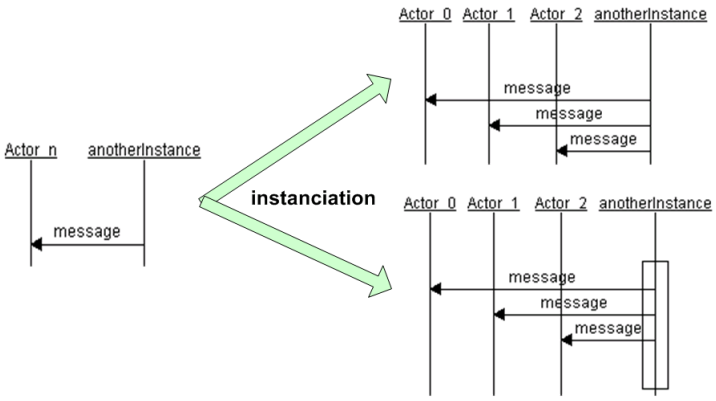

Figure 5. *Instanciation d'instances multiples*

en outre d'utiliser des corégions au niveau de l'instance non paramétrée communiquant avec une instance paramétrée (cf. f gure 5). Ces coregions ont l'avantage de modéliser un entrelacement entre les communications avec les différentes instances, mais engendrent bien entendu une plus grande complexité du graphe d'accessibilité du modèle.

– **Schéma n˚ 2 :** Il est possible de déclarer des variables parametrées. Ainsi il existe dans le système autant d'instances de cette variable que d'acteurs à laquelle la variable fait référence. Par exemple, *var_Actor = 0* signif e que *var_Actor_0 = 0*, que *var_Actor_1 = 0*, etc. Cette instanciation de variable peut intervenir dans toutes les expressions entières des instances paramétrées (cf. f gure 6).

– **Schéma n˚ 3 :** Les gardes de choix des IOD pouvant utiliser des variables, nous avons voulu rendre ces gardes paramétrables, c'est-à-dire que des variables paramétrées peuvent être utilisées dans ces gardes. La f gure 7 met en évidence l'instanciation d'un choix gardé avec une variable paramétrée. La garde *[anInstance / x_Actor == 2]* signif e que la garde ne s'applique qu'à l'instance *anInstance* et que la valeur de cette garde est *[x_Actor == 2]* : pour les instances autres que *anInstance*, le choix n'est pas gardé, il est donc considéré comme non déterministe. Aussi, l'instanciation de ce choix amène à modif er *Scenario1* : ce dernier est supposé comporter une ins-

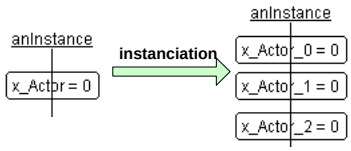

Figure 6. *Variables paramétrées*

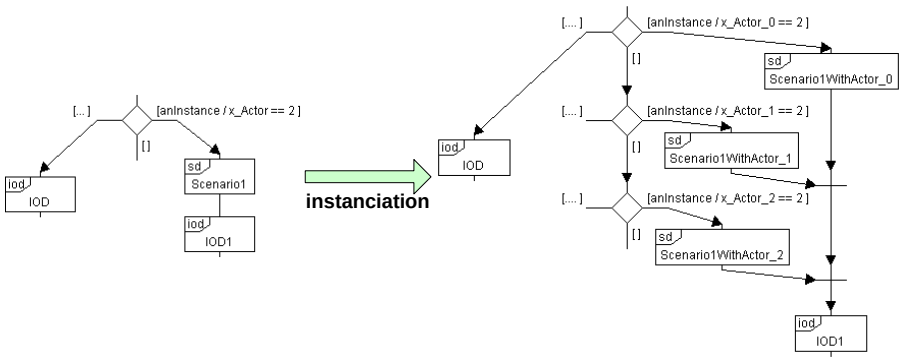

Figure 7. *IOD paramétré*

tance *Actor* qui, selon l'instanciation de la garde du choix, est transformée en *Actor_0*, *Actor_1* ou *Actor_2* dans le scénario suivant la garde paramétrée.

Finalement, ces différents schémas de paramétrage permettent d'exprimer des systèmes distribués de façon compacte (cela est mis en évidence au niveau de l'étude de cas de cet article), et de vérif er formellement le système pour différentes valeurs d'instanciation.

3.3.3. *Génération d'une spécification TIF*

TIF (*TURTLE Intermediate Format*) est un langage pivot formel utilisé par TTool et qui sert de base à la génération de code RT-LOTOS, LOTOS, UPPAAL et Java. TIF est construit autour d'éléments structurants (des *tclasses*) mis en relation de composition (parallélisme, synchronisation) et dont le comportement peut-être décrit au travers d'activités. TIF peut-être vu tout simplement comme une conception TURTLE (Apvrille *et al.*, 2004) qui n'admet pas de représentation graphique, et qui ne comporte donc pas les mêmes limitations : par exemple, le choix UML TURTLE possède au plus trois gardes (contraintes graphique), alors que le choix en TIF ne comporte pas cette contrainte. Notre approche basée sur ce langage pivot TIF confère ainsi à toute analyse (*i.e.* à un ensemble constitué d'IOD et de SD) une sémantique formelle (Apvrille *et al.*, 2005).

3.3.4. *Génération d'une spécification (RT-)LOTOS*

Cette génération est effectuée à partir du format intermédiaire en TIF. Cette génération a fait l'objet d'un article (Apvrille *et al.*, 2004).

3.4. *Propriété vérifiée par construction*

3.4.1. *Définitions*

Définition 1 *Un IOD se définit comme un n-uplet* $(N, N_0, t, succ)$ *avec :*

– N*, un ensemble fini de nœuds ;*

– N_0*, un nœud initial, avec* $t(N_0) =$ *START ;*

– t*, une fonction de typage des nœuds :* $t : N \mapsto Type$ *avec* $Type = \{$*START*, *STOP*, *CHOICE*, *SEQ*, *PAR*, *PREEMPT*, *JUNC*, *IOD*, *SD*$\}$*. Un seul nœud de type START peut appartenir à un IOD : ce nœud est* N_0*. Si le type d'un nœud est* IOD*, l'IOD référencé peut être obtenu avec une fonction* $iod : N \mapsto$ *IOD. De façon similaire, si le type d'un nœud est SD, le SD référencé par ce nœud peut être obtenu avec une fonction* $sd : N \mapsto$ *SD ;*

– $succ$*, une fonction de définition du successeur d'un nœud :* $succ : N \mapsto N$*. Les nœuds, par défaut, ne peuvent avoir qu'un seul successeur, à l'exception de CHOICE*, *SEQ*, *PAR*, *PREEMPT dont le nombre de successeurs n'est pas limité, et STOP qui ne peut avoir de successeur. On note* $succ_i(n)$ *le ième successeur du nœud* n*.*

PAR est utilisé pour représenter un nœud de parallélisme et JUNC pour représenter les jonctions entre branches d'IOD. Aussi, le nœud de type IOD référence un IOD, alors que le nœud SD référence un SD, qui peut être déf ni formellement ainsi :

Définition 2 *Un SD se définit comme un n-uplet* $(I, A, t, i, <)$ *avec :*

– I*, un ensemble fini d'instances*

– A*, un ensemble fini d'actions*

– t *une fonction de typage des actions. Pour simplifier,* $t : N \mapsto$ *TypeAction avec, pour simplifier,* $TypeAction = \{$*SEND*, *RECEIVE*, *INTERNAL*$\}$

– i*, une fonction qui associe à une action une instance :* $i : A \mapsto I$

– $<$*, une relation d'ordre total entre actions d'une même entité, c'est-à-dire* $\forall a0, a1 \in A, i(a0) = i(a1) \Rightarrow ((a0 < a1) \vee (a1 < a0))$

Aussi, on note par $last : E \times SD \mapsto A$ *la fonction qui retourne la dernière action de chaque scénario, et par* $first : E \times SD \mapsto A$ *la fonction qui retourne la première action de chaque scénario.*

Définition 3 *La fonction* $sousch : N \mapsto n_1 \ldots n_n$ *de sous-chemin d'un nœud* n_0 *associe à un nœud une suite ordonnée de nœuds. Le chemin d'un IOD IOD* $=$

$(N, N_0, t, succ)$ *est noté* ch(*IOD*) *avec* ch(*IOD*) $= N_0.sousch(N_0)$. *On note* $subch_i(ch)$ *les ièmes premiers éléments d'un chemin, et* ch_i *le ième nœud d'un chemin* ch. *En supposant que les opérateurs de préemption et de parallélisme possèdent au maximum deux successeurs, la fonction* $sousch$ *est définie de la façon suivante :*

– $t(n_0) =$ (*START* $\vee SEQ \vee$ *JUNC* $\vee SD) \Rightarrow sousch(n_0) = sousch(succ_0(n_0))$

– $t(n_0) =$ *CHOICE* $\Rightarrow sousch(n_0) = sousch(succ_0(n_0)) \vee sousch(succ_1(n_0)) \vee \ldots \vee \vee sousch(succ_n(n_0))$

– $t(n_0) =$ *IOD* $\Rightarrow sousch(n_0) = ch(IOD).sousch(succ_0(n_0))$

– $t(n_0) =$ *PAR* $\Rightarrow sousch(n_0) =$ *interleaving total de nœuds entre* $sousch(succ_0(n_0))$ *et* $sousch(succ_0(n_1)$

– $t(n_0) =$ *PREEMPT* $\Rightarrow (\mid subch(n_0) \mid finiteetsubch_0(n_0).sousch(succ_0(n_1)) \vee subch_1(n_0).sousch(succ_0(n_1)) \vee subch_2(n_0).sousch(succ_0(n_1)) \vee \ldots \vee subch(n_0)) \vee (midsubch(n_0) \mid finiteetsubch_0(n_0).sousch(succ_0(n_1)) \vee subch_1(n_0).sousch(succ_0(n_1)) \vee subch_2(n_0).sousch(succ_0(n_1)) \vee \ldots)$

– $t(n_0) = STOP \Rightarrow sousch(n_0) = \varnothing$

Enfin, on note $last(ch, E)$ *la dernière action d'une entité* E *sur un chemin* ch.

Un chemin est ainsi un parcours d'un IOD, en suivant les successeurs du nœud initial - sauf dans le cas de la préemption et du parallélisme - et en s'arrêtant toujours à un nœud de type STOP (sinon, le chemin est inf ni).

Définition 4 *Un IOD de haut niveau* IOD_h *se définit comme un n-uplet* (IOD_e, SD_e, IOD_0) *avec :*

– IOD_e*, un ensemble fini d'IOD*

– SD_e*, un ensemble fini de SD*

– IOD_0*, un IOD « initial », appelé aussi IOD de haut niveau.*

Un IODh est correctement construit ssi :

– $IOD_0 \in IOD_e$.

– $\forall iod = (N, N_0, t, succ) \in IOD_e, (n \in N \wedge t(n) =$ *IOD*$) \Rightarrow iod(n) \in IOD_e$

– $\forall iod = (N, N_0, t, succ) \in IOD_e, (n \in N \wedge t(n) =$ *SD*$) \Rightarrow sd(n) \in IOD_e$

3.4.2. *Propriété*

Considérons un IOD de haut niveau $IOD_h = (IOD_e, SD_e, IOD_0)$ avec $IOD_0 = (N, N_0, t, succ)$, et deux nœuds n_0 et n_1 avec $(n_0, n_1) \in N^2$, avec $n_1 = succ_i(n_0)$ et $t(n_0) = t(n_1) = IOD$ et $iod(n_0) = IOD_0$ et $iod(n_1) = IOD_1$. Par hypothèse, $\forall n \in ch(IOD_0), \forall iod \in IOD_e \setminus IOD_0, IOD_e, n \notin ch(iod)$ et $\forall n \in ch(IOD_1), \forall iod \in IOD_e \setminus IOD0, IODe, n \notin ch(iod)$.

Pour toute instance I telle que $\forall ch(IOD_0)\exists n_2/(t(n_2) = SD \wedge last(sd(n_2), I) \neq \varnothing) \wedge (\forall ch(IOD_1)\exists n_3/(t(n_3) = SD \wedge first(sd(n_3), I) \neq \varnothing)$, alors, pour tout chemin $ch \in ch(IODh)$, $\exists i/(t(ch_i) = SD \wedge first(sd(\text{SD}), I) \neq \varnothing \wedge ch_i \in ch(IOD1)) \Rightarrow (\exists j < i/t(ch_j) = SD \wedge last(sd(\text{SD}), I) \neq \varnothing \wedge ch_j \in ch(\text{IOD}))$.

Moins formellement, la propriété exprime que pour toutes les instances des scénarios qui effectuent au moins une action - c'est-à-dire un envoi de message, une réception de message ou une action interne sur une porte - dans chaque chemin possible de deux sous-IOD IOD1 et IOD2 référencé par l'IODh, si l'IODh ne comporte pas de comportements concurrents (pas d'opérateurs de parallélisme), et IOD1 et IOD2 n'ont pas de nœuds en commun avec les autres IODs de l'IODh, alors si une instance effectue une action lors de l'exécution de l'IOD2, il est garanti par construction que cette même instance a fait auparavant au moins une action de l'IOD1.

La preuve de cette propriété repose sur la sémantique de construction des chemins des IODs (notamment la préemption), le parallélisme étant exclu par hypothèse, et sur la possibilité de référencement de scénarios et d'IODs.

A titre d'exemple de cette propriété, considérons le patron du mode connecté présenté à la f gure 9. Si une instance *Client* effectue des actions dans tous les chemins possibles des sous-IOD, alors le patron garantit par construction que les données ne peuvent être reçues par cette instance - si cette réception est implantée dans l'IOD *Data exchange* - qu'une fois la connexion établie - si la connexion est réellement établie dans *Connection setup*.

Les règles d'utilisation des patrons, ainsi que la méthodologie générale présentées ci-avant permettent de donner un guide à la personne en charge de modéliser. Ce dernier a la possibilité, s'il ne modif e pas le patron, d'obtenir des propriétés de son modèle qui sont vraies par construction. Toutefois, notre approche ne permet pas de recalculer des propriétés vraies par construction dans le cas où le modeleur n'applique pas les règles citées précédemment (cf. section 3.1). Que la personne qui modélise se conforme ou non aux patrons, la sémantique des modèles d'analyse TURTLE permet d'effectuer des vérif cations formelles *a posteriori* sur les modèles.

L'impossibilité d'utiliser l'opérateur de parallélisme peut apparaître comme une limitation importante, notamment dans le cadre de protocoles pour lesquels des comportements concurrents existent : par exemple, la réception d'une donnée alors qu'un PDU est en cours de construction. Toutefois, la sémantique du prof l TURTLE, donnée en (RT-)LOTOS fait reposer l'opérateur de parallélisme des IOD sur celui de LOTOS, c'est-à-dire un entrelacement entre actions. Notre expérience d'utilisation des patrons présentés dans cet article a montré que l'entrelacement inhérent aux protocoles est en général aisé à exprimer en utilisant uniquement l'opérateur de choix, lequel supporte en outre notre paramétrisation de modèle. De plus, l'opérateur de parallélisme d'UML introduit d'autres problèmes, et notamment celui de la cohérence des données d'une instance.

3.5. *Outillage support aux patrons*

L'outil TTool (Apvrille, 2010) permet la sauvegarde et l'importation de librairies et modèles contenant des modélisations TURTLE faites sous TTool. C'est sous cette forme que les patrons présentés à la section suivante peuvent être chargés dans TTool puis utilisés. Ainsi, une fois TTool lancé, un utilisateur procèdera comme suit pour utiliser un patron :

1) il charge le patron correspondant ;

2) il complète le diagramme de cas d'utilisation du patron, comme expliqué dans la méthodologie exposée à la section précédente ;

3) il complète les différents IOD obligatoires et éventuellement ceux facultatifs, en ajoutant à la modélisation TURTLE les diagrammes globaux d'interactions et de séquences nécessaires, et résoud le paramétrage du modèle ;

A ce jour, TTool n'est pas capable de vérif er que la personne a respecté le patron, et a notamment fourni les diagrammes obligatoires. Par contre, TTool vérif e que la syntaxe des diagrammes est correcte, par rapport au métamodèle TURTLE. Une fois cette vérif cation syntaxique effectuée, TTool construit une modélisation interne en format TIF qui tient compte des diagrammes fournis par l'utilisateur mais aussi du patron. A partir de ce format interne, l'utilisateur a deux possibilités :

– effectuer une vérif cation formelle, par génération de code LOTOS ou RT-LOTOS. Cette vérif cation formelle peut-être réalisée directement depuis TTool sans connaissance particulière de ces langages formels, ni des outils de vérif cation associés ;

– générer automatiquement une première conception, qui tienne compte à la fois de la modélisation de l'utilisateur et du patron.

Ainsi, actuellement, l'approche générale précédemment présentée n'est que partiellement implémentée. Deux points manquent plus particulièrement :

– la vérif cation du fait que l'utilisateur s'est bien inscrit dans le patron qu'il a chargé dans TTool. Une vérif cation syntaxique suff rait pour cela, mais il faudrait aussi déf nir un langage - par exemple sous forme d'un métamodèle - permettant d'exprimer, pour chaque patron, ses règles syntaxiques propres, et d'appliquer les dites règles au niveau de l'outillage ;

– la gestion du paramétrage est possible en modélisation, mais la transformation automatique d'un modèle paramétré en un modèle instancié n'est pas encore possible.

4. Patrons TURTLE pour les protocoles

4.1. *Patron pour les protocoles en mode connecté*

Ce patron vise l'analyse des protocoles comportant une phase d'établissement de connexion avant tout échange de données. La phase d'échange de données se poursuit

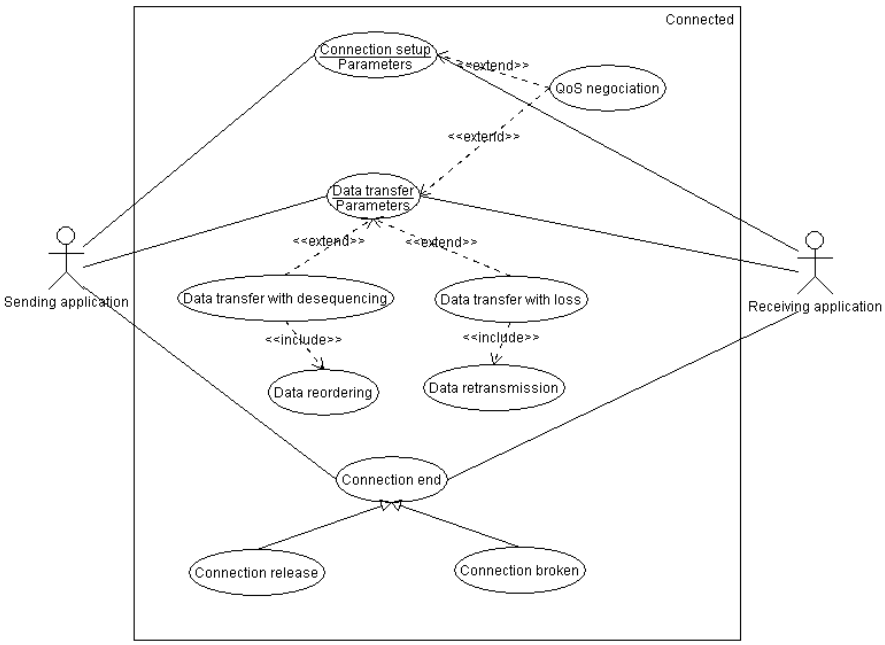

Figure 8. *Patron pour les protocoles en mode connecté : UCD*

jusqu'à la libération - voulue ou non - de la connexion. Le patron est construit comme suit :

– son diagramme de cas d'utilisation (voir f gure 8) met en évidence trois principales fonctions :

- l'établissement de la connexion (obligatoire, voir la section 3.1), avec négociation de la qualité de service (optionnel) ;

- l'échange de données (obligatoire), qui peut éventuellement comporter des déséquencements et des pertes de données, qui sont alors gérés par le protocole (par exemple *Data transfer with loss* inclut *Data retransmission*) ;

- la f n de la connexion qui est soit volontaire ou soit imputable à une rupture de connexion (la modélisation des deux est obligatoire). L'utilisation d'une relation d'héritage entre cas d'utilisation signif e qu'un des deux IOD correspondant à la terminaison de la connexion devra ultérieurement être renseigné et donc ne pas rester vide (cf. les propriétés des patrons, section 3.3) ;

– son diagramme global d'interactions comporte lui aussi trois parties principales (f gure 9) :

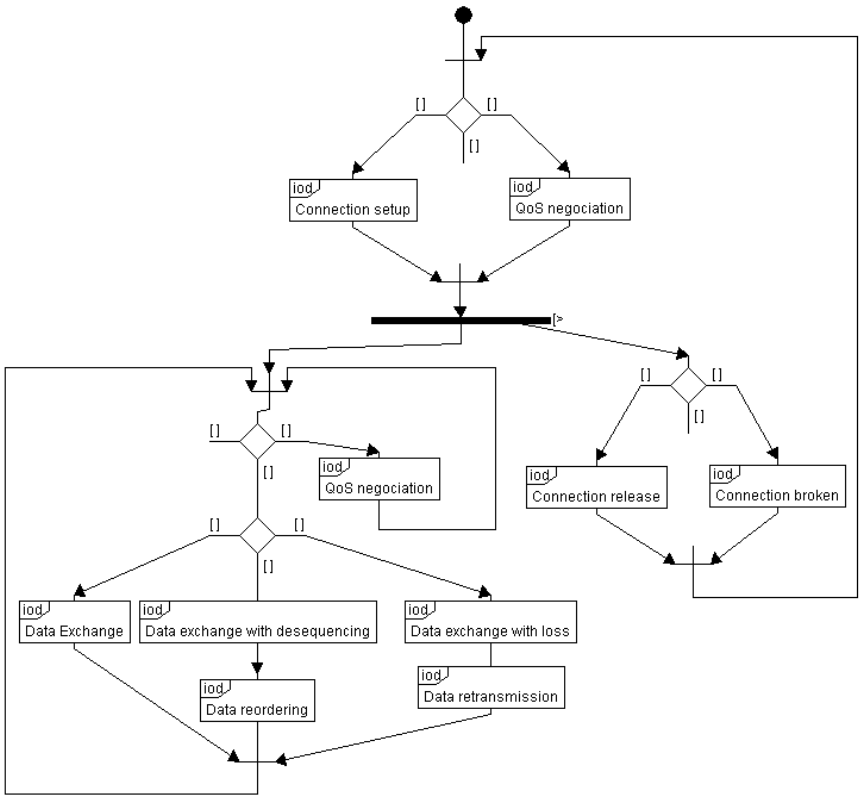

Figure 9. *Patron pour les protocoles en mode connecté : IODh*

- la partie du haut met en évidence le fait que l'ouverture de connexion se fait avec ou sans négociation de qualité de service ;

- la partie située en bas à gauche met en évidence les échanges de données, avec éventuellement renégociation de la qualité de service en cours de connexion ;

- enf n, la partie en bas à droite met en évidence que la connexion une fois établie peut être rompue ou relâchée involontairement. Notons l'opérateur de préemption - à la LOTOS - qui permet de modéliser cela. Cette extension aux diagrammes globaux d'interactions UML a été présentée dans (Apvrille *et al.*, 2005).

Une première remarque concernant le patron est que les fonctions incluses de l'UCD (*Data reordering*, *Data retransmission*) ne doivent être implantées sous forme d'IOD et de SD que si les fonctions optionnelles les incluant sont elles-mêmes renseignées. Aussi bien nos patrons que TTool ne font aucune vérif cation en ce sens, et rien n'interdit d'ailleurs de placer dans un IOD appelé *Connection setup* un échange de données :

les noms des fonctions de l'UCD et des IOD ne sont que des assistants à l'analyse qui ne peuvent brider les choix de l'utilisateur.

Une deuxième remarque sur ce patron est relative à la *propriété* des patrons. Si l'on suppose que nos scénarios comportent une instance *Server* et que cette instance *Server* effectue au moins une action dans chacun des IOD obligatoires, alors, par construction, on garantit que les actions effectuées dans *Data Exchange*, *Connection broken* et *Connection release* le sont forcément après les actions effectuées par *Server* dans *Connection setup*.

4.2. *Patron pour les protocoles de diffusion*

Le patron proposé est destiné à la modélisation des protocoles de diffusion d'un serveur vers n clients abonnés à un groupe de diffusion. Af n de gérer ces n clients, le patron est paramétré : nous discutons de ce paramétrage à la section 5.5. Ce patron est constitué :

– d'un diagramme de cas d'utilisation. Ce dernier met en évidence d'une part des fonctions de manipulation du groupe de diffusion, et d'autre part des échanges de données dans ce groupe de diffusion. En ce qui concerne la manipulation du groupe de diffusion, un groupe peut être créé - avec éventuellement des paramètres liés à de la qualité de service - ou détruit. Un client (*ReceivingApp*) peut s'ajouter à un groupe de façon standard, ou en précisant ses propres paramètres de qualité de service. Au niveau de l'échange de données, des pertes ou des déséquencements peuvent apparaître ;

– d'un diagramme global d'interactions (cf. f gure 11). Ce dernier met en évidence la création du groupe. Une fois le groupe créé, des ajouts ou retraits de clients peuvent intervenir sur le groupe. De même, des données peuvent être émises au sein du groupe. Enf n, la diffusion de données et l'ajout/le retrait de clients au sein du groupe cesse lorsque le groupe est supprimé (opérateur de préemption).

Enf n, un utilisateur ne pourra se joindre au groupe que si le nombre maximal d'utilisateurs du groupe n'est pas atteint. Cela se modélise par un simple ajout d'une garde sur le choix précédant l'ajout des utilisateurs. L'étude de cas traitée par la suite en est un exemple qui permet en outre de mieux expliciter ce patron.

5. Etude de cas : un protocole Multicast

5.1. *Présentation du protocole*

Dans cette étude de cas, nous avons repris le patron proposé dans le cadre de la diffusion de données, et nous l'avons appliqué à une version simplif ée d'un protocole de diffusion de données multimédias proposé dans le cadre du projet européen *Maestro* (Thalès-Alenia-Space, 2006). Dans ce protocole, un émetteur de f ux multimédias demande à un serveur principal la création d'un groupe de diffusion auquel des

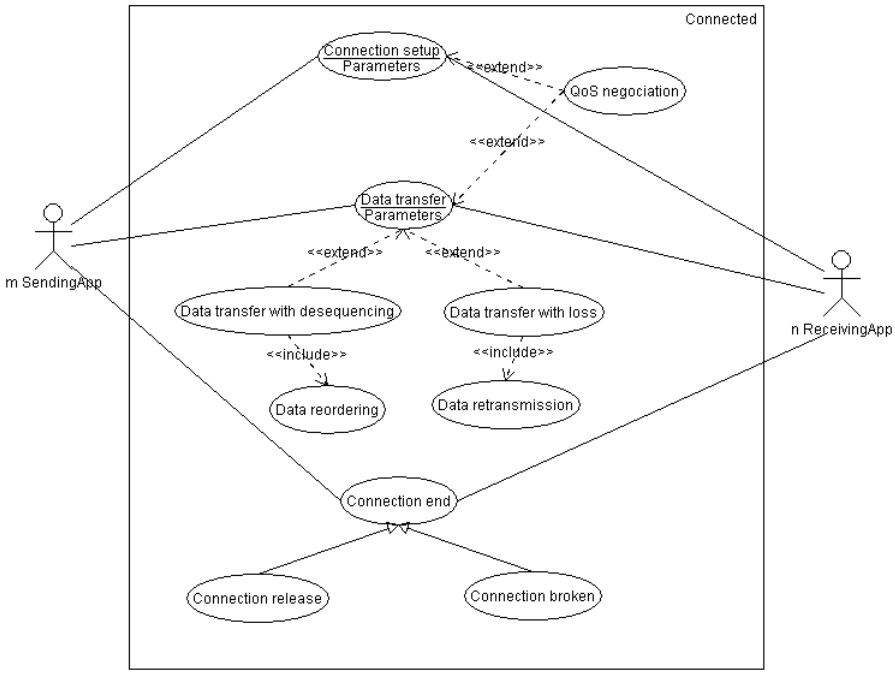

Figure 10. *Patron pour les protocoles de diffusion : diagramme de cas d'utilisation*

utilisateurs peuvent par la suite se joindre en contactant ledit serveur. Lorsqu'au moins un utilisateur est présent dans le groupe, les données sont régulièrement émises vers un satellite de télécommunication multifaisceau qui possède un routeur embarqué capable de diffuser des données vers certains faisceaux : la diffusion dans un faisceau donné n'a lieu que dans le cas où au moins un utilisateur de ce faisceau est inscrit au groupe de diffusion. Notons enf n que les données échangées entre les clients du groupe et le serveur sont effectuées par une voie retour terrestre.

5.2. *Analyse basée sur les patrons*

Cet exemple utilise le patron paramétré qui correspond aux systèmes multicast. Nous avons fourni les comportements des IOD suivants :

– *création d'un groupe*. Cette création est réalisée lorsqu'un serveur de données multimédias désire émettre des données sur le système satellite. Une seule application émétrice est considérée dans notre exemple ;

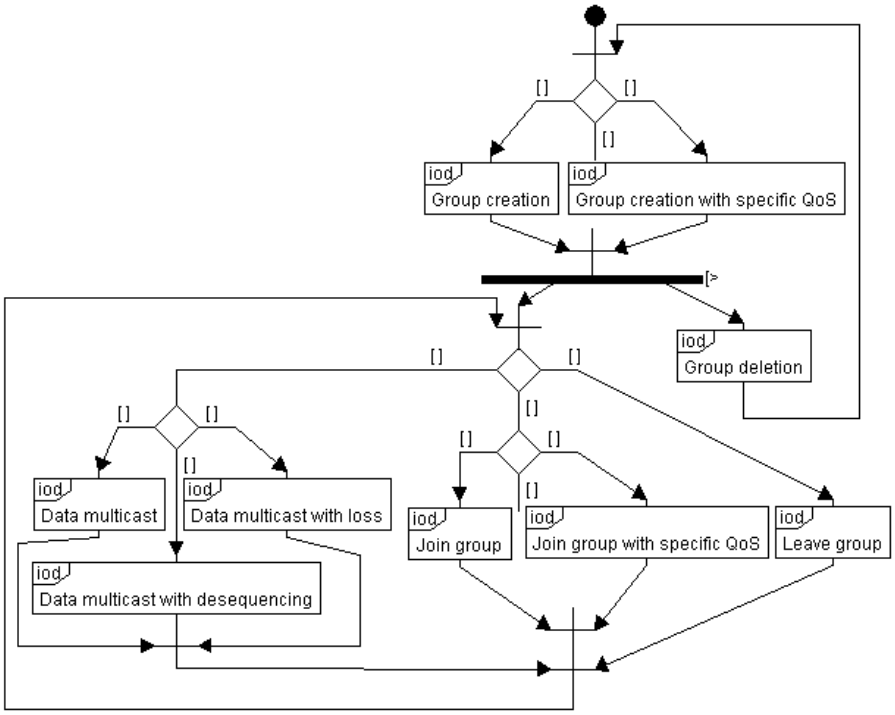

Figure 11. *Patron pour les protocoles de diffusion : diagramme global d'interactions (IOD)*

– *ajout d'un utilisateur à un groupe*. Nous nous sommes limités, pour des raisons de complexité, à des utilisateurs situés dans deux faisceaux différents ;

– *Retrait d'un utilisateur d'un groupe* ;

– *émission de données*. Le serveur multimédias met à jour les données à émettre au niveau du système satellite (*Gateway*). Ces données sont transmises si et seulement si l'un au moins des utilisateurs s'est ajouté au groupe. Si tel est le cas, les données sont envoyées, accompagnées de FEC[1], à bord du satellite, avec des informations statiques de routage. Les données et FEC sont alors routées vers les faisceaux correspondant aux utilisateurs du groupe. Notons que ces données sont émises vers tous les utilisateurs inscrits au groupe au moment de leur émission sur leur satellite, même si entre temps les utilisateurs se sont désinscrits. L'émission de données se termine par l'envoi d'acquittements des clients vers le serveur de groupe ;

1. *Forward Error Correcting*.

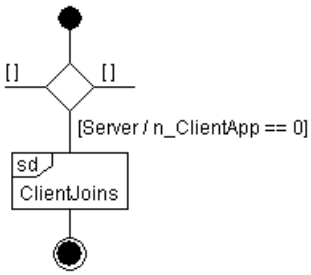

Figure 12. *IOD "A client joins the group"*

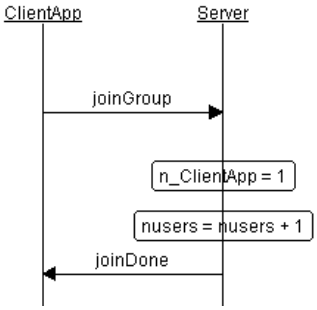

Figure 13. *Diagramme de séquence UML "ClientJoins"*

– *destruction d'un groupe*. Le serveur de groupe peut décider de fermer le groupe, soit pour cause d'erreur, soit parce que le serveur de données multimédias a terminé sa diffusion. Les clients sont informés de cette fermeture, mais peuvent neanmoins continuer à recevoir les dernières données émises par le serveur multimédias.

Plus précisément, et à titre d'exemple de base, nous fournissons le contenu de l'IOD *A client joins the group* à la f gure 12. Cet IOD comporte lui-même une référence vers un scénario parametré qui permet à chaque client de s'ajouter au groupe. Par exemple, la f gure 13 montre comment un client paramétré peut s'ajouter au groupe. Certains sous-IOD et scénarios développés pour cette étude de cas sont bien entendu beaucoup plus complexes que ceux fournis dans ces deux f gures.

5.3. *Vérification formelle*

Après avoir rempli le patron selon le mode décrit précédemment, nous avons utilisé le générateur automatique de code LOTOS de TTool (Apvrille, 2010) et l'outil

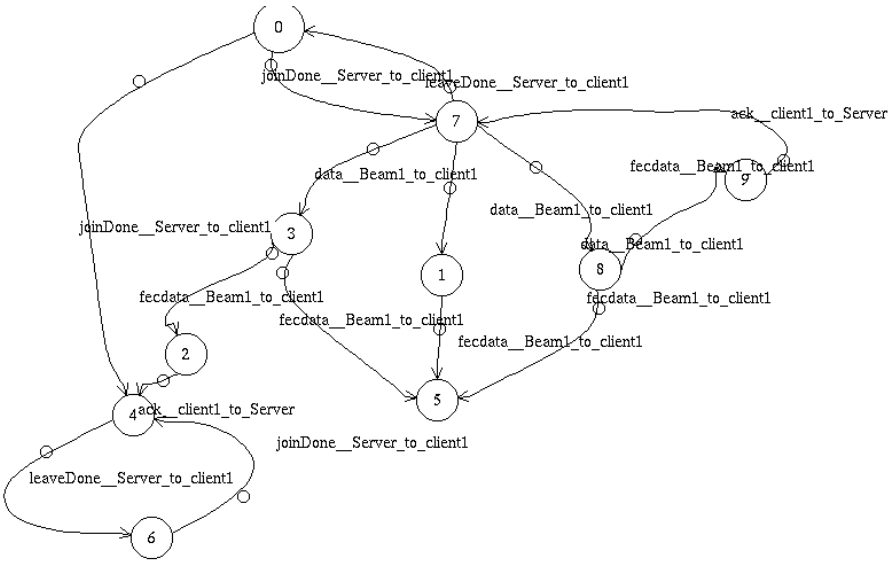

Figure 14. *Graphe d'accessibilité minimisé par rapport aux seules actions de client1*

CADP (INRIA-VASY, 2008) af n de construire le graphe d'accessibilité du modèle de notre application. Ce graphe comporte environ 6 millions d'états et 25 millions de transitions.

Nous avons également construit le graphe d'accessibilité d'une version de cette application dépourvue du rebouclage au niveau de l'IOD principal (*i.e.* le lien entre la destruction du groupe et sa création). La taille plus modeste du graphe résultant nous a permis de réaliser plus aisément des preuves de propriétés. A titre d'exemple, la f gure 14 représente le graphe minimisé par rapport aux actions de *client1*. Ce graphe met en évidence que les données ne sont reçues par *client1* que lorsque ce dernier fait partie du groupe (*joinDone* est toujours avant *data*). De plus, lorsque le système s'arrête, le client reçoit un dernier paquet de données non acquitté. Finalement, nous avons pu prouver ceci :

– un utilisateur - *i.e.* une *ReceivingApp* - ne peut pas recevoir de données s'il n'est pas membre du groupe ;

– un utilisateur peut recevoir au maximum une donnée supplémentaire après fermeture du groupe. Il s'agit de la donnée en cours d'émission avant l'ordre de fermeture du groupe.

Notons que ces propriétés ne sont pas prouvées par construction au niveau du patron. Leur preuve nécessite d'avoir recours à la génération de graphe d'accessibilité.

5.4. *Génération d'une conception*

TTool permet la génération d'une conception équivalente à l'analyse, aux problèmes de non-implémentabilité près (Apvrille *et al.*, 2005). Si l'on considère le système de cette étude de cas, la conception générée comprend 8 classes qui correspondent aux 8 instances des scénarios décrites lors de l'analyse (SD). Le comportement de ces classes est également généré de manière automatique depuis les IOD et scénarios de l'analyse.

5.5. *Discussions et limitations*

Nous avons appliqué avec succès notre approche par patron à un système distribué que l'on pourrait qualifier de complexe (plusieurs médiums de communication, routage entre faisceaux, bufferisation, gestion de groupe, etc.). Même si la vérification n'a pu se faire que dans un contexte avec assez peu d'utilisateurs, l'analyse du système, en se basant sur le patron, a été réalisée en un peu moins d'une journée, ce qui démontre la pertinence de l'approche. Toutefois, nous avons rencontré un certain nombre de limitations.

Tout d'abord, le problème de la non-implémentabilité, abordé auparavant dans le cadre de l'analyse TURTLE (Apvrille *et al.*, 2005). La non-implémentabilité d'un modèle d'analyse signifie qu'il n'existe aucune conception objets structurée de la même façon que le modèle d'analyse (*i.e.* la conception objets comporte un objet par instance contenue dans les scénarios) et sémantiquement équivalente au modèle d'analyse. Cette non-implémentabilité est due à une différence fondamentale entre les modèles d'analyse et les modèles objets au niveau de la sémantique des alternatives. En effet, l'alternative, au niveau des IOD, représente un choix effectué globalement par toutes les instances : tel scénario est exécuté, ou tel autre. Toutes les instances d'un scénario doivent prendre la même décision. Au niveau conception, les choix sont uniquement locaux aux objets : c'est-à-dire qu'un objet peut décider par lui-même de partir dans une branche d'une de ses alternatives (contrairement aux instances des scénarios). La non-implémentabilité existe ainsi lorsqu'au moins un choix global d'un IOD ne peut pas être traduit sous la forme de choix locaux à des objets : on parle alors de *choix distribué*. Les patrons proposés ne garantissent en rien cette implémentabilité. Cette dernière provenant de *choix distribués*, nous nous sommes efforcés lors de l'utilisation de ces patrons, et en particulier lors de l'étude de cas décrite dans cet article, de ne pas utiliser de tels choix. Cette contrainte est apparue comme assez forte, et nous a amenés à utiliser des messages synchrones plutôt que des messages asynchrones dans certaines situations, notamment dans la modélisation des interactions client - serveur de groupe.

Le deuxième problème rencontré concerne le rebouclage. Les patrons fournis font l'hypothèse d'un rebouclage vers une nouvelle connexion lorsque la connexion en cours est terminée. Ce rebouclage a été prévu pour s'inscrire dans un cadre général de modélisation, mais nuit à la vérification formelle. Par exemple, dans le cadre de

notre système, le graphe d'accessibilité avec rebouclage comporte plusieurs millions d'états, et seulement 30 000 sans ce rebouclage. Le problème d'explosion combinatoire est bien entendu plus général que celui de l'utilisation des patrons. Pour contrer ce problème, une solution serait d'offrir, au niveau des patrons, davantage de propriétés garanties par construction (par exemple, la vivacité).

La troisième limitation concerne le paramétrage, qui d'une part n'est pas automatisé, et est d'autre part limité à quelques schémas d'instanciation. En s'appuyant sur des travaux déjà faits dans un autre prof l UML (Ahumada *et al.*, 2007), supporté par TTool, et supportant du paramétrage de composants, nous souhaitons enrichir nos schémas d'instanciation.

6. Positionnement par rapport aux travaux du domaine

6.1. *Patrons prouvés*

Dans le sillage de l'ouvrage fondateur de Gamma (Gamma, 1995), de nombreux auteurs ont proposé des patrons d'analyse ou de conception dédiés à un domaine d'application particulier. Ainsi, Douglass (Douglass, 2002) et Rising (Rising, 2001) ont respectivement traité des systèmes temps réel et du logiciel de communication. Aujourd'hui, l'idée de partager sous forme de patron des artéfacts largement adoptés par une communauté de praticiens, ne se suff t plus à elle-même. Plusieurs auteurs expriment le besoin de formaliser les patrons et proposent d'adosser un travail sur les patrons à une méthode formelle. Ainsi, (Konrad *et al.*, 2005) associent patrons et logique temporelle et (Lecomte *et al.*, 2007) intègrent les patrons dans une démarche de "correct par construction" à base de langage B.

6.2. *Introduction d'un assistant dans un outil*

L'utilisation de patrons en modélisation de protocoles a été étudié pour SDL (ITU-T, 1996), un langage de conception qui permet de traiter les aspects *architectures* et aussi les parties *contrôle* et *données* d'un protocole. De ce point de vue, SDL peut être rapproché des diagrammes de classes et d'activités portés par TURTLE (cf. section 2.1). L'analyse en amont d'une conception SDL est conf ée aux MSCs (ITU-T, 1999) que nous pouvons rapprocher des SD TURTLE. Notons que l'approche combinée MSC, SDL ne dispose des diagrammes de cas d'utilisation, ni des IODs.

Certains des patrons proposés dans les travaux *SDL Pattern Approach* (Geppert, 2001) ont été implantés dans l'outil SPT (Dorsch *et al.*, 2005) lui-même intégré à l'outil TAU G2 (IBM, 2008). On peut donc qualif er SPT d'assistant méthodologique dédié à la modélisation de protocoles. L'utilisateur/trice peut aisément changer les identif cateurs dans les diagrammes MSCs ou SDL et se servir des patrons ainsi modif és comme point de départ pour la construction de mécanismes de protocoles plus complexes. SPT est exclusivement dédié à l'ingénierie des protocoles. A contrario, la

démarche proposée dans cet article relève en premier lieu d'une approche de génie logiciel qui découle de l'utilisation de diagrammes UML ; néanmoins, l'article en présente une application directe aux protocoles. Notre approche se démarque également d'autres travaux par le fait que l'utilisateur/trice de TTool utilise les patrons dans la seule phase d'analyse. Un algorithme de synthèse d'une architecture de classes/objets et des diagrammes d'activités décrivant le comportement de ces objets, permet à l'utilisateur/trice de TTool de générer automatiquement une première conception que cet utilisateur/trice peut faire évoluer pour la rendre plus proche du système réel. Une autre particularité du prof l TURTLE est que les objets communiquent par rendez-vous ; ceci n'empêche pas de créer des classes émulant telle ou telle politique de f les de messages. A l'inverse, d'autres outils UML tels que TAU G2 imposent d'emblée l'utilisation de f les (non bloquantes dans le cas du prof l UML/SDL supporté par TAU G2).

7. Conclusion

Si le fait de modéliser contribue à maîtriser la complexité des systèmes à concevoir, la diff culté d'appropriation d'un langage de modélisation d'une part et des outils et méthodes d'autre part peut désarçonner bien des praticiens. Ce constat s'applique en particulier à la notation UML 2 et aux prof ls qui la spécialise dans le domaine des systèmes temps réels et communicants. L'un de ces prof ls UML temps réel est TURTLE qui sert de langage support à l'approche de modélisation défendue dans cet article et outillée par le TURTLE toolkit (TTool en abrégé) en ajoutant un assistant méthodologique.

L'article propose tout d'abord d'introduire des patrons paramétrés alliant diagrammes de cas d'utilisation et diagrammes globaux d'interactions, respectivement dédiés à l'identif cation des fonctionnalités du système et à la structuration de scénarios exprimés par des SD. L'approche est appliquée aux protocoles en distinguant les modes « sans connexion », « connecté », et « multicast » (l'article ne détaille que les deux derniers, mais le premier est bien disponible sous TTool). Enf n, ce cadre méthodologique formel et sa spécialisation aux protocoles sont implantés dans TTool. Cet outil a été utilisé pour traiter le cas d'un protocole de diffusion de données multimédias.

L'implantation dans TTool utilise le gestionnaire de bibliothèque de l'outil pour gérer les patrons. Le travail d'implantation va se poursuivre par l'implantation des algorithmes de f ltrage et d'instanciation notamment. D'un point de vue plus conceptuel, le travail de formalisation du cadre méthodologique va se poursuivre par la réalisation de patrons pour les systèmes distribués, et par une réf exion plus générale sur la possibilité d'offrir, avec certaines restrictions, davantage de propriétés satisfaites par construction, avec une extension du cadre formel introduit dans cet article à d'autres diagrammes UML et SysML.

8. Bibliographie

Ahumada S., Apvrille L., Barros T., Cansado A., Madelaine E., Salageanu E., « Specifying Fractal and GCM Components With UML », *XXVI International Conference of the Chilean Computer Science Society (SCCC'07)*, Iquique, Chile, novembre, 2007.

Apvrille L., « TTool », 2010. Available at `.enst.fr/ttool`.

Apvrille L., Courtiat J.-P., Lohr C., de Saqui-Sannes P., « TURTLE : A Real-Time UML Prof le Supported by a Formal Validation Toolkit », *IEEE transactions on Software Engineering*, vol. 30, p. 473-487, juillet, 2004.

Apvrille L., de Saqui-Sannes P., Khendek F., « Synthèse d'une conception UML temps-réel à partir de diagramme de séquences », *Colloque Francophone sur l'Ingéniérie des Protocoles*, Bordeaux, France, mars, 2005.

Apvrille L., De Saqui-Sannes P., Pacalet R., Apvrille A., « Un environnement de conception de systèmes distribués basé sur UML », *Annals of the Telecommunications*, vol. 61, n° 11/12, p. 1347-1368, novembre, 2006a.

Apvrille L. et al., « A UML-based Environment for System Design Space Exploration », *13th IEEE International Conference on Electronics, Circuits and Systems (ICECS2006)*, Nice, France, décembre, 2006b.

Behrmann G., David A., Larsen K. G., A tutorial on UPPAAL, Technical report, novembre, 2004.

Courtiat J.-P., Santos C., Lohr C., Outtaj B., « Experience with RT-LOTOS, a Temporal Extension of the LOTOS Formal Description Technique », *Computer Communications*, vol. 23, p. 1104-123, 2000.

Dorsch J., Ek A., Gotzhein R., « SPT - The SDL Pattern Tool », *D. Amyot, W. Williams (Eds.), System Modeling and Analysis*, vol. 3319, Lecture Notes in Computer Science, Springer, St. Louis, MO, USA, p. 372-381, janvier, 2005.

Douglass B. P., *Real-Time Design Patterns : Robust Scalable Architecture for Real-Time Systems*, Addison wesley, 2002.

Fontan B., Méthodologie de conception de systèmes temps réel et distribués en contexte UML/SysML, thèse de doctorat, Université de Toulouse, Université Paul Sabatier, janvier, 2008.

Gamma E., *Design Patterns*, Addison wesley, 1995.

Geppert B., « The SDL Pattern Approach », *Dissertation, Fachbereich Informatik, Universität Kaiserslautern*, 2001.

Gherbi A., Khendek F., « UML Prof les for Real-time Systems and their Applications », *Journal of Object Technology*, vol. 6, p. 149-169, mai, 2006.

IBM, « the TAU G2 Toolkit », 2008. `http://www.ibm.com`.

INRIA-VASY, « The CADP Toolkit », 2008. Available at `http://www.inrialpes.fr/vasy/cadp`.

ISO-LOTOS, « A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour », *Draft International Standard 8807, International Organization for Standardization - Information Processing Systems - Open Systems Interconnection*, Geneva, juillet, 1987.

ITU-T, « Recommendation Z.100, Specif cation and Design Language (SDL) », 1996.

ITU-T, « Recommendation Z.120, Message Sequence Charts », 1999.

Konrad S., H.C.Cheng B., « Real-time specif cation patterns », *Proceedings of the 27th international conference on Software engineering*, St. Louis, MO, USA, p. 372-381, mai, 2005.

Lecomte T., Cansell D., Méry D., « Patrons de conception prouvés », *Journées NEPTUNE*, mai, 2007.

OMG, « UML Superstructure Specif cation », Geneva, 2009. Available at http ://www.omg.org/spec/UML/2.2/Superstructure/PDF/.

Rising L., *Design Patterns in Communications Software*, Cambridge University Press, 2001.

Thalès-Alenia-Space, « The Maestro european project, Mobile Applications and sErvices based on Satellite and Terrestrial inteRwOrking », 2006. `http://cordis.europa.eu/fetch?CALLER=PROJ\_ICT\&ACTION=D\&CAT=PROJ\&RCN=71246`.

The SystemC Community, « Open SystemC Initiative », 2010. `http://www.systemc.org`.

Ludovic Apvrille *est enseignant-chercheur à Télécom ParisTech. Ses travaux s'intéressent à la modélisation et à la vérification des systèmes-sur-puce et des systèmes embarqués. Il est le principal auteur de la plateforme TTool dont l'un des objectifs est de faciliter l'utilisation des techniques formelles depuis des diagrammes UML avec une approche de type presse-bouton.*

Pierre de Saqui-Sannes *est professeur à l'Institut Supérieur de l'Aéronautique et de l'Espace (ISAE) et chercheur au Laboratoire d'Architecture et d'Analyse des Systèmes (LAAS) du CNRS. Ses travaux allient les langages UML et SysML à des langages formels dotés d'outils de vérification de modèles, avec pour domaine d'application privilégié les systèmes temps réel et distribués.*