Un environnement formel d'assistance à la modélisation de protocoles

Ludovic Apvrille
Institut TELECOM, TELECOM ParisTech, LTCI
CNRS
2229 route des crêtes, B.P.193
F-06904, Sophia-Antipolis Cedex, France
ludovic.apvrille@telecom-paristech.fr

Pierre de Saqui-Sannes
Université de Toulouse, ISAE, LAAS-CNRS
10 avenue Edouard Belin, B.P. 54032
31055 Toulouse Cedex 4, France
pdss@isae.fr

ABSTRACT

Les outils UML applicables en conception de protocoles n'intègrent généralement aucun logiciel d'assistance à la mise en œuvre d'une méthodologie adaptée. Ce constat s'appliquait jusqu'ici au profil UML TURTLE (Timed UML and RT-LOTOS Environment) supporté par l'outil TTool. Cet article décrit l'introduction dans TTool d'un assistant méthodologique basé en particulier sur des "patterns" largement acceptés. L'article focalise la discussion sur les diagrammes d'analyse que TTool utilise pour générer automatiquement les diagrammes de conception. L'outil TTool étendu trouve une application directe dans l'ingénierie des protocoles, et notamment son enseignement. La démarche proposée peut s'appliquer à d'autres langages de modélisation et processus guidés par les cas d'utilisation.

Categories and Subject Descriptors

H.4 [**Information Systems Applications**]: Miscellaneous; D.2.11 [**Software Engineering**]: Software Architectures—*Data abstraction, Domain-specific architectures, Information hiding, Languages (e.g., description, interconnection, definition), Patterns (e.g., client/server, pipeline, blackboard)*

General Terms

Languages

Keywords

Modélisation objets, patrons, protocoles, techniques de description formelle, vérification formelle

1. INTRODUCTION

La complexité des systèmes répartis justifie le recours à des langages de modélisation supportés par des outils de vérification formelle. La définition de profils UML temps réel adossés à des méthodes formelles [?] répond à ce besoin en dotant la notation UML de l'OMG [11] 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 [3] et étendu en [4].

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 [?] ou sur des exemples d'une part, et l'outil *Open Source* TTool [9] qui supporte le profil d'autre part, reste à établir.

C'est pourquoi le présent article jette les bases du développement d'un assistant méthodologique à intégrer à TTool. A titre de comparaison, cet assistant sera débrayable par la personne qui modélise en TURTLE, à l'instar du GPS installé sur nombre de voitures. Il ne remettra par ailleurs pas en cause le fait que tous les diagrammes TURTLE ont une sémantique formelle et que l'environnement TTool est fortement, mais pas exclusivement, orienté vers les phases amont du cycle de développement et la vérification formelle d'exigences temporelles.

L'article propose de démarrer l'assistance méthodologique en phase d'analyse. Cela passe par une meilleure compréhension des trois diagrammes UML 2.1 que sont les diagrammes de cas d'utilisation (UCD), les diagrammes de vue globale (IOD) et les diagrammes de séquences (SD). En effet, si les UCD et SD sont relativement bien connus, il n'en va pas de même des IOD, en dépit de l'intérêt de ces derniers lorsqu'il s'agit de structurer (sous forme d'organigrammes) les scénarii qui documentent les cas d'utilisation. Une première contribution de l'article réside dans la proposition de patrons (patterns en anglais) qui accouplent UCD et IOD 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 et l'article décrit sa mise en oeuvre sur un protocole issu du projet européen *Mastro*.

L'article est structuré de la manière suivante. La section 2 passe en revue des travaux du domaine. La section 3 présente le profil TURTLE en ciblant les diagrammes d'analyse utilisés dans la suite de l'article. La section 4 propose des patterns accouplant UCD et IOD. La section 5 spécialise la proposition vers les protocoles. La section 6 présente une étude de cas réalisée avec la première version de l'outil TTool étendue. La section 7 conclut l'article et annonce les travaux à venir.

NOTERE'08 June 23-27, 2008, Lyon, France

2. TRAVAUX CONNEXES

2.1 Patterns prouvés

Dans le sillage de l'ouvrage fondateur de Gamma [7], de nombreux auteurs ont proposé des patterns d'analyse ou de conception dédiés à un domaine d'application particulier. Ainsi, Douglass [6] et Rising [12] ont respectivement adressé les systèmes temps réel et le logiciel de communication. Aujourd'hui, l'idée de partager sous forme de pattern des artéfacts largement adoptés par une communauté de praticiens, ne se suffit plus à elle-même. Plusieurs auteurs expriment le besoin de formaliser les patterns et proposent d'adosser un travail sur les patterns à une méthode formelle. Ainsi, [8] associe patterns et logique temporelle. [10] intègre les patterns dans une démarche de "correct par construction" à base du langage B.

2.2 Introduction d'un assistant dans un outil

3. LE PROFIL UML TURTLE

3.1 Vue d'ensemble du profil

Comme le suggère l'acronyme TURTLE (Timed UML and RT-LOTOS Environment), le profil UML temps réel TURTLE est adossé à l'algèbre de processus temporisée RT-LOTOS [5]. Sa sémantique est donnée par traduction vers RT-LOTOS. L'association à l'outil TTool (*TURTLE Toolkit* supportant le profil [9]) de l'outil de vérification formelle RTL (supportant RT-LOTOS) fait de TURTLE un langage de modélisation particulièrement adapté à la vérification d'exigences temporelles. Un générateur de code Java ouvre des perspectives vers le prototypage d'applications réparties.
Supportant les diagrammes d'exigences étendus par des chronogrammes aptes à représenter des exigences temporelles, TTool permet de traiter les exigences avant d'entamer la phase d'analyse. Celle-ci démarre classiquement par un diagramme de cas d'utilisation qui définit le périmètre du système à modéliser, isole les acteurs et identifie les grandes fonctionnalités que le système doit offrir. Les cas d'utilisation représentatifs de ces fonctionnalités sont documentés par des scenarii représentés à l'aide de diagrammes de séquences[1] (voir la Figure 8 pour un exemple de SD). Là interviennent les IOD[2] qui permettent de structurer les SD sous forme d'organigramme. Généralement, une première famille de scenarii est réalisé fin de mettre en évidence les interactions acteurs-système. Dans un deuxième temps, une itération méthodologique supplémentaire permet d'éclater les premier scenarii et de faire apparaître la structure interne du système. Cette analyse fonctionnelle assistée au besoin d'une recherche des objets - par exemple par la traditionnelle méthode des *mots dans le texte* - débouche ensuite sur une conception objets.
La définition de l'architecture statique du système repose sur un diagramme de classes/objets TURTLE qui étend ceux d'UML 2.1 pour formaliser le type de relation entre objets (parallélisme, synchronisation, préemption) et autoriser ces mêmes objets à communiquer par rendez-vous. Les comportements des objets sont exprimés par des diagrammes d'activités qui, outre les actions de synchronisation par rendez-vous, offrent des opérateurs temporels aptes à exprimer des délais fixes et des intervalles temporels, mais aussi de limiter temporellement les offres de rendez-vous. La définition de diagrammes d'activités rend le modèle TURTLE exécutable. Une dernière étape consiste enfin à effectuer un déploiement du système à l'aide de diagrammes de déploiement TURTLE.
Le principe de TURTLE est de fournir une sémantique formelle à des ensembles de diagrammes TURTLE. Ainsi, une analyse TURTLE (i.e. un IOD principal et tout ce qui est lié depuis cet IOD), une conception TURTLE (i.e. un diagramme de classes et un ensemble de diagrammes d'activités) ou enfin un diagramme de déploiement possèdent une sémantique formellement définie en (RT-)LOTOS (le RT s'applique si le système utilise des opérateurs temporels). L'outil TTool sait effectuer cette traduction automatique, et invoquer les outils de vérification formelle sous-jacents (RTL [2], CADP [1]). Une interface conviviale permet ainsi de confronter les diagrammes aux exigences temporelles.

La suite de cet article focalise la discussion sur les deux premiers diagrammes d'analyse, à savoir les diagrammes de cas d'utilisation et les IOD.

3.2 Diagrammes de cas d'utilisation

Un diagramme de cas d'utilisation met en évidence le système à modéliser sous la forme de fonctions et son interface avec l'environnement extérieur représenté par des acteurs (cf. la Figure 1).

- Une frontière représentée sous la forme d'un rectangle délimite clairement les acteurs des fonctions.
- Les fonctions peuvent être mises en relation d'inclusion (<< *include* >>), d'extension (<< *extend* >>, l'extension est en général définie par rapport à un point d'extension), ou de spécialisation (connecteur UML pour modéliser l'héritage). L'inclusion permet de modéliser des sous-fonctions d'une fonction ; l'extension permet de modéliser les fonctions optionnelles d'une fonction de plus haut niveau.
- Les acteurs peuvent être mis en relation avec des fonctions avec lesquelles ils inter-agissent.

3.3 IOD

Un diagramme IOD s'apparente à un diagramme d'activités UML (un organigramme, en fait) dont les actions seraient remplacées par des références à des diagrammes de séquences ou à d'autres IOD. Notons que depuis un diagramme de séquences, il n'est pas possible de référencer un IOD.
Ces organigrammes sont constitués de noeuds (choix, séquence, parallélisme). TURTLE ajoute un noeud de préemption à ces IODs afin d'autoriser la préemption d'une branche d'un IOD par une branche d'un IOD. Un exemple d'IOD est donné à la Figure 4. Cet IOD met en évidence des références vers d'autres IODs (par exemple, *Connection setup* est une référence vers un IOD), des choix entre IODs (le losange), et un opérateur de préemption (la barre horizontale avec le symbole [> à son extrémité droite) qui permet de spécifier que les IODs *Connection release* ou *Connection broken* peuvent à tout moment interrompre la branche partant de la gauche de l'opérateur de préemption.
Finalement, les IODs, bien que relativement méconnus et donc peu usités, ont la bonne propriétés de pouvoir structurer entre eux des scenarii, et donc de réduire par là même la

1. Sequence Diagram, noté SD dans la suite du texte
2. Interaction Overview Diagram, ou IOD dans la suite du texte

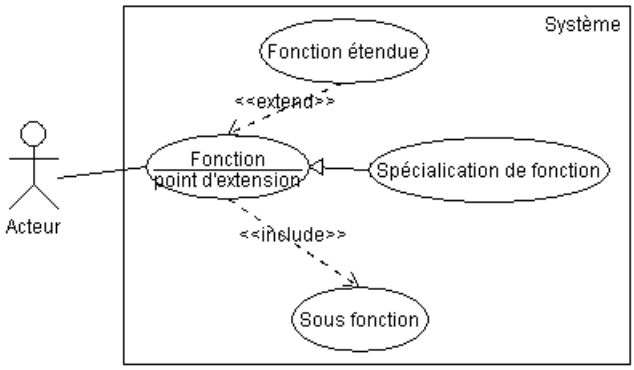

Fig. 1 – ***Exemple d'un diagramme de cas d'utilisation***

complexité de ces scenarii. En effet, ces derniers ont comme avantage d'être un guide visuel d'échanges d'informations entre instances. Les IODs ajoutent la possibilité visuelle de pouvoir structurer des scenarios, en omettant la visualisation directe de ces "détails" d'échanges d'information entre instances. Malheureusement, En UML, certains opérateurs d'IODs ont la même sémantique que ceux d'opérateurs de scénario, ce qui implique que le modeleur a la possibilité de faire figurer certaines informations soit au niveau de l'IOD, soit au niveau de scénario, ce qui casse un peu ce modèle dual structure / comportement. En TURTLE, nous avons limités certains de ces possibilités en rendant obligatoire l'emploi d'un opérateur d'IOD ou de Sd pour une sémantique données (par exemple, les références des scénarios ont été supprimées des scénarios, mais sont conservées au niveau des IODs). Nous avons par contre gardé l'opérateur d'alternative aux deux niveaux de modélisation (IOD et SD).

4. UNE MÉTHODOLOGIE BASÉE SUR LES PATRONS

4.1 Approche générale

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 :

- A 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 est associée au moins une référence vers un IOD dans l'IODh qui porte le nom de cette fonction.
- 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), ajouter des fonctions optionnelles, etc. 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 représentées uniquement au travers de références vers des sous-IODs (ou SDs) de IODh.
- Dans l'IODh, les références vers des IODs correspondant à des fonctions de haut niveau doivent être obligatoirement complété i.e. un diagramme IOD doit être fourni. En ce qui concerne les autres références à des IODs de l'IODh, cette procédure est facultative, c'est à dire qu'ils peuvent être laissés vides.
- L'IODh ne peut être modifié par l'utilisateur, sauf au niveau des noeuds 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.

Ces patrons ont vocation à être utilisés comme suit (voir la Figure 2) :

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.).
2. L'utilisateur peut éventuellement modifier le diagramme de cas d'utilisation selon les règles listées plus haut. 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* $>>$ avec les 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 lui : compléter cet IOD ou bien le laisser vide. Le fait de compléter un IOD veut dire lui donner une sémantique au travers des enchaînements d'autres IODs ou de scénarios. Les fonctions ajoutées au diagramme de cas d'utilisation doivent se retrouver au niveau des IODs référencées par l'IODh, soit sous la forme d'un scénario, soit sous la forme d'un IOD. Notons enfin que la structure générale de l'IODh ne peut pas être modifiée par l'utilisateur.
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 afin notamment de débarrasser l'IODh de références vers des IODs vides, et d'éliminer un éventuel paramétrage des diagrammes de séquences référencés par ces IODs (voir plus loin).
5. Nos patrons garantissent *par construction* un certain

nombre de propriétés discutées par la suite, et montrées sur des exemples. Bien entendu, des vérifications formelles supplémentaires peuvent être effectuées par génération de code (RT-)LOTOS.

6. La dernière étape est la génération automatique d'une conception à partir de l'analyse effectuée. Le principe de cette génération automatique n'est pas traitée dans cet article mais peut-être consultée dans [4]. Rappelons que cette génération automatique préserve les propriétés du modèle d'analyse si et seulement si ce dernier est *implémentable*.

4.2 Sémantique

Les patrons que nous proposons possèdent une sémantique formellement définie par traduction en (RT-)LOTOS. Cette traduction est effectuée en trois étapes :

1. **Filtrage**
   L'IODh est "filtré" selon la procédure suivante. Tout d'abord, les références vers des IODs qui ont été laissées vides sont éliminées de l'IODh, c'est à dire que les connecteurs menant et partant de ces références sont supprimées, et les références elles-même sont supprimées de l'IODh. L'UCD est utilisé par cet algorithme de filtrage pour savoir si une référence vers un IOD correspondant à une fonction de haut niveau - ou pas. Si une référence correspondant à une fonction de haut niveau a été laissée vide, alors la traduction ne peut avoir lieu.
   Une autre procédure a lieu à cette étape, que nous ne détaillons dans cet article. Elle concerne le fait que les diagrammes de séquences peuvent être paramétrés. Plus exactement, une instance par scenario peut-être paramétrée, c'est à dire qu'elle correspond en fait à n instances, une instance étant un élément communiquant des scénarios. Cette paramétrisation est particulièrement utile dans le cas de systèmes de diffusion de données vers n clients. L'algorithme de filtrage prend en entrée pour chaque instance paramétrée une valeur d'instantiation et utilise ces valeurs pour instancier autant de fois que nécessaire chaque instance, et dupliquer les messages arrivant et partant de ces instances paramétrées.
2. **Génération d'une spécification TIF**
   TIF - TURTLE Intermediate Format - est un format intermediaire formel utilisé par TTool et qui sert de base à la génération de code RT-LOTOS, LOTOS, UPPAAL et Java. Notre approche confère ainsi à tout ensemble IODs, Diagrammes de séquences une sémantique formelle [4].
3. **Génération d'une spécification (RT-)LOTOS**
   Cette génération est effectuée depuis le format intermédiaire en TIF.

Ces traduction formelle confère par construction à tout modèle basé sur nos patrons la propriété suivante :

- **Propriété 1**
  Pour toutes les instances des scénarios qui effectuent au moins une action dans chaque chemin possible des sous-iods de l'IODh, et si ce chemin est unique (pas de parallélisme), et si les références entre IODs ne comportent de références vers des IODs référencés par l'IODh, alors la séquence entre des actions effectuées dans des sous IODs en séquence, et pour chaque instance, est garantie par construction.
  Par exemple, si l'on considère le patron du mode connecté - présenté dans la section suivante -, si l'instance cliente effectue des actions dans tous les chemins possibles des sous-iods, alors le patron garantit par construction que les données ne peuvent être reçues - 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*.

La preuve de cette propriété repose notamment sur la sémantique des noeuds des IODs (choix, séquence, premption), le parallélisme étant exclu par hypothèse, et sur les possibilité de référencement de scénario et d'IOD. En effet, de façon assez informelle, si l'on considère deux IODs $IOD1$ et $IOD2$ référencés dans l'$IODh$, qui sont en séquence l'un après l'autre. Si l'on considère une instance $instance1$ d'un scénario qui effectue au moins une action $action1$ dans tous les chemins possibles de $IOD1$ et $IOD2$. Dans chaque chemin possible de l'$IOD1$, $instance1$ fait au moins une action $action1$, et en raison de l'absence de parallélisme, et l'absence de référencement de $IOD2$ depuis $IOD1$ (hypothèse), lorsque $instance1$ fera une action $action2$ dans $IOD2$, alors $instance1$ ne possédera plus aucune action en cours dans IOD1. Donc $action2$ ne pourra être effectuée qu'après toute action $action1$.
Notons que la propriété précédente reste valable même si les gardes des choix de l'IOD du patron sont renseignées (et non laissées vides i.e. totalement non déterministes).
Nous avons coutume de comparer nos patrons à un GPS de voiture. Les règles d'utilisation des patrons, ainsi que la méthodologie générale présentées ci-dessus permettent de donner un guide au modeleur. Ce dernier a la possibilité aussi, s'il ne modifie pas le patron, d'obtenir des propriétés de son modèle qui sont vraies par construction. A la différence d'un GPS de voiture qui peut recalculer en temps réel un chemin à suivre si la personne dévie du trajet recommandé, notre approche ne permet pas de re-calculer des propriétés vraies par construction si le modeleur n'applique pas les règles citées. Par contre, la sémantique des modèles d'analyse TURTLE reste vraie - si la personne se conforme aux règles syntaxique TURLE - : l'utilisateur pourra toujours effectuer des vérifications formelles a posteriori sur son modèle.

4.3 Outillage supports aux patrons

L'outil TTool [9] permet la sauvegarde et l'importation de librairies / modèles contenant des modélisations TTool. C'est sous cette forme que les patrons présentés à la section suivante peuvent être chargés dans TTool puis utilisés. Ainsi, un utilisateur, une fois TTool lancé, pourra utiliser un patron comme suit :

1. il charge le patron correspondant ;
2. il complète le diagramme de cas d'utilisation du patron, comme expliqué dans la méthodologie exposée lors de la section précédente ;
3. il complète les différents IODs obligatoires, et éventuellement ceux facultatifs, en ajoutant à la modélisation TTool les diagrammes globaux d'interactions et de séquences nécessaires ;
4. TTool n'est à ce jour pas capable de vérifier que la personne a respecté le patron, et a notamment fourni les diagrammes obligatoires. Par contre, TTool vérifie que la syntaxe des diagrammes est correcte, par rapport au méta-modèle TURTLE. Une fois cette vérification syntaxique effectuée, TTool construit une modélisation in-

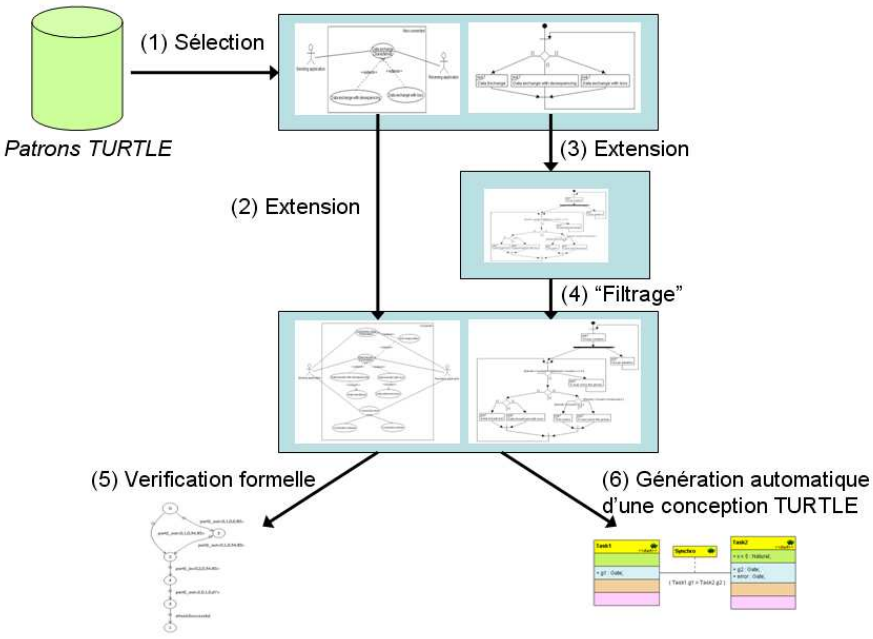

FIG. 2 – *Méthodologie générale d'utilisation des patrons*

terne en format TIF qui tient compte des diagrammes fournis par l'utilisateur, et qui tient compte du patron. A partir de ce format interne, l'utilisateur a deux possibilités :

- Effectuer de la vérification formelle, par génération de code LOTOS ou RT-LOTOS ; Cette vérification formelle peut-être réalisée directement depuis TTool sans aucune connaissance de ces langages formels, ou des outils de vérification associés.
- Générer automatiquement une premier conception, qui tient compte bien entendu de la modélisation de l'utilisateur, et du patron.

Ainsi, à ce jour, l'approche générale précédemment présentés n'est que partiellement implémentée. Deux points manquent plus particulièrement:

- la vérification que l'utilisateur s'est bien inscrit dans le patron qu'il a chargé dans TTool ;
- la gestion du paramétrage n'est pas encore possible. Ainsi, actuellement, une solution consiste à utiliser un modèle comportant un nombre d'utilisateur fixe. Notons que le problème du paramétrage n'est pas un problème lié à l'approche par patrons en général, mais un problème lié à la sémantique du profil TURTLE. Par exemple, les diagrammes de conception TURTLE ne sont pas paramétrés. On peut utiliser la notion d'objets, ou instancier dynamiquement des objets dans les conceptions TURTLE, mais pas paramétrer les classes, par exemple, exprimer que n clients sont connectés à m serveurs.

5. PATRONS TURTLE POUR LES PROTOCOLES

Dans cette section, l'idée n'est pas de faire un catalogue des patrons que nous proposons sous TTool, mais plutôt de mettre en évidence l'intérêt de ces patrons comme guide méthodologique formel de conception de protocoles et de systèmes distribués.

5.1 Patron pour les protocoles en mode connecté

Ce patron vise l'analyse des protocoles comportant une phase de connexion avant tout échange de données. Cette phase d'échange de données a lieu jusqu'à la fermeture - voulue ou non - de la connexion.
Le patron est construit comme suit :

- Son diagramme de cas d'utilisation (voir Figure 3) met en évidence trois principales fonctions :
  1. l'établissement de la connexion, avec négociation de la qualité de service (option) ;
  2. l'échange de données, 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*) ;
  3. la fin de la connexion qui peut être soit volontaire, soit due à une rupture de connexion. L'utilisation d'une relation héritage entre cas d'utilisation signifie qu'un des deux IODs correspondant à la terminaison de la connexion devra ultérieurement être renseigné i.e. être non vide (cf. les propriétés des patrons).
- Son diagramme global d'interactions comporte lui aussi trois principales parties :
  1. la partie du haut met en évidence le fait que l'ouverture de connexion se fait avec ou non négociation de service ;
  2. 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 ;

3. enfin, la partie en bas à droite met en évidence qu'une fois la connexion établie, elle peut être rompue ou relâchée non volontairement. 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 [4].

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'IODs et de SDs que si les fonctions optionnelles les incluant sont-elles même renseignées. Aussi bien nos patrons que TTool ne font aucune vérification en ce sens, et rien n'interdit d'ailleurs d'aller mettre dans un l'IOD appelé *Connection setup* un échange de données: les noms des fonctions de l'UCD et des IODs ne sont que des assistants à l'analyse qui laissent libre cours à l'utilisateur, à l'instar d'un GPS de voiture dont les indications sont débrayables.
Une deuxième remarque sur ce patron est relative à la *propriété 1* 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 IODs obligatoires, alors, par construction, l'on garantit que les actions effectuées dans *Data Exchange*, dans *Connection broken* et dans *Connection release* le sont forcément après les actions effectuées par *Server* dans *Connection setup*.

5.2 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. Ce patron est ainsi paramétré, nous expliquons ce paramétrage par la suite.
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 d'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 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 déséquencements peuvent apparaître.
- D'un diagramme global d'interactions (cf. Figure 6). Ce dernier met en évidence la création du groupe. Une fois le groupe créé, des ajouts ou retrait de clients peuvent intervenir sur le groupe. Aussi, des données peuvent être émises au sein du groupe. Enfin, la diffusion de données et l'ajout / le retrait de clients au sein du groupe cesse lorsque le groupe est supprimé (opérateur de préemption).

Il sera possible de dire qu'un utilisateur ne peut se joindre au groupe que si le nombre maximal d'utilisateur du groupe n'est pas atteint par un simple ajout d'une garde sur le choix précédant l'ajout des utilisateurs. C'est d'ailleurs ce que nous avons fait dans l'étude de cas à suivre, qui permet en outre de mieux expliciter ce patron.

6. ETUDE DE CAS : UN PROTOCOLE MULTICAST

6.1 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 simplifiée d'un protocole de diffusion de données multimédia proposé dans le cadre d'un projet Européen (projet Maestro).
Dans ce protocole, un émetteur de flux multimédia demande à un serveur principal la création d'un groupe de diffusion auxquels des utilisateurs peuvent par la suite se joindre en contactant le dit serveur. Lorsqu'au moins un utilisateur est présent dans le groupe, les données sont régulièrement émises vers un satellite de diffusion multi-faisceaux 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 enfin que les données échangées entre les clients du groupe et le serveur sont effectuées par une voie retour terrestre.

6.2 Analyse basée sur les patrons

Le patron que nous avons utilisé dans cet exemple est celui correspondant aux systèmes multicast. Nous avons fourni les comportement des IODs suivants:

- *Création d'un groupe.* Cette création est réalisée lorsque un serveur de données multimédia désire émettre des données sur le système satellite
- *Ajout d'un utilisateur à un groupe.* Nous nous sommes limités, pour des raisons de complexité, à trois utilisateurs situés dans deux faisceaux différents
- *Retrait d'un utilisateur d'un groupe.*
- *Émission de données.* Le serveur multimédia met à jour les données à émettre au niveau du système satellite (*Gateway*). Ces données ne sont réellement transmises que si au moins un utilisateur s'est ajouté au groupe. Si tel est le cas, les données sont envoyées, accompagnées de FEC[3], à bord du satellite, avec des informations statiques de routage. Les données et FEC sont alors routés vers les faisceaux correspondants 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 utilisateur se sont désinscrits. L'émission de données se termine par l'envoi d'acquittements des clients vers le serveur de groupe.
- *Destruction d'un groupe.* Le serveur de groupe peut décider de fermer le groupe, soit pour une raison d'erreur, soit parce que le serveur de données multimédia a terminé sa diffusion. Les clients sont informés de cette fermeture, mais peuvent continuer à recevoir les dernières données émises par le serveur multimédia.

Plus précisément, et à titre d'exemple basique, nous fournissons le contenu de l'IOD *A client joins the group* à la Figure 7. Cet IOD comporte lui-même trois références vers des scénarios. Ces scénarios permettent à chaque client de s'ajouter au groupe. Par exemple, la Figure8 montre comment le *client1* peut s'ajouter au groupe . Certains sous IODs et scénarios sont bien entendu plus complexes que ceux fournis dans ces deux figures.

6.3 Vérification formelle

3. Forward Error Correcting

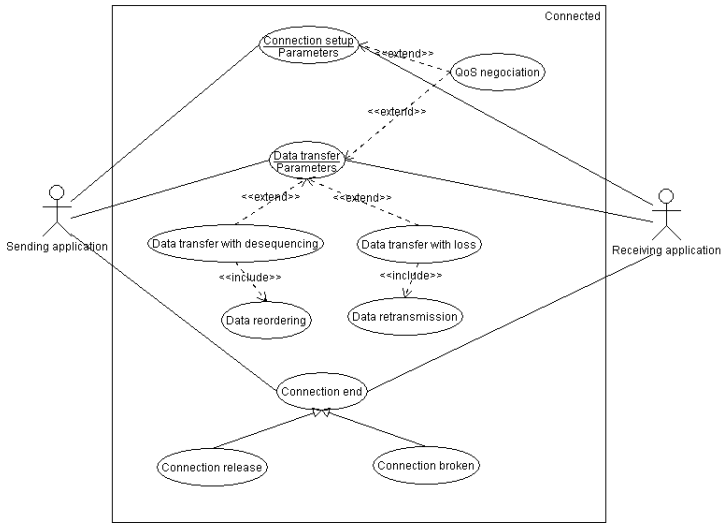

Fig. 3 – *Patron pour les protocoles en mode connecté : UCD*

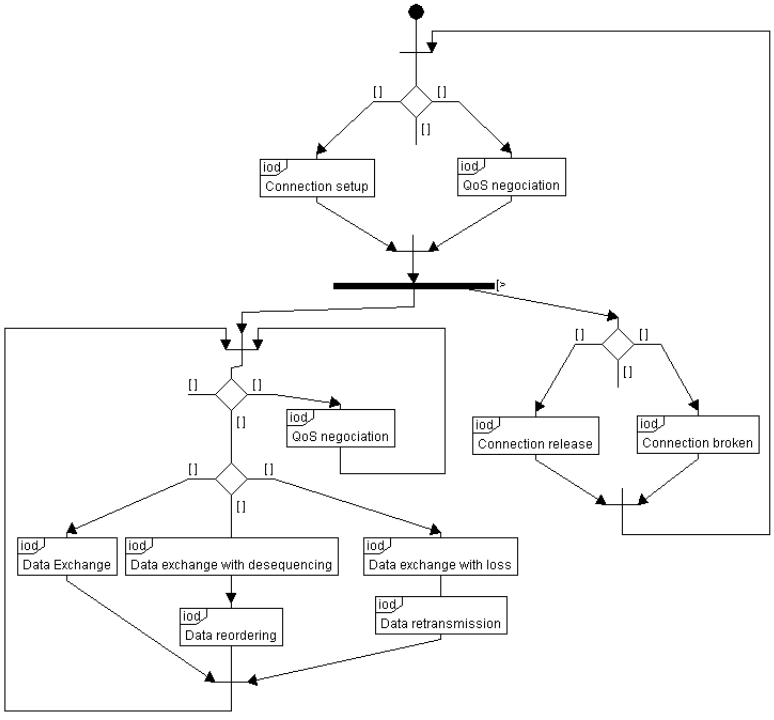

Fig. 4 – *Patron pour les protocoles en mode connecté : IODh*

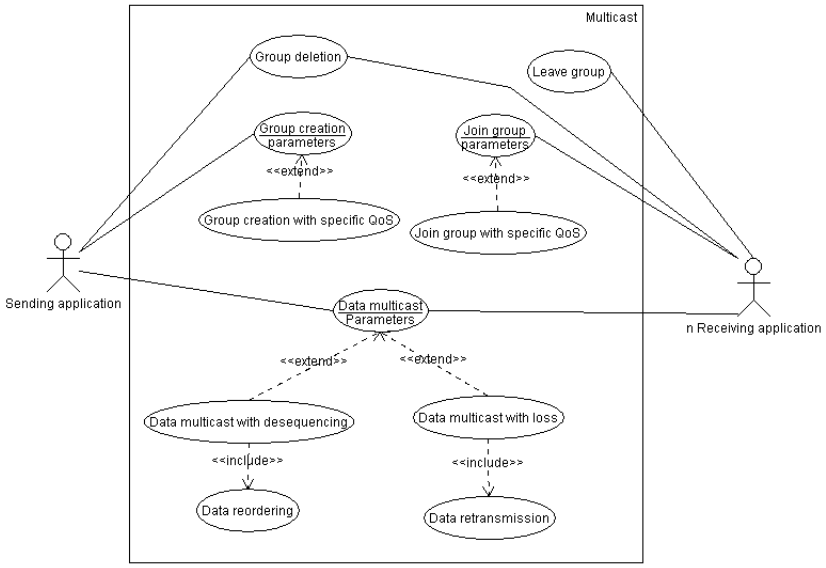

Fig. 5 – *Patron pour les protocoles de diffusion : diagramme de cas d'utilisation*

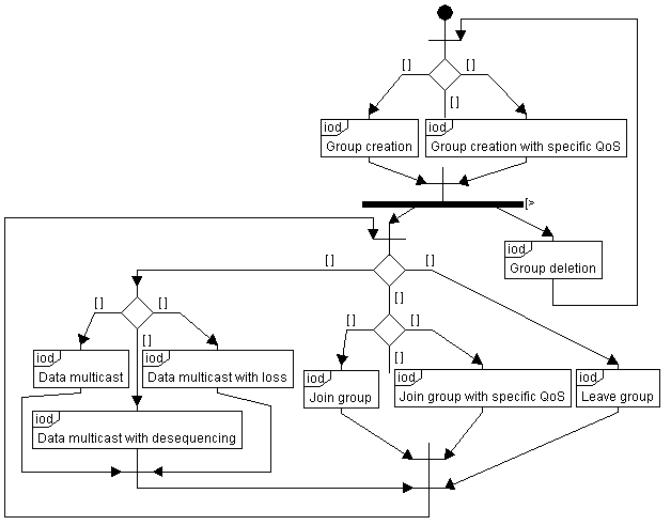

Fig. 6 – *Patron pour les protocoles de diffusion : diagramme global d'interactions (IOD)*

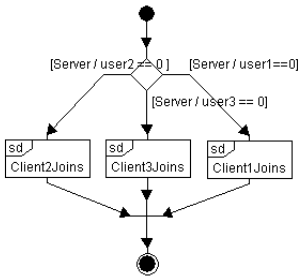

Fig. 7 – *IOD "A client joins the group"*

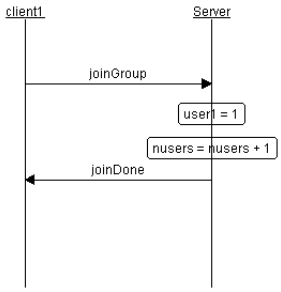

FIG. 8 – *Diagramme de séquence UML "Client1Joins"*

Après avoir rempli le patron tel que décrit précédemment, nous avons utilisé le générateur automatique de code LOTOS de TTool et l'outil CADP afin de construire le graphe d'accessibilité de notre application. Ce dernier comporte environ 6 millions d'états et 25 millions de transitions.
Nous avons aussi réalisé le graphe d'accessibilité de cette application dépourvu du rebouclage au niveau de l'IOD principal (i.e. le liens entre la destruction du groupe, et sa création). La taille plus modeste du graphe résultant nous a permis de réaliser des preuves de propriétés. A titre d'exemple, la Figure 9 représente le graphe minimisé 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, l'utilisateur reçoit un dernier paquet de données non acquitté.
Finalement, nous avons pu prouver, que:

- un utilisateur ne peut 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.

6.4 Génération d'une conception

TTool permet la génération d'une conception équivalente à l'analyse, aux problème de non-implémentabilité près [4]. Pour notre système, nous avons ainsi généré une conception qui comprend 8 classes qui correspondent aux 8 instances des scénarii décrites lors de l'analyse (diagrammes de séquences). Le comportant de ces classes est calculé à partir des IODs et scénarios de l'analyse.

6.5 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 statique, 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 a priori la pertinence de l'approche. Toutefois, nous avons rencontré un certains nombre de limitations qui sont discutées par la suite
Tout d'abord, le problème de la non-implémentabilité, qui a été abordé auparavant [4] dans le cadre de l'analyse TURTLE. Les patrons que nous fournissons ne garantissent en rien cette non-implémentabilité. Cette dernière provenant de choix dits *distribués*, nous nous sommes efforcés lors de l'utilisation de notre patron, et dans notre étude de cas, de ne pas utiliser de tels choix. Cette contrainte est apparue comme assez forte, et nous a amené 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 30000 sans ce rebouclage. Ce problème est bien entendu plus général que celui de l'utilisation des patrons, et est plutôt inhérent à la vérification de systèmes décrits avec des LTS.
Le troisième limitation concerne le paramétrage. Dans le cadre de systèmes distribués, il est utile de pouvoir spécifier un nombre n d'utilisateurs du système. Malheureusement, l'approche UML ne comporte pas de scénarios paramétrables, au sens où le nombre d'instance de ces scénarios pourrait être paramétré. Cela fait partie de nos travaux futurs que d'ajouter ce paramétrage à la fois au niveau de l'analyse TURTLE, et au niveau de nos patrons.

7. CONCLUSIONS

Faire ses premiers pas en modélisation de protocoles dans un environnement UML est une tâche déroutante pour qui se confronte aux treize diagrammes supportés par la norme 2.1 et à une simple description écrite d'un processus méthodologique. L'idée défendue dans cet article est d'inclure aux ateliers logiciels UML autorisant la validation de modèles de conception, un assistant méthodologique qui serait à la modélisation UML outillée ce qu'est un GPS à la conduite automobile. Sans perte de généralité, cette idée est mise en oeuvre sur le profil UML temps réel TURTLE supporté par l'outil TTool.
L'article propose tout d'abord d'introduire des patrons alliant diagrammes de cas d'utilisation et diagrammes d'aperçu général, respectivement dédiés à l'identification des fonctionnalités du système et à la structuration de scénarii exprimés par des diagrammes de séquences. 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). Enfin, ce cadre méthodologique formel et sa spécialisation aux protocoles sont implantés dans TTool. Cet outil

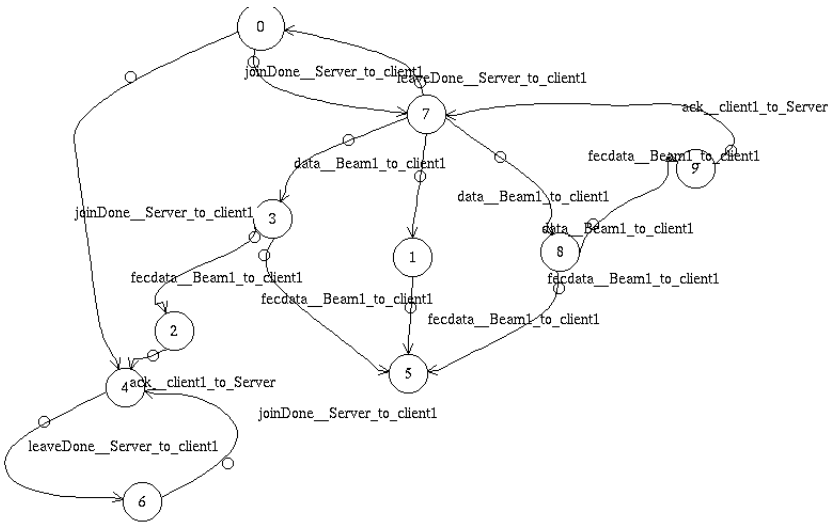

FIG. 9 – ***Graphe d'accessibilité minimisé aux seules actions de client1***

a été utilisé pour traiter le cas d'un protocole de diffusion de données multimédia.
L'implantation dans TTool utilise le gestionnaire de bibliothèque de l'outil pour gérer les patrons. Le travail d'implantation se poursuivra par l'implantation de l'algorithme de filtrage, et par la possibilité de paramétrer les instances des scenarii.
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 réflexion plus générale sur la possibilité d'offrir, avec certaines restriction, plus de propriétés satisfaites par construction.

8. REFERENCES

[1] The cadp toolkit. http://www.inrialpes.fr/vasy/cadp.

[2] The rtl toolkit. http://www.laas.fr/RT-LOTOS/index.html.en.

[3] L. Apvrille, J.-P. Courtiat, C. Lohr, and P. de Saqui-Sannes. TURTLE: A Real-Time UML Profile Supported by a Formal Validation Too lkit. In *IEEE transactions on Software Engineering*, volume 30, pages 473–487, Jul 2004.

[4] L. Apvrille, P. de Saqui-Sannes, and F. Khendek. Synthèse d'une conception UML temps-réel à partir de diagramme de séquences. In *Colloque Francophone sur l'Ingéniérie des Protocoles*, Bordeaux, France, mar 2005.

[5] J.-P. Courtiat, C. Santos, C. Lohr, and B. Outtaj. Experience with RT-LOTOS, a Temporal Extension of the LOTOS Formal Description Technique. In *Computer Communications*, volume 23, pages 1104–123, 2000.

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

[7] E. Gamma. *Design Patterns.* Addison wesley, 1995.

[8] S. Konrad and B. H.C.Cheng. Real-time specification patterns. In *Proceedings of the 27th international conference on Software engineering*, pages 372–381, St. Louis, MO, USA, May 2005.

[9] LabSoc. The TURTLE Toolkit. In *http://labsoc.comelec.enst.fr/turtle/ttool.html.*

[10] T. Lecomte, D. Cansell, and D. Méry. Patrons de conception prouvés. In *Journées NEPTUNE*, May 2007.

[11] OMG. UML 2.0 Superstructure Specification. In *http://www.omg.org/docs/ptc/03-08-02.pdf*, Geneva, 2003.

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