Conception basée modèle des systèmes temps réel et distribués

Pierre de Saqui-Sannes

Habilitation à Diriger les Recherches

Institut National Polytechnique de Toulouse

Conception basée modèle des systèmes temps réel et distribués

Pierre de Saqui-Sannes
Enseignant/Chercheur à l'ENSICA
Chercheur au LAAS-CNRS

Habilitation soutenue le 7 juillet 2005 devant le jury :

Jean-Pierre Courtiat	Correspondant
Richard Castanet	Rapporteur
Michel Diaz	Examinateur
Rachida Dssouli	Rapporteur
Patrick Hébrard	Invité
Elie Najm	Rapporteur
Patrick Sallé	Président

Avant-Propos

Je tiens à remercier ici Messieurs Sintès, Fréson, Michaut et Darrenougué, directeurs successifs de l'ENSICA pour m'avoir accueilli au sein de cette Ecole et renouvelé leur confiance à maintes occasions. Mes remerciements vont de pair à Messieurs Estève, Costes, Laprie et Ghalab, directeurs successifs du LAAS-CNRS qui m'ont accueilli en tant que doctorant puis intégré au corps des enseignants-chercheurs de leur laboratoire.

Le partenariat de recherche qui lie l'ENSICA et le LAAS-CNRS trouve une déclinaison naturelle dans les relations privilégiées qu'entretiennent le Département Mathématiques appliquées et Informatique de l'ENSICA et le groupe Outils et Logiciels pour la Communication du LAAS-CNRS. Je remercie Michel Diaz, responsable du groupe OLC du LAAS-CNRS jusqu'en décembre 2003 et Patrick Sénac, chef du département DMI de l'ENSICA, pour m'avoir accueilli et offert un cadre privilégié pour développer mon activité de recherche. Ces remerciements s'étendent à Jean-Pierre Courtiat qui a pris depuis peu les rênes du groupe OLC..

Je tiens à exprimer ma gratitude à Madame Rachida Dssouli et Messieurs Richard Castanet, Jean-Pierre Courtiat, Michel Diaz, Patrick Hébrard, Elie Najm et Patrick Sallé, qui m'ont honoré en acceptant de participer à ce jury. Je remercie tout particulièrement Richard Castanet, Rachida Dssouli et Elie Najm d'avoir accepté la charge de Rapporteur et Jean-Pierre Courtiat qui au travers de son rôle de « correspondant au près de l'INPT » se porte garant de ce travail.

Je tiens à souligner combien je me sens redevable envers Jean-Pierre Courtiat qui m'a initié au monde de la recherche, puis épaulé et encouragé tout au long de ma carrière d'enseignant-chercheur. Je ne me risquerai pas à dresser ici un portrait forcément réducteur de ses qualités scientifiques et humaines mais je tiens à l'assurer de ma très sincère amitié.

Le travail présenté dans ce mémoire résulte non d'une pensée unique mais d'une alchimie de partenariats.

Merci tout d'abord aux étudiants avec lesquels j'ai eu l'occasion de travailler et en particulier Vitório, Marcelo, Rosvelter, Michel, Christophe, Ludovic, Tarek, Benjamin et Na. Puisse Ludovic Apvrille continuer à me réserver de belles surprises sur l'outillage du profil TURTLE !

Merci à mes collègues et amis du DMI de l'ENSICA que je nommerai ici dans l'ordre alphabétique: Bernard, Fabrice, Jérôme, Laurent, Patrick, René, Sébastien, Tanguy et Yves. A tous ceux qui en forment le vœux je souhaite de passer un jour l'HDR. J'exprime des vœux de même nature aux doctorants. La vie sociale

au DMI et le tissus humain qui la sous-tend font de ce département un lieu de travail hautement privilégié dans une collectivité ENSICA dont je salue bien chaleureusement tous les membres.

J'associe à ces remerciements tous les membres du groupe OLC du LAAS et tous les services qui concourent au bon fonctionnement de ce laboratoire.

Mes remerciements s'étendent à toute la communauté scientifique toulousaine et en particulier mes collègues de l'ENSEEIHT, INSA et SUPAéRO. Philippe Marthon m'a bien souvent seriné « Et l'HDR ? ».Puisse-t-il mainteant ne pas me demander si je me sens « vulnérable » (référence au jeu de bridge).

Le parcours de recherche retracé en ce mémoire ne s'expliquerait pas sans mention de l'année post-doctorale passée à l'Université de Montréal dans le groupe de Grégor von Bochmann que j'associe sans hésitation à la liste des personnes qui ont contribué à forger mon parcours professionnel et personnel. Un réseau tricotté-serré s'est créé à cette occasion et je remercie toutes celles et tous ceux qui me témoignent encore aujourd'hui leur indéfectible amitié transatlantique.

Je contiendrai ici mes remerciements et salutations au giron professionnel pour témoigner en direct mon affection à d'autres cercles familiaux et amicaux.

Chapitre 1

Positionnement

Cible d'application privilégiée des travaux de recherche présentés dans ce mémoire, les systèmes temps réel et distribués justifient de par leur complexité le recours à des langages de modélisation et à des techniques de validation a priori permettant de détecter les erreurs de conception au plus tôt dans le cycle de développement. Au long de ce chapitre, nous allons passer en revue les langages de modélisation et les techniques d'ingénierie basée modèle que nous avons effectivement pratiquées. Nous n'avons donc pas la prétention de couvrir ici l'intégralité du cycle de développement ni de rechercher l'exhaustivité vis-à-vis des techniques de modélisation rencontrées dans la littérature.

Ce chapitre est structuré de la manière suivante. Le paragraphe 1 identifie les activités de l'ingénierie basée modèle qui seront abordées dans la suite du document. Le paragraphe 2 poursuit avec un panorama des techniques étudiées au cours de notre activité de recherche, sans prendre parti pour tel ou tel langage de modélisation. Le discours se spécialise ensuite au paragraphe 3 vers le langage UML [OMG 03].

1 Positionnement dans le cycle de développement

Inspiré de [BRU 03], le cycle de vie décrit par la figure 1 met en évidence le rôle nourricier des exigences vis-à-vis des grandes étapes que sont l'analyse, la conception, le codage, le test et la maintenance d'un système à logiciel prépondérant. La notion d'« exigences » utilisée en génie logiciel englobe les « propriétés » et autre notion de « service » d'usage plus ancien dans les communautés « temps réel » et « ingénierie des protocoles ».

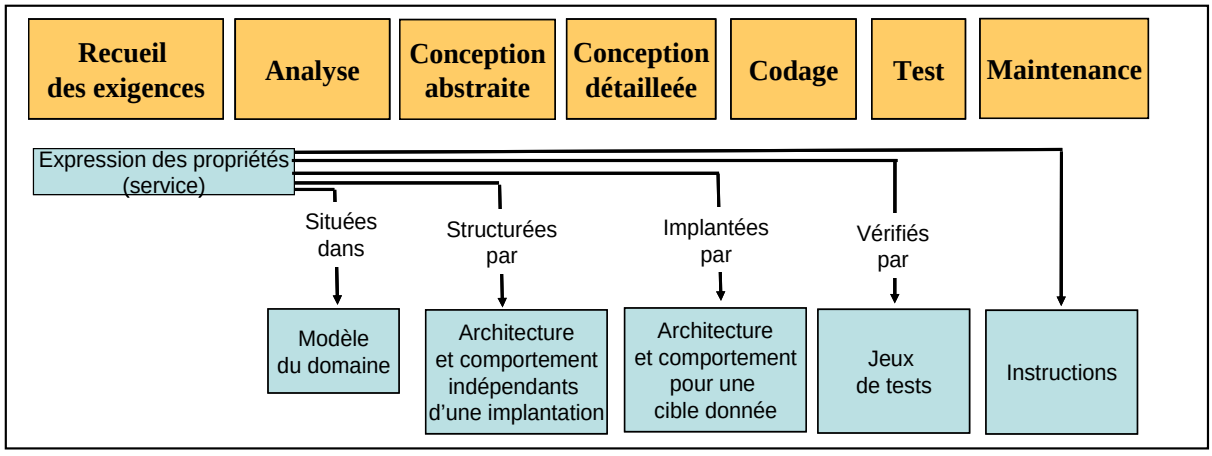


Figure 1. Cycle de vie et productions associées

La palette des techniques basées modèle disponibles au long des étapes identifiées par la figure 1 est trop large pour prétendre en dresser ici un panorama « complet ». Aussi

centrons-nous immédiatement la discussion sur les cinq activités identifiées par la figure 2 et qui feront l'objet de développements techniques dans la suite de ce mémoire. Cette même figure identifie la place de ces activités en génie logiciel au sens large et, plus restrictivement, en ingénierie des protocoles.

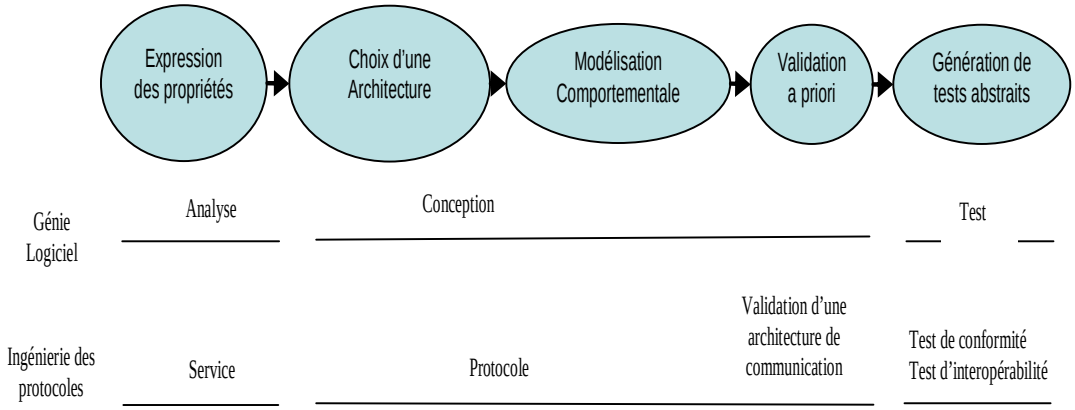


Figure 2. Activités basées modèle abordées dans ce mémoire

Le prochain paragraphe accorde un sous paragraphe à chacune des activités identifiées ci-dessous.

2 Techniques d'ingénierie basées modèle

2.1. Expression des exigences

2.1.1. Introduction

Nous opérons ici une distinction entre exigences exécutables et non exécutables. En ingénierie des protocoles, nous associerons les exigences exécutables à l'idée d'une modélisation sous forme de scénarios. Le recours aux scénarios [SOM 96] rend l'expression d'exigences abordable à une large classe de concepteurs de systèmes temps réel et distribués. De plus, l'utilisation de scénarios permet de mettre en œuvre une modélisation incrémentale dans laquelle on traite en premier lieu le fonctionnement nominal caractéristique d'un système plongé dans un environnement « idéal » (réseau sous-jacent sans perte, par exemple) pour ensuite traiter les cas dégradés (recouvrement de pertes ou de données corrompues, par exemple). Ces différents scénarios doivent être organisé et vérifiés pour détecter les incohérences intra- et inter-scénarios tant du point de vue de l'ordonnancement temporel des messages que de la cohérence des contraintes temporelles.

Selon le langage de modélisation considéré, les scénarios peuvent être utilement complétés par une description monolithique du service. Un exemple en est donné en [CS 95]. Cet article applique la méthodologie du projet Lotosphere à un service de transfert de données. [CS 95] décrit ce service sous la forme d'un unique processus LOTOS offrant des choix entre plusieurs séquences de primitives de service et donc implicitement de scénarios. Dans un modélisation en UML 2.0, cette description monolithique prendrait la forme d'un IOD (*Interaction Overview Diagram*).

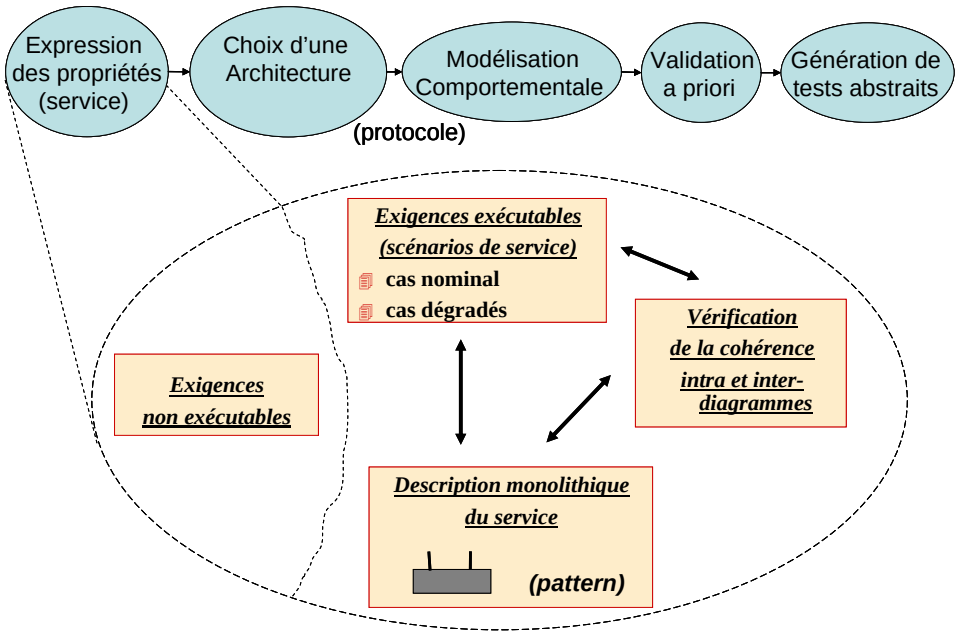


Figure 3. Phase d'analyse

2.2. Choix d'une architecture

2.1.1. Langages de modélisation

Dans la classe de systèmes que nous visons, une architecture doit permettre de représenter le caractère distribué et asynchrone du système.

Les modèles de base de type machines à états finies ou réseaux de Petri ne disposent pas en natif d'éléments structurants (découpage en modules et canaux de communication entre ces entités). En particulier, la notion de composition fait défaut aux réseaux de Petri. Au contraire, les algèbres de processus de type CCS ou CSP offrent en natif des opérateurs dits

« de composition ». Il n'y a pas à notre connaissance de consensus sur la composition des réseaux de Petri, fussent-il temporels [DIA 01], alors que cette question est tranchée pour LOTOS [LOT 89], ses extensions temporelles telles que RT-LOTOS [COU 00], et plus généralement pour les algèbres de processus temporisées. C'est une des raisons pour lesquelles nous avons retenu RT-LOTOS comme langage sous-jacent au profil UML temps réel TURTLE (Cf. chapitre III-3).

Le caractère réactif des systèmes que nous étudions fait que nous devons privilégier un langage de modélisation dans lequel une architecture est formée d'entités (modules, objets, etc.) dotée d'interfaces de communication avec leur environnement. Une telle architecture doit faire apparaître des canaux de communication véhiculant les messages ou signaux émis et reçus par ces entités. D'où la nécessité de doter les entités de ports de communication. Les connexions établies entre ports permettent de composer les comportements instanciés par les entités. En dehors de la famille des algèbres de processus, ces compositions sont souvent implicites. C'est le cas en Estelle (instruction *connect*), en SDL et plus récemment en UML 2.0.

La communication entre entités de même niveau relève effectivement de la composition « directe ». Dans une architecture hiérarchisée où les entités complexes sont décomposées, on ressent le besoin d'un mécanisme de délégation ou renommage. Il en existe en Estelle (instruction *attach*), en SDL et en UML 2.0.

Etant donné un langage d'entités communicantes muni d'opérations de composition et de renommage, l'on peut se demander si une architecture décrite en ce langage doit rester statique ou bien si elle peut-être dynamique dans le sens où à la fois les entités et les connexions entre ports de communication pourraient être crées ou détruites au cours de la vie du système. Cette possibilité existait, par exemple, en Estelle.

2.1.2. Eléments de méthodologie

Indépendamment du langage de modélisation retenu, la construction d'une architecture repose sur l'identification des entités qui la composent et des interfaces typées caractérisant les interactions entre ces entités. Dans le cas de systèmes communicants, nous identifierons les entités de protocole et leur environnement. Nous devons également être capable de faire un bilan des messages échangés entre les entités de protocoles.

Cette indentification des entités communicantes et des messages échangés peut être menée dans un langage par passage de messages tels qu'Estelle ou dans un contexte objet

avec UML. Notons bien ceci : c'est la culture de modélisation de protocole qui nous préoccupe ici et non la génération de langages de modélisation. Aussi, nous nous référons immédiatement aux trois *patterns* d'architecture décrits par la figure 4. On retrouve sur cette figure un modèle à un niveau caractéristique d'un couplage direct, suivi d'un modèle à deux niveaux dans lequel intervient le service sous-jacent aux entités de protocole, pour aboutir finalement à un modèle à trois niveaux dans lequel deux entités de protocole s'appuient sur un service de communication existant pour offrir un service avec valeur ajoutée à leurs utilisateurs. Le choix entre ces trois *patterns* relève de la décision de la personne qui modélise. Dans tous les cas, l'approche de modélisation retenue reste fondamentalement celle d'une modélisation par couche de protocole.

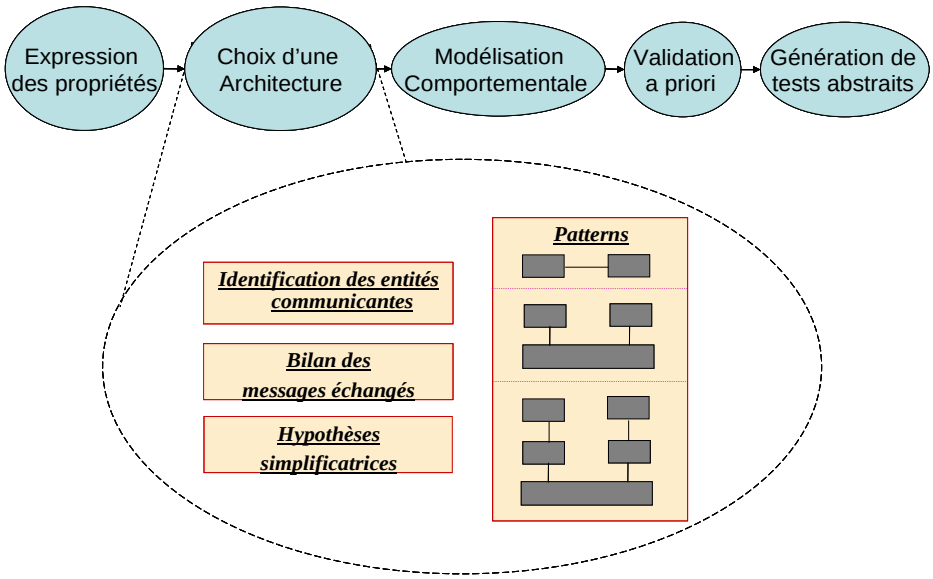


Figure 4. Eléments de base dans le choix d'une architecture

Pour un même système communicant, il sera possible d'utiliser un ou plusieurs des *patterns* représentés à la figure 4 en fonction de l'intérêt de représenter ou pas l'environnement des entités de protocole (interfaces inférieure et supérieure). Cette prise en compte progressive de l'environnement s'inscrit dans une démarche de modélisation incrémentale partant d'un ensemble d'hypothèses simplificatrices qui seront progressivement levées pour rendre le modèle plus réaliste et ce, tant que les techniques de validation a priori présentées plus loin dans ce chapitre, permettront de traiter le modèle en question.

Rappelons pour conclure ce paragraphe quelle est l'origine du modèle à 3 niveaux qui apparaît sur la figure 4. Du modèle OSI [ZIM 80] nous conservons encore aujourd'hui l'idée

que la conception d'une architecture de communication doit être centrée sur la notion de « couche de protocole ». D'un point de vue purement conceptuel et même si l'implantation ultérieure a toutes les chances de « mélanger » plusieurs couches (ce, à des fins d'amélioration de performances), nous préconisons de ne modéliser qu'une couche de protocole à la fois. Pour se placer dans la mouvance « objet », la figure 5 attribue le nom de « pattern à 3 niveaux » à ce modèle d'architecture connu de longue date [COU 84] [BOC 90] et porteur de la dualité service/protocole apparue avec le modèle OSI [ZIM 80].

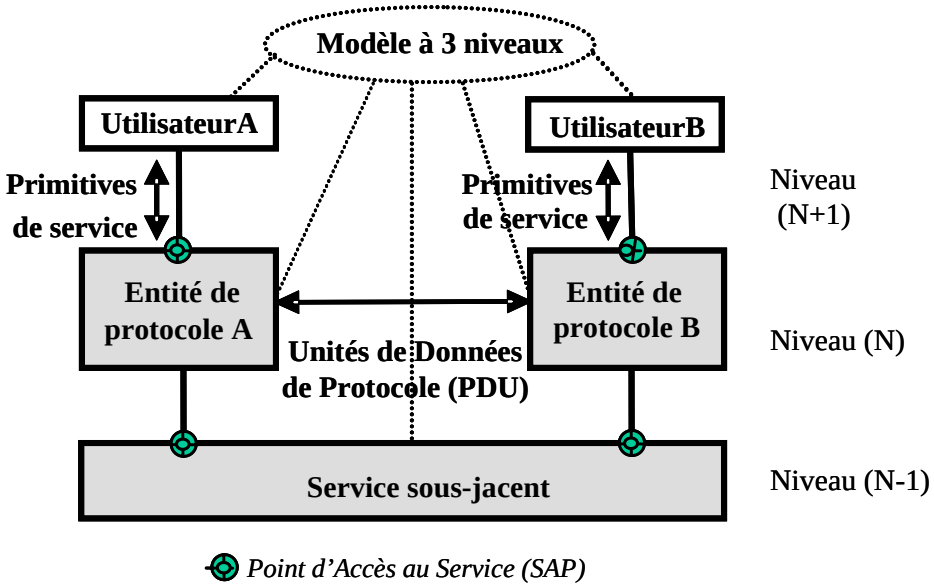


Figure 5. *Pattern* d'architecture pour la conception de protocoles

2.3. Modélisation comportementale

De la partie « description comportementale » d'un langage de modélisation de système temps réel ou distribué, l'on attend une capacité à (1) manipuler des variables locales (y compris un état majeur lorsqu'on se trouve dans une paradigme états/transitions), (2) exprimer synchronisation et communication au travers de variables partagées pour la décomposition locale et au travers d'interactions avec échanges de données pour ce qui relève des communications entre entités distribuées, et (3) une manipulation explicite du temps au travers de mécanismes dont les temporisateurs sont un exemple de base.

Ce triptyque (état local, interactions entre entités localement regroupées ou distribuées, gestion explicite du temps) se décline de manières diverses et variées dans une multitude de langages impossibles à recenser ici. Les langages (machines à états communicantes, réseaux

de Petri, Estelle, Proméla [HOL 97], SDL, UML 2.0) qui implémentent un paradigme états/transitions sont parmi les plus faciles à appréhender pour un concepteur de systèmes qui réagit aux stimuli de son environnement.

Note : dans ce mémoire, nous nous spécialisons sur les langage « asynchrones » par opposition aux langages « synchrones » tels que Lustre [CAS 87], Estérel [BS 91] ou Signal [GBP 86] dont l'hypothèse de modélisation vérifie l'hypothèse dite de « synchronisme ». Notre choix s'explique d'une part, par le caractère distribué et donc asynchrone de la majorité des systèmes que nous avons eu à traiter dans le cadre de notre activité de recherche et d'autre part, la difficulté de s'assurer que le système que l'on entend modéliser et concevoir vérifiera effectivement l'hypothèse de synchronisme (et donc que le recours à un langage synchrone sera fondé).

2.4. Validation a priori par simulation et vérification

2.4.1. Besoins

Bien des interprétations circulent sur le sens des mots « vérification » et « validation ». Dans ce mémoire, nous parlons de « validation a priori » opérée sur un modèle du système dans le but de détecter au plus tôt les erreurs de conception. Cette validation se base sur la complémentarité entre une simulation qui analyse partiellement l'espace d'états du système modélisé et une vérification qui après construction de cet espace d'états[1] permet de confronter une conception basée modèle à un ensemble de propriétés attendues du système. En ingénierie des protocoles, ces propriétés sont à rattacher à la notion de « service ». Dans ce cas, le but de la validation a priori est de montrer qu'une couche de protocole rend effectivement le service attendu par ses utilisateurs.

Au-delà des protocoles, nous incluons la simulation et la vérification de modèles dans un processus incrémental (figure 6) dans lequel les exigences sont introduites étapes par étapes dans le modèle de conception formé de l'architecture et des comportements. Chaque inclusion d'exigences est suivie de la validation de la conception qui en découle sans occulter la question de la non régression (par exemple, pour s'assurer qu'un protocole qui compense les pertes du réseau sous-jacent continue à traiter correctement le cas où ce réseau sous-jacent ne perd aucun message).

[1] En supposant le système borné.

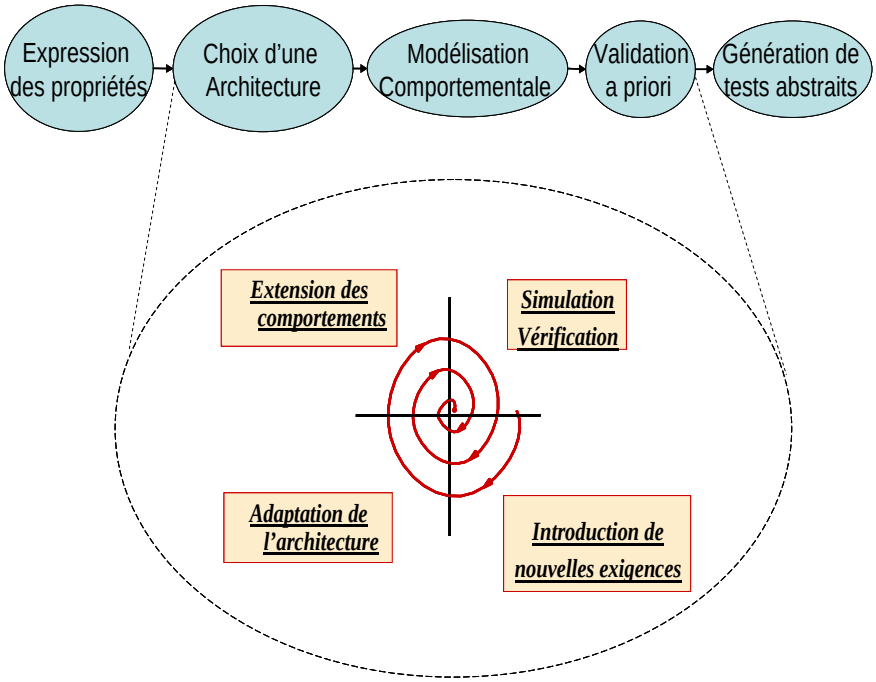


Figure 6 Méthodologie de conception incluant la validation a priori

2.4.2. Simulation

« Simulation » : là encore, voici un mot aux sens multiples. Dans ce mémoire, nous ne nous préoccuperons pas dévaluation de performances. Nous supposons disposer d'un langage de modélisation qui soit exécutable et dont la sémantique formelle garantit à chaque construction du langage une interprétation unique. A partir de là, la simulation permet d'explorer partiellement le comportement du système modélisé. Le temps consacré à se rapprocher d'une exploration « totale » des comportements possibles dépend des ressources humaines et matérielles mises à disposition. S'y ajoute le problème de l'exploitation des traces de simulation. Que faire en effet de « rubans » de scénarios (MSC ou diagrammes de séquences UML) obtenus brut de fonderie en sortie d'un simulateur SDL ou UML 2.0 ?

Les simulateurs autorisant l'analyse comportementale de systèmes temps réel ou distribués offrent en règle générale trois modes de simulation : la simulation interactive qui relève du déboguage, la simulation dite « aléatoire » qui choisit les transitions sur la base d'une fonction pseudo aléatoire dont le germe permet au besoin de rejouer des scénarios de simulation et, enfin, la simulation guidée soit par un programme externe, soit par une entité (module, objet, etc.) incluse dans le modèle du système. Cette entité de guidage de

simulation peut se ramener à une sonde (*probe* en anglais) qui se contente de tracer l'évolution de l'état des entités du modèle (état de la machine de protocole, variables ou attributs, etc). Cette sonde jusqu'à présent supposée non intrusive peut le devenir si par la modification d'un élément du modèle (attribut d'un objet par exemple) elle en vient à influencer le cours de la simulation. Au delà des sondes, il est possible de définir de véritables observateurs [ACD 82] [JAR 88] [DJC 94] qui décrivent des propriétés dont la validité ou l'invalidité peut être vérifiée à la volée [GRO 00].

Indépendamment du mode de simulation retenu, il reste à exploiter les résultats de simulation en termes de traces et en tenant compte de la couverture de la simulation par rapport au nombre d'états et transitions visitées.

2.4.3. Vérification

Poursuivant dans l'optique d'une validation comportementale, nous abordons maintenant la vérification que l'on suppose ici reposer sur une analyse exhaustive[2] de l'espace d'états du système modélisé. En termes d'exhaustivité, nous parlerons à présent « d'analyse d'accessibilité » et par conséquent du graphe d'accessibilité qui caractérise l'ensemble des états stables que le système peut atteindre à partir de son état initial. Deux problèmes se posent : d'une part, peut-on construire le graphe d'accessibilité et comment ? D'autre part, une fois le graphe construit, comment l'exploiter pour vérifier des propriétés générales (par exemple, l'absence de *deadlocks*) ou spécifiques (par exemple le fait que tout message envoyé par une tour de contrôle sera acquitté dans les dix minutes qui suivent).

Plaçons-nous à partir de maintenant dans l'hypothèse favorable où le graphe d'accessibilité du modèle du système étudié peut être engendré dans un temps acceptable. Le problème se pose alors de la disponibilité des ressources mémoires nécessaires au stockage des états rencontrés et à la comparaison des états entre eux. Plus le langage de modélisation est « riche », par exemple lorsque l'on travaille avec des réseaux de Petri temporels [BER 04], plus la structure des états à stocker et à comparer devient complexe. Holzmann a implanté dans l'outil SPIN [HOL 97] une approche particulièrement

[2] Nous ne traiterons pas ici des techniques « à la volée » (*on the fly*) qui permettent de vérifier une propriété en évitant de parcourir l'espace d'état du système dans son intégralité, sauf si la propriété est vraie (dans ce cas, l'ensemble de l'espace d'état est parcouru).

performante et reprise par d'autres outils pour coder les états en utilisant des fonctions de *hashing*. A cette piste orientée « codage optimisé des états », plusieurs chercheurs ont préféré une piste « exploration partielle » de l'espace d'états du système avec pour contrainte que la propriété ou la classe de propriétés recherchée soit préservée par l'abstraction réalisée. L'on trouvera en [VER 01] un comparatif des techniques d'exploration partielle et en particulier la méthode des pas couvrants proposée par l'auteur et implantée depuis dans l'outil TINA [BER 04] développé au LAAS-CNRS pour l'analyse des réseau de Petri.

Si l'on en vient maintenant à l'exploitation du graphe d'accessibilité à des fins de preuve de propriété, deux techniques nous intéressent : le contrôle de modèle (*model-checking*) et la vérification par abstraction.

Le contrôle de modèle consiste à « poser une question » qui appelle une réponse de type oui/non permettant de vérifier si la propriété attendue est valide ou non. L'outil KRONOS [KRONOS] est un exemple de *model-checker*. Le problème posé par ce type d'outil réside essentiellement dans l'expression des propriétés par une personne non formée à la logique temporelle et ses dérivées. Pour cette raison, nous n'exploiterons pas le contrôle de modèle dans la suite de ce mémoire et privilégierons la vérification par abstraction.

Cette approche de vérification dite « par abstraction » vise à observer une partie seulement du système modélisé pour s'abstraire d'éléments relégués au rang de détail et obtenir ainsi une vue (partielle) du comportement du système étudié focalisée sur des événements jugés importants par le/la concepteur/trice de ce système. Indépendamment de la génération de langage de modélisation (Estelle au chapitre III-2 ou UML au chapitre III-3), nous avons toujours basé la vérification par abstraction sur l'analyse d'accessibilité d'un modèle. L'idée est de décorer les transitions du graphe d'accessibilité par des événements jugés d'intérêt et en l'occurrence déclarés « observables » par opposition au reste des événements qui seront considérés de facto comme « internes » ou tout au moins non observables depuis l'environnement. Ainsi, le graphe d'accessibilité est traité comme un système de transitions étiquetées que l'on minimise par rapport à une relation d'équivalence et par rapport à un ensemble d'événements que l'on veut voir préservés par la minimisation. Le résultat d'une telle minimisation est un automate quotient qui donne une vue abstraite du système modélisé. A ce jour, l'outil Aldébaran [CADP] est la référence en matière de minimisation de systèmes de transitions étiquetées. Parmi les relations d'équivalence implantées par l'outil Aldébaran, nous avons régulièrement privilégié l'équivalence

observationnelle de Milner (MIL80] pour sa faculté à préserver certains choix non déterministes [GLA 90] .

2.5. Test

Nous souhaitons rappeler ici que nous n'avons pas pratiqué le test ou tout au moins la génération de séquences de tests comme une activité première à partir d'une modélisation, mais que nous l'avons toujours adossée à la vérification. Ceci ne nuit en rien à la généralité des principes qui vont maintenant être énoncés.

2.5.1. Quel type de test ?

La validation formelle est un premier filtre d'erreurs qui ne dispense pas de tester l'implantation du système. Il existe différents types de test qui sont autant de procédés complémentaires pour gagner en confiance dans une implantation et, dans des domaines tels que l'aéronautique, passer avec succès l'étape de certification. Dans ce mémoire, nous privilégions les tests de conformité et d'interopérabilité par rapport aux tests de robustesse et de performance. Le test de conformité vise à montrer qu'une implantation sous test est conforme à la spécification. Le test d'interopérabilité vient ensuite qui permet d'établir si deux ou plusieurs implantations séparément réputées conformes à la spécification arrivent bel et bien à interagir et donc à communiquer et coopérer.

2.5.2. Génération de séquences de test

Alors que la vérification formelle relève souvent de préoccupations académiques, le rôle du test de conformité est bien compris des industriels pour des raisons évidentes de mise sur le marché. Dès lors que ces mêmes praticiens industriels nous disent consacrer jusqu'à 50% du coût d'un projet à la production des jeux de tests, il n'est pas surprenant que la première des activités de recherche liées au test soit la « génération de séquences de tests » [FUJ 91].

En la matière, il ne suffit pas de trouver une façon « astucieuse » de parcourir le modèle du système pour en dériver des séquences de test. Définir un cadre formel de génération de séquences de tests suppose d'aborder les points suivants :

1. L'exécutabilité des tests. Dans un modèle états/transitions, les contraintes associées à chaque transition traversée par un cas de test doivent être satisfaites par les variables utilisées dans ce cas de test.
2. Le modèle de fautes. Ce modèle doit caractériser l'ensemble des fautes que l'on s'attend à rencontrer lors de l'exécution des tests. Les fautes de sortie et de transferts sont communes à la plupart des modèles étudiés dans la littérature [PET 01]. Les modèles de fautes intégrant une dimension temporelle (par exemple une attente hors des bornes d'un intervalle) sont moins répandus et en tout cas objets de publications récentes (voir par exemple [ENN 02]).
3. La couverture de fautes des tests. On en vient à parler ici du pouvoir de détection des fautes sur une implantation donnée. Ce critère sert à comparer les techniques de génération de séquences de test qui partagent le même modèle de fautes.
4. Une relation de conformité qui donne son sens à l'idée de conformité d'une implantation par rapport à une spécification. La formalisation de la relation de conformité suppose que l'implantation et la spécification puissent être décrites dans le même langage formel.

Ces quatre points forment le « socle » des études dédiées à la génération de séquences de test. Nous y ajoutons une liste complémentaire tenant compte des publications les plus récentes dans notre domaine d'application et des spécificités des langages et outils de modélisation que nous mettrons en avant au prochain chapitre.

1. Le lien entre la génération de tests et la vérification. On perçoit intuitivement des analogies entre l'analyse énumérative qui sous-tend la construction d'un graphe d'accessibilité et le « parcours » d'un modèle (de type machine à états finie par exemple) qui permet de générer des tests.
2. La non prolifération des séquences de tests. Rien ne sert de produire des tests trop nombreux ou du moins si nombreux qu'ils en deviennent inapplicables. Un moyen de restreindre l'espace d'états du modèle du système à tester est de réaliser un produit synchrone entre le modèle en question et des objectifs de test. Ces derniers permettent de guider des tests en affichant des priorités tenant compte en particulier de l'expérience de la personne en charge de mettre en œuvre les tests.
3. La prise en compte de l'architecture de test. Si le langage de modélisation le permet, les séquences de tests devront utiliser des Points de Contrôle et d'Observation (PCO au sens de la norme IS 9646). La question du test au sein d'une architecture pose le problème du test d'un composant non plus pris

individuellement mais dans bien des cas embarqué dans un système qui limite l'atteignabilité de ce composant.

4. Le niveau d'abstraction des tests et le passage de tests abstraits dérivés d'un modèle à des tests concrets, exprimés par exemple dans la notation TTCN (The Testing and Test Control Notation [TTCN]) et de fait exécutables sur un testeur réel.

Enfin, l'on comprendra aisément qu'une implantation d'un système sera d'autant plus testable que la conception de ce système aura pris en compte le problème fondamental de la testabilité (on parle en anglais de *design for testability* [KUR 90]).

3 L'approche UML

Les promoteurs du langage de modélisation UML (Unified Modeling Language) présentent la notation aujourd'hui normalisée par l'OMG (Object management Group) [OMG 03] comme le compromis le plus abouti entre les « meilleures pratiques » du génie logiciel. Ce langage qui est en fait une notation et en aucun cas une méthode, résulte de compromis entre lobbyistes à l'OMG. La version 2.0 adoptée par cet organisme en août 2003 inclut pas moins de 13 diagrammes. Une difficulté en résulte pour qui veut dispenser un enseignement sur UML 2.0 : trouver un compromis ou tout au moins un équilibre entre égrenage des constructions syntaxiques des différents diagrammes et éléments de méthodologie permettant de comprendre comment utiliser UML dans un domaine d'application donné. Nous allons maintenant tenter de trouver cet équilibre pour présenter UML 2.0 et porter un regard critique sur UML 2.0 concernant son adéquation à traiter des systèmes temporellement contraints.

3.1. La notation UML 2.0

En amont du cycle de développement d'un système complexe, la notation UML n'offre aucune facilité pour le traitement textuel des exigences et la première difficulté réside dans le passage du cahier des charges aux diagrammes d'analyse offerts par la notation de l'OMG. Si les processus de développement tels que le Processus Unifié [JBR 00] suggèrent de démarrer l'analyse par un diagramme du domaine, la norme UML ne définit pas ce type de diagramme. UML permet de mener une analyse fonctionnelle au moyen de diagrammes

de cas d'utilisation, la transition vers la conception objet se faisant au moyen de diagrammes de séquences complétés d'un premier diagramme de classes.

Le diagramme de cas d'utilisation permet de délimiter le périmètre du système que l'on entend modéliser. Il identifie un ensemble de cas d'utilisation qui sont autant de fonctionnalités ou services que le système doit offrir à ses utilisateurs externes matérialisés par des « acteurs ». Dans le cas d'un protocole de communication orienté connexion, nous aurons ainsi trois cas d'utilisation correspondant aux phases du protocole, à savoir l'établissement de connexion, le transfert de données et la libération de connexion. Ce diagramme peut être raffiné pour exprimer par exemple que l'établissement de connexion inclut une négociation de qualité de service. Par contre, ce diagramme de cas d'utilisation n'exprime pas la dynamique du système ; ainsi, il ne permet pas d'exprimer le fait que la libération de connexion peut à tout moment interrompre l'une des deux autres phases. Des diagrammes états/transitions peuvent suppléer à ce besoin si l'on souscrit à l'idée que des éléments de dynamique du système ont leur place dans des diagrammes d'analyse. Cette idée a été défendue par des auteurs qui disposait d'un langage de spécification que nous qualifierons de « unique » par opposition aux différents paradigmes des diagrammes du langage UML qui est – lui – « unifié ». Ainsi en est-il du langage LOTOS et des styles de spécification définis dans le cadre du projet LOTOSPHERE [BOL 95] utilisés en [COU 95] (article dédié à la construction incrémentale d'un service de transfert de données). La version normalisée d'UML 2.0 et les outils de supports tels que TAU G2 n'ont pas un niveau d'abstraction nécessaire à la transposition de la méthodologie LOTOSPHERE.

A chaque cas d'utilisation (parfois à plusieurs d'entre eux), l'on va associer un ou plusieurs diagrammes de séquences qui sont autant de scénarios et donc de vues partielles de la dynamique du système. Un diagramme de séquences trace des lignes de vies temporelles pour matérialiser les interactions (échanges de messages ou appels de méthodes) entre objets qui forment le système. Des extensions temporelles ont été proposées qui permettent de représenter des temporisateurs et des intervalles temporels associés aux actions entreprises par les objets. Un problème important reste la question de la structuration des scénarios. UML 2.0 y apporte une solution avec les *Interaction Overview Diagrams,* forme d'organigrammes dont les actions sont des références à des diagrammes de séquences et donc à des scénarios. Dans le cas général, nous aurons un IOD structurant plusieurs diagrammes de séquences caractérisant respectivement le cas nominal (utilisation du système dans un environnement quasi parfait, par exemple avec un réseau sous-jacent sans perte ni duplication) et les cas dégradés (réseau avec perte par exemple).

La construction du diagramme de séquences aide à trouver les objets dont les types sont définis au niveau du diagramme de classes. Ce dernier met en évidence les types d'objets avec leurs attributs et leurs méthodes, sans oublier les ports de communication apparus avec UML 2.0. Un diagramme de classes exprime également les relations entre classes. Outre les relations « simples » dont la sémantique est donnée par le concepteur, la composition permet de structurer une classe en « sous-classes » et la relation d'héritage encourage la réutilisation de fragments de conception. Lorsqu'une classe est composée de « sous-classes », ces dernières peuvent communiquer entre elles ; ceci s'exprime au moyen du diagramme de structure composite apparu avec UML 2.0. La description de systèmes complexes s'opère ainsi en deux niveaux au moins, au travers de diagrammes de classes et de structure composite, respectivement.

Une fois définie la structure du système en incluant les entités qui forment ce système et leurs interfaces, l'on en vient à décrire le comportement interne des objets. Il serait effectivement possible de coder ce comportement en Java ou dans un autre langage mais les systèmes temps réels se prêtent fort bien à l'utilisation des diagrammes états/transitions qu'UML a empruntés aux Statecharts. Outre un paradigme états/transitions avec gestion des entrées/sorties, ces diagrammes intègrent la notion d'historique permettant de retrouver l'état antérieur à un déroutement mais aussi le concept d'état composite et de raffinement de comportement. Le profil UML2.0/SDL implanté par l'outil TAU G2 de Telelogic permet de plus de décrire des machines de protocoles dans une syntaxe SDL. Notons enfin que rares sont les outils qui vont au-delà d'un aspect purement documentaire dans le placement des objets dans des composants eux-mêmes répartis sur les nœuds d'un réseau (diagrammes de composants et de déploiement). Cette remarque s'applique aux outils UML temps réel des quatre principaux acteurs industriels du marché, à savoir Telelogic (outil TAU G2), I-logix (outil Rhapsody), IBM-Rational (outil ROSE-RT) et Artisan Software.

3.2. De l'adéquation d'UML à modéliser des systèmes temps réel

La question a été maintes fois posée de savoir si la notation UML peut répondre ou pas aux besoins de la conception des systèmes temps réel [GT 03]. Les efforts de l'OMG dans ce sens ont abouti à l'adoption de la norme UML 2.0 après définition d'un profil temps réel qui n'est pas une extension de la notation UML 1.5 mais une représentation UMLienne des concepts (ressources, temporisateurs, qualité de service, …) d'usage fréquent dans les

systèmes d'exploitation temps réel. La norme UML 2.0 reflète le compromis entre lobbyistes à l'OMG et l'on n'est pas surpris d'y retrouver outre les Statecharts inclus depuis UML 1.5, des constructions des langages ROOM et SDL. Dans les faits, nous pouvons même dire que le pouvoir d'expression d'UML 2.0 vis-à-vis de contraintes temporelles, ne surpasse pas celui du langage SDL. Ainsi, les diagrammes états/transitions permettent de gérer des temporisateurs (*timers*) mais en aucun cas des intervalles temporels. Notons que dans la version UML 1.5 d'UML qui a servi de point de départ à la définition du profil TURTLE présenté au chapitre III-3, il y avait équivalence entre Statecharts et diagrammes d'activité. Cette équivalence a d'ailleurs été exploitée par l'outil Rhapsody qui permettait de définir les comportements des objets indifféremment par un Statechart ou un diagramme d'activité. Au niveau du profil TURTLE, nous avons retenu les diagrammes d'activité.

Nous évoquions là les limitations en termes de comportement. Au niveau de la structuration, il faut noter que la norme UML 2.0 relègue la question de communication entre objet au titre de point de variation sémantique. De fait, les développeurs d'outils ont été amenés à développer des solutions propriétaires, par exemple une sémantique à la SDL pour ce qui est de l'outil TAU G2 de Télelogic. La version d'UML 2.0 implantée par TAU G2 est un profil UML/SDL [BJO 00]. Dans une optique de modélisation abstraite et de validation de modèle en indépendance d'une plateforme particulière, il nous semblerait préférable de pouvoir composer des fonctionnalités par un mécanisme abstrait ne préfigurant pas du mécanisme de communication qui sera réellement implanté. Ajoutons à ceci qu'en termes de composition, UML 2.0 garde au parallélisme entre objets son caractère « naturel » et donc implicite. De manière plus générale, l'absence de sémantique formelle de la progression du temps est un frein supplémentaire à la modélisation de systèmes temps réel et par extension à l'application de techniques de validation formelle.

Certaines limitations rencontrées avec la version 1.5 d'UML perdurent donc avec UML 2.0. De ce fait, l'apparition de la norme UML 2.0 rend tout sauf caduques les recherches que nous avons menées pour adosser UML à un langage formel, en l'occurrence l'algèbre de processus temporisée RT-LOTOS [COU 00]. Ces travaux liant UML et RT-LOTOS participent d'un mouvement plus général pour donner une sémantique formelle à UML.

3.3. Une sémantique formelle pour UML ?

Au niveau de l'OMG, la formalisation de la sémantique d'UML ne va pas au-delà de la définition du méta-modèle et de règles sémantiques écrites en anglais. La norme UML 2.0 utilise régulièrement le terme de « semantic varia point » pour désigner des choix sémantiques laissés à l'appréciation des développeurs d'outils (en particulier). L'on ne sera pas alors surpris de trouver des outils UML 2.0 qui implantent différemment la communication asynchrone entre objets en prenant soit une sémantique bloquante face aux réceptions non spécifiées, soit une sémantique non bloquante à la SDL. Ainsi, des pans entiers de la notation UML restent « partiellement » définis. C'est le cas par exemple du parallélisme entre objets.

Tout ceci explique le foisonnement depuis d'articles qui proposent une formalisation de la notation UML ou tout au moins d'une partie de cette notation, par exemple les Statecharts. On retrouve alors les deux approches classiques pour donner une sémantique formelle à un langage de haut niveau : soit par définition d'un modèle mathématique (voir, par exemple [BHH 97] [BF 98]), soit par traduction vers un langage déjà pourvu d'une sémantique formelle ([CM 00], [DB01], [TRA 00], projet OMEGA [GOO 05b], projet TURTLE [ACL 04], …).

Parmi les initiatives visant à donner une sémantique formelle à UML et regroupant plusieurs institutions de recherche sur la durée, nous pouvons mentionner p-UML (*Precise UML* [PUML]) et STL (*UML 2.0 Semantics Project* [STL]), la deuxième allant jusqu'à proposer la définition d'une « machine virtuelle UML ».

Chapitre 2

L’approche Estelle

Les techniques de Description Formelle Estelle [Est 89], SDL [SDL 89] et LOTOS [LOT 89] concrétisent le passage de modèles de base de types « automates » et « algèbres de processus » à une forme langage. L'investissement en termes de recherche et l'effort de normalisation qui ont abouti à la définition de ces trois langages traduisent le besoin ressenti par la communauté « ingénierie des protocoles » de disposer de langages de haut niveau pour la conception de systèmes distribués et de protocoles de communication en particulier [BAB 02]. Une description formelle en Estelle a pour vocation à servir de point de départ pour la validation a priori d'une conception mais aussi pour la génération de code et l'automatisation de la production de jeux de test.

Ce chapitre dédié est organisé de la manière suivante. Le paragraphe 1 donne un aperçu de ce langage et de son dialecte Estelle* (prononcer « Estelle-étoile »), promu par le LAAS-CNRS [COU 87b]. Le paragraphe 2 explique la méthodologie basée Estelle* que nous avons proposé d'appliquer à la validation d'architectures de communication. Le paragraphe 3 discute l'approche proposée. Enfin, le paragraphe 4 explique comment l'expérience menée sur Estelle a influencé notre approche ultérieure des techniques de modélisation et de validation de modèles.

1. Le langage Estelle* et l'outil ESTIM

Le langage Estelle permet de décrire une architecture dynamiquement modifiable d'entités communicantes et de préciser le comportement interne de chacune de ces entités sous la forme d'une machine de protocole. Langage par passage de messages, Estelle implante une composition de machines à états communicantes et repose sur le langage Pascal au niveau des structures de données.

1.1. Estelle*

La version normalisée d'Estelle [EST 89] a été critiquée [COU 87a] à l'aune d'expériences antérieures sur la modélisation de protocoles au moyen des réseaux de Petri [DIA 82] [COU 84]. Le langage Estelle* était né. Si l'on ne parlait pas à l'époque de « profil » au sens utilisé aujourd'hui par la communauté UML, Estelle* possédait cependant les ingrédients d'un « profil Estelle » : restriction sur l'usage de constructions litigieuses en termes d'interprétation du parallélisme, extension en termes de communication par rendez-

vous inspiré de la fusion de transition en réseaux de Petri, sémantique formelle donnée sur la base des mêmes RdP et enfin un couple (outil [CS 92], méthodologie [CDM 91]) orienté vers la validation d'architectures de communication.

1.2. L'outil ESTIM

L'outil ESTIM (*Estelle Simulator based on an Interpretative Machine* [CS 92]) était dédié à la validation formelle de descriptions Estelle*. La version initiale du prototype ESTIM offrait des fonctionnalités de simulation interactive ou aléatoire [SC 88]. Le moteur de simulation basé sur un interpréteur a été étendu à l'analyse d'accessibilité. Le lien avec les outils de minimisation de systèmes de transitions étiquetées PIPN [LLO 90] et Aldébaran [CADP] a permis de proposer une approche de vérification par abstraction[3] [SC 89] sur la base d'une génération de graphe d'accessibilité étiqueté par les seuls échanges de messages que l'utilisateur désirait observer.

Ultimement, nous avons basé la génération de séquences de test d'interopérabilité sur les automates quotients de services issus de vérifications par abstraction. Le protocole de messagerie industrielle MMS (*Manufacturing Messaging System*) a servi d'étude de cas [CSC 92a]. Note : les modèles MMS n'utilisant pas de clause *delay*, le test n'était pas « temporel ».

2. Estelle* appliquée aux protocoles : éléments de méthodologie

L'objectif de ce paragraphe est de montrer comment les techniques présentés au chapitre III-1 ont été instanciées dans le contexte du langage Estelle*. Il ne s'agit pas de prodiguer un cours de modélisation de protocoles mais de livrer quelques éléments de méthodologie caractéristiques d'une approche de validation d'architectures de communication appliquées à maintes occasions (protocoles de niveau Transport, protocole MMS [CCS 92b], cellule flexible d'assemblage [MAZ 91], API multimédia [CSC 03]).

[3] Le terme de « projection » a parfois été utilisé pour qualifier ce type d'abstraction.

2.1. Expression du service

Le langage Estelle offre une syntaxe textuelle pour définir une architecture et des comportements. C'est un langage de conception d'architecture distribuée et de modélisation des protocoles qui sous-tendent ces architectures. Par contre, Estelle n'offre aucune facilité particulière pour définir un service et plus généralement pour exprimer propriétés et exigences.

Faute de quoi, nous avons exprimé les services sous diverses formes ad-hoc. Dès lors que la vérification par abstraction a été implantée dans l'outil ESTIM, nous avons pris l'habitude de décrire le service attendu sous forme d'automates, facilitant ainsi leur comparaison (manuelle) avec les automates quotients résultant de la minimisation du graphe d'accessibilité étiqueté par des échanges de primitives de service.

Notons cependant que les automates caractérisant le service attendu peuvent s'accompagner d'une description Estelle* de ce service. Dans ce cas, nous aurons une description formée d'un seul module portant le nom du service et doté d'autant de points d'interactions que de points d'accès au service. On peut décrire l'ordonnancement temporel des primitives de service sous forme de machines à états Estelle* ne faisant aucune mention des PDUs échangés. Cette description monolithique du service peut aider à combler le fossé entre description du service et description du protocole [CDM 91a].

2.2. Conception du protocole

La définition du service attendu sous forme d'automates permet de définir un ensemble d'entités communicantes et des messages qu'échangent ces entités. Cette information est à la base d'une architecture Estelle qui définit[4] un ensemble de « boites » éventuellement raffinées en « sous-boites » et en tout état de cause interconnectées pour communiquer.

Notons que le langage Estelle permet dans sa définition normalisée de grouper les entités communicantes en « systèmes ». Nous nous placerons ici dans la configuration d'une description Estelle* formée d'un unique système attribué par *systemactivity* et raffiné en modules attribués par *activity*. La forme de parallélisme alors mise en œuvre – la seule

[4] Sous une forme exclusivement textuelle, ce qui a amené les auteurs de descriptions Estelle à accompagner leurs sources Estelle de dessins d'architectures donnés dans une notation ad-hoc construites sur des boites et des connecteurs.

implantée par l'outil ESTIM – est compatible avec les modèles formels, réseaux de Petri par exemple, les plus répandus parmi les techniques de modélisation qui implantent une sémantique d'entrelacement.

De plus, Estelle* ajoute au mécanisme de files FIFO d'Estelle un mécanisme de communication par rendez-vous (à deux). Dans une architecture à trois niveaux centrée sur deux entités de protocole s'appuyant sur un service existant pour rendre à leur tour un service à leurs utilisateurs, la communication entre les entités de protocole et un service datagramme se fera par file FIFO alors que l'on privilégiera le rendez-vous pour le communications entre entités de protocoles et utilisateurs.

Lorsque les choix architecturaux auront été stabilisés, on en viendra à décrire les machines de protocoles dans un style états/transitions avec traitement des entrées/sorties, manipulation de variables simples ou structurées, gestion de temporisateurs et – si besoin est – définition de priorités entre transitions simultanément sensibilisées.

2.3. Validation de l'architecture de communication

La finalité première d'une modélisation de protocole en Estelle* est la validation d'une architecture de communication. Dans l'architecture à trois niveaux précédemment évoquée, nous déclarons d'intérêt les échanges (par rendez-vous) de primitives de services et voulons nous abstraire tant des échanges de PDUs sur les canaux FIFO entre entités de protocole et service existant, que des évolutions des variables locales à ces entités. Traité comme un système de transitions étiquetées ayant les primitives de services pour seuls événements observables, le graphe d'accessibilité de la description Estelle* du protocole peut être minimisé en utilisant l'outil Aldébaran. L'automate quotient alors obtenu caractérise le service fourni par la couche de protocole. Selon l'équivalence retenue pour cette minimisation, l'on obtient soit un graphe étiqueté par les seuls échanges de primitives de service (équivalence langage) ou bien un automate quotient étiqueté non seulement par les primitives de service mais aussi des événements internes préservés pour caractériser des choix non déterministes (équivalence observationnelle de Milner).

La figure 1 résume notre approche de modélisation de protocole en Estelle* en support à la validation d'une architecture de communication.

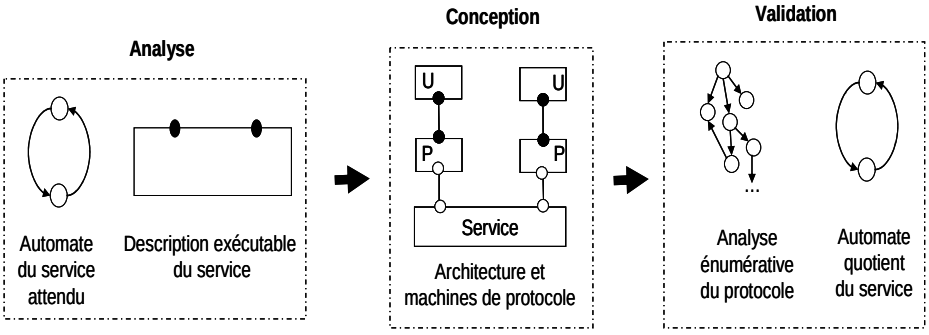


Figure 1. Validation d'une architecture de communication modélisée en Estelle*

3. Contrôle versus données

Comme ses « compétiteurs » SDL et LOTOS, Estelle sied parfaitement aux protocoles de niveau Transport [Boch 90]. Il n'en va pas de même aux extrémités de la pile protocolaire ISO. Ainsi, la manipulation des données en Pascal est peu commode pour décrire des protocoles de la couche liaison de données, voire de la couche physique. A l'autre extrême, le langage Pascal ne satisfaisait pas davantage les développeurs des couches applicatives. C'est pourquoi, nous avions proposé une étude [COE 89] sur l'introduction de la notation ASN.1 dans Estelle* et ESTIM.

Le paragraphe précédent pourrait laisser penser que l'effort a été concentré sur les structures de données. En fait, la simulation et la vérification par abstraction s'appliquaient en priorité à la partie « contrôle » d'un protocole et permettaient d'étudier les communications, synchronisations et les temporisations (tout au moins les « timeouts » à la Estelle). L'approche de validation proposée en [SAQ 90] supposait que les « points durs » du protocole eussent été localisés dans la partie « contrôle » et réactive. L'outil ESTIM eût été de peu de secours pour un protocole simple en termes d'échanges de messages mais dépendant d'algorithmes de cryptage complexes qui auraient « mérité » des techniques de preuves telles qu'on les trouve sur des outils dédiés au langage B [ABR 96], par exemple.

4. Incidences sur la suite de notre activité de recherche

Avec le recul, il apparaît que ce travail de thèse dédié au langage Estelle* et au prototypage d'ESTIM a sub-consciemment prédéfini nos prédilections en matière de

langages de modélisation et orienté les travaux de recherche exposés dans la suite de ce mémoire. Ainsi :

- La description d'une architecture et d'un comportement à des fins de conception a prévalu sur l'expression de propriétés et de scénarios.
- La partie « contrôle », réactive et temporelle a été approfondie « au détriment » de la partie « données ».
- La validation a priori de modèles a été menée en associant exploration partielle par simulation, analyse d'accessibilité et vérification par abstraction.
- La génération de séquences de tests a été abordée en complémentarité avec la vérification.
- La recherche de convergences avec des techniques et outils développés dans notre entourage immédiat de recherche a été une préoccupation constante.
- Dans le couple (outil, méthodologie) qui conditionne le « succès » d'un langage de modélisation, la méthodologie est un élément déterminant.
- Cette expérience ne vaut que si elle est partagée, en particulier au travers de publications et d'activités d'enseignement.

Chapitre 3

Le projet TURTLE

Timed UML and RT-LOTOS Environment

Les promoteurs du langage UML ont fait adopter à l'OMG une notation à caractère généraliste dotée de mécanismes d'extension qui permettent d'adapter cette notation à un domaine d'application particulier. Cette personnalisation d'UML s'opère au travers de la définition d'un « profil » qui spécialise le méta-modèle UML [OMG 03]. Un profil est composé d'un certain nombre d'éléments sélectionnés depuis le méta-modèle de référence, de mécanismes d'extension, d'une description de la sémantique du profil, et enfin, de règles permettant la présentation, la traduction et la validation du modèle.

Après avoir normalisé UML 1.5 et peu avant l'adoption d'UML 2.0, l'OMG (Object Management Group) a développé un profil UML temps réel très orienté « ordonnancement » qui représente un certain nombre de concepts (ressources, temporisateurs, qualité de service) majoritairement orientés « système d'exploitation ». Ce profil n'apporte pas d'extension au langage UML et diffère en cela d'UML 2.0 qui intègre – entre autres – des concepts empruntés aux langages ROOM et SDL.

Le profil UML temps réel TURTLE qui a été au centre de notre activité de recherche sur la période 2000-2005, a été défini pour sa partie « conception » sur la base d'UML 1.5 ; la partie « analyse » est postérieure à l'adoption d'UML 2.0 et se base sur cette nouvelle « norme[5] ». Outre la formalisation, la force du profil tient en particulier au développement par Ludovic Apvrille (Télécom-Paris, Sophia Antipolis) d'un outil dédié – TTool (TURTLE Toolkit) – placé en amont de RTL, outil de validation formelle de spécifications RT-LOTOS. La sémantique du profil TURTLE est en effet donnée par traduction vers le langage formel RT-LOTOS. Ceci nous a permis de réutiliser au bénéfice de TURTLE les outils disponibles pour RT-LOTOS et au premier chef RTL.

Le développement de TTool se poursuit à l'heure où nos écrivons ces lignes. Nous aurons au cours de ce chapitre l'occasion de distinguer les extensions déjà implantées dans TTool de celles déjà publiées mais non encore outillées.

Ce chapitre est organisé comme suit. Le paragraphe 1 rappelle les principales motivations qui ont prévalu à la définition d'un profil UML temps réel adossé au langage formel RT-LOTOS. Le paragraphe 2 donne un bref aperçu du langage RT-LOTOS. Le paragraphe 3 présente le profil UML temps réel « natif » applicable en phase de conception (diagramme de classes et diagrammes d'activité). Le paragraphe 4 traite ensuite des diagrammes d'interactions en phase d'analyse. Le paragraphe 5 décrit l'environnement logiciel qui

[5] Les guillemets reflètent le fait qu'à ce jour les documents définissant UML 2.0 ont été adoptés sans atteindre le statut de « norme », lequel sera accordé après finalisation du processus de révision.

supporte le profil à la fois dans sa partie « conception » déjà expérimentée sur plusieurs études de cas et dans sa partie « analyse » qui est en bétha-test à ce jour. Le paragraphe 6 situe nos propositions par rapport au processus de normalisation à l'OMG. Enfin, le paragraphe 7 explique le positionnement de l'approche TURTLE par rapport aux travaux de recherche dédiés à UML temps réel.

1. Contexte, problème et solution

Face à un processus de normalisation à l'OMG qui fit dans un premier temps [OMG 03a] abstraction des spécificités des systèmes temps réel [OMG 03b], le développement du profil UML temps réel TURTLE (*Timed UML and RT-LOTOS Environment* [ACL 04]) répond au besoin de doter la notation UML d'une sémantique formelle et d'adosser la validation formelle de modèles UML temps réel à des techniques éprouvées. Le profil TURTLE est supporté par une chaîne d'outils qui va de l'édition de diagrammes à la validation d'un modèle par combinaison d'une exploration partielle sous forme de simulation et de vérification à base d'analyse d'accessibilité exploitée au moyen de « model checking » ou de vérification par abstraction. Ce prototype s'adresse à tout concepteur/trice de système temps réel, système embarqué ou protocole de communication désireux/se de détecter au plus tôt les dysfonctionnements logiques et temporels potentiels sur un modèle de haut niveau indépendant d'un langage d'implantation ou d'un système d'exploitation spécifique.

2. RT-LOTOS

LOTOS [BOL 87] est une Technique de Description Formelle et une norme internationale [IS 8807] pour la spécification et la conception de systèmes de traitement distribués. Une spécification LOTOS se présente sous la forme d'un processus structuré en d'autres processus. Un processus LOTOS est une boite noire qui communique avec son environnement au travers de portes et sur le principe d'une offre de rendez-vous. Des échanges mono- ou bi-directionnels de valeurs sont autorisés lors de la synchronisation.

Le parallélisme et la synchronisation entre processus s'expriment par des opérateurs de composition : mise en séquence, synchronisation sur toutes les portes, synchronisation sur certaines portes, choix non déterministe et entrelacement (composition parallèle sans synchronisation).

L'acronyme LOTOS (*Language of Temporal Ordering of Sequences*) indique sans ambiguïté la finalité de ce langage : décrire l'ordonnancement temporel des actions. Ainsi, LOTOS n'offre ni opérateur temporel, ni mécanismes temporels de base tels qu'un « temporisateur ». Il est cependant possible de modéliser un temporisateur en LOTOS à la condition de ne pas avoir besoin d'exprimer explicitement la progression du temps et de traiter cette dernière comme un événement interne.

Faire de LOTOS un langage formel avec prise en compte explicite du temps a été une préoccupation de plusieurs laboratoires de recherche dont le LAAS-CNRS qui a défini et outillée RT-LOTOS [COU 00]. D'autres extensions temporelles à LOTOS sont recensées en [LOH 02].

RT-LOTOS étend LOTOS avec trois opérateurs temporels (Tableau 2). La combinaison d'un délai déterministe et d'un délai non déterministe permet de traiter les intervalles temporels. Le langage ainsi étendu conserve la partie « contrôle » de LOTOS, mais remplace les types de données algébriques par des implantations en C++ ou Java [COU 00].

Opérateurs Temporel	***Description***
a{T}	Offre limitée dans le temps.
delay(t1)	Délai déterministe.
latency(t2)	Délai non déterministe.

Figure 2. Opérateurs temporels de RT-LOTOS

RT-LOTOS se distingue d'autres extensions temporelles de LOTOS et en particulier de ET-LOTOS [LL 97] par son opérateur *latency* qui propose pour l'expression de l'intéderminisme temporel une solution plus générale [COU 00] que ET-LOTOS.

3. Un profil UML temps réel orienté conception

Dans sa version initiale [ALS 01], le profil UML temps réel TURTLE personnalise les diagrammes de classes et d'activité UML pour adresser les volets « structure » et « comportement » d'une conception de système temps réel ou distribués.

3.1. Architecture : diagrammes de classes étendus

Une conception TURTLE natif repose sur un diagramme de classes pour définir l'architecture du système comme un ensemble de *Tclasses* (TURTLE classes). Ces Tclasses communiquent par rendez-vous sur des portes (*gates*). Chaque Tclasse se voit associer un diagramme d'activité représentant son comportement interne décrit par un diagramme d'activité dont les actions font références soit à ces portes, soit aux attributs. Un diagramme d'activité TURTLE autorise la représentation de choix non déterministes. Il permet enfin est surtout de décrire des mécanismes et des contraintes temporelles.

3.1.1. Stéréotype Tclass et portes de communication

Aux classes UML de base caractérisées par un nom, des attributs et des méthodes, le profil TURTLE ajoute des « *Tclasse*[6]*s* » (cf. Figure 3) dotées d'attributs particuliers appelés « gates » qui sont des portes de communication. Ces portes, instances du type abstrait *Gate* (figure 4) sont bidirectionnelles. La spécialisation de *Gate* en *InGate* et *OutGate* permet la description de ports unidirectionnels. La définition du profil TURTLE a en quelque sorte « anticipé » l'adoption par l'OMG de la norme UML 2.0 qui consacre l'idée qu'une classe UML puisse avoir des portes de communication. Contrairement à la norme UML 2.0, la définition du profil TURTLE n'entretient pas de flou sur la sémantique de communication associée aux « gates » et adopte un rendez-vous à la LOTOS. Ce rendez-vous est moins orienté implantation que la sémantique du profil UML2.0/SDL implanté par l'outil TAU G2.

Ajoutant à ceci que toute *Tclasse* contient un diagramme d'activité modélisant son comportement, nous pouvons identifier les différentes zones d'une Tclasse sur le schéma suivant (3).

[6] Les *Tclasses* sont construites par le mécanisme d'extension UML appelé « stéréotype ».

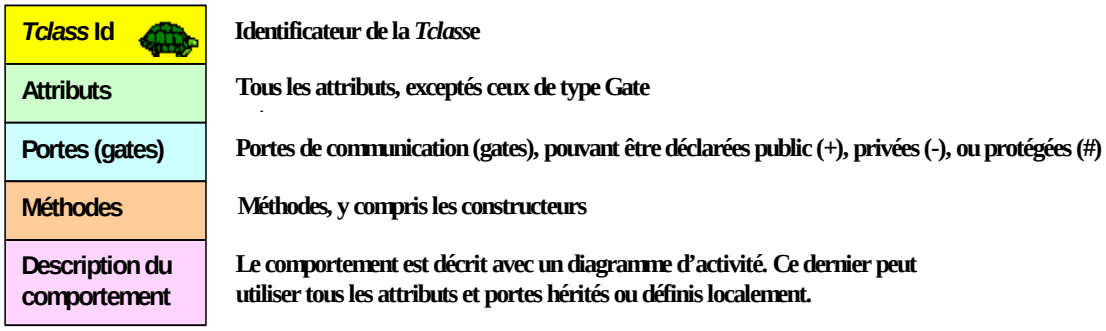


Figure 3. Structure d'une *Tclasse*

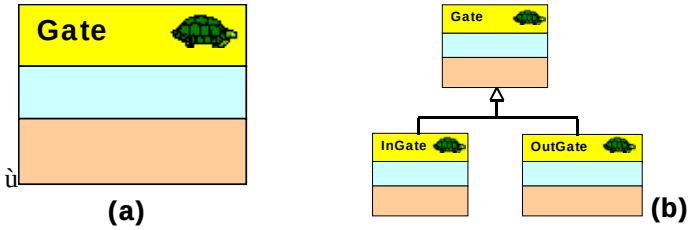


Figure 4. Le type abstrait *Gate* (a) et la distinction entre *InGate* et *OutGate*

3.1.3. Propriétés d'une Tclasse

La figure 3 décrit la structure générale d'un stéréotype *Tclass* : les attributs de type *Gate* y sont séparés des autres attributs. Les propriétés principales des *Tclass* sont les suivantes :

◊ prop1 : tous les attributs, sauf ceux de type *Gate*, doivent être déclarés privés (-) ou protégés (#).

◊ prop2 : toutes les méthodes, sauf les constructeurs, doivent être déclarées privées (-) ou protégées (#). Les constructeurs peuvent être déclarés publics.

◊ prop3 : un diagramme d'activité[7] doit accompagner chaque *Tclass* et décrire ainsi son comportement. Ce diagramme peut utiliser tout attribut ou classe déclarés dans la *Tclass* ou hérités.

◊ prop4 : les communications entre *Tclass* sont nécessairement réalisées au travers des portes de communication (*Gate*). Synchrones, ces communications sont les seules à être prises en compte lors de la validation. Ainsi, une *Tclass* peut communiquer avec des classes normales par appel de méthode, modification d'attributs, signaux, etc., mais ces communications sont ignorées pour la validation.

[7] Seuls les diagrammes d'activité sont pris en considération dans cet article. Nos travaux futurs aborderont les Statecharts.

◊ prop5 : des attributs de type *Gate* peuvent être déclarés dans tout type de classe. Néanmoins, seules les *Tclass* peuvent les utiliser à des fins de communication.

◊ prop6 : une *Tclass* T peut hériter d'une classe C ssi C et tous ses ascendants satisfont les propriétés prop1 et prop2.

◊ prop7 : les relations d'association, de composition et d'héritage sont prises en compte lors de la validation ssi celles-ci concernent deux *Tclass*.

◊ prop8 : deux *Tclass* ne peuvent être mises en relation que par une seule relation UML. Si cette relation est une association, alors, elle doit attribuée avec une classe associative de type *Composer*.

◊ prop9 : une classe ou une *Tclass* peut-être composée de *Tclass* ou de classes.

◊ prop10 : la relation d'agrégation n'est pas supportée dès lors qu'une *Tclass* est impliquée dans la relation.

◊ prop11 : Supposons que la *Tclass* T1 composée de la *Tclass* T2, que T1 possède une porte publique g1 et T2 une porte publique g2. Une *Tclass* T3 qui veut communiquer avec T2 via g2 a deux solutions : soit T3 communique directement avec T2 sur la porte g2, soit T3 communique avec T1 sur la porte g1, à la condition qu'une formule OCL associe g1 et g2. Cette formule doit être attachée à la relation de composition entre T1 et T2. Elle doit alors être de la forme : *{T1.g1 = T2.g2}*.

3.1.4. Opérateurs de composition

Constatant le caractère « implicite » du parallélisme entre classes UML et l'absence d'opérateurs de composition qui font la force de langages de modélisation basés sur les algèbres de processus, TURTLE introduit la notion d'opérateur de composition par l'entremise de classes associatives attachées aux associations (bipoints) entre *Tclasses*. Les opérateurs de TURTLE natif sont les suivants : *Parallel* (Figure 5.a) pour deux classes qui s'exécutent en parallèle sans aucune communication ; *Sequence* pour deux classes qui s'exécutent séquentiellement (sur la Figure 5.b, remarquez le sens de navigation pour exprimer que T2 s'exécute suite à la terminaison de T1) ; *Synchro* (Figure 5.c) pour que deux *Tclasses* puissent se synchroniser et éventuellement échanger des données à cette occasion ; enfin, *Preemption* pour permettre à une *Tclasse* d'interrompre une autre *Tclasse*.

Note : L'opérateur de choix [] de LOTOS n'est pas repris au niveau des diagrammes de classes TURTLE. On le rencontrera uniquement dans des diagrammes d'activité. Faire un

choix relève en effet de la vue comportementale et non pas de la vue structurelle statique que donne le diagramme de classes.

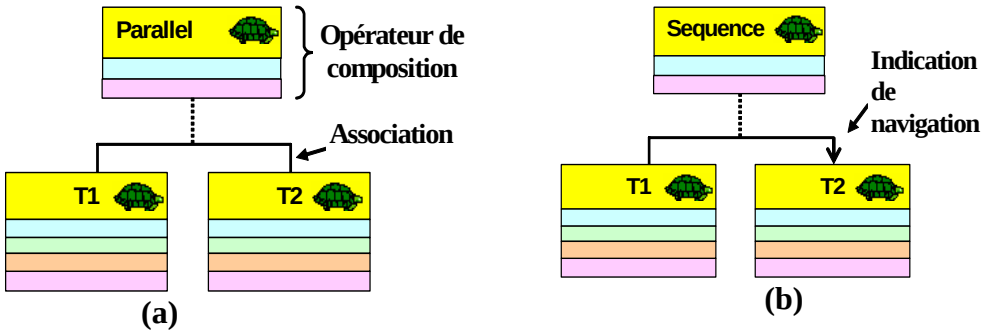


Figure 5. Introduction d'opérateurs de composition dans un diagramme de classes

Toute association entre deux *Tclasses* doit avoir une seule sémantique et ainsi n'être attribuée que par une seule classe associative de type *Composer*. Les classes suivantes héritent de *Composer* :

Parallel Les deux *Tclasses* mises en relation par une association à laquelle est attribué cet opérateur sont exécutées en parallèle et sans synchronisation. Les deux *Tclasses* doivent être des classes actives (classes ayant leur comportement propre).

Synchro Les deux *Tclasses* mises en relation par une association à laquelle est attribué cet opérateur, réalisent des synchronisations entre elles dans deux flux d'exécution séparés. Cette synchronisation peut donner lieu à un échange de données dont le format est précisé lors de l'appel, dans le diagramme d'activité. Si l'association entre les deux *Tclass* comporte un sens de navigation, alors l'échange de données ne peut se réaliser que dans le sens indiqué par la navigation.

Deux *Tclasses* doivent se synchroniser sur deux portes de type Gate, qui doivent être listées dans une formule OCL. Par exemple, supposons que les portes g1 et g2, de type *Gate* et de la *Tclass* T1, se synchronisent respectivement avec les portes g3 et g4 de type *Gate* et de la *Tclass* T2. Dans ce cas, la formule OCL qui accompagne la relation d'association entre T1 et T2 doit être *{T1.g1 = T2.g3 and T1.g2 = T2.g4}*. A chaque fois que T1 réalise une action sur g1, elle doit attendre que la classe T2 réalise une action sur g3 et réciproquement. Lorsque l'action est enfin réalisée par les deux classes, l'échange de données a lieu, et les deux classes passent à

l'instruction suivante de l'activité respective.

Invocation Considérons deux *Tclasses*, T1 et T2, mises en relation par une association dirigée de T1 à T2 et attribuée par la classe associative *Invocation*. On dit alors que T2 peut-être invoquée par T1. Tout comme dans la relation de synchronisation, une porte de chaque *Tclasses* doit être impliquée dans chaque invocation. Soit g1 (resp. g2) une porte de T1 (resp. T2). Considérons que la formule OCL suivante est associée à la relation : *{T1.g1 = T2.g2}*. Alors, quand T1 réalise un appel sur g1, elle doit attendre que T2 réalise un appel sur g2. Quand T2 réalise cet appel, les données sont échangées uniquement dans le sens de la navigation (c.à.d. de T1 à T2). T1 est bloquée jusqu'à ce que T2 réalise de nouveau un appel sur g2. Dans ce deuxième cas, un échange de données peut avoir lieu dans le sens contraire à celui indiqué par la navigation. Cette invocation est similaire à l'appel de méthode du paradigme objet. Ainsi, le code de T2 qui est invoqué par T1 est exécuté dans le flux d'exécution de T1.

Sequence Les deux *Tclasses*, mises en relation par une association à laquelle est attribué cet opérateur, sont exécutées l'une après l'autre, dans le sens donné par la navigation de l'association. Dans la relation (T1 Sequence T2), T1 doit se terminer[8] avant que T2 ne démarre. T2 est exécutée dans un nouveau flux d'exécution si T2 est une classe active.

Preemption La *Tclasses* T2, désignée par la navigation de l'association reliant deux *Tclasses* T1 et T2, et dont l'opérateur de composition associé est *Preemption*, peut interrompre définitivement et à n'importe quel instant l'autre *Tclasse*. Cette interruption se produit lorsque la première instruction de T2 est réalisable (synchronisation sur une porte par exemple). Les deux *Tclasses* doivent être des classes actives.

Remarque importante : un opérateur de composition (*Parallel*, *Synchro*, …) est lié à une association entre deux objets. Dans ce mémoire comme dans l'éditeur de diagrammes de l'outil TTool (Cf. paragraphe 5.2), nous faisons une « simplification » : lorsqu'une Tclasse n'est instanciée qu'une fois, nous assimilons l'objet qu'est cette unique instance et la

[8] On dit qu'une *Tclass* se termine lorsque toutes les activités associées à cette classe ont atteint leur point de terminaison.

Tclasse qui lui donne son type. C'est pourquoi les diagrammes TURTLE que nous avons appelé jusqu'ici « diagrammes de classes » pourraient donner l'impression que l'on compose des Tclasses alors que ce sont bel et bien les objets que l'on compose. On peut constater ce fait sur la figure 6 qui décrit le diagramme de classes d'une machine à café dans lequel la classe *Button* est instanciée par deux objets *teaButton* et *coffeButton* alors que ce même diagramme de classes inclut un seul portefeuille (classe/objet *Wallet*), un seul contrôleur (classe/objet *Wallet*) et un seul contrôleur (classe/objet *Machine*).

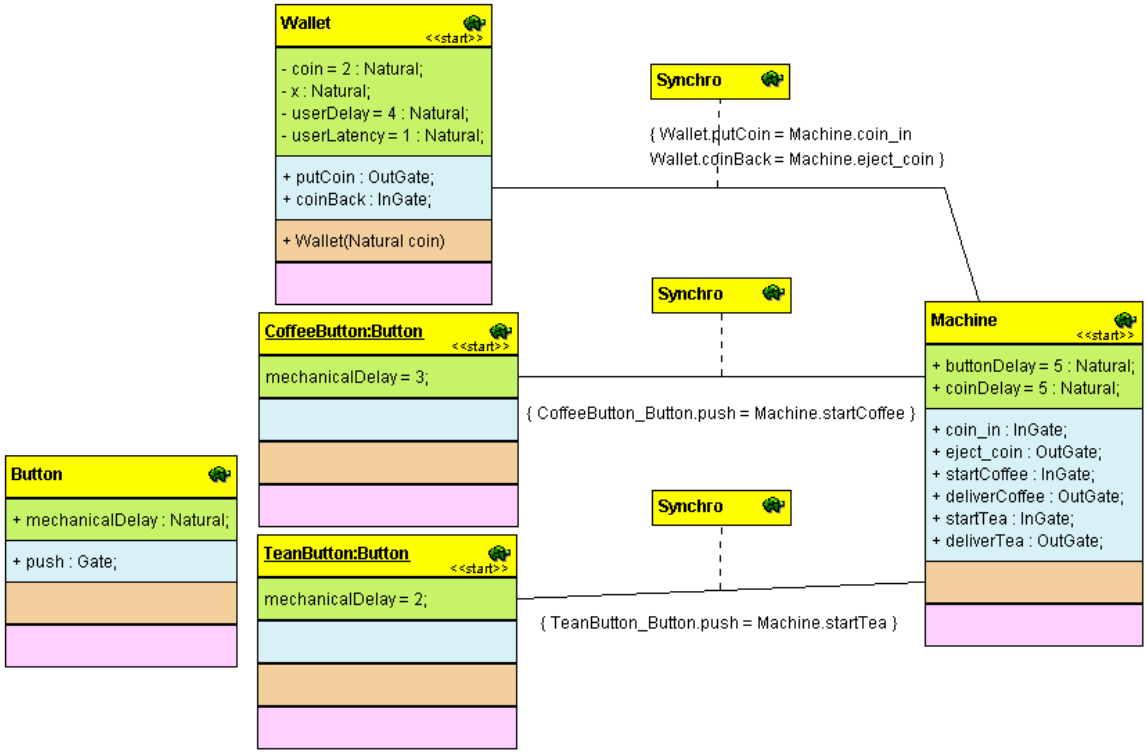


Figure 6. Diagramme de classes d'une machine à café

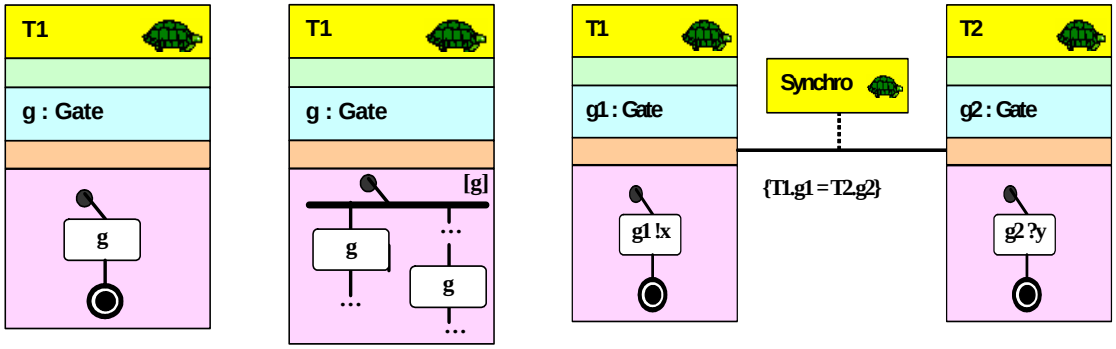
3.2. Comportement : diagrammes d'activité étendus

3.2.1. Principe

Après avoir défini l'architecture, nous en venons à décrire le comportement interne des objets. Ici, le choix a été fait de traiter ces comportements sous la forme de diagrammes d'activité. Cela revient à décrire les comportements dans un style « organigramme » où les actions sont des synchronisations sur portes avec possibles échanges bi-directionnels de données ou bien des manipulation d'attributs. Ces diagrammes d'activité prennent leur caractère « temps réel » dans l'introduction de trois opérateurs temporels. Tout d'abord un

délai déterministe correspondant à un retard fixe. Ensuite, un délai non déterministe basé sur l'opérateur *latency* de RT-LOTOS. Enfin une offre limitée dans le temps qui évite à un objet de se bloquer indéfiniment sur un rendez-vous.

L'introduction de l'opérateur *Synchro* dans le diagramme de classes a impacté sur les diagrammes d'activité par l'ajout de symboles pour la synchronisation sur porte avec réception (?) et/ou émission (!) de données. La plupart du temps, une porte sera utilisée par une Tclasse pour se synchroniser avec une autre Tclasse. Ce schéma classique est illustré par la figure 7. La deuxième utilisation d'une porte est à usage interne d'une Tclasse. Les figures 7a et 7b montrent respectivement une Tclasse qui effectue une synchronisation sur une porte g et la figure 7b montre qu'on peut synchroniser sur une porte g deux branches d'exécution internes à une Tclasse.



(a) (b) (c)

Figure 7. (a) Action interne sur g. (b) Synchronisation interne sur g.

(c) Synchronisation externe sur g1/g2.

A titre d'illustration de la syntaxe d'un diagramme d'activité TURTLE, la figure 8 décrit le comportement du contrôleur de machine à café représenté par l'objet *Machine* sur la figure 6. Ce contrôleur de machine à café attend d'avoir reçu deux pièces pour donner un choix de sélection entre café et thé avant de préparer la boisson correspondante et de revenir enfin à son état initial d'attente de pièces. Dans les rectangles d'actions apparaissent les synchronisation sur les portes *coin_in*, *eject_coin*, *start_coffee*, *deliver_coffee*, *start_tea*, et *deliver_tea*. Le losange représente le choix – purement non déterministe au niveau d'un tel modèle abstrait – entre l'appui sur le bouton du café et celui de sélection du thé.

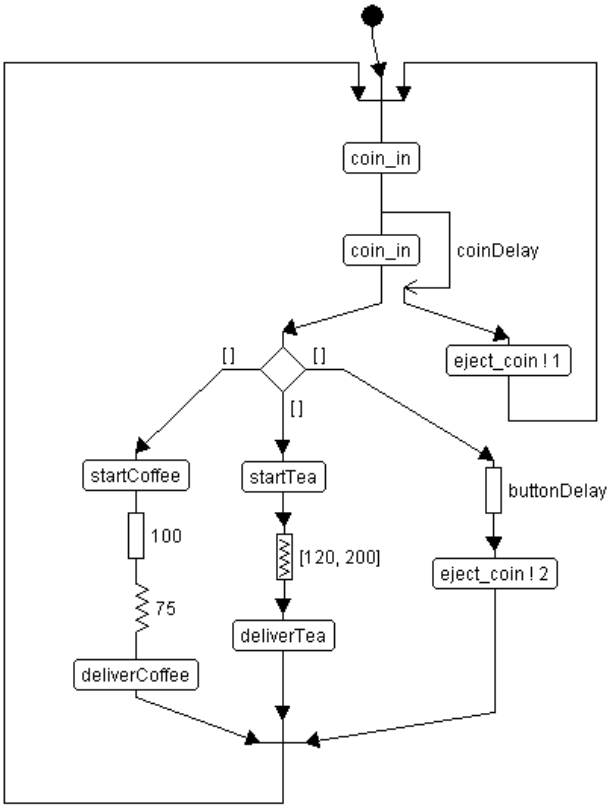
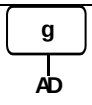
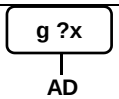
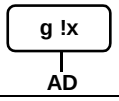


Figure 8. Diagramme d'activité d'une machine à café

Le tableau ci-dessous illustre, pour toutes les constructions des diagrammes d'activité [DOU 00], la sémantique associée par transposition en LOTOS. Soit AD la dénomination pour un diagramme d'activité, et **τ**(AD) le processus RT-LOTOS correspondant à la transcription de AD.

Diagramme d'activité TURTLE	***Description***	***Traduction LOTOS***
	Début d'un diagramme d'activité. En d'autres termes, début de la traduction.	τ(AD)
	Appel sur la *Gate* **g**. AD est ensuite interprété.	g ; τ(AD)
	Appel sur la *Gate* **g** avec réception de la valeur x. AD est ensuite interprété.	g ?x:nat ; τ(AD)
	Appel sur la *Gate* **g** avec émission de la valeur x. AD est ensuite interprété.	g !x ; τ(AD)
	Appel sur la *Gate* **g** avec émission de la valeur x et réception de la	g !x ?y:nat ; τ(AD)

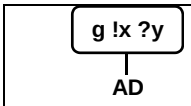
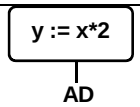
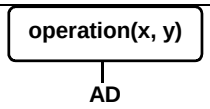
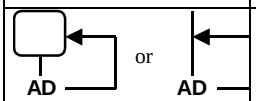
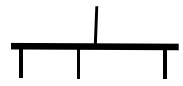
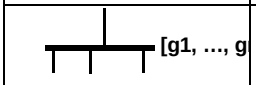
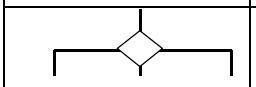
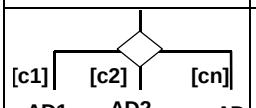
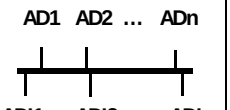
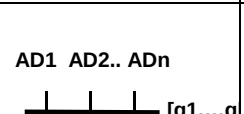
	valeur y. AD est ensuite interprété.	
	Assignation d'une valeur à un attribut. AD est ensuite interprété.	let y : YType = x*2 in τ(AD)
	Appel de méthode. AD est ensuite interprété.	τ(AD) (appel de méthode non traduit)
	Structure de type boucle. AD est interprété à chaque fois que l'interpréteur entre dans la boucle.	process LabelX[g1,…gn] : noexit := τ(AD) >>LabelX[g1,...gn] end proc;
	Parallélisme entre n sous-activités décrites par AD1, AD2, …, ADn. A des fins de documentation, les sous-activités peuvent être séparées par des *swinlanes*.	τ(AD1) ||| τ(AD2) ||| … ||| τ(ADn) (*swinlanes* non traduites)
	Synchronisation sur les portes g1, g2, …, gm entre n sous-activités décrites par AD1, AD2, …, ADn.	τ(AD1) |[g1,…gm]| τ(AD2) |[g1,…gm]| … |[g1,…gm]| τ(ADn)
	Sélection de la première activité prête à s'exécuter parmi les n activités décrites par AD1, AD2, … , ADn.	τ(AD1) [] τ(AD2) [] ... [] τ(ADn)
	Sélection de la première activité prête à s'exécuter et dont la garde est vraie, parmi les n activités décrites par AD1, AD2, … , ADn.	[c1] -> τ(AD1) [] [c2] -> τ(AD2) [] … [] [cn] -> τ(ADn)
	Une fois que toutes les activités AD1, AD2, …, ADn ont terminé leur exécution, les m activités décrites par AD'1, AD'2, …, AD'm sont exécutées en parallèle (entrelacement).[9]	(τ(AD1) ||| τ(AD2) ||| … τ(ADn)) >> (τ(AD'1) ||| τ(AD'2) ||| … τ(AD'm))
	Une fois que toutes les activités AD1, AD2, …, ADn ont terminé leur exécution, les m activités décrites par AD'1, AD'2, …, AD'm sont exécutées en synchronisation sur les k portes g1, g2, …, gk.	(τ(AD1) ||| τ(AD2) ||| … τ(ADn)) >> (τ(AD'1) |[g1, …gk]| τ(AD'2) |[g1, …gk]| … τ(AD'm))
	Terminaison d'une activité	Exit

Tableau A. Opérateurs non temporels des diagrammes d'activité TURTLE

[9] Si le cardinal des ADi vaut 1 et que le cardinal des AD'j vaut aussi 1, alors une simple flèche est utilisée entre AD1 et AD'1.

Le tableau qui suit présente les pictogrammes retenus pour les opérateurs temporels. S'y ajoutent un opérateur de délai applicable à un intervalle temporel, cumulant ainsi les effets d'une durée fixe égale à la borne inférieure de l'intervalle et une latence égale à la différence entre les bornes supérieure et inférieure de l'intervalle.

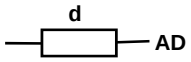
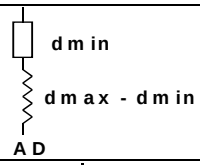
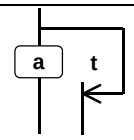
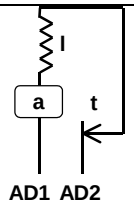
Opérateurs temporels TURTLE	***Description***	***Traduction RT-LOTOS***
	Retard déterministe. AD est interprété après d unités de temps.	**delay**(d) τ(AD)
	Retard non déterministe. AD est interprété après au plus t unités de temps.	**latency**(t) τ(AD)
	Délai non déterministe. AD est interprété après au moins dmin et au plus dmax unités de temps.	**delay**(dmin,dmax) τ(AD)
	Offre limitée dans le temps. L'offre sur la *Gate* **a** est offerte pendant une période inférieure ou égale à **t**. Si l'offre est réalisée, alors AD1 est interprété, sinon AD2 est interprété.	**a**{t, τ(AD2)}; τ(AD1)
	Offre limitée dans le temps. L'offre sur la *Gate* **a** est offerte pendant une période inférieure ou égale à **t**. A noter que la latence commence en même temps que la limitation sur l'offre. Si l'offre est réalisée, alors AD1 est interprété, sinon AD2 est interprété.	**latency**(l) **a**{t, τ(AD2)}; τ(AD1)

Tableau B. Opérateurs temporels des diagrammes d'activité TURTLE

3.3. Sémantique formelle

Nous avons déjà mentionné le fait que la sémantique du profil TURTLE natif est exprimée en RT-LOTOS [COU 00]. En pratique, toute modélisation TURTLE peut être traduite en une spécification RT-LOTOS par utilisation des algorithmes définis dans [LOH 02]. De nouveaux opérateurs de composition pourront être ajoutés ultérieurement aux diagrammes de classes TURTLE. Leur sémantique sera alors donnée en TURTLE natif qui a lui-même une sémantique exprimable en RT-LOTOS. Cette approche a déjà été

expérimentée [LOH 03] pour introduire les opérateurs *Periodic* et *Suspend* qui permettent de décrire des tâches périodiques et des opérations de suspension/reprise de tâches. Mais revenons pour l'instant à TURTLE natif.

Une modélisation TURTLE structure un système en *Tclasses* et associe un diagramme d'activité à chaque *Tclasse*. L'algorithme de traduction des diagrammes TURTLE en RT-LOTOS comporte deux étapes [LOH 02]. Tout d'abord, un processus RT-LOTOS est construit pour chaque diagramme d'activité. Ensuite, les processus RT-LOTOS issus de la première étape sont composés entre eux en utilisant l'information fournie par le diagramme de classes. Cet algorithme en deux étapes peut être esquissé comme suit:

1. **Diagrammes d'activité.** Chaque *Tclasse* contient un diagramme d'activité qui est traduit en un processus RT-LOTOS. Ce processus peut lui-même être composé de sous-processus qui implémentent un opérateur, une boucle, un choix (gardé ou non), une jonction, une action, la modification d'attributs ou une structure parallèle/synchronisation. Les opérateurs temporels de TURTLE (délai déterministe, délai non déterministe, offre limitée dans le temps et enregistrement de date) ainsi que la synchronisation sur des portes présentent une traduction directe en RT-LOTOS. A contrario, certains opérateurs tels que le parallélisme et la synchronisation entre sous-activités nécessitent une traduction faisant appel à des synchronisations multiples et à la création de sous-processus.
2. **Diagrammes de classes.** Les associations entre *Tclasses* peuvent être attribuées avec des opérateurs de composition. Soit par exemple *T1* une *Tclasse* impliquée dans une relation attribuée par un opérateur de composition, et *P1.1* le processus RT-LOTOS obtenu à partir de la traduction du diagramme d'activité de *T1* à l'étape 1. À l'étape 2, un nouveau processus RT-LOTOS *P1.2* prenant en compte les opérateurs de composition impliquant *T1* est créé. Le corps du processus *P1.2* peut appeler *P1.1* en utilisant des portes RT-LOTOS appropriées (déclaration ou renommage). Il peut également appeler d'autres processus *Pn.2* où le *Tn* désigne une autre *Tclasse* liée à *T1* par un opérateur de composition TURTLE. Considérons un diagramme de classes, les *Tclasses* sont traitées dans l'ordre suivant, déterminé par notre pratique de la modélisation en TURTLE :
    a. *Tclasses* a l'origine d'opérateurs *Preemption*,
    b. *Tclasses* a l'origine d'opérateurs *Sequence*,
    c. *Tclasses* pointées par des opérateurs *Preemption* ou *Sequence*,
    d. *Tclasses* a l'origine a la fois d'opérateurs *Sequence* et *Preemption*, et

e. Les sous-ensembles de *Tclasses* reliées entre elles par des opérateurs *Parallel* ou *Synchro*.

Considérons par exemple trois *Tclasses*, *T1*, *T2* et *T3*. Les *Tclasses T2* et *T3* se synchronisent sur *g*, et il y a deux relations de *Sequence* orientées de *T1* vers *T2*, et de *T1* vers *T3*. L'algorithme de traduction produit six processus : *P1.1*, *P1.2*, *P2.1*, *P2.2*, *P3.1* et *P3.2*. Les processus *P1.1*, *P2.1* et *P3.1* contiennent la traduction du diagramme d'activité de *T1*, *T2* et *T3*, respectivement. Les processus *P2.2* et *P3.2* se contentent juste d'appeler *P2.1* et *P3.1*, respectivement. Cependant, le corps de *P1.2* est plus complexe. En effet, après exécution de son activité, *T1* exécute *T2* et *T3* conjointement, *T2* et *T3* étant synchronisées sur la porte *g*. Ainsi, le corps de *P1.2* est :

P1.1 > > (P2.2|[g]|P3.2).

Pour finir, pour chaque instance de *Tclasse Tn* désignée comme active au démarrage du système, la spécification RT-LOTOS instancie le processus *Pn.2*.

3.4. Extensions orientées « système d'exploitation »

Si TURTLE répond aux lacunes d'UML en termes de description du comportement temps-réel d'une classe, ce profil ne propose pas d'opérateurs spécifiques temps-réel au niveau « structuration » (c'est-à-dire le diagramme de classes). Pour remédier à ce problème, TURTLE a été étendu pour exprimer explicitement le comportement périodique et l'ordonnancement des tâches au sein d'un système temps-réel [LOH 03]. Pour cela, deux opérateurs de composition ont été introduits, à savoir *Periodic* et *Suspend / Resume*.

L'opérateur *Periodic* permet de caractériser, d'une part, le comportement répétitif d'une tâche selon une période donnée et, d'autre part, une limite maximale de temps pour l'exécution de chaque période de cette tâche. Cet opérateur s'applique soit à deux *Tclasses* (cf. figure 9.a), soit à une *Tclasse* (cf. figure 9.b).

Dans le premier cas, la tâche T2 ne démarre qu'après terminaison de la tâche T1 ; le comportement de T2 est périodique (la période est précisée dans la formule OCL qui accompagne la relation de composition). Ainsi, lorsque T1 achève son exécution, la tâche T2 de la Figure a admet une période de 100 unités de temps et une limite de temps de 50 unités de temps pour chaque exécution de son diagramme d'activité.

Dans le deuxième cas, la classe T3 admet un comportement périodique dès qu'elle est démarrée, peu importe l'objet qui l'instancie.

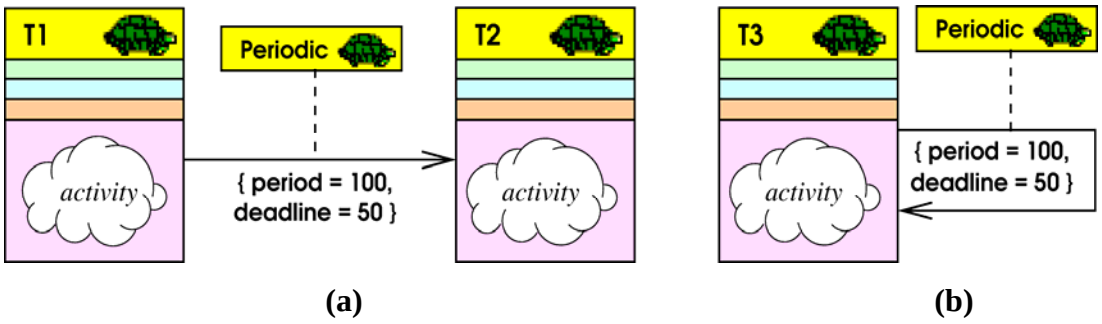
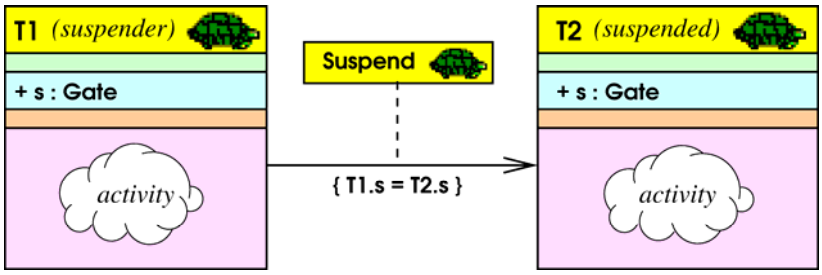


Figure 9. Opérateur Periodic, (a) entre deux *Tclasses*, (b) pour une seule *Tclasse*

L'opérateur *Suspend / Resume* permet d'exprimer la suspension d'une tâche par une autre. Sur la figure 10, la classe T1 peut suspendre l'activité de la classe T2 par un appel sur la porte s. La classe T2 est alors dans l'état « suspendu ». Un deuxième appel sur la porte s de la part de la classe T1 permet de réactiver la classe T2 dans l'état d'exécution dans lequel elle se trouvait avant suspension. Son état est alors qualifié d'« actif ».



f 10. Relation de suspension entre deux *Tclasses*

Appliqué plusieurs fois et à plusieurs Tclasses distinctes, ce mécanisme de suspension permet de modéliser le comportement d'ordonnancement d'un système Par exemple, sur la figure 11 une classe *TaskManager* suspend et active alternativement deux autres Tclasses, *TaskA* et *TaskB*, par appel sur les portes *switchA* et *switchB*, respectivement.

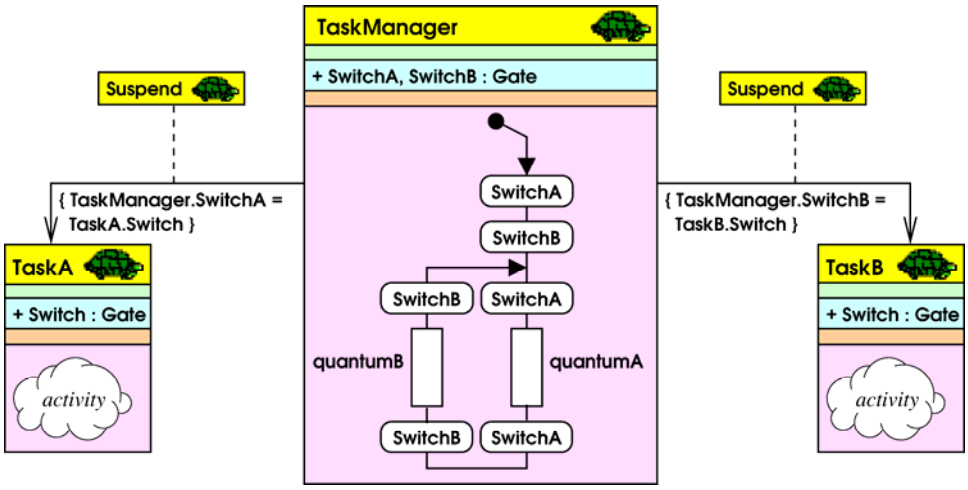
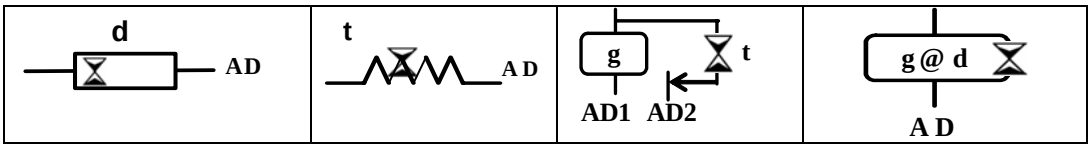


Figure 11. Modélisation d'un séquenceur de tâches avec l'opérateur Suspend

L'utilisation d'un opérateur permettant la suspension d'une *Tclasse* soulève le problème de la sémantique des opérateurs temporels internes aux *Tclasses*. En effet, ces derniers ont été définis avec l'hypothèse d'un temps universel qui ne peut être interrompu. Aussi, les opérateurs temporels de TURTLE natif sont utilisés pour modéliser des mécanismes temporels tels que des temporisateurs mais également pour modéliser le profil temporel de comportements logiques, par exemple, pour modéliser le temps de traitement d'un algorithme. Si le délai d'un temporisateur doit continuer de s'écouler lorsque sa *Tclasse* est interrompue, a contrario, le délai modélisant l'exécution d'un algorithme doit être interrompu. Ce problème nous a amené à rendre les opérateurs temporels de TURTLE natif non interruptibles, et à définir de nouveaux opérateurs temporels interruptibles. Ces derniers se distinguent graphiquement des opérateurs originaux par la présence d'un sablier (cf. figure). Pour les trois premiers opérateurs de la figure, l'écoulement du temps est interrompu lorsque la classe appelant l'opérateur est elle-même interrompue. Pour le dernier opérateur (relevé de temps interruptible), le délai d correspond au temps écoulé par la *Tclasse* dans l'état actif entre la sensibilisation de l'action g et sa réalisation.



Délai déterministe interruptible	Délai non déterministe interruptible	Offre limitée dans le temps interruptible	Relevé de temps interruptible
(a)	**(b)**	**(c)**	**(d)**

Figure 12. Opérateurs temporels interruptibles

3.5. TURTLE-P : une extension orientée « systèmes distribués »

TURTLE natif autorise une conception préliminaire relativement abstraite basée sur les diagrammes de classes et d'activité. L'objectif de l'extension TURTLE-P initialement présentée à CFIP'03 [ASK 03] et stabilisée par la suite [ASS 04] est de prendre en compte les diagrammes de composants pour regrouper des Tclasses, mais aussi les diagrammes de déploiement pour conforter l'applicabilité du profil aux systèmes distribués. Mieux encore, TURTLE-P permet d'associer des paramètres de qualité de service (délai de transmission, bande passante, taux de perte) aux liens entre nœuds d'un diagramme de déploiement. Les diagrammes de déploiement se voient dotés de « sondes » (*probes*) permettant d'observer les liens entre les nœuds.

TURTLE-P est formellement défini par des algorithmes de traduction vers TURTLE natif. Cette extension n'est pas encore implantée dans les outils TURTLE. C'est pourquoi nous avons choisi de présenter TURTLE-P en annexe de ce mémoire.

4. Diagrammes d'analyse et synthèse de conceptions TURTLE

4.1. Principe général

La méthodologie habituellement supportée par TURTLE repose sur la réalisation d'une conception TURTLE constituée de diagrammes de classes et d'activité qui décrivent la structuration du système et son comportement. A ce jour, l'analyse du système pouvait se réaliser à l'aide de diagrammes de séquences [ASK 04] qui, dépourvus de sémantique formelle, se voyaient cantonnés à un rôle de documentation. Les travaux présentés en (ASK 05] ont amené les avancées suivantes :

- Tout d'abord, la synthèse automatique d'une conception TURTLE à partir des

diagrammes SD, IOD et CSD. Cette synthèse a pour objectif de générer un système sous la forme de diagrammes de classes et d'activité avec un comportement équivalent en traces à celui décrit par les trois diagrammes SD, IOD et CSD lorsque certaines conditions sont vérifiées [ASK 05].

- Par là même, cette synthèse nous permet de donner une sémantique formelle aux IOD, CD et CSD. Notons au passage que nous avons enrichi ces diagrammes avec des opérateurs spécifiquement dédiés aux systèmes temps-réel et aux protocoles.

Le principe de la synthèse est le suivant (cf. figure 13). L'analyste du système décrit les différents scénarios de fonctionnement du système à l'aide d'un IOD et de plusieurs SD. Les SD représentent les scénarios en eux-mêmes alors que l'IOD modélise les enchaînements possibles entre ces scénarios. La synthèse automatique génère à partir de cet ensemble de diagrammes une conception TURTLE, c'est à dire un ensemble de diagrammes TURTLE constitué d'un diagramme de classes (appelé CD sur les figures) et d'un ou plusieurs diagrammes d'activité (AD). En outre, l'utilisateur peut guider la génération de cette conception en fournissant un diagramme de structure composite pour décrire les canaux de communication entre instances. Le fait de donner une sémantique par défaut à l'architecture de communication rend cette étape facultative.

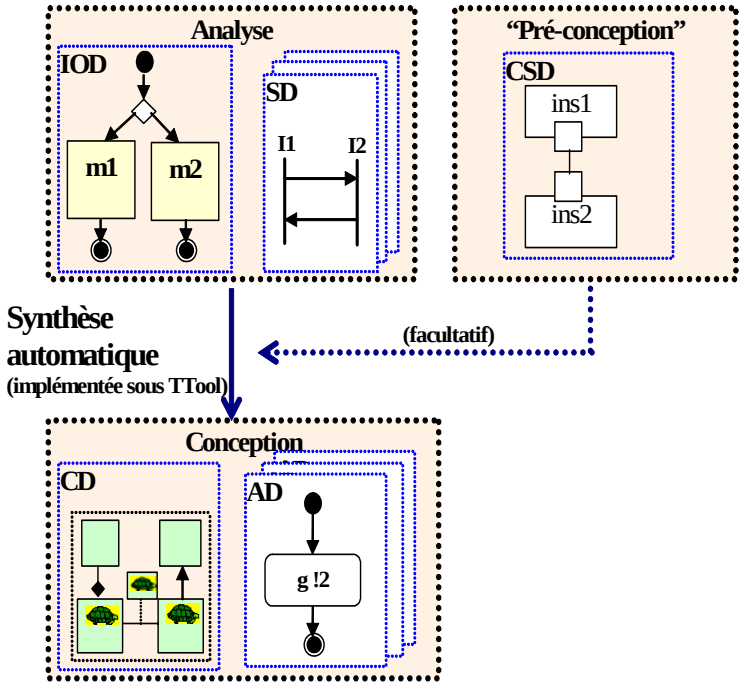


Figure 13. Principe méthodologique de la synthèse d'une conception TURTLE

4.2. Diagrammes d'analyse

4.2.1. Diagramme d'interactions

Les diagrammes d'interactions permettent de conduire une analyse du comportement d'un système à un haut niveau d'abstraction. Ces IOD apparus avec la norme UML 2.0 [OMG 03a] sont relativement similaires à des diagrammes d'activité. Ils supportent des opérateurs de choix, de parallélisme, de synchronisation et de boucle en y ajoutant les références à des scénarios.

Au niveau de l'extension orientée « analyse » du profil TURTLE discutée dans cet article, nous reprenons les opérateurs UML 2.0 et ajoutons un opérateur de préemption intéressant pour décrire des scénarios susceptibles d'en interrompre d'autres à tout instant. Par exemple, lors de la description d'un scénario modélisant un protocole d'échange de données en mode connecté, cet échange peut-être interrompu à tout moment par un scénario décrivant la libération de la connexion (cf. figure 14).

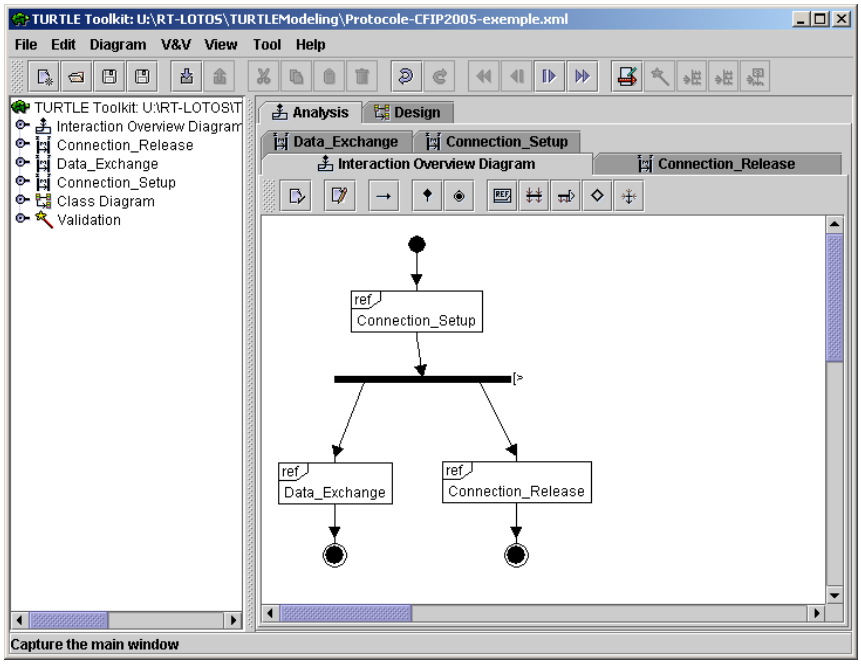


Figure 14. Exemple d'un diagramme d'interactions (réalisé avec TTool)

4.2.2. Diagramme de séquences

Les diagrammes de séquences UML 2.0 enrichissent nettement ceux de la norme 1.5. Outre les classiques messages synchrones et asynchrones, et les opérateurs de boucle et d'alternative, ces diagrammes supportent à présent des opérateurs temps-réel permettant

d’une part, d’associer à un événement un intervalle de dates absolues d’occurrence de cet événement et d’autre part, d’associer à deux événements un intervalle de temps représentant les bornes inférieures et supérieures de temps pouvant s’écouler entre l’occurrence de ces deux événements. Ces opérateurs sont issus de la norme dite MSC’2000 [MSC 99].

Du point de vue TURTLE, nous avons repris la majorité des opérateurs logiques de ces diagrammes UML 2.0, notamment les opérateurs d’envoi et de réception de messages asynchrones et la synchronisation. Nous ne supportons pas à ce jour certains opérateurs logiques tels que les références internes à d’autres scénarios (les problèmes de sémantique qui s’y rattachent seront traités ultérieurement). En effet, dans un premier temps, nous avons tout particulièrement mis l’accent sur les opérateurs temps-réel : nous supportons les opérateurs de temps absolu et relatif, et nous avons en outre intégré explicitement la notion de « timers » au niveau de ces diagrammes.

La figure 14 illustre l’utilisation de plusieurs de ces opérateurs temporels. L’émission de « data_req2 » a lieu exactement 1 unité de temps (date absolue) après le lancement de l’application. Par la suite, la deuxième émission du message « data » a lieu entre 5 et 10 unités de temps après la première émission du message « data ». De plus, l’instance « client » positionne un timer nommé « timer1 » à 30 unités de temps. Dans ce scénario, « server » envoie à « client » des messages « data » qui parviennent avant l’expiration du timer.

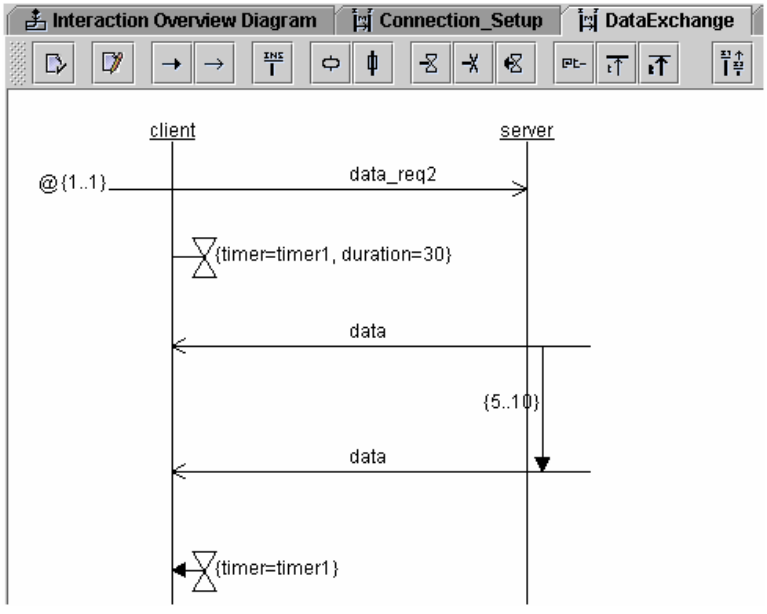


Figure 15. Exemple d'un diagramme de séquences (réalisé avec TTool)

4.2.3. Diagramme de structure composite

Le nouveau diagramme UML 2.0 de structure composite a pour objectif la description des canaux de communication entre des « parts ». Ces « parts » représentent un niveau d'abstraction intermédiaire entre les classes et les objets : ce sont les abstractions des composées d'une classe. Le diagramme de structure composite permet la description de ces parts ainsi que la description des canaux de communication entre ces parts. Ces canaux connectent des ports de communication qui sont attribués aux parts. En outre, chaque port peut se voir associer une liste de signaux entrants et sortants (interfaces requises et fournies par le port).

Dans le cas de la synthèse de conception TURTLE, l'objectif est de préciser la nature des liens de communication entre les instances modélisées au niveau des scénarios. Notamment, il s'agit de préciser si les signaux échangés entre une instance et une autre transitent sur les mêmes canaux (architecture dite mono-canal) ou sur des canaux distincts (architecture dite multi-canaux).

D'un point de vue TURTLE, nous supportons la déclaration de parts (que nous assimilons aux instances). Ces parts peuvent être connectées par des canaux de communication eux-mêmes reliés à des ports. Les messages émis par une instance doivent être déclarés, comme UML 2.0 le préconise, dans une interface fournie (demi-cercle) alors que les messages reçus par une instance doivent être déclarées dans une interface requise (disque). Par ailleurs, afin de renforcer l'analyse temps-réel du système avec TURTLE, nous donnons la possibilité d'attribuer les canaux de communication avec un intervalle de temps représentant le délai minimal et maximal de transmission d'un message sur un lien. Notons aussi que lorsque les canaux de communication visés répondent à la sémantique donnée par défaut lors de la synthèse (une sémantique plus complète de ces canaux est précisée dans la section suivante) alors la réalisation d'un diagramme de structure composite est facultative.

A titre d'exemple, la Figure 16 représente une analyse TURTLE constituée d'un diagramme d'interactions, d'un diagramme de séquences et d'un diagramme de structure composite (nommé Archi1). Notons le canal de communication entre I1 et I2 dont le délai de transmission de I1 au buffer de sortie du canal est de [3, 4] unités de temps. Le canal peut transmettre des messages « a » accompagnés d'un entier et des messages « b » accompagnés d'un entier et d'un booléen. Notons aussi que l'interconnexion entre les deux ports p1 et p2 peut avoir lieu car leurs interfaces sont compatibles ; en effet, l'interface requise par p2 est identique à l'interface fournie par p1.

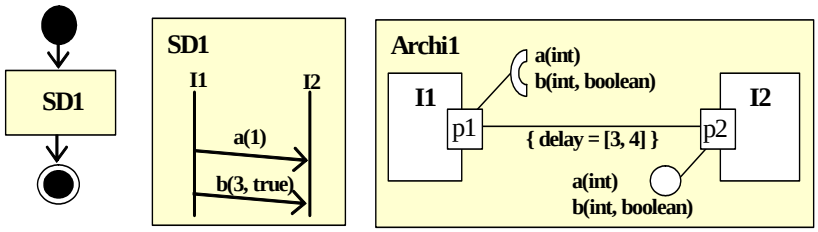


Figure 16. Analyse TURTLE comprenant un diagramme de structure composite

4.3. Sémantique

4.3.1. Principe general

La sémantique d'une analyse TURTLE constituée des trois diagrammes UML précédemment décrits est donnée sous forme d'une conception TURTLE. L'idée est de proposer des algorithmes capables de construire une conception TURTLE à partir des diagrammes d'interactions et de séquences, et par utilisation d'un diagramme de structure composite. Cette conception TURTLE formée d'un diagramme de classes et de diagrammes d'activité a une sémantique formelle en RT-LOTOS [ACLS 04].

Notons que la synthèse d'une conception à partir d'une modélisation sous forme de scénarios n'est pas toujours possible [KGBG 98].

4.3.2. Algorithmes

Les algorithmes de synthèse sont relativement complexes en raison de la nature très différente des diagrammes UML d'analyse et de conception. L'objectif de cet article n'est pas de donner une vue exhaustive de ces algorithmes[10] mais plutôt de donner la philosophie générale d'utilisation et de transformation des différents éléments graphiques des diagrammes considérés.

L'algorithme comporte les étapes fondamentales suivantes :

1. Pour chaque instance distincte, une classe TURTLE est générée. Si I_i est une instance d'un SD, alors soit T_i la Tclasse associée.
2. La traduction des messages synchrones ne pose pas de difficulté : les messages synchrones échangés entre instances sont modélisés par des opérateurs de

[10] L'implantation comporte plus de 6000 lignes de code Java.

composition « Synchro » qui attribuent les associations entre les Tclasses correspondant aux instances.

3. La traduction des messages asynchrones est moins immédiate. Ainsi, pour chaque message asynchrone « m_k » émis entre deux instances I_i et I_j, un canal de communication est généré entre T_i et T_j (mode par défaut en l'absence de diagramme de structure composite). Pour cela, deux Tclasses Tin_ij_mk et Tout_ij_mk intermédiaires sont utilisées entre T_i et T_j. Tin_ij_mk se synchronise avec T_i et Tout_ij_mk, et Tout_ij_mk avec Tin_ij_mk et T_j. Tin_ij_mk modélise l'émission du message sur le canal et Tout_ij_mk modélise le buffer d'arrivée. Notons que les messages sont ordonnés sur le même canal. Dans le cas où un diagramme de structure composite est fourni, une autre méthode de génération des canaux est utilisée. Si le canal est partagé et s'il est déjà créé, on réutilise une émission / réception sur ce canal. Si un tel canal n'est pas déjà créé, la classe intermédiaire représentant la sémantique de ce canal est générée. Par défaut, la synthèse utilise des canaux totalement ordonnés, sans délai, avec buffer de taille « infinie » à l'arrivée.
4. Chaque timer est modélisé par une Tclasse qui offre trois portes de synchronisation : *set*, *reset*, et *exp* (pour expiration) selon la définition des timers dans le profil UML temps-réel défini par l'OMG [OMG 03b]. Chaque timer se synchronise avec la ou les Tclasses qui utilisent les fonctions *set*, *reset* et *exp* de ce timer.
5. A l'instar des timers, les contraintes de temps sont chacune modélisées par une Tclasse. Dans le cas d'une contrainte de temps absolue @[t1, t2], la Tclasse offre une action sur une porte de synchronisation nommée « end_tc » entre t1 et t2. Pour faire simple, la Tclasse de l'instance ayant cette contrainte de temps doit se synchroniser sur « end_tc » avant de pouvoir exécuter l'action sur laquelle cette contrainte de temps est spécifiée. Dans le cas d'une contrainte de temps relative, la Tclasse modélisant la contrainte offre deux portes de synchronisation « begin_tc » et « end_tc ». Lorsque la contrainte relative de temps commence, une synchronisation doit être réalisée avec la Tclasse de cette contrainte sur la porte « begin_tc ». De même que pour les contraintes absolues de temps, la synchronisation sur la porte « end_tc » n'est offerte que pendant le bon intervalle temporel.
6. Le comportement interne des Tclasses T_i est construit comme suit. Tout d'abord, pour chaque evt_{ij} de l'instance I_i, un sous-diagramme d'activité ad_{ij} est construit. Ces sous-diagrammes d'activité sont interconnectés comme suit. Si deux evts evt_{ij} et evt_{ik} sont ordonnés dans un même diagramme de séquences, alors un lien est établi entre les

deux sous-diagrammes ad_{ij} et ad_{ik}. Sinon, si ces deux événements ne sont pas ordonnés (corégion, scénarios distincts) les sous diagrammes de ces événements sont reliés par un opérateur de parallélisme. De plus, les interconnexions entre les sous-diagrammes tiennent compte des relations d'ordre spécifiées au niveau des diagrammes d'interactions.

5. La chaîne d'outils TURTLE

Nombreux sont les outils UML qui permettent d'éditer des diagrammes de classes et d'activité au besoin étendus par des stéréotypes. Ces outils suffiraient à produire des diagrammes TURTLE à des fins documentaires. Or, les attentes vis-à-vis du profil TURTLE se situent avant tout dans un support à la validation d'architectures de communication et plus généralement de confrontation de modèles de systèmes temps réel à une série d'exigences logiques et temporelles.

De manière très pragmatique, nous avons cherché à réutiliser l'existant. C'est ainsi que nous avons adossé le profil TURTLE au langage formel RT-LOTOS et avons réutilisé l'outil de validation formelle RTL développé au LAAS-CNRS. Cela supposait de développer en amont de RTL un *front end* UML qui permette d'éditer des diagrammes TURTLE et de générer une spécification RT-LOTOS qui puisse être validée par RTL. L'idée principale derrière cette opération était et demeure de rendre l'usage de RT-LOTOS aussi transparent que possible pour l'utilisateur/trice qui ne doit voir, d'une part, que le modèle TURTLE qu'il/elle a conçu et, d'autre part, les résultats de simulation et vérification. Ces résultats doivent être présentés de manière avenante et très facilement exploitable même par qui ne connaît pas les arcanes du formalisme RT-LOTOS. Le besoin ici exprimé trouve une réponse dans l'outil TTool (TURTLE Toolkit [TTOOL]) développé par Ludovic Apvrille à l'ENST-SophiaAntipolis.

5.1. Les outils disponibles au préalable : RTL et Aldébaran

5.1.1. RTL

RTL (*RT-LOTOS Laboratory* © LAAS-CNRS [RTL]) admet en entrée une description RT-LOTOS et propose deux approches de validation complémentaires : la simulation

intensive qui explore partiellement l'espace d'état du système modélisé et l'analyse d'accessibilité. Cette analyse d'accessibilité s'applique aux systèmes bornés et de taille « raisonnable ». Des interfaces développées pour les outils KRONOS et Aldébaran permettent d'exploiter le graphe construit par RTL à des fins de *model checking* et de minimisation, respectivement.

L'analyse d'accessibilité d'une description RT-LOTOS passe par la compilation de cette dernière en un DTA (Dynamic Time Automaton) [COU 95]. Ce dernier est un automate temporise étiqueté qui opère une distinction entre actions urgentes et non urgentes. Le DTA sert de point de départ à la génération du graphe d'accessibilité préservant les propriétés CTL (Computational Tree Logic) [YAN 93]. Un nœud (ou classe) définit à la fois un état de contrôle et une région représentée par un polyèdre convexe dont la dimension est égale au nombre d'horloges de l'état de contrôle. Un arc correspond soit à l'occurrence d'une action RT-LOTOS, soit à une progression du temps (arc étiqueté par *t*).

Enfin, RTL permet de synthétiser à partir du graphe d'accessibilité, un automate d'ordonnancement (TLSA – *Timed Labeled Scheduling Auotmaton* [LOH 02]) qui donne un échéancier d'implantation dans lequel chaque état dispose d'une horloge propre et unique.

5.1.2. Aldébaran

Aldébaran [CADP] propose une boite à outils pour les systèmes de transitions étiquetées et permet, d'une part, la minimisation d'un STE par rapport à une relation d'équivalence et, d'autre part, la comparaison de deux STE par rapport à une équivalence. Dans ce mémoire, nous ne considérerons que les opérations de minimisation.

5.2. Ttool : un front-end UML pour RTL

TTool (*TURTLE Toolkit* © Telecom-Paris) est un outil *Open Source* qui inclut un éditeur graphique de diagrammes TURTLE archivés dans un format XMI, un analyseur syntaxique, un générateur de code RT-LOTOS et, enfin, des outils de visualisation et d'analyse des résultats de simulation et de validation retournés par RTL. TTool est entièrement programmé en Java et est indépendant de tout autre outil UML industriel ou académique. Initialement focalisé sur l'étape de conception et sa validation par combinaison de simulation et analyse d'accessibilité [ACL 04], TTool a été étendu pour offrir la vérification par abstraction. La version 2005 de l'outil ne se limite plus aux diagrammes de

classes et d'activité de la phase de conception ; elle supporte également des diagrammes d'interaction (*Interaction Overview Diagrams*) et de séquences [ASK 05].

La version 0.82 de l'outil TTool permet d'aborder la modélisation d'un système temps réel soit par une analyse (diagramme d'interaction et de séquences), soit par une conception (diagrammes de classes et d'activité). A partir des diagrammes d'analyse, TTool peut générer soit directement un code RT-LOTOS prêt à être validé en utilisant RTL, soit une conception TURTLE synthétisée sous forme d'un diagramme de classes et de diagrammes d'activité. Notons que si l'on choisit de synthétiser une conception TURTLE à partir des diagrammes d'analyse, les diagrammes de classes et d'activité générés peuvent être validés après traduction en RT-LOTOS. Dans tous les cas de figure, on en vient à utiliser (de manière transparente pour l'utilisateur de TTool) l'outil RTL et, pour la vérification par abstraction, les outils RTL et Aldébaran. L'éventail de ces possibilités apparaît sur la figure qui suit.

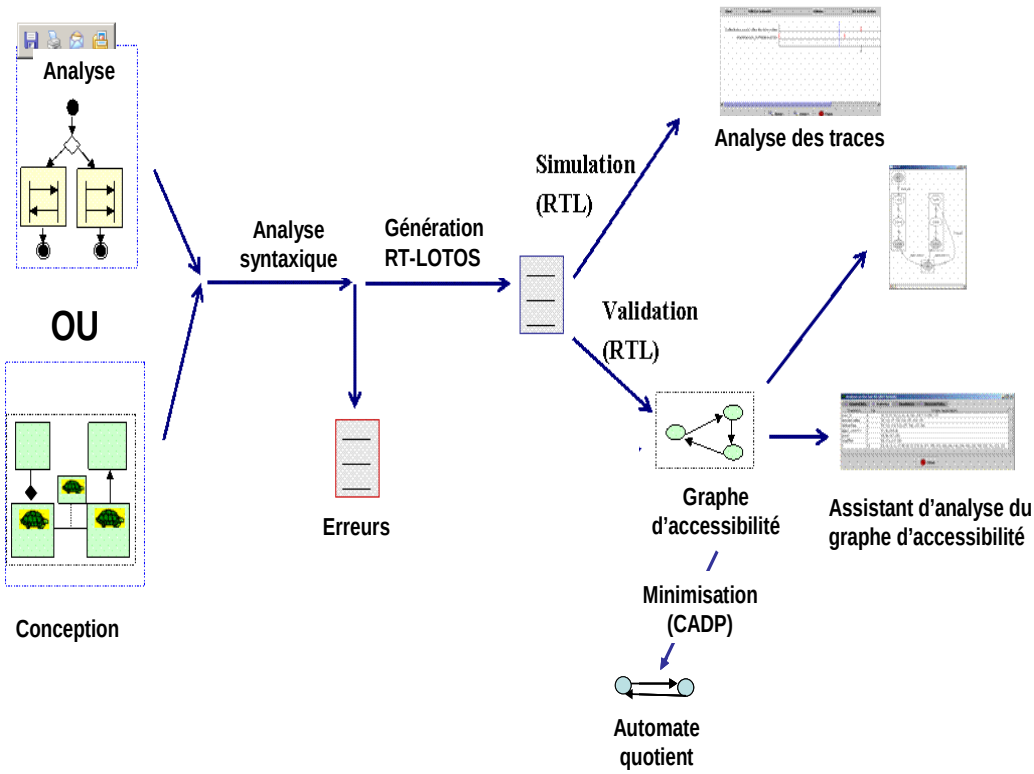


Figure 17. Que peut-on faire avec la chaîne d'outils TTool-RTL-Aldébaran ?

L'interface entre les outils TTool et RTL permet à un(e) ingénieur(e) de générer des traces de simulation ou un graphe d'accessibilité de façon transparente depuis TTool, c'est à dire sans écrire ou examiner une seule ligne de code RT-LOTOS.

TTool facilite la phase de validation en établissant une correspondance entre les actions des traces de simulation ou du graphe d'accessibilité et les actions du modèle TURTLE.

En termes de vérification, le rôle de RTL est de construire un automate temporisé et un graphe d'accessibilité. Ces derniers peuvent être exploités par des outils externes. Citons deux exemples. Tout d'abord, Kronos (KRONOS) pour vérifier sur un automate temporel dérivé du DTA la satisfaction de formules logiques décrivant les propriétés requises. Ensuite, Aldébaran (CADP) qui permet de minimiser le graphe d'accessibilité par rapport à une relation d'équivalence, ou d'effectuer des bisimulations entre deux graphes engendrés par RTL.

5.2.1. Simulation à base d'observateurs

Pour faciliter l'analyse, TTool permet d'adjoindre au diagramme de classes des observateurs modélisés par des classes TURTLE non intrusives. La recherche des transitions d'erreur de ces observateurs au niveau des traces de simulation ou du graphe d'accessibilité permet de mettre en évidence la violation de la propriété observée. Cette technique a été exploitée pour prouver la continuité de service dans une procédure de reconfiguration dynamique de logiciel embarqué à bord de satellite [ASS 04].

5.2.2. Vérification par abstraction

Sous le vocable « vérification par abstraction » nous entendons la possibilité pour le concepteur d'extraire du graphe d'accessibilité caractérisant le comportement global du système, une vue abstraite mettant en avant un nombre restreint d'items du modèle qui suffisent à répondre à un problème de vérification bien délimité. Ainsi, dans une classique architecture à trois niveaux incluant deux entités de protocoles qui s'appuient sur un service existant, le concepteur de la couche de protocole regroupant les deux entités cherchera à caractériser le service rendu par cette couche en faisant abstraction des communication entre entités de protocole et réseau sous-jacent. L'on peut observer un service local à une entité de protocole ou un service global aux deux entités.

Pour illustrer la vérification par abstraction en contexte TURTLE, nous utilisons à la figure 18 un extrait du mécanisme d'ouverture de connexion d'un protocole Transport extrêmement simplifié. Par souci d'efficacité, nous avons regroupé sur une même figure le

diagramme de classes/objets et les diagrammes d'activité associés aux objets en question. La figure suivante présente en sa partie gauche le graphe d'accessibilité obtenu pour cet exemple à vocation purement pédagogique. Les automates quotients obtenus pour l'équivalence de langage et l'équivalence observationnelle, sont respectivement présentés au centre et à la droite de la figure 19.

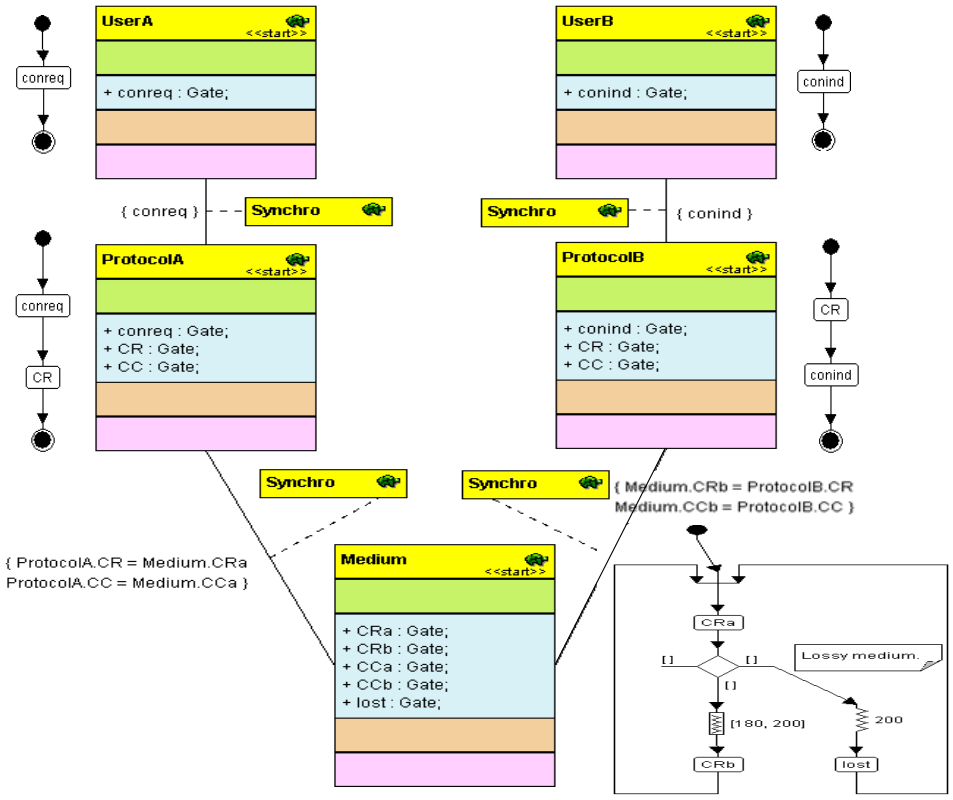


Figure 18. Architecture et comportements TURTLE d'une ouverture de connexion

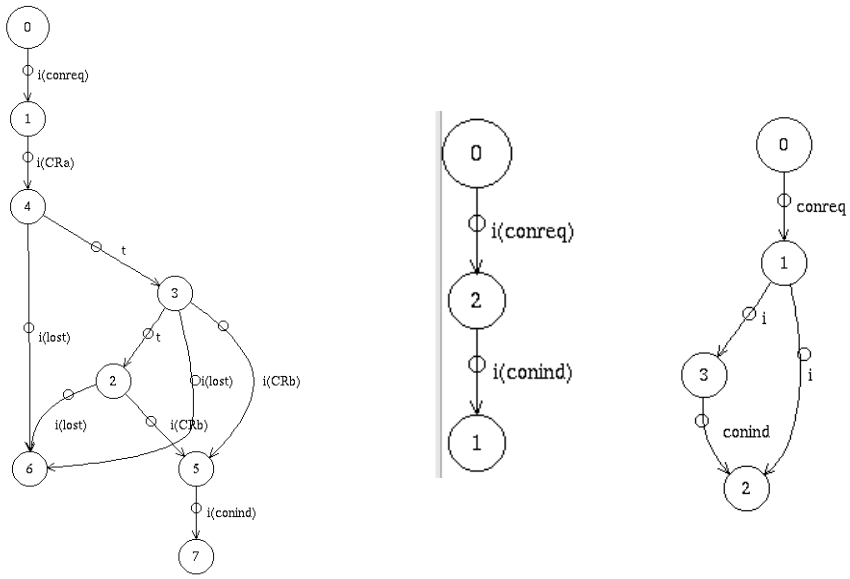


Figure 19. Graphe d'accessibilité et automates quotients

6. Positionnement par rapport aux travaux de normalisation à l'OMG

6.1. Positionnement par rapport à UML 1.5

Le profil TURTLE étend la notation UML au niveau des diagrammes de classes et des diagrammes d'activité, respectivement dédiés à la structuration d'un système en classes et à la description des comportements internes aux dites classes. Il s'agit en l'espèce de Tclasses qui sont autant de classes stéréotypées dotées de portes de communication par rendez-vous. Les associations entre Tclasses sont attribuées par des opérateurs de composition, ce qui permet d'exprimer explicitement les concepts de parallélisme, séquence, synchronisation, invocation et préemption entre tâches décrites par des Tclasses. Les diagrammes d'activité ont été étendus par des opérateurs temporels pour décrire (1) un délai fixe, (2) un délai non déterministe qui associé au premier permet de travailler avec des intervalles temporels, et (3) une offre limitée dans le temps.

6.2. Positionnement par rapport à UML 2.0

La présentation du profil TURTLE à la conférence UML'2001 [ACL 01] a précédé de trois ans l'adoption par l'OMG d'UML 2.0 qui entérine l'idée d'associer des ports de communication à des classes UML. Le document [OMG 03] ne définit aucune sémantique de communication et les concepteurs d'outils industriels se sont enfoncés dans cette brèche pour promouvoir leurs solutions antérieures : Statecharts pour Rhapsody [RHA] et SDL pour TAU G2 [TAU]. Ce deuxième outil (utilisé par ailleurs à l'ENSICA pour l'enseignement) implémente une sémantique de file non bloquante à la SDL. TURTLE reste sur sa ligne initiale : la composition de fonctionnalités instanciées par des entités qui communiquent par rendez-vous. Par ailleurs, la comparaison des opérateurs temporels est favorable à TURTLE qui surpasse le « timeout » à la SDL de TAU G2 en offrant la possibilité de décrire explicitement l'indéterminisme temporel (Cf. l'opérateur de « latence » de RT-LOTOS) et de le prendre en compte au niveau d'une validation formelle. De plus, il n'existe à notre connaissance aucun outil industriel labellisé UML 2.0 qui propose une analyse énumérative des modèles de conception sans parler de la validation de diagrammes d'analyse et de synthèse de diagrammes de conception à partir de ces mêmes diagrammes d'analyse.

6.3. Positionnement par rapport au profil SPT

Ce paragraphe présente un extrait d'un rapport [ASC 04] dans lequel nous avons montré comment transcrire en contexte TURTLE les concepts du profil SPT (*Scheduling, Performanc*e *and Time*) de l'OMG. Nous nous limitons ici aux constructions supportées par TTool et proposées avec l'outil sous forme d'une bibliothèque de Tclasses. Nous ne détaillerons pas les opérateurs Periodic et Watchdog qui ne sont pas encore implantés. Par contre, nous allons montrer les possibilités d'analyse d'ordonnancement offertes par le TURTLE toolkit.

6.3.1. Ressources et concurrence

Le profil SPT de l'OMG propose de modéliser la concurrence par le biais de deux stéréotypes, à savoir <<CRasynch>> et <<CRconcurrent>>. Si ces stéréotypes peuvent être utilisés dans tous les diagrammes UML, le profil de l'OMG n'en fait usage sous forme

d'exemples que dans les diagrammes de séquence. De son côté, le profil TURTLE modélise les concurrences à la fois au niveau des diagrammes de classes/objets (par exemple, l'opérateur de composition *Parallel*) et au niveau des diagrammes d'activité (opérateur *fork* ou *join*).

Considérons tout d'abord les communications entre Tclasses. TURTLE se focalise sur des communications synchrones (rendez-vous) qui se traduisent au niveau des diagrammes de classes TURTLE par des opérateurs de composition *Synchro*. Pour modéliser une communication asynchrone entre deux Tclasses T1 et T2, il est nécessaire d'utiliser une troisième classe qui modélise la sémantique de cet asynchronisme. Cette troisième Tclasse doit se synchroniser à la fois avec T1 et T2. Afin de nous conformer à l'approche du profil SPT de l'OMG, nous proposons de stéréotyper les opérateurs de composition *Synchro* avec <<CRsynch>> (cf. figure 20) alors que les classes modélisant une sémantique d'asynchronisme doivent être stéréotypées <<CRasynch>> (cf. figure 21).

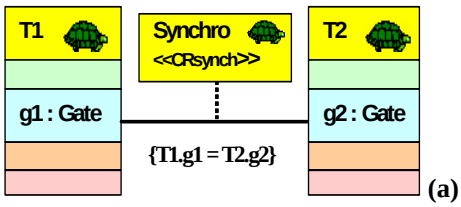


Figure 20. Modélisation d'une communication synchrone

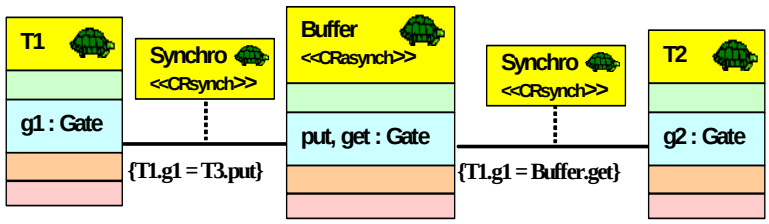


Figure 21. Modélisation d'une communication asynchrone

En ce qui concerne la modélisation du partage de ressource, nous proposons d'introduire un stéréotype <<mutex>> dont l'usage sur une Tclasse signifie qu'au plus une *Invocation*[11] peut être réalisée à un instant donné sur cette Tclasse. Notons que TURTLE n'autorise pas la connexion d'une Tclasse stéréotypée <<mutex>> à une autre Tclasse par une relation Synchro.

[11] L'opérateur de composition *Invocation* offre une contrepartie à l'appel de méthodes entre classes UML dans le contexte de Tclasses TURTLE qui se synchronisent sur leurs portes de communication.

Enfin, TURTLE ne prend pas en compte le stéréotype <<atomicity>> défini dans le profil SPT de l'OMG. En effet, l'exécution d'un ensemble E d'opérations atomiques oblige à empêcher toute autre action dans le système tant que les actions de E ne sont pas terminées.

6.3.2. Mécanismes temporels

Comme son nom l'indique, le profil SPT met l'accent sur la définition de mécanismes temporels (stéréotypés <<RTtimingMechanism>>). Ces mécanismes sont de deux types : les timers (Timer) et les horloges (Clock). Ils possèdent tous deux trois attributs (*stability*, *drift* et *skew*) et cinq méthodes (*set*(), *get*(), *reset*(), *start*() et *pause*()). De plus, la classe *Timer* définit un attribut *isPeriodic*.

Ces mécanismes temporels se retrouvent en TURTLE sous la forme de bibliothèques de Tclasses (*TTimer* et *TTclock*) et d'opérateurs de composition (*Periodic*, *Watchdog*).

6.3.3. Bibliothèques de mécanismes temporels

TURTLE introduit deux bibliothèques *TClock* et *TTimer* (stéréotype <<RTtimingMechanism>> dont l'objectif est de générer respectivement des interruptions d'horloge à intervalle régulier et des interruptions programmables. Ces interruptions peuvent être capturées par synchronisation sur une porte publique de ces classes (*tick* pour *TClock* et *interrupt* pour *TTimer*). De plus, ces classes définissent cinq portes de synchronisation correspondantes aux cinq méthodes (*set*, *get*, etc.) définies dans les mécanismes temporels du profil de l'OMG. Ces classes ne définissent pas les attributs *skew*, *drift* et *stability*. Notons que *skew* et *drift* peuvent par contre être modélisés au niveau du comportement des Tclasses par utilisation des opérateurs de délai déterministe et non déterministe.

La figure représente les deux classes TTimer et TClock et la classe Ticker, dont l'objectif est de générer une action de synchronisation sur *tick* toutes les *increment* unités de temps.

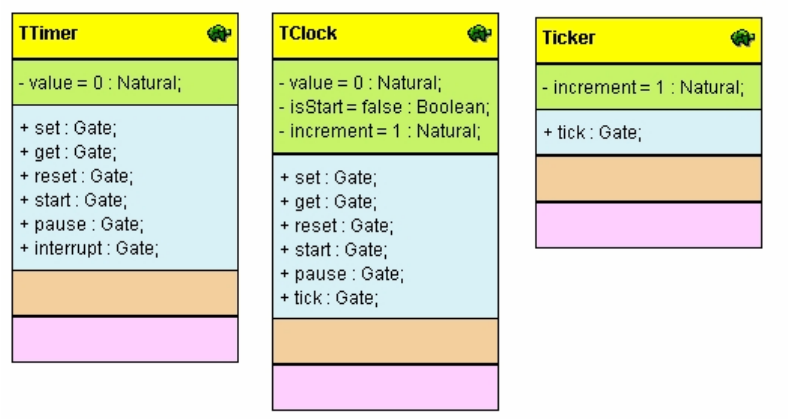


Figure 22. Librairies de mécanismes temporels

6.4. Ordonnancement : modélisation et analyse

Le profil SPT de l'OMG introduit un ensemble d'extensions pour la modélisation des ordonnancements statiques et dynamiques. Plus particulièrement, ce profil définit des stéréotypes qui permettent de modéliser les actions et les ressources des systèmes spécifiquement dédiées à l'ordonnancement.

TURTLE inclut un opérateur de composition – nommé Suspend/Resume - dédié à la modélisation de l'ordonnancement. Cet opérateur est généralement utilisé comme suit : chaque tâche du système est modélisée par une Tclasse. L'ordonnanceur est modélisé par une série de Tclasses, dont une joue un rôle particulier : elle représente le mécanisme de préemption et l'algorithme d'ordonnancement. Cette classe principale est reliée aux Tclasses qui représentent les tâches par une relation de type Suspend/Resume (cf. figure, la classe principale étant nommée *SchedulingMainClass*).

Le fait que l'ordonnancement soit statique (par ex. Rate Monotonic) ou dynamique (par exemple Deadline Monotonic) et les actions liées à l'ordonnanceur sont modélisés au niveau du diagramme d'activité de la classe SchedulingMainClass. Lorsque une action de ce diagramme d'activité est conforme aux <<SAction>> telles que définies dans le profil de l'OMG, alors il est possible d'attacher à ces actions une note UML contenant <<SAction>>. Aussi, dans le cas d'un ordonnancement dynamique, le diagramme d'activité de la classe principale modélise généralement la décision d'ordonnancement, qui utilise notamment les priorités des tâches. C'est pourquoi, conformément à ce qui est défini dans le profil STP, la classe doit être stéréotypée <<SAschedRes>>. De plus, puisque cette classe modélise le dispatching des tâches sur la ressource matérielle d'exécution, elle doit être stéréotypée <<SAEngine>>.

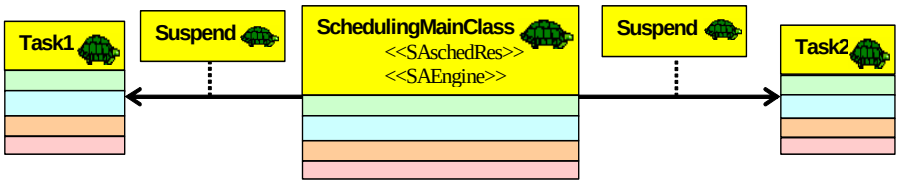


Figure 23. Modélisation de l'ordonnancement dans un diagramme de classes

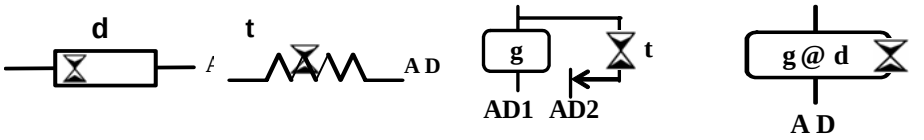
L'algorithme de traduction d'une modélisation TURTLE vers une spécification RT-LOTOS utilise à l'heure actuelle l'opérateur '@' de RT-LOTOS lorsque des contraintes d'ordonnancement sont utilisées dans la modélisation TURTLE. Notons que seul le mode simulation (et non vérification) peut-être utilisé avec RTL lorsque la spécification RT-LOTOS contient un opérateur '@'. Ainsi, l'analyse d'ordonnançabilité du système se limite à des simulations intensives mais non exhaustives. Ces simulations permettent de mettre en évidence la date d'occurrence de toutes les actions de type <<SAction>>, ce qui permet d'analyser l'utilisation des ressources d'ordonnancement.

6.5. Performance et qualité de service

Le profil SPT de l'OMG suggère de modéliser les contraintes de performance et de qualité de service au niveau des diagrammes de séquences et d'activité, et d'analyser ces performances avec un outil adapté.

En TURTLE, les contraintes de performance se modélisent au niveau des diagrammes d'activité. Elles peuvent ensuite être analysées en phase de vérification. Ces contraintes de performance sont modélisées par utilisation des opérateurs temporels interruptibles (cf. figure 20). Ces opérateurs temporels interruptibles se distinguent des opérateurs temporels classiques dans la mesure où les opérateurs interruptibles sont interrompus lorsque la Tclasse qui les utilise est suspendue (opérateur *Suspend*) et réactivés lorsque la Tclasse est réactivée. A contrario, les opérateurs temporels classiques ne sont pas affectés par les opérateurs de suspension. Ces derniers opérateurs sont donc particulièrement bien adaptés à la modélisation de timers, alors que les opérateurs interruptibles sont plutôt adaptés à la modélisation de temps d'exécution (performance des algorithmes, et plus généralement du système) : de tels opérateurs sont nommés *responseTime* et *delay* dans le profil STP. De plus, les *interval* du profil SPT peuvent modéliser à l'aide de délais non déterministes classiques et les *repetition* avec l'opérateur de boucle des diagrammes d'activité TURTLE. Enfin, pour être encore plus en conformité avec le profil SPT, nous donnons la possibilité

d'attacher une note UML aux opérateurs temporels interruptibles qui modélisent des temps de réponse afin de préciser s'ils sont utilisés ou non à des fins d'étude de performance.



Délai déterministe interruptible	Délai non déterministe interruptible	Offre limitée dans le temps interruptible	Relevé de temps interruptible

Figure 24. Opérateurs temporels interruptibles

Une fois modélisées, les contraintes de performance peuvent être analysées au moyen des fonctions de vérification et de simulation adjointes à l'outil TTool.

Premièrement, les traces de simulation sont temporisées ; en d'autres termes, chaque action de la trace comporte un numéro d'ordre et une date d'occurrence. L'outil TTool permet la visualisation de ces traces sous la forme de chronogrammes (cf. figure 25). Notons que les actions synchronisées étant effectuées simultanément (même numéro ordre, même date d'occurrence), elles sont représentées sur le même axe. Par exemple, les actions *resetTimer* de *task1* et *reset* de *timer1* sont synchronisées.

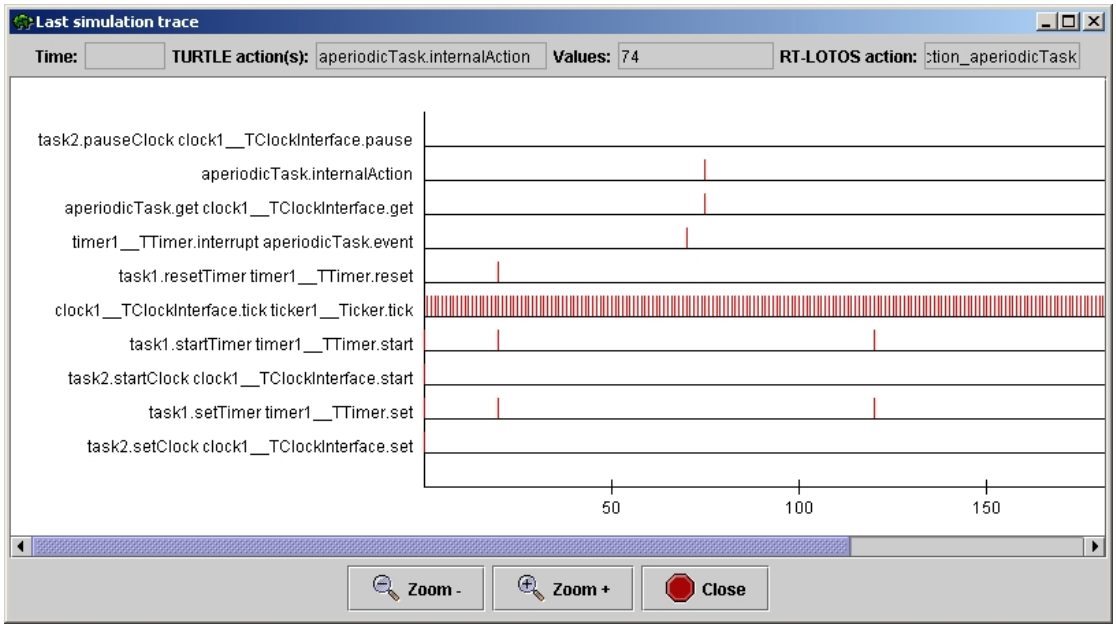


Figure 25. Analyse d'une trace de simulation avec TTool

Deuxièmement, le graphe d'accessibilité que RTL peut générer contient des informations temporelles telles que les horloges des processus LOTOS. Ces informations permettent de déduire les intervalles temporels dans lesquels les actions LOTOS peuvent être réalisées. Par utilisation, dans TTool, d'un arbre de correspondance entre les actions de ce graphe

temporisé et les actions TURTLE, il est possible d'analyser sur ce graphe les contraintes de performance de la modélisation TURTLE.

7. Positionnement par rapport aux travaux de recherche du domaine

Le profil TURTLE a été introduit pour la modélisation et la validation des systèmes embarqués, des protocoles et des systèmes distribués.

Pour la modélisation des systèmes critiques, deux approches ont été proposées : l'introduction d'opérateurs logiques (principalement des opérateurs pour composer les classes UML) et des opérateurs temporels (principalement au niveau de la description comportementale des classes). Dans la majorité des profils UML temps réel, la composition logique repose sur une communication asynchrone [CM 00] [DOU 02] [GTT 02] [SEL 01]. De plus, cette composition est généralement masquée

et implicite (cf. les outils Rose RT [SEL 01] et TAU G2 [TAU]). TAU G2 implante un profil UML/SDL [BJO 00]. A l'opposé des ces compositions asynchrones qui anticipent sur des choix d'implantation (communication par file), TURTLE intègre une composition explicite et formellement définie, basée sur une communication synchrone (rendez-vous à la LOTOS). Du point de vue temporel, l'approche classique repose sur l'utilisation de temporisateurs (Selic 2001) qui ne permettent pas l'expression d'une latence indépendante de l'implémentation. Au contraire, TURTLE permet grâce à ses opérateurs temporels de base d'exprimer des contraintes temporelles et des phénomènes tels que la gigue fréquemment constatée dans les transmissions de flux multimédia.

La modélisation en UML des protocoles et des systèmes distribués a été étudiée dès les premières versions d'UML d'abord au travers d'exemples académiques [JJP 98] puis sur des systèmes plus avancés tels que des systèmes coopératifs [JS 00], des systèmes de commerce électronique [END 01] et des systèmes multi-agents [KKB 03] [LIN 01]. Les contributions dans ce domaine se focalisent principalement sur l'introduction en UML de mécanismes de description des interactions entre composants, et sur la description des contraintes relatives à ces interactions. Par exemple, AUML [WCW 01] introduit les *Protocol diagrams* qui étendent les diagrammes de séquence avec des opérateurs logiques exprimant explicitement la causalité, la synchronisation et la multi diffusion. UMLSec [JUR 04] étend les diagrammes de déploiement en donnant la possibilité d'exprimer des contraintes relatives à la sécurité au niveau des liens entre noeuds. De son côté, TURTLE permet l'expression

explicite de liens synchrones et asynchrones entre entités du système. La sémantique des liens asynchrones étant modélisée avec un diagramme d'activité, il est possible de leur donner un comportement contraint logiquement et temporellement. Une extension de TURTLE, nommée TURTLE-P [ASK 05] et présentée en annexe de ce mémoire, aborde le déploiement de composants TURTLE sur des liens physiques et permet d'attribuer des contraintes de qualité de service aux liens de communication entre les unités physiques.

Parmi les projets qui placent la validation a priori basée modèles au centre du processus de conception de systèmes temps réel et embarqués, nous pouvons mentionner le projet OMEGA [OME] et son profil UML temps réel : OMEGA-RT [GOO 05]. Les modèles édités dans un des trois outils (externes au projet OMEGA) que sont ROSE-RT, Rhapsody et Argo sont traduits dans le formalisme IF déjà supporté par des outils de validation formelle. La démarche est similaire à celle que nous avons mis en œuvre pour le profil TURTLE et la traduction vers RT-LOTOS. Les articles [GOO 05a] et [OGO 05] nous ont permis de comparer les profils OMEGA-RT et TURTLE. Au niveau des diagrammes de classes, OMEGA-RT supporte l'orientation objet d'UML en termes d'héritage alors que ce mécanisme n'est pas encore implanté dans TTool (Notons au passage que la traduction vers IF aplatit les modèles contenant de l'héritage). Par contre, la notion d'opérateur de composition que TURTLE hérite des algèbres de processus n'est pas présente dans les diagrammes de classes OMEGA-RT. Ajoutons à cela qu'OMEGA-RT dote les objets de ports de communication asynchrone (à l'exception des observateurs qui sont synchrones au modèle) alors qu'une modélisation TURTLE repose sur l'idée de composition de fonctionnalités décrites par des classes dotées de ports de communication par rendez-vous. En termes de diagrammes de comportement, OMEGA-RT traite les contraintes temporelles au niveau des événements alors que TURTLE offre trois opérateurs temporels génériques dont l'opérateur *latency* pour exprimer de un indéterminisme temporel qui semble difficile à reproduire en OMEGA-RT.

La validation a priori de modélisation UML est une caractéristique recherchée tout particulièrement chez les concepteurs de systèmes temps-réel. Par exemple, le projet OMEGA adjoint une sémantique formelle à un sous-ensemble d'UML et propose une méthodologie adaptée aux systèmes embarqués temps-réel [HOO 02]. La plateforme OMEGA traduit des modèles UML temps réel vers le formalisme IF qui disposait déjà d'outils de validation formelle. L'approche est de même nature que le projet TURTLE qui traduit un UML temps réel dans le langage formel RT-LOTOS. Par contre les opérateurs

temporels de TURTLE sont plus puissants que ceux des UML temps réel considérés dans le cadre du projet OMEGA.

ACCORD [GTT 02] est aussi un exemple de méthodologie UML pour les systèmes temps-réel basé sur Objecteering. Son originalité réside surtout dans ses règles de conception.

La méthodologie TURTLE s'appuie de son côté sur un processus de validation formelle automatisé et supporté par un outillage (TTool et RTL) qui permet de s'affranchir totalement de la connaissance du langage formel RT-LOTOS. RTL implémente des concepts avancés de validation formelle tels que la génération de graphes d'accessibilités temporisés dont la génération repose sur l'utilisation de régions temporelles et la génération d'automates d'ordonnancement [LOH 02].

Pour conclure ces paragraphes dédiés au positionnement du profil TURTLE, nous devons élargir la discussion au-delà des outils de vérification liés à un profil UML temps réel. Prenons tout d'abord le cas des langages synchrones tels qu'Esterel [BS 91] ou Lustre [CAS 87]. Ces deux langages se fondent sur une hypothèse de synchronisme que le système modélisé doit vérifier dans les faits pour que les outils dédiés à ces langages soient applicables. Contrairement aux SyncCharts qui lient les Statecharts UML au langage Esterel, l'approche TURTLE est résolument asynchrone, en conformité avec le domaine d'application visé que sont les systèmes distribués. Ces systèmes ont depuis longtemps modélisés en utilisant des outils de vérification tels que SPIN [HOL 97] qui est basé sur le langage Promela. SPIN intègre du contrôle de modèle ou « model checking » en anglais. A ce jour nous n'avons pas encore appliqué du contrôle de modèles aux diagrammes TURTLE ; en particulier, la liaison entre l'outil RTL et l'outil KRONOS n'a pas été exploitée. Parmi les « model checkers » les plus souvent cités dans la littérature, il serait intéressant de comparer des modélisations TURTLE et leur vérification à celles opérées sur des automates temporisés en utilisant l'outil UPPAAL [BEN 96]. Notons que nous comptons développer le volet « contrôle de modèles » en bénéficiant des derniers développements de l'analyseur de réseaux de Petri temporels TINA développé au LAAS-CNRS. Ceci suppose que nous soyons capables de traduire les spécifications RT-LOTOS dans le formalisme supporté par l'outil TINA. Cette voie est l'une de celles que nous proposons d'explorer au prochain chapitre dédié aux perspectives à court, moyen et long terme de notre travail.

Chapitre IV

Conclusions et Perspectives

1. Conclusion générale

1.1. Démarche scientifique

« Modéliser est un acte d'ingénieur ». Cette phrase péremptoire ne saurait être démentie dans bien des branches d'activité qui concourent à l'élaboration de systèmes complexes. Pour autant, bien des concepteurs et développeurs de systèmes à logiciels prépondérant demeurent retors à l'idée de modéliser avant de produire du code. La complexité reconnue aux systèmes temps réel et distribués, ajoutée à la diffusion des approches *Model Driven* [MDA 04] dans le continuité de la normalisation de la notation UML, contribue à faire évoluer les mentalités des praticiens industriels.

Depuis trois décennies au moins, ces mêmes systèmes temps réel et distribués ont offert un sujet d'étude quasi inépuisable aux équipes de recherche qui développent des techniques de modélisation - formelles ou non - et leurs outils de support. Les travaux présentés dans ce mémoire s'inscrivent dans cette mouvance. Parmi les approches successivement étudiées au cours de notre parcours de recherche, à savoir Estelle, Mondel, les réseaux de Petri à flux temporels et UML temps réel, nous avons dans l'exposé scientifique de ce mémoire choisi de mettre l'accent sur le langage Estelle et le profil UML temps réel TURTLE. Ce profil est au cœur de notre contribution scientifique la plus récente sur la période 2000-2005.

Nos recherches sur ces langages procèdent d'une même idée directrice : proposer des techniques de validation de modèles facilitant la détection des erreurs de conception au plus tôt dans le cycle de développement d'un système complexe. Promouvoir la validation a priori et basée modèle a été et demeure notre préoccupation ; d'une part, dans la recherche de techniques et outils appréhendables par le plus grand nombre de praticiens du domaine et, d'autre part, du point de vue de l'enseignement en cursus ingénieur et Master Recherche. Le choix d'intégrer des concepts RT-LOTOS dans un moule UML et le fait de mener de fronts activités de recherche et de formation utilisant soit le profil TURTLE, soit le profil UML2.0/SDL, sont également des caractéristiques de cette démarche.

2. Synthèse des contributions

Nous ne reviendrons pas ici sur les aspects « encadrements » et « projets » identifiés sur les diagrammes de la section « Parcours de Recherche » de ce mémoire. Nous adoptons en cette conclusion un point de vue « description thématique » et aboutissons pour les activités liées à Estelle et TURTLE à la figure qui suit. L'arbre ainsi constitué identifie entre crochets quelques publications de référence.

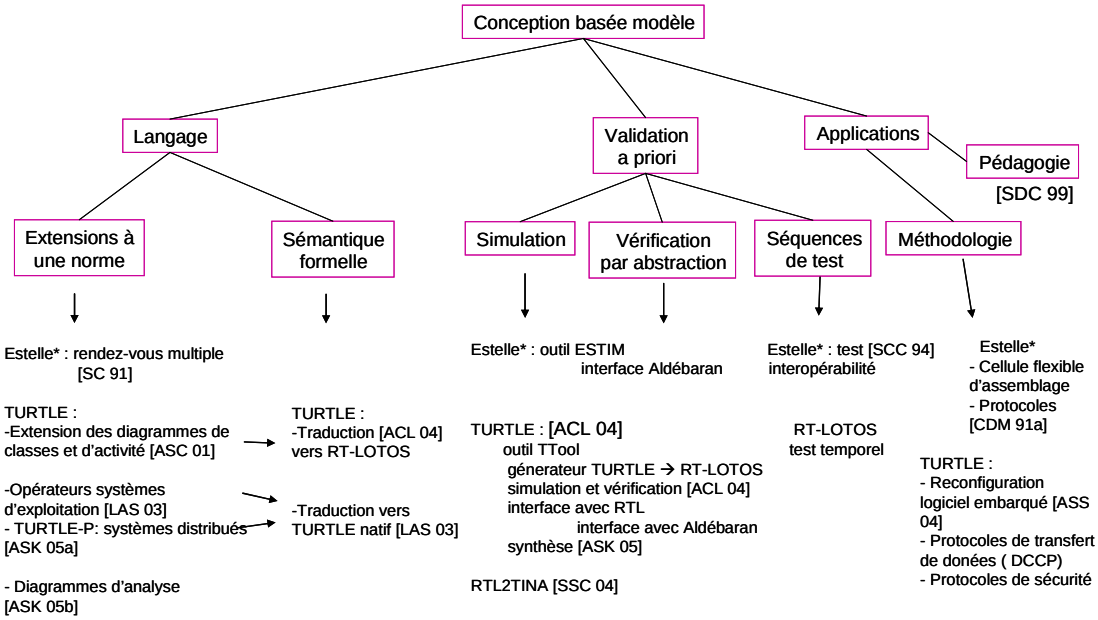


Figure 1. Classification thématique des activités de recherche

Le profil UML temps réel TURTLE et la chaîne d'outils qui le supportent nous permettent d'offrir aujourd'hui un environnement d'analyse et de conception de systèmes temps réel et distribués avec validation de modèle à la clé. L'étude de cas sur la reconfiguration dynamique de logiciel embarqué à bord de satellite [ASS 04] a montré l'applicabilité en contexte industriel. Le partenariat avec Télécom-Paris à Sophia Antipolis et le développement de l'outil TTool par Ludovic Apvrille ont assuré la « survie » du projet TURTLE en offrant une plateforme d'expérimentation pour des extensions au profil.

Le profil TURTLE a été appliqué avec succès à une étude fournie par la société Alcatel Space : la preuve de continuité de service dans la reconfiguration dynamique de logiciel embarqué à bord de satellite. Plus récemment le profil TURTLE et ses outils ont été utilisés

par la société UDCAST et dans le cadre du projet européen MAESTRO**.** L'étude a porté sur un protocole de diffusion de données multimédia à modéliser et valider. D'autres contacts industriels sont établis en complémentant des manifestations d'intérêt émanant du secteur académque.

3. Perspectives

Nos perspectives de travail de recherche à court, moyen et long terme s'entendent dans le prolongements de travaux jusqu'ici exposés qui accordent une place privilégiée à la conception basée modèle et à la validation formelle des systèmes temps réel et distribués.

3.1. Prolongements immédiats

En termes de langage support à notre activité, nous entendons poursuivre le développement du profil UML temps réel TURTLE en revisitant les techniques de validation formelle et de génération de tests basées sur le langage RT-LOTOS.

En termes de validation formelle, des schémas de traduction de la partie « contrôle » de RT-LOTOS vers les réseaux de Petri temporels ont été définis et un premier prototype de traducteur réutilisant l'analyseur lexico-syntaxique de RTL et engendrant un réseau de Petri temporel au format de l'outil TINA est opérationnel. Les premières expérimentations sont encourageantes qui montrent un gain significatif en temps d'exécution non seulement par rapport à RTL mais aussi à l'outil CAESAR développé pour la version non temporelle de LOTOS. Ces schémas de traduction de RT-LOTOS vers les réseaux de Petri temporels doivent maintenant être étendus pour prendre en compte la partie « données » de RT-LOTOS. Pour faire court, nous dirons que nous cherchons à traduire un modèle en langage de haut niveau TURTLE vers un « assembleur » RT-LOTOS lui-même compilé dans ce « binaire » que sont les réseaux de Petri.

Par ailleurs, une activité a démarré sur le thème du test temporel pour RT-LOTOS. Il s'agit de définir un cadre formel pour ce test temporel et de doter RTL d'un module de génération de séquences de test temporisées que l'on dérivera automatiquement à partir de l'automate d'ordonnancement appelé TLSA.

En termes d'étude de cas et de mise à l'épreuve du profil TURTLE et de son outillage, nous avons démarré une étude sur le protocole DCCP qui bénéficie de la mise en œuvre de la vérification par abstraction dans l'environnement TURTLE.

3.2. Perspectives à moyen terme

Au-delà des protocoles de transfert de données tels que le protocole DCCP que nous venons de mentionner, nous souhaitons éprouver le profil TURTLE et ses outils sur dans le domaine des protocoles de sécurité. Une action en ce sens a démarré dans le cadre du projet SAFECAST dédié à la sécurité dans les communications de groupe.

Le volet sécuritaire des systèmes à logiciel prépondérant nous paraissent en effet fournir un champ d'application demandeur de techniques de modélisation et validation formelles. Il en va de même des systèmes aéronautiques et plus généralement des systèmes critiques en vies humaines dont nous entendons faire un champ d'application privilégié. Dans la sécurité comme dans l'aéronautique, la proposition d'un outillage formel dans un moule UML est de nature à favoriser une acceptation en contexte industriel dès lors que nous saurons intégrer langage et outils à une méthodologie acceptable par les praticiens du domaine.

La prise en compte de systèmes de plus en plus complexes rend hautement nécessaire la poursuite du travail sur la traduction de RT-LOTOS vers les réseaux de Petri temporel plus exactement vers une extension de ce modèle qui sera intégrée à l'outil TINA pour supporter le traitement des données. Par ailleurs, l'introduction de « stopwatchs » dans TINA amènera certainement à revoir l'opérateur de préemption de RT-LOTOS pour traiter in fine les opérateurs TURTLE orientés « systèmes d'exploitation temps réel » proposés en [LAS 03].

Notons enfin que la chaîne d'outils TURTLE n'a pas vocation à concurrencer les outils UML industriels. Elle constitue avant tout un démonstrateur des « potentialités » du profil par rapport à la norme de l'OMG. Néanmoins, il serait souhaitable de ne pas se limiter à la validation formelle et de proposer à terme un générateur de code, Java par exemple. Par ailleurs, le niveau d'abstraction d'un modèle TURTLE et l'implantation d'une procédure d'analyse d'accessibilité fond de ce profil un candidat naturel à modéliser et analyser exhaustivement des mécanismes de cœur d'un système complexe, en amont d'une conception détaillée et davantage orientée « implantation » comme la propose l'outil TAU G2 dans son implantation d'un profil UML/SDL.

3.3. Ligne directrice sur le plus long terme

Les travaux sur le profil UML temps réel TURTLE procèdent d'une démarche d'intégration de langages formels dans un processus de conception acceptable par des praticiens industriels.

Le TURTLE toolkit aujourd'hui disponible permet de valider une conception UML temps réel en masquant autant que faire se peut le langage formel RT-LOTOS. Cette conception peut être synthétisée à partir de scénarios. L'effort doit se poursuivre pour proposer aux concepteurs de systèmes temps réel et distribués des langages plus puissants en termes d'expression de propriétés à satisfaire par le système.

Nous devons également suivre l'actualité des techniques de validation formelle pour ce qui tient au passage à l'échelle et au problème bien identifié de l'explosion combinatoire inhérente à l'analyse énumérative.

Enfin, nous souhaitons poursuivre l'activité sur le test en ajoutant le test de robustesse aux tests d'interopérabilité et de conformité.

Annexe

TURTLE-P

1. Description du déploiement de composants logiciels

Le diagramme de déploiement UML permet de mettre en évidence les configurations d'exécution d'un système. Ce diagramme est fort peu mis en valeur dans les méthodologies de développement de logiciels. De plus, il n'est utilisé qu'à des fins de documentation. Pourtant, nous pensons que ce diagramme est bien adapté à la description d'une architecture distribuée et nous voulons lui donner une sémantique formelle.

Un diagramme de déploiement est constitué de :

- ***Nœuds***. Ces nœuds constituent les différents sites physiques d'exécution du système considéré.
- ***Composants* logiciels**. D'après la norme UML, ces composants logiciels sont des éléments logiciels déployables qui encapsulent en leur sein des fonctions et qui offrent à leur environnement des interfaces.
- **Liens de communication**. Ces liens relient un composant logiciel à l'interface d'entrée d'un autre composant logiciel.

La Figure 1 présente un diagramme de déploiement UML dans lequel un composant logiciel *client* communique avec un autre composant logiciel *serveur* situé sur un autre site physique d'exécution.

Figure 1. Exemple d'un diagramme de déploiement UML.

Nous proposons d'améliorer le diagramme de déploiement UML comme suit :

- Les composants logiciels sont constitués de classes TURTLE et deviennent ainsi « validables » ;
- Une multiplicité peut être déclarée au niveau des nœuds afin de faciliter la modélisation des systèmes distribués comportant de nombreux nœuds identiques sur lesquels les mêmes composants sont déployés ;
- Les liens de communication peuvent être caractérisés par des paramètres (notamment, délai de transmission et gigue) qui sont pris en compte lors du processus de validation formelle.

2. Composants logiciels du diagramme de déploiement

2.1. Définition des composants

Un composant logiciel regroupe des classes qui réalisent une fonction logicielle sur un même site d'exécution. C'est pourquoi nous proposons de définir ces composants logiciels comme un sous-ensemble des classes modélisées dans le diagramme de classes. Soit C un composant logiciel tel que C soit un ensemble de n *Tclasses* = $\{c_i, i \in 1..n\}$. C doit respecter la propriété suivante :

> $\forall\, i \in 1..n, c_i \in C, \forall\, c \in$ *Diagramme de classes,* $c \notin C \Rightarrow c$ *et* c_i *ne sont pas en relation de Synchronisation ou d'Invocation.*

En effet, deux classes ne peuvent communiquer via une synchronisation ou une invocation que dans le cas où elles appartiennent toutes deux au même composant logiciel.

S'il existe entre deux *Tclasses* C1 et C2 d'un diagramme de classes, une relation de *Parallélisme*, *Séquence* ou *Préemption* et si un composant logiciel ne contient qu'une seule de ces deux *Tclasses*, alors la relation entre ces deux *Tclasses* est ignorée au niveau de ce composant. Notons aussi que deux composants logiciels distincts peuvent contenir la même *Tclass*.

Par exemple, considérons le diagramme de classes représenté à la Figure 2.

- C_a = {C1, C3} ne constitue pas un composant logiciel car C1 et C2 sont en relation de synchronisation et C2 n'appartient pas à C_a.
- C_b = {C1, C2, C4} est un composant logiciel valide car C4 s'exécute en parallèle au regard de C1 et C2 et aucune classe de C_b n'est en relation de Synchronisation / Invocation avec une classe n'appartenant pas à C_b.
- Cc = {C3} est aussi un composant logiciel valide. C3 ne pourra jamais être préempté par C4 puisque C4 n'appartient pas à ce composant, mais cela ne « nuit » pas à l'exécution de C3.

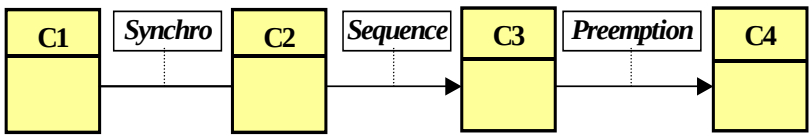


Figure 2. Exemple d'un diagramme de classes TURTLE.

2.2. Interface des composants

Les composants logiciels sont constitués de *Tclasses*. Pour communiquer, ces *Tclasses* offrent à leur environnement des portes de communication. Nous proposons que les interfaces des composants soient donc constituées des portes de communication des *Tclasses* de ce composant.

Cependant, certaines des portes de communication des *Tclasses* sont impliquées dans des relations de synchronisation à l'intérieur d'un même composant. Afin de ne pas pouvoir réutiliser ces mêmes portes avec des relations extérieures, seules les portes déclarées publiques (+) et non impliquées dans ces relations de synchronisation internes au composant constituent les interfaces des composants.

De plus, il nous faut distinguer les interfaces d'entrée des interfaces de sortie. Seuls les attributs de type *InGate* peuvent être considérés comme des interfaces d'entrée. De même, seules les portes de type *OutGate* peuvent être considérées comme des interfaces de sortie.

2.3. Représentation des composants

En UML, les composants logiciels représentés au niveau du diagramme de déploiement peuvent être définis au sein d'un diagramme de composants. Nous rendons cette démarche obligatoire.

Au niveau du diagramme des composants, seuls sont représentés les composants logiciels déployés au niveau du diagramme de déploiement et les interfaces des composants utilisées dans ce même diagramme de déploiement. Une porte *g* de la classe *c* donnera une interface nommée *g_c*. Enfin, une multiplicité indiquée au niveau du nœud précise combien de fois l'ensemble des composants du nœud sont déployés sur des nœuds distincts. Par exemple, considérons le diagramme de déploiement représenté à la Figure 3. Le diagramme de composants correspondant est représenté à la Figure 4. Ce diagramme sert à mettre en évidence les classes logicielles contenues dans le composant, sans rappeler les liens sémantiques entre ces classes. La représentation des interfaces au niveau du diagramme des composants nous apparaît comme facultative dans la mesure où ces interfaces (c-à-d les portes des *Tclasses*) sont déjà spécifiées au niveau du diagramme de classes.

Notons que le nœud Client de la Figure 3 possède une multiplicité de *n* ce qui signifie que le composant C*b* est déployé sur *n* nœuds de type *Client*.

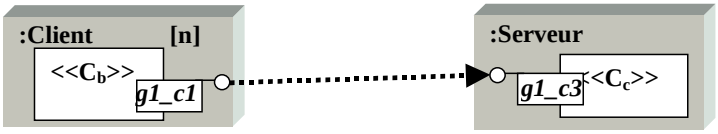


Figure 3. Exemple d'un diagramme de déploiement TURTLE-P

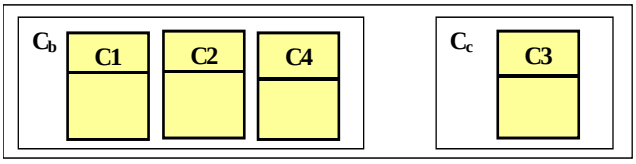


Figure 4. Diagramme de composants.

2.4. Liens du diagramme de déploiement

En UML 1.5, les liens des diagrammes de déploiement ne sont donnés qu'à titre d'information. Notre objectif est de leur donner une sémantique formelle.

Nous nous limitons ici à des liens de transmission asynchrone et unidirectionnels et attribués par quatre paramètres :

- un délai minimal (*delay_min*) et maximal (*delay_max*) de transmission ;
- une bande passante (*max_msg*) ;
- un taux de perte exprimé en pourcentage (*loss_rate*).

La Figure 5 représente un lien unidirectionnel entre un client et un serveur. La durée minimale de transmission sur ce lien est de 10 unités de temps, alors que la durée maximale de transmission est de 15 unités de temps. Au plus 100 messages peuvent être transmis simultanément sur ce lien. De plus, en moyenne 0,001 % des messages de ce lien sont perdus. La multiplicité de 10 précise que 10 clients peuvent cohabiter simultanément dans le système à l'exécution. Le fait d'introduire une multiplicité au niveau des nœuds soulève le problème du transit des messages : les messages entre un client et un serveur transitent-t-il sur le même lien que les messages entre un autre client et ce même serveur ? Par défaut, nous proposons que les liens soient dupliqués i.e. que les messages ne transitent pas sur le même lien. Graphiquement, cette duplication de lien est identifiée par un disque évidé, cf. Figure 5. Dans le cas contraire où les messages transitent tous sur le même lien, ce lien unique est représenté graphiquement par un disque plein (cf. *Client2* sur la Figure 5).

De plus, un lien qui relie un nœud de multiplicité *n* vers un nœud de multiplicité *m* signifie que tout message émis sur ce lien est reçu par les *m* nœuds récepteurs. Si l'interface de réception du lien possède un disque plein, alors le message est considéré dupliqué à la réception. Sinon, il est dupliqué à l'émission.

Par exemple, dans le cas de la Figure 5, le nœud *Client1* est dupliqué à 10 exemplaires comportant chacun une instance du composant C_b, les instances communiquant avec le serveur via un lien indépendant. Par contre, les cinq composants C_b du nœud Client2 utilisent le même lien de communication pour envoyer un message au serveur.

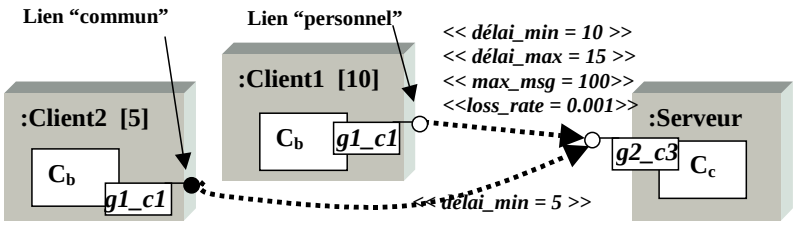


Figure 5. Modélisation d'un lien avec gigue et multi-clients.

3. Sémantique formelle des diagrammes de déploiement

Les algorithmes [LOH 02] de génération d'une spécification RT-LOTOS à partir d'une modélisation TURTLE acceptent en paramètre un diagramme de classes constitué de classes TURTLE dites *Tclasses* auxquels s'ajoutent des diagrammes de comportement inclus dans chacune de ces *Tclasses*. Nous proposons de nous appuyer sur les algorithmes définis dans TURTLE [LOH 02] dans le cadre de TURTLE-P.

La première étape de la traduction en RT-LOTOS consiste à transformer les diagrammes de déploiement en diagrammes de classes et d'activitéj TURTLE par les algorithmes de traduction de TURTLE-P : ce sont les nouveaux algorithmes que nous présentons ci-dessous.

Dans une deuxième étape, nous réutilisons les algorithmes de TURTLE pour générer la spécification RT-LOTOS à partir des diagrammes construits à l'étape précédente.

Les algorithmes de traduction TURTLE-P ne seront pas exposés in extenso compte tenu du nombre de pages alloué à l'article. Nous allons cependant donner une vue générale du processus de traduction.

Le diagramme de déploiement de TURTLE-P est un 2-uple <*N*, *L*> avec *N*, une liste de nœuds = $\{n_i, i \in 1..n\}$et *L*, une liste de liens = $\{l_i, i \in 1..m\}$.

Un nœud n_i est un t-uple <*C*, *mult*> avec *C* un ensemble de composants = $\{c_j, j \in 1..n_{ij}\}$ et *mult* la multiplicité du nœud.

Un lien est un 10-uple <n_o, n_d, *g1*, *g2*, *dmin*, *dmax*, *max_msg*, *loss_rate*, *type_o*, *type_d*> où n_o et n_d dénotent respectivement les nœuds origines et destinations, *g1* et *g2* dénotent les interfaces initiale et destination du lien, *dmin* et *dmax* représentent les délais minimum et

maximum de transit des messages, *max_msg* le nombre maximal de messages en transit à un instant t sur le lien et *loss_rate* le taux de perte moyen du lien. Enfin, *type_o* ∈ {*lien_personnel*, *lien_commun*} et *type_d* ∈ {*dup_à_origine*, *dup_à_destination*}.

Nous notons *Dc* le diagramme de classes TURTLE natif que nous construisons à partir de la modélisation TURTLE-P. Ce diagramme de classes est construit comme suit :

// Génération des classes de Dc
Pour chaque nœud n_i de N, pour chaque composant c_j de de n_i
    Pour k variant de 1 à la multiplicité de n_i
        Les classes de c_j sont renommées de « nom_initial » à « ni_k_cj_nom_initial »
        Les classes de c_j sont ajoutées à Dc
    Fin
Fin

// Construction des liens
Pour tout lien l_s = <n_{op}, n_{dp}, $g_x\_c_p$, $g_y\_c_q$, $dmin_p$ $dmax_p$, max_msg_p, $loss\_rate_p$, $type\_origine_p$, $type\_dest_p$> de L
    Si $type\_origine_p$ = *lien_personnel* alors
        Pour k variant de 1 à la multiplicité de n_{op}
            On ajoute à Dc une classe $Lg_x\_k\_c_p$ tel que le comportement de $Lg_x\_k\_c_p$ reflète le comportement du lien i.e. la gigue, la bande passante, le taux de perte et l'éventuelle duplication des messages avant ou après transmission (voir après l'algorithme)
            On ajoute une relation de Synchronisation entre $Lg_x\_k\_c_p$ et la classe c_q. La formule OCL {g_y} est ajoutée à l'association
        Fin
    Sinon
        On ajoute à Dc une classe $Lg_x\_c_p$ tel que le comportement de $Lg_x\_c_p$ représente le lien
        Pour k variant de 1 à la multiplicité de n_{op}
            On ajoute une relation de Synchronisation entre la classe *ni_k_cj_cp* et la classe $Lg_x\_c_p$. La formule OCL {$g_x = g_k$} est ajoutée à l'association
        Fin
        On ajoute une relation de Synchronisation entre $Lg_x\_c_p$ et la classe c_q. La formule OCL {g_y} est ajoutée à l'association
        Fin
    Fin
Fin

Finalement, un lien se caractérise par une relation de synchronisation entre, d'une part, la classe origine du lien et une classe C dont le comportement représente celui du lien et, d'autre part, la classe C et la classe désignée par le lien. Il n'est pas possible de détailler dans le cas général le comportement de la classe C, mais simplement, par exemple, dans le cas d'un lien entre une classe c1, porte g1, et une classe c3, porte g2. Le diagramme de déploiement correspondant est celui représenté à la Figure 3 (avec *n* qui vaut 1). Le diagramme de classes construit par les algorithmes précédents est représenté à la Figure . Le

lien est ainsi modélisé par deux relations de synchronisation et une classe *Lg1_c1* dont le comportement émule celui du lien représenté à la figure ?

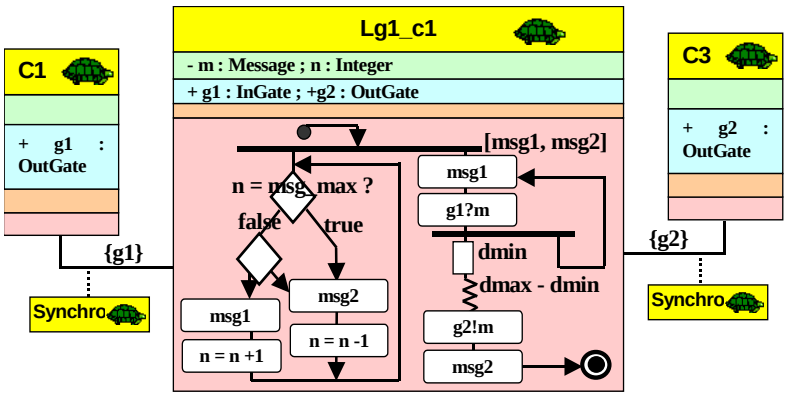


Figure 6. Représentation graphique en TURTLE d'un lien exprimé en TURTLE-P.

Bibliographie de l’auteur

1. Thèse

[SAQ 90] P. de Saqui-Sannes, Prototypage d'un environnement de validation de protocoles : application à l'approche Estelle, Doctorat de l'Université Paul Sabatier, Avril 1990.

2. Contributions à ouvrages (avec comité de lecture)

[SC 89] P. de Saqui-Sannes, J.-P. Courtiat, Rapid Prototyping of an Estelle Simulator, in M. Diaz, J.-P. Ansart, J.-P. Courtiat, P. Azéma, V. Chari, The Formal Description Technique Estelle – Results of the SEDOS Project, North Holland, January 1989.

[WSS 01] R. Willrich, P. de Saqui-Sannes, P. Sénac, M. Diaz, HTSPN: an Experiment in Formal Modeling of Multimedia Applications Coded in MHEG or Java, Invited paper, *Design Management of Multimedia Information Systems: Opportunities and Challenges*, S.M.Rahman, Idea Group Publishing, pp.380-411, 2001.

3. Revues scientifiques (avec comité de lecture)

[CS 87] J.-P. Courtiat, P. de Saqui-Sannes, Relations sémantiques entre Estelle et les réseaux de Petri, *Bigre+Globule*, N°55, juillet 1987.

[BSB 92a] M. Barbeau, P. de Saqui-Sannes, G. von Bochmann, Conception et spécification par objets du contrôle centralisé d'un système de transmission, *Réseaux et Informatique Répartie*, Vol.2, N°1, pp.7-20, 1992.

[CS 92] J.-P. Courtiat, P. de Saqui-Sannes, ESTIM: an integrated environment for the simulation and verification of OSI protocols specified in Estelle*, *Computer Networks and ISDN Systems*, Vol.25, pp.83-98, 1992.

[SDS 94] P. Sénac, M. Diaz, P. de Saqui-Sannes, Towards a Formal Specification of Multimedia Synchronization Scenarios, *Annales des Télécommunications*, N°5-6, pp.297-314, 1994.

[SDL 96] P. Sénac, M. Diaz, A. Léger, P. de Saqui-Sannes, Modeling Logical and Temporal Synchronization in Hypermedia Systems, *IEEE Journal on Selected Areas in Communications*, Vol.14, N°1, pp.84-103, January 1996.

[DAS 01] K. Drira, P. Azéma, P. de Saqui-Sannes, Testability Analysis in Communicating Systems, *Computer Networks*, Vol.36, N°5-6, pp.671-693, 2001.

[WSS 02] R. Willrich, P. de Saqui-Sannes, P. Sénac, M. Diaz, Multimedia authoring with hierarchical timed stream Petri nets and Java, *Multimedia Tools and Applications*, Vol.16, N°1-2, pp.7-27, 2002.

[SAL 02] P. de Saqui-Sannes, L. Apvrille, C. Lohr, P. Sénac, J.-P. Courtiat, UML and RT-LOTOS: an Integration for Real-time System Validation, *Journal Européen des Systèmes Automatisés*, Vol.36, N°36, pp.1029-1042, 2002.

[ASS 04] L. Apvrille, P. de Saqui-Sannes, P. Sénac, C. Lohr, Verifying Service Continuity in a Dynamic Reconfiguration Procedure: Application to a Satellite System, *Automated Software Engineering*, Vol.11, No.2, April 2004.

[ACL 04] L. Apvrille, J.-P. Courtiat, C. Lohr P. de Saqui-Sannes, TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit, *IEEE Transactions on Software Engineering*, Vol. 30, no. 7, pp.473-487, July 2004.

[ASK 05a] L. Apvrille, P. de Saqui-Sannes, F. Khendek, TURTLE-P: a UML Profile for the Formal Validation of Critical and Distributed Applications, to appear in *SOSYM, Journal of Software and System Modeling*, 2005.

4 Conférences internationales avec comité de programme et actes publiés chez un éditeur

[SC 88] P. de Saqui-Sannes, J.-P. Courtiat, ESTIM : an Interpreter for the Simulation of Estelle descriptions, in K.J. Turner, 1st International Conference on Formal Description Techniques (FORTE'88), Stirling, UK, North Holland.

[SC 89] P. de Saqui-Sannes, J.-P. Courtiat, From the simulation to the verification of Estelle specifications, Invited paper, 2nd Int. Conf. on Formal Description Techniques (FORTE'89), Vancouver, Canada, Dec. 1989, pp.524-541 Formal Description Techniques II, S.T.Vuong (Ed.), North Holland, 1990, pp.393-407.

[SC 89a] P. de Saqui-Sannes, J.-P. Courtiat, ESTIM: Simulating Estelle Descriptions of OSI Protocols, Singapore International Conference on Networs (SICON'89), Singapore, July 1989.

[BCS 90a] M. Baretto, J.-P. Courtiat, P. de Saqui-Sannes, Experience in using ESTELLE* for the specification and verification of a field bus protocol : FIP, COMNET'90, Budapest, Hungary, May 1990. Computer Networking, Ed. L.Csaba, T.Szentuvanyi, K.Tarnay, North Holland, 1990, pp.295-304.

[CDM 91a] J.-P. Courtiat, M. Diaz, V.B. Mazzola, P. de Saqui-Sannes, Description formelle de protocoles et de services OSI en ESTELLE et ESTELLE*. Expérience et méthodologie, Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'91), Pau, France, Septembre 1991, pp.65-98, Hermès.

[SC 91] P. de Saqui-Sannes, J.-P. Courtiat, An extension of the multi-way synchronization mechanism concealed by ESTELLE, 11th IFIP Symposium on Protocol Specification, Testing and Verification, Stockholm, Sweden, June 1991, pp.68-79.

[CSC 92] L.F.R. da Costa Carmo, P. de Saqui-Sannes, J.-P. Courtiat, Basic synchronisation concepts in multimedia systems, 3rd IEEE Int. Workshop on Network and Operating System Support for Digital Audio and Video, San Diego, USA, Nov. 1992, pp.85-96.

[CCS 93a] L.F. Rust da Costa Carmo, J.-P. Courtiat, P. de Saqui-Sannes, Towards multimedia communication services, 4th Workshop on Future Trends of Distributed Computing Systems, Lisbon, Portugal, pp.83-89, Sept. 1993.

[DSS 93] M. Diaz, P. Sénac, P. de Saqui-Sannes, Un modèle formel pour la spécification de la synchronisation multimédia en environnement distribué, CFIP'93 (Colloque Francophone sur l'Ingénierie des Protocoles), Montréal, Canada, Septembre 1993.

[SKC 93] P. de Saqui-Sannes, K. Drira, J.-P. Courtiat, P. Azéma, Séquences de test et testeurs canoniques dérivables de spécifications Estelle*: une expérience avec le protocole MMS, Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'93), Montréal, Canada, Septembre 1993.

[YSS 94] J. Yee, P. Sénac, P. de Saqui-Sannes, Resource Synchronization Specification and Modeling Based on Timed Stream Petri Nets, ISMM International Conference Distributed Multimedia Systems and Applications, Hawaii, Aug. 1994, pp.171-175.

[SCC 94] P. de Saqui-Sannes, J.-P. Courtiat, R. Casadessus, Verification by abstraction as a preamble for interoperability test suite generation, Protocol Specification, Testing and Verification (PSTV), Vancouver, Canada, pp.145-160, June 1994.

[SSW 95] P. Sénac, P. de Saqui-Sannes, R. Willrich, Hierachical Time Stream Petri Net: a Model for Hypermedia Systems, 16th International Conference on Application and Theory of Petri Nets, Turin, Italy, June1995, pp.451-470.

[WSD 96] R. Willrich, P. Sénac, M. Diaz, P. de Saqui-Sannes, A formal framework for the specification, analysis and generation of standardized hypermedia documents, 3rd IEEE International Conference on Multimedia Computing and Systems, Hiroshima, Japan, June 1996, pp.399-406.

[WSS 96a] R. Willrich, P. de Saqui-Sannes, P. Sénac, M. Diaz, Hypermedia document design using the HTSPN model, 3rd International Conference on Multimedia Modeling (MMM'96), Toulouse, France, Novembre 1996, pp.151-166.

[WSS 96b] R. Willrich, P. de Saqui-Sannes, P. Sénac, M. Diaz, Une approche formelle pour le développement de documents hypermédias normalisés, Colloque Francophone sur l'Ingénierie des Protocoles (CFIP'96), Rabat, Maroc, Octobre 1996, pp.393-407.

[SCS 98] C.A.S. Santos, J.-P. Courtiat, P. de Saqui-Sannes, A design methodology for the formal specification and verification of hypermedia documents, FORTE-PSTV'98, Paris, France, in S.Budkowski, A.Cavalli, E.Najm (eds) Formal Description Techniques and Protocol Specification, Testing and Verification, Kluwer Academic Publishers, 1998, pp.163-178.

[SDC 99] P. de Saqui-Sannes, L. Dairaine, C. Chassot, Expérimentation d'outils pédagogiques destinés à l'ingénierie des protocoles, CFIP'99 (Colloque Francophone sur l'Ingénierie des Protocoles), Nancy, France, avril 1999, Hermès, pp. 49-63.

[SDS 93] P. Sénac, M. Diaz, P. de Saqui-Sannes, A formal environment for the specification and design of multimedia synchronisation scenarios, 4th Int. Workshop on Network and Operating System Support for Digital Audio and Video, Lancaster, UK, November 1993, pp.77-82.

[ASC 01] L. Apvrille, P de Saqui-Sannes, C. Lohr, P. Sénac, J.-P. Courtiat, A New UML Profile for Real-time System Formal Design and Validation, 4th Int. Conf. on the Unified Modeling Language (UML'2001), Toronto (Canada), in M.Gogolla, C.Kobryn (eds), LNCS 2185, Springer, pp.287-301, 2001.

[ASS 01] L. Apvrille, P de Saqui-Sannes, P. Sénac, M. Diaz, Formal Modeling and Validation of Space-Based Software in the Context of Dynamic Reconfiguration, conference on DAta Systems in Aerospace (DASIA'2001), Nice, France, 2001, 6p.

[SAC 01] P de Saqui-Sannes, L. Apvrille, C. Lohr, P. Sénac, J.-P. Courtiat, UML et RT-LOTOS : Vers une intégration informel/formel au service de la validation de systèmes temps réel, MSR'2001 (Colloque Francophone sur la Modélisation des Systèmes Réactifs), Toulouse, France, Octobre 2001, Hermès, pp.513-528.

[ASS 02] L. Apvrille, P. de Saqui-Sannes, P. Sénac, C. Lohr, Reconfiguration dynamique de protocoles embarqués à bord de satellites, CFIP'2002 (Colloque Francophone sur l'Ingénierie des Protocoles), Montréal, Canada, Hermès, pp.441-454, 2002.

[ASK 03] L. Apvrille, P. de Saqui-Sannes, F. Khendek, TURTLE-P : un profil UML pour la validation d'architectures distribuées, CFIP'2003 (Colloque Francophone sur l'Ingénierie des Protocoles), Paris, France, hermès, p. 17-42, octobre 2003.

[LAS 03] C. Lohr, L. Apvrille, P. de Saqui-Sannes, J.-P. Courtiat, New Operators for the TURTLE Real-Time UML Profile, FMOODS'03, Paris, France in E. Najm, U.Nestmann, P.Stevens, LNCS 2884, Springer, pp.214-228, 2003.

[ASK 05b] L. Apvrille, P. de Saqui-Sannes, F. Khendek, Synthèse d'une conception UML temps-réel à partir de diagrammes de séquences in R. Castanet (éd.), CFIP'2005 (Colloque Francophone sur l'Ingénierie des Protocoles), Bordeaux, France, mars 2005, Hermès.

[SSC 05a] T. Sadani, P. de Saqui-Sannes, J.-P. Courtiat, Validation de spécifications RT-LOTOS Une interface vers l'outil TINA, accepté pour publication à MSR'05, Autrans, France.

[SSC 05b] T. Sadani, P. de Saqui-Sannes, J.-P. Courtiat, From RT-LOTOS to Time Petri Nets : New Foundantions for a Verification Platform, accepted for publication at SEFM'05, Koblentz, Germany.

5 Conférences internationales avec comité de programme et actes à diffusion restreinte

[BSC 92b] M. Barbeau, P. de Saqui-Sannes, G. von Bochmann, Conception et Spécification par objets du contrôle centralisé d'un système de transmission, MCSEAI'92, Tunis, Tunisie, Avril 1992.

[SDS 94] P. Sénac, M. Diaz, P. de Saqui-Sannes, Un modèle formel pour la spécification et la vérification de systèmes temps-réels, Real-Time Systems (RTS'94), Paris, France, Janvier 1994, pp.35-46, Teknea.

[SSD 96] P. Sénac, P. de Saqui-Sannes, M. Diaz, F. Fabre, Un modèle formel unificateur pour les systèmes temps réels, Real-Time and Embedded Systems (RTS & ES'96), Paris, France, 11-12 Janvier 1996, pp.197-207.

[SS 97] P. de Saqui-Sannes, A.-E.-K. Sahraoui, General Purpose vs. Message Passing Specification Methods : a Case Study with Statecharts and Estelle on an Avionics Systems, International Symposium on Dynamic Modeling of Information Systems (ISDMIS'97), Yamagata, Japan, November 1997.

[SC 98] P. de Saqui-Sannes, J.-P. Courtiat, Teaching Estelle: Lessons from Experience, Estelle'98, INT, Evry, France, November 1998.

[SCC 00] P de Saqui-Sannes, C. Lohr, J.-P. Courtiat, P. Sampaio, TURTLE: A Timed UML and RT-LOTOS Environment, Workshop on Formal Methods for Real-Time UML, UML'2000 Conference, York, UK, October 2000.

[SSC 04] T. Sadani, P. de Saqui-Sannes, J.-P. Courtiat, Formal Validation of RT-LOTOS Specifications: New Directions and Preliminary Results, RTSS'04, Work in Progress Session, Lisbon, Portugal, 2004.

6 Conférences nationales avec comité de programme et actes à diffusion restreinte

[SC 89b] P. de Saqui-Sannes, J.-P. Courtiat, Some Impacts of the Estelle Approach for the Specification and Verification of OSI Protocols, 1st Maghrebin Conference on Artificial Intelligence and Software Engineering, Constantine, Algeria, Sept. 1989.

[BCS 90b] M.L. Barretto, J.-P. Courtiat, P. de Saqui-Sannes, Utilizando Estelle* para a especifiacacao e verificacao de um protocolo para redes locais industriais : FIP, 8° Congresso Brasileiro de Automatica, Belem, Brazil, pp.591-598, Septembre 1990.

[BSB 91] M. Barbeau, P. de Saqui-Sannes, G. von Bochmann, Design Formal Specification and Validation of Centralized and Distributed Control in a Transmission System, Bell Quality Engineering Workshop, October 1991.

[MRS 93] V.B. Mazzola, L.F. Rust da Costa Carmo, P. de Saqui-Sannes, J.-P. Courtiat, Utilização da técnica Estelle na validação de protocolos de alto nível: metodologia, ferramentas e experiência, 11° Sympósio Brasileiro de Redes de Computadores, Maio de 1993.

[WSS 96c] R. Willrich, P. Sénac, P. de Saqui-Sannes, M. Diaz, Towards hypermedia documents design, XIV Brazilian Symposium on Computer Networks (SBRC/96), Fortaleza, Brasil, May 1996, pp.473-491.

[WS 97] R. Willrich, P. de Saqui-Sannes, Concepção Formal de Applicações Multímídia Java, 15° sympósio Brasileiro de Redes de Computadores, São Carlos, Brasil, Maio de 1997.

[SAC 04] P. de Saqui-Sannes, L. Apvrille, C. Lohr, J.-P. Courtiat, TURTLE : un pont entre UML et RT-LOTOS, Approches Formelles pour le Développement du Logiciel (AFADL'04), Besançon, France, juin 2004.

[SAL 04] P. de Saqui-Sannes, L. Apvrille, C. Lohr, J.-P. Courtiat, Derniers développements autour du profil UML temps réel TURTLE, Journées FAC (Formalisation des Activités Concurrentes), Toulouse, France, mars 2004.

[SCS 05a] T. Sadani, J.-P. Courtiat, P. de Saqui-Sannes, Validating RT-LOTOS Specifications using the TINA tool, Journées FAC'05 , Toulouse, France, février 2005.

7. Rapports de contrats

[SC 86] P. de Saqui-Sannes, J.-P. Courtiat, ESTIM : An Interpretative Machine for the Simulation of Estelle Descriptions, Report SEDOS/B3/083, November 1986.

[Saq 87] P. de Saqui-Sannes, LAAS Contribution to the Definition of the Estelle Workstation, ESPRIT/SEDOS-Estelle-Demonstrator Report LAA-T11-001, January 1987.

[SC 87a] P. de Saqui-Sannes, J.-P. Courtiat, ESTIM : an Interpreter for the Simulation of Estelle Descriptions, ESPRIT-SEDOS/B3/115, November 1987.

[SC 87 b] P. de Saqui-Sannes, J.-P. Courtiat, Note on the Integration of the Rendez-Vous mechanism in Estelle, ESPRIT-SEDOS-Estelle-Demonstrator Report LAA-T14-001, June 1987.

[CCS 92a] L.F.R. da Costa Carmo, J.-P. Courtiat, P. de Saqui-Sannes, Modélisation et vérification du protocole MMS Contrat EDF/DER-LAAS, Décembre 1992.

[CCS 92b] L.F.R da Costa Carmo, J.-P. Courtiat, P. de Saqui-Sannes, Dérivation de séquences de test à partir de spécifications Estelle : application aux modèles MMS, Contrat LAAS-EDF, Novembre 1992.

[CCE 92] L.F.R. da Costa Carmo, J.-P. Courtiat, R. El Neklaoui, P. de Saqui-Sannes, Modélisation et vérification du protocole MMS, Contrat LAAS-EDF, Juillet 1992.

[CCS 93b] J.-P. Courtiat, L.F.R. da Costa Carmo, P. de Saqui-Sannes, L. Besse, L. Dairaine, E. Horlait, Synchronisation multimedia. Terminologie et concepts de base, Contrat CNET FT N°92.1B.178. Lot 2, Avril 1993.

[CLD 93] C. Chassot, A. Lozes, M. Diaz, P. de Saqui-Sannes, Spécification en Estelle d'un sous ensemble du protocole de transport haute vitesse XTP et mesures de performances, Contrat CNET FT N°92.1B.178. Lot 1, Avril 1993, 59p. Contrat CNET FT N°92.1B.178. Lot 1, Avril 1993.

[SC 93] P. de Saqui-Sannes, J.-P. Courtiat, Evaluation de la méthodologie Estelle pour le test d'applications réparties, Contrat EDF/DER-LAAS, Décembre 1993.

[BCD 94] V. Thomas-Baudin, G. Cicchelero, M. Diaz, P. Sénac, P. de Saqui-Sannes, Modélisation et implantation d'applications multimédia, Contrat Région Midi-Pyrénées N°9300088, Septembre 1994.

[CFA 94] J.-P. Courtiat, M. Filali-Amine, A. Léger, R.C. de Oliveira, P. Sénac, R. Willrich, Modèles pour la synchronisation multimédia et relations entre modèles, Contrat CNET FT N°92.1B.178. Projet CESAME. Lot 2, Déc. 1994.

[FSS 95] F. Fabre, P. Sénac, P. de Saqui-Sannes, M. Diaz, Un modèle et un atelier pour la spécification formelle de systèmes hypermédias, Contrat Région Midi-Pyrénées N° RECH/9407495, Décembre 1995.

[DSS 98] M. Diaz, P. de Saqui-Sannes, P. Sénac, R. Willrich, Modélisation et atelier logiciel pour applications multimédias, Contrat Région Midi-Pyrénées N°9407495, Mars 1998.

Bibliographie du domaine

[ABD 03] S. Abdellatif, G. Juanole, Evaluation de performances de protocoles de communications, Vérification et mise en oeuvre des réseaux de Petri, Hermes Science, *Traité IC2 Information-Commande-Communication*, 2003, Chapitre 11, pp.357-384.

[ABR 96] J.-R. Abrial, *The B-Book: Assigning Programs to Meanings*, Cambridge University Press, 1996.

[ALL 94] R. Allur, D. Dill, A Theory of Timed Automata, Theoretical Computer Science, 126:183-235, 1994.

[APV 02] L. Apvrille, Contribution à la reconfiguration dynamique de logiciels embarqués temps-réel : application à un environnement de télécommunication par satellite, Doctorat, Institut National Polytechnique de Toulouse, Juin 2002.

[BAB 02] F. Babich, L. Deotto, Formal Methods for Specification and Analysis if Communication Protocols, *IEEE Communications Surveys*, Vol.4, No.1, 2002.

[BEN 96] J. Bengtsson, K. Larsen, F. Laarson, P. Pettersson, W. Yi, UPPAAL—a tool suite for automatic verification of real-time systems, Proc. of the DIMACS/SYCON workshop on Hybrid systems III: verification and control: verification and control, Springer Verlag, 1996.

[BER 91] B. Berthomieu, M. Diaz, Modeling and Verification of Time-Dependant Distributed Systems Using Time Petri Nets, *IEEE Transactions on Software Engineering*, Vol.17, N°3, pp.259-273, Mars 1991.

[BER 04] B. Berthomieu, P.O. Ribet, F. Vernadat, The TINA Tool: Construction of Abstract State Space for Petri Nets and Time Petri Nets, *International Journal of Production Research*, Vol.42, N°14, pp.2741-2756, 2004.

[BF 98] J-M. Bruel, R.B.France, Transforming UML models to formal specifications, UML'98, LNCS, Springer, 1998.

[BHH 97] R. Breu, U. Hinkel, C. Hofmann, C. Klein, B. Paech, B. Rumpe, V. Thurner Towards a Formalization of the Unified Modeling Language, ECOOP'97 - Object-Oriented Programming, Mehmet Aksit, Satoshi Matsuoka (ed.), Jyväskylä, Finland, June 1997, Springer Verlag, LNCS 1241.

[BJO 00] M. Bjorkander, Real-Time Systems in UML (and SDL), *Embedded System Engineering*, October/November 2000.

[BOC 89] G. von Bochman, M. Barbeau, A. Bean, M. Erradi, L. Lecomte L, The Specification language MONDEL, Object-Oriented Databases: Modelling and Specification of Applications in the field of Network Management, CRIM/BNR report, May 1989.

[BOC 90] G. von Bochman, Specifications of a Simplified Transport Protocol Using Different Formal Description Techniques, *Computer Networks*, Vol. 18, No.5, June 1990.

[BOL 95] T. Bolognesi, J. van Lagemaat, C. Vissers, LOTOSsphere: Software Development with LOTOS, Kluwer Academic Publishers, January 1995.

[BRI 04] E. Brinksma, L. B. Brionès, Testing Time on Model-Driven Test Generation for Nondeterministic Real-Time Systems, Int. Conf. on Application of Concurrency to System Design (ACSD'2004), Hamilton, Canada, June 2004.

[BS 91] F. Boussinot, R. de Simone, The ESTEREL language, *Proceedings of the IEEE*, Vol. 79, No. 9, September 1991.

[BRU 03] B. Bruegge, A. H. Dutoit, Object-oriented Software Engineering: Using UML, Patterns and Java, Prentice-Hall, September 2003.

[CADP] http://www.inrialpes.fr/vasy/cadp/

[CAS 87] P. Caspi et al., LUSTRE: A Declarative Language for Programming Synchronous Systems, 14th Ann ACM Symp on Princ Prog Langs, 1987.

[CAS 03] R. Castanet, M Mackaya, A. Cavalli et al., Une plateforme de validation multi-protocoles et multi-services – résultats d'expérimentation, CFIP'03 (Colloque Francophone sur l'Ingénierie des Protocoles), Paris, France, oct. 2003,Hermès.

[CM 00] Clarck R.G., Moreira A.M.D., Use of E-LOTOS in Adding Formality to UML, Journal of Universal Computer Science, Vol.6, No. 11, p. 1071-1087, 2000.

[COU 84] J.-P. Courtiat, J.-M. Ayache, B. Algayres, Petri Nets are Good for Protocols, *Computer Communication Review*, Vol.14, n°2, 1984.

[COU 87a] J.-P. Courtiat, How Could Estelle Become a Better FDT?, PSTV'87 (Protocol Specification Testing and Verification, Zurich, Switzerland, 1987.

[COU 87b] J.-P. Courtiat, Contribution à la description formelle de protocoles, Thèse d'Etat, Institut National Polytechnique de Toulouse, 1987.

[COU 95] J.-P. Courtiat, D.-E. Saidouni, A Case-Study on Protocol Design, in T. Bolognesi, J. van de Lagemaat, C. Vissers (eds.), *LOTOSphere: Software Development with LOTOS*, Kluwer Academic Publishers, pp. 201-217, 1995.

[COU 00] J.-P. Courtiat, C.A.S. Santos, C. Lohr, B. Outtaj, "Experience with RT-LOTOS, a Temporal Extension of the LOTOS Formal Description Technique", *Computer Communications*, Vol. 23, No. 12, p. 1104-1123, 2000.

[COU 03] J.-P. Courtiat, "Formal Design of Interactive Multimedia Documents", Invited talk, 23rd IFIP WG 6.1 International Conference on Formal Techniques for Networked and Distributed Systems (FORTE'2003), Berlin, Germany, Sep-Oct. 2003, LNCS 2767, Springer, pp.351-366.

[DIA 82] M. Diaz, Modeling and Analysis of Communication and Cooperation Protocols using Petri Nets Based Models, *Computer Networks*, Vol.6, No.6, December 1982.

[DIA 89a] M. Diaz, C. Vissers, SEDOS: Designing Open Distributed Software, *IEEE Software*, Vol.6, No.6, November 1989.

[DIA 89b] The Formal Description Technique Estelle – Results of the SEDOS Project, Edited by M. Diaz, J.-P. Ansart, J.-P. Courtiat, P. Azéma, V. Chari, North-Holland, January 1989.

[DIA 89c] M. Diaz, J. Dufau, R. Groz, Experiences using ESTELLE within SEDOS ESTELLE Demonstrator, FORTE'89, Vancouver, BC, Canada, December 1989, pp.596-613.

[DIA 93] M. Diaz, L'explosion des technologies multimédias, *La Recherche*, Vol.24, N°258, pp.1126-1127, Octobre 1993

[DIA 94] M. Diaz, G. Pays, The CESAME project. Formal Design of High Speed Multimedia Co-operative Systems, *Annales des Télécommunications*, N°5-6, pp.220-229, Mai-Juin 1994.

[DIA 01] M. Diaz, P. Sénac, Composition temporelle et réseaux de Petri à flux autonomes, Les Réseaux de Petri. Modèles fondamentaux, Hermes Science,

Traité IC2 Information-Commande-Communication, N°ISBN 2-7462-0250-6, 2001, Chapitre 6, pp.199-219.

[DJP 05] Werner Damm, Bernhard Josko, Amir Pnueli, Angelika Votintseva A discrete-time UML semantics for concurrency and communication in safety-critical applications In Science of Computer Programming 2005.

[DOL 03] L. Doldi, *UML2 illustrated: Developing Real-Time and Communications Systems*, TMSO, October 2003.

[DOR 04] J. Dorsch, A. Ek, R. Gotzhein, SPT: the SDL Pattern Tool, Proceedings of SAM'04, 4th SDL and MSC Workshop, Ottawa, Canada, June 2004.

[DOU 02] Douglass B.P., Real-Time UML Tutorial, OMG Real-Time and Embedded Distributed Object Computing Workshop, Arlington, VA, USA, July 2002.

[DRI 92] K. Drira, Transformation et composition de graphes de refus : analyse de la testabilité, Doctorat de l'Université Paul Sabatier, Toulouse, Octobre 1992.

[DB 01] S. Dupuy, L. du Bouquet, A Multi-formalism Approach for the Validation of UML Models, *Formal Aspects of Computing*, No.12, p.228-230, 2001.

[END 01] Espinosa J. M., Nabuco O., Drira K., "A UML Model for Session Management in Collaborative Design for Space Activities", 8th European Concurrent Engineering Conference (ECEC'2001), Valence, Spain, April 2001.

[ENN 02] A. En-Nouaary, R. Dssouli, F. Khendek, "Timed Wp-Method: Testing Real-Time Systems", *IEEE Transactions on Software Engineering*, Vol. 28, No. 11, November 2002.

[ENN 03] A. En-Nouaary, R. Dssouli, A Guided Method for Testing Timed Input Output Automata, Conference on Testing Communication Systems (TestCom'2003), Sophia Antipolis, France, May 2003.

[EST 89] ISO 9074 -- Information processing systems -- Open Systems Interconnection -- Estelle: A formal description technique based on an extended state transition model, 1989.

[EXP 03] E. Exposito, Spécification et mise en oeuvre d'un protocole de transport orienté Qualité de Service pour les applications multimédias, Doctorat de l'Institut National Polytechnique, Toulouse, Décembre 2003.

[FEL 03] P. Félix, R. Castanet, Objectifs de test pour systèmes temporisés, CFIP'03 (Colloque Francophone sur l'Ingénierie des Protocoles), Paris, France, octobre 2003, ed. Hermès.

[FUJ 91] S. Fujiwara, G. von Bochmann, F. Khendek, M. amalou, A. Ghedamsi, Test Selection Based on Finite State Models, *IEEE Trans. on Software Engineering*, Vol. 17, No.6, 1991.

[GLA 90] The Linear Time – Branching Time Spectrum, The Semantics of Concrete, Sequential Processes, Proceedings CONCUR '90, Amsterdam (J.C.M. Baeten & J.W. Klop, eds.), LNCS 458, Springer-Verlag, 1990, pp. 278-297.

[GBP 86] P. Le Guernic, A. Benveniste, P. Bournai, T. Gautier, Signal: a Data flow language for signal processing, *IEEE Transactions on Signal Processing*, Vol.34, No.2, April 1986.

[GOO 05a] I. Ober, S. Graf, I. Ober, Validating Timed UML models by Simulation and Verification, to appear in *Software Tools For Technology Transfer*, 2005.

[GOO 05b] S. Graf, I. ,Ober, I. Ober, A Real-Time UML profile, submitted for publication to *Software Tools For Technology Transfer*, 2005.

[GOD 04] J. Godskesen, B. Nielsen, A. Skou, Connectivity Testing Through Model-Checking Formal Techniques for Networked and Distributed Systems (FORTE 2004), Madrid, Spain, Sept. 2004, Springer, pp. 167-184.

[GRA 04] R. Grammes, R. Gotzhein, Towards the Harmonisation of UML and SDL, Formal Techniques for Networked and Distributed Systems (FORTE 2004), Madrid, Spain, Sept. 2004, Springer, pp. 61-78.

[GTT 02] S. Gerard, F. Terrier, Y. Tanguy, Using the Model Paradigm for Real-Time Systems Development: ACCORD/UML, Proceedings of the Advances in Object-Oriented Information Systems, OOIS 2002, LNCS 2426, pp.260-269.

[GT 03] S. Gérard, F. Terrier, UML for real-time: which native concepts to use?, UML for real: design of embedded real-time systems, Kluwer Academic Publishers, Norwel, MA, USA, 2003.

[GRO 00] R. Groz, Hypothèses pyrrhoniennes sur le génie logiciel pour les télécommunications, mémoire d'Habilitation à Diriger les Recherches, Université de Bordeaux I, 2000.

[GWS 03] Z. Gu, S. Wang, K.G. Shin, Issues in Mapping from UML Real-Time Profile to OSEK API, University of Michigan, October 2003.

[HOL 97] G.J. Holzmann. The Model Checker SPIN. *IEEE Transactions on Software Engineering*, 23(5):279-295, 1997.

[HOO 02] Hooman J., Towards Formal Support for UML-based Development of Embedded Systems, Proceedings of PROGRESS 2002, Utrecht, The Netherlands, October 2002.

[JAR 88] C. Jard, J.-F. Monin, R. Groz, Development of Véda, a Prototyping Tool for Distributed Algorithms, *IEEE Transactions on Software Engineering*, 14(3), pp. 339-352, 1988.

[JJP 98] C. Jard, J.-M. Jézéquel, F. Pennaneach, Vers l'utilisation d'outils de validation de protocoles dans UML, *Techniques et Sciences Informatiques*, Vol.15, No. 11, pp. 1-15, 1998.

[JBR 00] Y. Jacobson, G. Booch, J. Rumbaugh, Le processus de développement unifié du logiciel, éditions Eyrolles, 2000.

[JS 00] Jaragh M., Saleh K.A., Modeling Communications protocols using the Unified Modeling Language, TENCON'2000, Intelligent Systems and Technologies for the New Millenium, Kuala Lumpur, Malaysia, Sept. 2000.

[JUR 04] Jan Jürjens, *Secure Systems Development with UML*, Springer-Verlag 2004.

[KKB 03] Kavi K., Kung D.C., Bhaambhani H., Pancholi G., Kanikarla M., Sah R., "Extending UML to Modeling and Design of Multi-Agent Systems"; Workshop on Software Engineering for Large Multi-Agents Systems, associated with ICSE'2003.

[KHF 05] E. Kohler, M. Handley, S. Floyd, Datagram Congestion Protocol, IETF draft, March 2005, http://www.icir.org/kohler/dcp/draft-ietf-dccp-spec-11.txt.

[KRONOS] http://www-verimag.imag.fr/TEMPORISE/kronos/

[KUR 90] A. Kurokawa, R. Dssouli, A. Das, A Design Framework for Highly Available Communication Systems, *Computer Standards & Interfaces*, Vol.14, North-Holland, 1993, pp.375-391.

[LL 97] G. Leduc, L. Léonard, A Timed LOTOS supporting a dense time domain and including new timed operators, FORTE'92, 5th International Conference on Formal Description techniques, Lannion, France, 1992.

[LEG 91] P. Le Guernic, T. Gautier, M. Le Borgne, C. Le Maire, Programming Real-Time Applications with Signal, *Proceedings of the IEEE*; 79(9):1321-1336, Sept. 1991.

[LIN 01] Lind J., Specifying Agent Interaction Protocols with Standard UML, Second International Workshop on Agent-Oriented Software Engineering (AOSE-2001), LNCS 2222.

[LLO 90] J.-C. Lloret, Réseaux prédicat/transition étiquetés pour la modélisation et la vérification des systèmes informatiques répartis, Doctorat de l'Université Paul Sabatier, Toulouse, Juillet 1990.

[LOT 89] ISO 8807 -- Information processing systems -- Open Systems Interconnection – LOTOS: A formal description technique based on the temporal ordering of observational behaviour, 1989.

[LOH 02] C. Lohr, Contribution à la conception de systèmes temps-réel s'appuyant sur la technique de description formelle RT-LOTOS, Doctorat de l'Institut National Polytechnique, Toulouse, Décembre 2002.

[LQV 01] L. lavazza, G. Quaroni, M. Venturelli, Combining UML and formal notations for modelling real-time systems, Proceedings of the 8th European software engineering conference held jointly with 9th ACM SIGSOFT international symposium on Foundations of software engineering, 2001.

[MAZ 91] Contribution à la conception de systèmes flexibles de production. Application de la technique de description formelle Estelle, Doctorat de l'Université Paul Sabatier, Toulouse, Novembre 1991.

[MDA] http://www.omg.org/mda/

[MIL 80] R. Milner, A Calculus of Communicating Systems, LNCS 92, Springer.

[MLG 01] G. Martin, L. Lavagno, J. Louis-Guerin, Embedded UML: a Merger of Real-Time UML and Co-design, Cadence Design Systems, March 2001.

[OME] IST-2001-33522 OMEGA, Correct Development of Real-Time Embedded Systems, http://www-omega.imag.fr/.

[PET 01] A. Petrenko, Fault model-driven test derivation from finite state models: annotated bibliography, in Modeling and verification of parallel processes, pp. 196 – 205, 2001.

[PUML] http://www.puml.org/

[QUE 04] J. Quémada, Formal Description Techniques and Software Engineering: Some Reflections after 2 Decades of Research, *Formal Techniques for Networked and Distributed Systems (FORTE 2004)*, Madrid, Spain, Sept. 2004, Springer, pp. 23-42.

[RHA] http://www.ilogix.com/rhapsody/rhapsody.cfm

[RUS 94] L.F. Rust da Costa Carmo, Méthodes de synchronisation dans les systèmes répartis multimédia : une approche intégrant relations de causalité et contraintes temporelles, Doctorat de Université Paul Sabatier, Toulouse, France, Novembre 1994.

[SAF 05] Projet SAFECAST, Sécurité des communications de groupe, http://www.telecom.gouv.fr/rnrt/rnrt/projets/safecast.htm.

[SAL 04] A. Salah, R. Mizouni, R. Dssouli, B. Parreaux, Formal Composition of Distributed Scenarios, *Formal Techniques for Networked and Distributed Systems (FORTE 2004*), Madrid, Spain, Sept. 2004, Springer, pp. 213-228.

[SDL 99] ITU-T Z.100, Specification and Description Language (SDL), http://www.itu.int/ITU-T/studygroups/com10/languages/Z.100_1199.pdf,1999.

[SEL 01] B. Selic, A UML Profile for Modeling Complex Real-Time Architectures, http://www.omg.org/news/meetings/workshops/presentations/realtime2001/6-3%20Selic.presentation.pdf.

[SEN 96] P. Sénac, Contribution à la modélisation des systèmes multimédias et hypermédias, Doctorat de INPT, Toulouse, Juin 1996.

[SOM 96] S.S. Somé, R. Dssouli, J. Vaucher, Un cadre pour l'ingénierie des exigences avec des scénarios, CFIP'96 (Colloque Francophone sur l'Ingénierie des Protocoles), octobre 1996, Rabat, Maroc.

[STL] http://www.cs.queensu.ca/home/stl/internal/uml2/index.html

[TAU] http://www.telelogic.com/products/tau/tg2.cfm

[TTCN] http://www.etsi.org/ptcc/ptccttcn3.htm

[TRA 00] I. Traoré, An Outline of PVS Semantics for UML Statecharts, Journal of Universal *Computer Science*, Vol. 6, No. 11, p. 1088-1108, 2000.

[UML 03] Object Management Group, Unified Modeling Language Specification, Version 1.5, http://www.omg.org/docs/formal/03-03-01.pdf, March 2003.

[VER 01] F. Vernadat, Contribution à la modélisation et à la vérification des systèmes communicants, Habilitation à Diriger les Recherches, Université Paul Sabatier, Toulouse, Mai 2001.

[WIL 96] Conception formelle de documents hypermédias portables, Doctorat de l'Université Paul Sabatier, Toulouse, Septembre 1996.

[YAN 93] M. Yannakakis, D. Lee, An efficient algorithm for minimizing real-time transition system, CAV'93, *LNCS* 697, Springer, 1993.

[YOV 97] S. Yovine, KRONOS: a verification tool for real-time systems International Journal on Software Tools for Technology Transfer, Vol. 1, no. 1-2, December 1997.

[ZHE 03] T. Zheng, F. Khendek, Time Consistency of MSC-2000 Specifications, *Computer Networks*, Elsevier, Vol. 42, No. 3, pp. 303-322, June 2003.

[ZIM 80] H. Zimmermann, OSI Reference Model--The ISO Model of Architecture for Open Systems Interconnection, *IEEE Transactions on Communications*, Vol. 28, No. 4, April 1980.

[WAN 04] F. Wan, Formal Verification of Timed Services: A Survey and Perspective, *Proceedings of the IEEE*, Vol. 92, No.8, August 2004.

[WCW 01] Wei J., Cheung S.C., Wang X., "Exploiting Automatic Analysis of E-Commerce Protocols", 25th Annual Computer Software and Application Conference, Chicago, USA, Oct. 2001.