Dédié à André Danthine, pionnier.

Spécification et conception de systèmes communicants : une approche rigoureuse basée sur des scénarios d'usage.

D. Amyot et L. Logrippo
Université d'Ottawa, Groupe de recherche en télécommunications et génie logiciel
150 Louis-Pasteur, Ottawa, Ontario, Canada K1N 6N5
Courrier électronique : damyot@csi.uottawa.ca, luigi@csi.uottawa.ca

R.J.A. Buhr
Carleton University, Department of Systems and Computer Engineering
Ottawa, Ontario, Canada K1S 5B6
Courrier électronique : buhr@sce.carleton.ca

Résumé. *L'obtention d'une spécification à partir d'exigences informelles peut s'avérer une tâche fastidieuse et parsemée d'embûches si une approche méthodique ou rigoureuse n'est pas employée. Un nombre croissant de concepteurs s'intéressent aux approches basées sur les scénarios, ce qui leur permet de se concentrer sur les principaux aspects fonctionnels du système à spécifier. Nous démontrons comment une approche basée sur les scénarios d'usage peut mener à une spécification formelle tout en procurant des avantages indéniables tels que la documentation de la conception, l'évaluation d'architectures alternatives et la génération de tests fonctionnels. Nous présentons cette approche à l'aide d'un exemple : un serveur de communication pour groupes.*

Mots clés : *Scénarios, Use Case Maps, ingénierie des exigences, architecture, spécifications formelles, LOTOS, conception, tests.*

Abstract. *Deriving a specification from informal requirements can be a tedious and dangerous task unless a methodical or rigorous approach is used. An increasing number of designers are interested in scenario-driven approaches that allow them to focus on the main functional aspects of the system to be specified. We show how our approach, based on use cases (scenarios), leads to a formal specification and provides a way to evaluate alternative architectures and to generate functional test cases. We present it using an example : a Group Communication Server.*

Keywords : *Scenarios, Use Case Maps, requirements engineering, architecture, formal specifications, LOTOS, design, tests.*

1. Introduction

Les premières étapes de tout processus de conception sont parmi les plus importantes. C'est à ce moment-là que se prennent des décisions majeures, presque instinctivement, à partir d'exigences informelles. Ces dernières sont souvent incomplètes,

contradictoires, et en constante évolution. Une approche de conception rigoureuse se doit donc d'éclaircir rapidement les zones sombres qui couvrent les exigences. L'ingénierie des exigences est un domaine de recherche qui vise justement à trouver des solutions utiles et pragmatiques à ce genre de problèmes.

Le passage d'exigences informelles à la spécification formelle d'une conception représente bien souvent un ravin dans lequel il est facile de trébucher. De la même façon qu'il n'est pas conseillé de « coder » un programme à partir des exigences, nous ne devrions pas « coder » directement une spécification à partir des exigences. Il faut tout d'abord s'assurer que ces dernières sont bien comprises. Afin de fournir un pont plus aisément franchissable aux concepteurs, des approches basées sur les scénarios, qui visent justement à morceler les exigences en parties plus compréhensibles, sont en pleine émergence.

D'une part, les approches traditionnelles de conception sont souvent basées sur des processus itératifs, où les composantes (logicielles ou fonctionnelles) d'un système sont décrites et raffinées par étapes successives. Dans le monde des spécifications formelles, la méthodologie LOTOSphere constitue un bon exemple [QAP 95][BLV 95]. Nous croyons cependant que ces approches incitent le concepteur à s'engager trop tôt envers une architecture et des détails relatifs aux échanges de messages entre composantes.

D'autre part, les approches orientées scénarios semblent plus intuitives, et permettent de se concentrer sur les aspects importants des exigences en repoussant les décisions liées aux architectures et aux échanges de messages. Elles s'intègrent bien dans un processus de développement itératif et incrémental, facilitent l'ajout ou la modification de fonctionnalités, et ajoutent enfin une plus value à la documentation du système. Par contre, elles n'ont pas encore le niveau de maturité des approches traditionnelles, et certains défis nouveaux sont de taille. Parmi ceux-ci, nous retrouvons la vérification de la complétude et de la cohérence des scénarios, les niveaux d'abstraction qu'ils adressent (qui doivent être semblables d'un scénario à l'autre), et la synthèse de composantes à partir de scénarios.

Cet article propose une approche de conception rigoureuse basée sur les scénarios d'usage. Plusieurs types de systèmes, tels que les systèmes communicants (distribués) et réactifs, s'y prêtent particulièrement bien. Nous présenterons un processus de conception menant rapidement à un prototype LOTOS [ISO 88], spécifiant un ensemble de scénarios définis avec une notation appelée *Use Case Maps* [BC 95]. Nous décrirons les scénarios en termes de séquences causales de responsabilités, à un niveau d'abstraction plus élevé que celui des échanges de messages entre composantes d'un système. Cette nouvelle approche permet entre autres de considérer rapidement différentes architectures sous-jacentes, et de dériver un ensemble de tests fonctionnels permettant de valider la spécification formelle obtenue par rapport aux exigences. Nous utiliserons un exemple afin de concrétiser cette approche : un serveur de communication pour groupes. En insérant des sondes dans la spécification, nous mesurerons la couverture structurelle offertes par notre ensemble de tests. Nous discuterons les propriétés de notre approche, présenterons quelques techniques similaires, et introduirons enfin les travaux futurs considérés.

2. Approche orientée scénarios

Au cours des dernières années, nous avons remarqué un certain engouement pour l'utilisation de scénarios dans les méthodes de conception de systèmes. L'émergence

des scénarios d'usages (*use cases*) [JCJO 93] dans le monde de l'orienté objet confirma cette tendance. De nombreuses méthodologies sont maintenant proposées. Cependant, selon les approches, le terme « scénario » porte en son sein des sémantiques bien différentes les unes des autres. On associe parfois les scénarios à des traces (d'événements internes et/ou externes), à des échanges de messages entre composantes, à des séquences d'interactions entre un système et l'utilisateur, ou encore à des ensembles plus ou moins génériques de telles séquences, pour ne mentionner que ces quelques définitions. Nombreuses sont aussi les notations utilisées, qu'elles soient basées sur une grammaire textuelle plus ou moins formelle, sur des automates, ou sur des diagrammes d'échanges de messages semblables aux MSC [ITU 96]. Les approches diffèrent donc sur plusieurs points en fonction de la définition retenue.

Afin de bien établir le contexte de notre travail, nous présentons dans cette section notre définition d'un scénario, suivi de l'approche rigoureuse de conception que nous préconisons. Quelques approches reliées à la nôtre seront discutées à la section 5.2.

2.1 Use Case Maps : une notation pour les scénarios d'usage

Un scénario est une séquence de responsabilités (événements et activités), internes ou externes, qui sont reliées de façon causale dans le but d'offrir une certaine fonctionnalité. Nous avons sélectionné une notation appelée *Use Case Maps* (**UCM**) [BC 95] pour la description de nos scénarios. Quoique encore informelle sous certains aspects[1], elle a l'avantage de permettre au concepteur de se concentrer sur les fonctionnalités sans avoir à se limiter trop tôt au niveau de l'architecture sous-jacente et des échanges de messages. Les UCM permettent de décrire une séquence de responsabilités de bout en bout du système, et dans ce sens ils ne sont pas centrés vers les composantes.

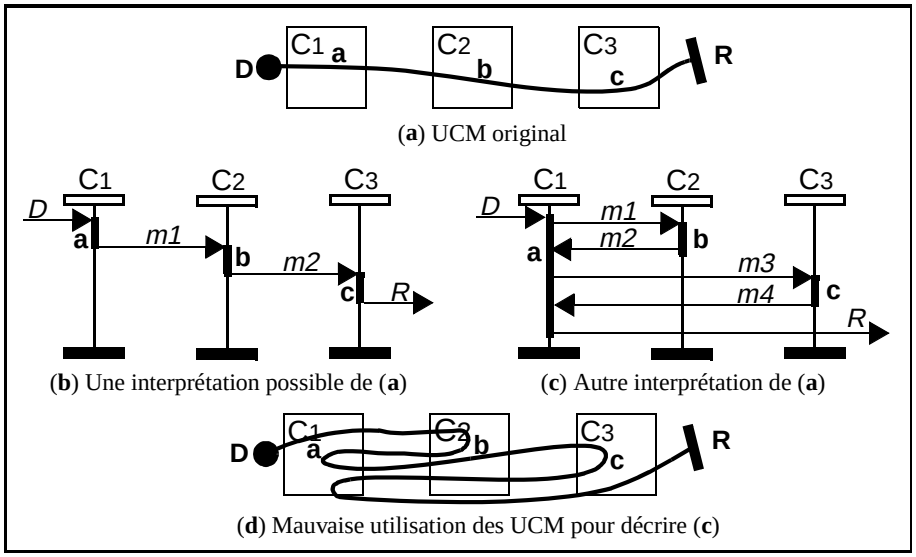
La figure 1(a) présente un UCM très simple ne contenant qu'un seul *chemin*, qui relie une cause à un effet. Un événement déclencheur (ou une pré-condition) D cause le démarrage de l'activité a dans la composante C1. Le tout cause par la suite le départ de b dans C2, suivi de celui de c dans C3, pour enfin se terminer par un événement résultant (ou une post-condition) R. Les événements déclencheurs sont représentés par des cercles pleins, et les événements résultants par des barres perpendiculaires au chemin. Les responsabilités sont reliées de façon causale par une ligne courbe, qui peut être superposée sur une architecture. Sommairement, un événement déclencheur peut mener à un ou plusieurs événements résultants, de façon concurrente ou encore mutuellement exclusive. La notation permet de décrire des choix et des séquences concurrentes[2]. Il est aussi possible de composer plusieurs chemins, qui sont des scénarios initiés par des événements déclencheurs différents. La composition se fait en insérant, sur une ligne courbe, des points d'attente déclenchés (de façon synchrone ou asynchrone) par un autre chemin. Par exemple, l'événement résultant d'un chemin peut devenir l'événement déclencheur d'un autre chemin.

Une relation causale peut être raffinée de plusieurs façons en échanges de messages, tout dépendant des types de composantes, des canaux de communication disponibles et des protocoles utilisés. Par exemple, deux diagrammes d'interaction (cf. figures 1(b) et 1(c)) pourraient être considérés comme des implémentations valides

1. Dans [Amy 94], l'auteur a suggéré un langage de description pour UCM (appelés *Timethread Maps* à l'époque) ainsi qu'une sémantique formelle des UCM basée sur LOTOS. Aucun aspect relié à l'architecture n'était cependant considéré.
2. La notation UCM contient plusieurs constructeurs (*OR-fork*, *OR-join*, *AND-fork*, *AND-join*, etc.) qui permettent d'exprimer de tels scénarios. Voir [BC 95].

de la relation causale décrite en 1(a). Celle de gauche représente l'interprétation la plus simple, alors que celle de droite insiste sur le fait que le déroulement des activités b et de c est sous le contrôle l'activité de a, située dans C1.

Les UCM font donc abstraction de ces détails, qui peuvent être déterminés lors d'une étape ultérieure du processus de développement. Ceci permet aussi de réutiliser plus aisément les scénarios en les superposant sur différentes architectures, comme des schémas de conception (*design patterns*). L'un des pièges à éviter consiste justement à considérer les UCM pour décrire un flot de contrôle ou de données (comme à la figure 1(d)), et ainsi embrouiller inutilement la conception avec une perspective trop près des détails de l'implémentation.



(**a**) UCM original

(**b**) Une interprétation possible de (**a**)

(**c**) Autre interprétation de (**a**)

(**d**) Mauvaise utilisation des UCM pour décrire (**c**)

Figure 1: *Use Case Maps : causalité et échanges de messages*

2.2 Une nouvelle approche

Nous croyons que l'utilisation des UCM dans une approche orientée scénario représente un choix judicieux pour décrire les systèmes réactifs et communicants. Ils s'intègrent bien dans l'approche que nous proposons à la figure 2, qui cherche à bâtir un pont entre les exigences informelles et la première conception formelle du système.

Comme nous prévoyons l'instabilité des exigences, nous utilisons un modèle itératif (en spirale) qui rend possible le prototypage rapide du système, de même que la génération de tests, directement à partir des scénarios. La figure 2 présente un survol de l'approche où nous constatons que le cycle principal consiste à décrire les scénarios et l'architecture (qui peut se faire de façon indépendante), et à les joindre pour enfin synthétiser (manuellement) une spécification LOTOS, notre prototype. Parallèlement, des tests peuvent être générés à partir des scénarios pour être ensuite appliqués à la spécification. Les résultats nous permettent de déterminer si d'autres tests sont nécessaires pour obtenir la couverture désirée, et nous pouvons alors constater s'il y a correspondance entre les exigences et le prototype généré.

Pour la représentation formelle de nos systèmes, nous avons opté pour l'utilisation de la norme internationale LOTOS [ISO 88]. Ce langage permet l'intégration de comportements et d'architectures dans un même modèle exécutable. Nous pouvons y spécifier des systèmes réactifs et communicants à plusieurs niveaux d'abstraction. De plus, LOTOS se prête bien au prototypage rapide, il est fondé sur une sémantique formelle sur laquelle de solides théories de vérification sont basées, et de nombreux outils de simulation, de tests et de vérification sont disponibles [BLV 95].

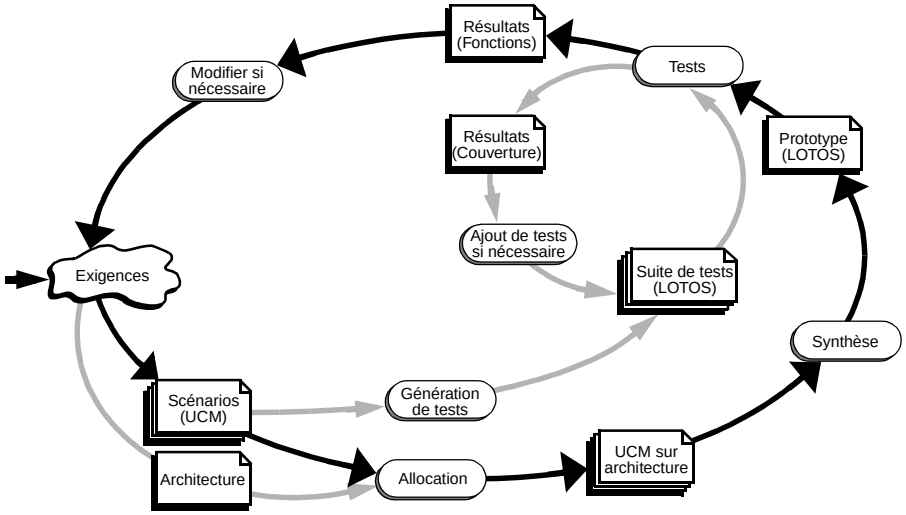


Figure 2: *Approche suggérée*

Nous avons constaté plusieurs avantages à cette approche rigoureuse :

- **Séparation de l'architecture et des fonctionnalités** : comme les scénarios sont formalisés à un niveau d'abstraction plus élevé que celui des messages, différentes architectures sous-jacentes peuvent être considérées avec plus de souplesse pour la génération de prototypes. Les scénarios deviennent alors des entités hautement réutilisables.
- **Possibilité de générer rapidement des prototypes** : une fois l'architecture et les scénarios compris et documentés, et une fois les responsabilités alloués aux composantes, un prototype peut alors être rapidement généré.
- **Génération de tests** : les scénarios se prêtent bien à la génération de tests, qui sont utiles car ils se rapportent presque directement aux exigences.
- **Documentation de la conception** : la documentation de la conception se fait au fur et à mesure que nous avançons dans le cycle. Bien souvent, les concepteurs ne documentent une conception que s'ils en sont obligés. Nous croyons que cette approche incite à la documenter méthodiquement.

Cette approche a déjà été utilisée dans [Amy 97], et nous l'illustrerons à l'aide d'un exemple dans la section qui suit.

3. Un exemple : le serveur de communication pour groupes

Nous décrivons tout d'abord informellement ce système, puis nous discuterons quelques alternatives architecturales. Viendront ensuite un exemple de scénario et quelques explications sur la spécification LOTOS qui résulte de notre approche.

3.1 Description informelle

Fonctionnalités

Sommairement, un serveur de communication pour groupes (*Group Communication Server* ou ***GCS***) permet la diffusion (*multicast*) d'un message aux membres d'un groupe. Les services offerts représentent un noyau utilisable pour l'implémentation d'un serveur de liste de courrier électronique, de *Internet Relay Chat*, ou encore de vidéoconférence. Les utilisateurs peuvent assumer deux rôles : requérant (*sender*) et récepteur (*receiver*). Les groupes peuvent être créés et détruits au besoin.

Les utilisateurs peuvent joindre et quitter un ou plusieurs groupes. Les messages peuvent être de différents types (voix, texte, vidéo, données, etc.) et seront diffusés via des canaux de communication adaptés aux exigences du groupe. Un groupe peut avoir un *administrateur* responsable de la gestion des membres et de la destruction du groupe. L'un des membres peut aussi être nommé *modérateur.* Il sera alors responsable d'approuver ou de rejeter les messages retransmis aux membres du groupe. Il est possible de modifier l'administrateur et le modérateur d'un groupe. Les utilisateurs peuvent obtenir en tout temps la liste des groupes gérés par le serveur, de même que la liste des membres d'un groupe lorsqu'ils ont les privilèges d'accès nécessaires.

Chaque groupe aura quatre attributs : administré ou non administré, modéré ou non modéré, ouvert (n'importe qui peut diffuser) ou fermé (seuls les membres peuvent diffuser), public (tous peuvent joindre le groupe) ou privé (seul l'administrateur peut ajouter/retirer un membre du groupe).

Architecture

D'un point de vue architectural, deux types d'entités sont prédéfinis :

- **MGCS** : le gestionnaire de GCS. Entité unique responsable de créer de nouveaux groupes et de fournir la liste des groupes disponibles.
- **GCS** : une instance de groupe, avec ses attributs et sa liste membres. Le GCS est responsable de sept fonctionnalités : ajouter un membre, retirer un membre, détruire le groupe, fournir la liste des membres du groupe, modifier l'administrateur, modifier le modérateur et diffuser un message.

Les descriptions plus détaillées de ces fonctionnalités et des paramètres requis ne sont évidemment pas toutes présentées ici, mais nous incluons celle qui servira d'exemple : la modification d'un administrateur.

Modification de l'administrateur (CHANGEADMIN)

Cette requête doit être accompagnée d'un paramètre, soit l'identifiant du nouvel administrateur (*NewAdmin*, du type *MID*). Si le groupe n'est pas administré, alors le requérant reçoit un message d'erreur : NOADMINGROUP. Les alternatives suivantes présupposent que le groupe est effectivement administré. Si le requérant est bien l'administrateur et que le nouvel administrateur proposé est valide (il doit être membre du groupe), alors la modification est apportée et le GCS retourne ADMINCHANGED. Si le requérant n'est pas administrateur, alors il reçoit le message d'erreur NOTADMIN. Si le nouvel administrateur n'y est pas membre, alors le requérant s'en voit averti par MEMBERNOTINGROUP. Un groupe administré peut être rendu non administré en ne spécifiant aucun nouvel administrateur (*Nobody*). L'inverse n'est cependant pas permis.

3.2 Architecture

Notation utilisée

Les architectures que nous considérons ici sont relativement abstraites en ce sens qu'elles ne contiennent principalement qu'une topologie de composantes interreliées. Nous pouvons évidemment détailler leur description en ajoutant sémantique ou attributs aux composantes (type, cardinalité, dynamique/statique, etc.). Dans cet exemple, nous utilisons la notation de [BC 95]. Trois types de composantes y sont inclus :

- Les **processus** (*processes*), représentés par des parallélogrammes. Ce sont des objets actifs, fonctionnant séquentiellement à l'interne, comparables à des processus ou des tâches dans un système d'exploitation. Ils contrôlent les objets passifs.
- Les **objets** (*objects*), représentés par des rectangles arrondis. Ce sont des objets passifs, comparables à des fonctions ou encore à des bases de données.
- Les **équipes** (*teams*), représentées par des rectangles. Ils regroupent processus, objets et équipes, et fournissent une certaine encapsulation (*information hiding*).

Les composantes en pointillés indiquent qu'elles peuvent être créées ou détruites dynamiquement, alors que les piles de composantes indiquent que plusieurs instances peuvent cohabiter de façon concurrente. Nous ajoutons des liens, entre les composantes, qui peuvent être interprétés comme des canaux de communication permettant l'échange de messages. Les liens pointillés représentent des canaux créés au besoin. Nous traitons ces architectures à un haut niveau d'abstraction, au-delà des protocoles de communications ou des types de canaux reliant les composantes.

Alternatives architecturales

Il est possible de considérer différentes alternatives architecturales pour notre système, avant, pendant ou après la génération des scénarios. Des contraintes dues à la présence de composantes obligatoires ou déjà existantes, aux ressources limitées, de même que des exigences de performance et de robustesse peuvent guider notre choix. Considérons seulement deux alternatives simples présentées à la figure 3.

L'option **(a**) fusionne le MGCS et le GCS en un seul processus monolithique, qui reçoit les requêtes, les traite, et retourne les réponses en conséquences. Il n'y a ni parallélisme, ni distribution dans ce modèle (qui est simplifié à l'extrême).

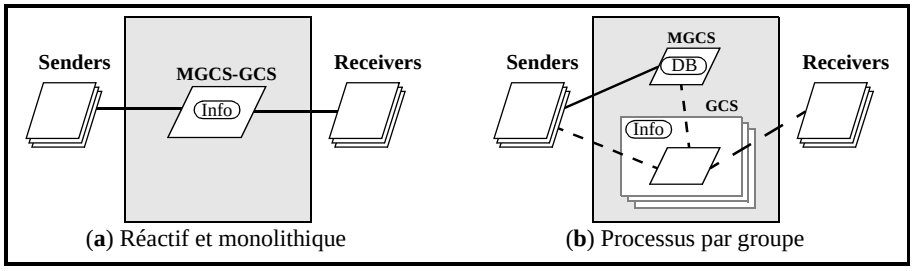


Figure 3: *Deux alternatives architecturales pour le GCS*

La seconde option (**b**) découple le MGCS des instances de GCS. Ces dernières s'exécutent de façon concurrente, et ont leurs propres canaux de communication les reliant aux requérants et aux récepteurs. Le MGCS communique avec les GCS en utilisant un canal interne. Deux objets passifs permettent de séparer les bases de don-

nées des attributs de chaque groupe (*info*) de celle qui maintient la liste des groupes (*DB*). Chaque groupe peut donc traiter ses requêtes indépendamment des autres groupes et du MGCS.

Il est possible d'étudier le problème architectural sous différentes facettes (performance, concurrence, distribution, complexité, etc.), tout dépendant des exigences initiales. D'autres notations peuvent tout aussi bien être utilisées, que ce soit ROOM [SGW 94], des langages de description d'architectures (ADL) ou simplement un ensemble de boîtes, interreliées ou non. Notre intention ici n'est pas de justifier notre choix architectural pour cet exemple, mais plutôt de démontrer comment différentes notations et alternatives peuvent être considérées à tout moment. Les scénarios que nous développons peuvent être superposés sur l'une ou l'autre de ces architectures, et il est donc toujours temps de revenir sur nos pas pour en considérer une autre.

Pour la suite de l'exemple, nous avons retenu l'option (**b**) pour décrire l'architecture présentée en arrière plan de la figure 4. Nous avons nommé les objets, les processus, les équipes et les liens. Un tampon bidirectionnel (*BiDirBuffer*) permet d'accumuler les requêtes et les réponses. Lors de la diffusion d'un message, un processus *Multicast* est dynamiquement associé à chaque membre du groupe. Chaque processus est responsable de la transmission du message au membre qui lui est assigné, et il est détruit lorsqu'il se termine.

3.3 Scénarios décrits en UCM

Chaque UCM regroupe les scénarios qui sont reliés logiquement. Souvent, ceux-ci sont des alternatives ou de multiples réponses **déclenchées par le même événement initial** ou la même pré-condition. Pour notre exemple, toutes les exigences reliées à la modification de l'administrateur ont été regroupées à la section 3.1, et nous devrons donc les considérer ici. De fait, chacune des neuf fonctionnalités assumées par ce système ferait l'objet d'un UCM indépendant. Il est donc possible de voir, d'un simple coup d'œil, quels seront les dénouements possibles causés par un même événement initial (paramétrisé). Les scénarios sont par la suite superposés sur l'architecture sélectionnée, et les activités sont associées aux composantes.

Description de l'UCM pour CHANGEADMIN

La figure 4 illustre l'UCM pour notre exemple. Les activités et conditions (**a**) à (**f**) sont associées à la composante GCS (*allocation* des responsabilités dans la figure 2). Sur réception de CHANGEADMIN, le GCS vérifie ses attributs locaux (**a**). Si le requérant est l'administrateur et que l'administrateur proposé (*NewAdmin*) est bien membre du groupe (**b**), alors l'attribut *Admin* est modifié en conséquence et le tout résulte en ADMINCHANGED.

Une telle requête, envoyée cette fois à un groupe non administré (**d**), aura pour effet de générer NOADMINGROUP comme message d'erreur. Si ce n'est pas le cas mais que le requérant n'est pas l'administrateur (**e**), alors NOTADMIN est renvoyé comme erreur. Le nouvel administrateur se doit aussi de faire partie du groupe, sinon (**f**) MEMBERNOTINGROUP est retourné.

Un tableau accompagne habituellement une telle figure afin de décrire plus substantiellement les responsabilités (activités, événements, conditions, etc.) de l'UCM avec leur type, leurs paramètres et des commentaires supplémentaires. Comme il ne

faut pas surcharger inutilement les UCM avec trop de détails, ce type de table devient rapidement nécessaire à leur documentation.

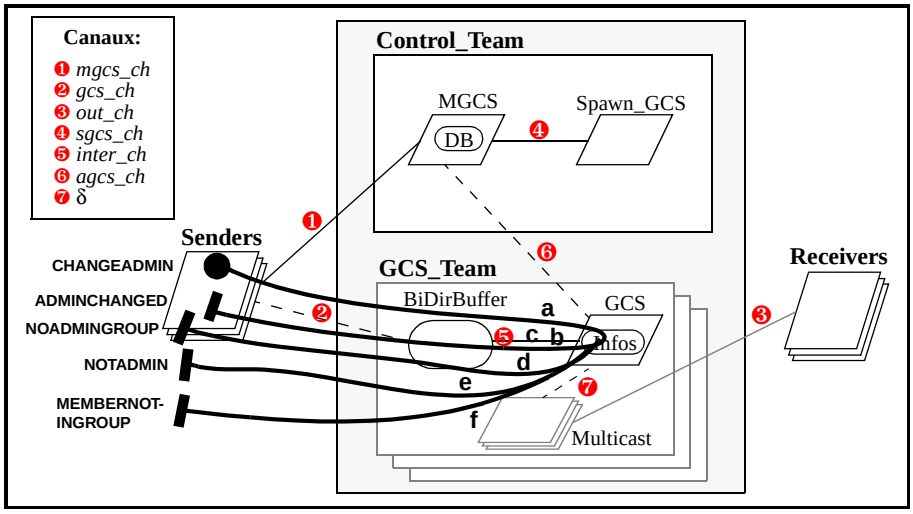


Figure 4: *Modification de l'administrateur (Change Administrator UCM)*

Pour la plupart des scénarios critiques ou complexes, il semble adéquat de documenter plus à fond les différents comportements en relation avec les conditions à vérifier. Pour ce faire, nous suggérons l'utilisation d'un type spécial de table de décision proposé par Parnas [PMI 94] où sont spécifiés les variables modifiées et les messages émis en fonction des conditions rencontrées. Ces tables permettent de s'assurer que toutes les alternatives sont couvertes (complétude) et qu'elles sont mutuellement exclusives (déterminisme). Notre exemple s'y prête particulièrement bien, étant donné le nombre élevé de conditions à considérer. Nous croyons que l'utilité de ce type de table est indéniable pour la documentation d'une fonctionnalité ou d'un processus, pour la construction d'un processus correspondant dans la spécification formelle, et pour la dérivation de cas de tests.

3.4 Spécification formelle en LOTOS

Une fois l'architecture et l'ensemble des UCM disponibles, nous procédons à une distribution du comportement décrit dans les scénarios sur la topologie de composantes. La spécification LOTOS résultante doit permettre le **retraçage** direct entre sa structure et l'architecture choisie, et entre les messages/actions et les activités provenant des scénarios.

Nous nous concentrons sur la spécification du serveur de groupes uniquement, sans égard aux clients (requérants et récepteurs). Ces derniers peuvent être de multiples natures (IRC, vidéoconférence, courrier électronique, etc.) étant donné l'utilisation générique du serveur.

Comme pour une spécification orientée ressource, chaque composante sera spécifiée par un processus LOTOS, et les canaux de communication deviendront les portes sur lesquelles se synchroniseront les processus. Chaque événement et chaque paramètre sera décrit à l'intérieur de types de données abstraits. Tous les noms sont donc

préservés. Des structures d'événements (porte et types de messages) fixes sont définies, ce qui permet ainsi un retraçage simple entre les traces d'exécution et les scénarios ou les exigences. Dans notre exemple, les protocoles d'échange de messages utilisés pour réaliser les relations de causalité sont relativement simples. Les clients communiquent à la manière d'un appel distant RPC avec le serveur. Par contre, le protocole de diffusion (*multicast*) pourrait être aussi complexe que le désire le concepteur (temporisé ou non, avec ou sans retransmission, etc.), pour la même relation causale.

Structure de la spécification

Tel que décrit dans l'architecture (cf. figure 4), nous retrouvons, au niveau supérieur : `Control_Team := MGCS |[sgcs_ch, agcs_ch]| Spawn_GCS`. Une structure similaire s'applique aussi au processus *GCS_Team*. La création dynamique de groupes se fait en utilisant *Spawn_GCS*, qui s'appelle récursivement tout en créant en parallèle une instance de *GCS_Team*. La même structure s'applique aussi à la création dynamique d'instances du processus *Multicast*, lors de la diffusion d'un message aux membres du groupe.

```
process Spawn_GCS[sgcs_ch, agcs_ch, gcs_ch, out_ch] :noexit:=
  sgcs_ch !CREATEGROUP ?id:GID ?mbrL:MemberList ?infos: msg;
    ( GCS_Team[agcs_ch, gcs_ch, out_ch] (id, mbrL, infos)
      |||
      Spawn_GCS[sgcs_ch, agcs_ch, gcs_ch, out_ch] )
endproc (* Spawn_GCS *)
```

Comportement d'une fonctionnalité

Le processus *GCS*, étant donné sa complexité, sera décomposé en un choix entre plusieurs sous-processus (un pour chacune des sept fonctionnalités dont le GCS est responsable), de façon à rendre plus modulaire la spécification résultante. *Req_ChangeAdmin* est justement l'un de ces processus. L'utilité des commentaires `(*_PROBES_*)` est expliquée à la section 4.3

```
process Req_ChangeAdmin[inter_ch, out_ch](sender: MID, msg: Msg, id:GID,
                                          mbrL:MemberList, infos:msg) :exit:=
    (* Le requérant a les privilège requis. *)
    [IsAdmin(infos) and (Admin(infos) eq sender)] ->
        (
            [(Admin(msg) IsIn mbrL) or (Admin(msg) eq Nobody)] ->
                inter_ch !sender !ADMINCHANGED; (*_PROBE_*)
                    GCS[inter_ch, out_ch](id, mbrL, SetAdmin(Admin(msg), infos))
            []
            [(Admin(msg) NotIn mbrL) and (Admin(msg) ne Nobody)] ->
                inter_ch !sender !MEMBERNOTINGROUP; (*_PROBE_*)
                    GCS[inter_ch, out_ch](id, mbrL, infos)
        )
    []
    (* Le requérant n'a pas les privilège requis pour modifier l'admin. *)
    [IsAdmin(infos) and (Admin(infos) ne sender)] ->
        inter_ch !sender !NOTADMIN; (*_PROBE_*)
            GCS[inter_ch, out_ch](id, mbrL, infos)
    []
    (* Le groupe n'est pas administré. *)
    [not(IsAdmin(infos))] ->
        inter_ch !sender !NOADMINGROUP; (*_PROBE_*)
            GCS[inter_ch, out_ch](id, mbrL, infos)
endproc (* Req_ChangeAdmin *)
```

À titre d'indication, la spécification que nous avons obtenue à partir des 9 UCM décrits dans [Amy 97] contient 640 lignes commentées de types de données (ADT), et 570 lignes commentées pour le comportement (les processus).

4. Tests et couverture de la spécification

Nous pouvons générer des tests fonctionnels directement à partir des scénarios, tirés préalablement des exigences. Comme les tests dérivés seront très près des exigences, ils permettront de valider partiellement le prototype que nous avons informellement synthétisé. Cet ensemble de tests représente aussi un investissement intéressant car il peut être réutilisé lors des modifications apportées à la spécification (tests de régression), ou même lorsqu'il y a des modifications à l'architecture.

Étant donné la complexité des types de données utilisés et le comportement dynamique des processus, provoquant ainsi une énorme explosion du nombre d'états possibles, les techniques de vérification basées sur le modèle sous-jacent (tel que le *model checking*) n'ont pu être utilisées. Nous nous sommes donc concentrés sur les tests pour valider notre spécification.

4.1 Génération des tests

En LOTOS, nous testons une spécification en définissant un ensemble de processus de test qui seront composés tour à tour au comportement principal. Il s'agit d'une approche « boîte noire », où nous n'avons accès qu'à l'interface offerte par le système. Pour chacun des UCM, nous avons créé deux ensembles de tests, composés de traces cherchant pour l'un à valider les comportements corrects (*acceptance tests*) et pour l'autre à invalider les comportements incorrects (*rejection tests*). Chaque trace est construite en parcourant un chemin possible de l'UCM et en fournissant les paramètres nécessaires à la satisfaction des conditions qui se retrouvent sur ce parcours.

Voici par exemple l'une des cinq séquences de test utilisées pour la validation du comportement correct de la modification de l'administrateur. Dans ce cas-ci, le groupe est administré, le requérant en est l'administrateur, mais le nouvel administrateur proposé n'en est pas membre. Un message d'erreur (`MEMBERNOTINGROUP`) averti alors le requérant de ce problème :

```
process TestUCMadminA_1 [mgcs_ch, gcs_ch, success]:noexit :=
    mgcs_ch !4 of MID !CREATEGROUP !d of GID (* Requête *)
        !Encode(Mail, 4, Administered, 4, Opened, Public, NonModerated, Nobody);
    mgcs_ch !4 of MID !GROUPCREATED !d of GID; (* Réponse du MGCS *)
    gcs_ch !4 of MID !d of GID !CHANGEADMIN !Encode(1 of MID);
    gcs_ch !4 of MID !MEMBERNOTINGROUP !d of GID; (* Réponse du GCS *)
    success; (* Événement qui sera recherché par LOLA. Indique un succès. *)
    stop
endproc (* TestUCMadminA *)
```

Les tests ainsi générés proviennent des scénarios, et donc indirectement des exigences. Nous cherchons à valider la conception par rapport aux exigences informelles, et non à faire des tests de conformance, où une spécification (ou une implémentation) est validée par rapport à une autre. D'autres théories plus formelles permettent de générer des tests vérifiant les équivalences entre états et transitions des automates correspondants [CKM 93]. Néanmoins, nous croyons que nous pouvons réutiliser nos tests dans les étapes ultérieures du processus de développement, jusqu'aux tests de l'implémentation.

4.2 Résultats

Pour l'exécution des tests dérivés, nous avons utilisé l'outil LOLA [PL 90], développé à l'Université Polytechnique de Madrid. Il permet d'exécuter, de façon simple et efficace, une suite de tests inclus comme processus LOTOS dans la spécification.

Le tableau 1 présente un sous-ensemble des résultats obtenus. Les ensembles de tests #0 à #17 furent directement générés à partir des scénarios, à raison de deux ensembles par UCM. LOLA fait l'expansion du LTS résultant de la composition du test et de la spécification, et vérifie si l'événement de succès (*success*) est toujours atteint (dans le cas d'un *Must test*) ou toujours inatteignable (dans le cas d'un *Reject test*).

Test #	Processus	Must / Reject	# de tests	# Exec.	# Exec. sondes
		...			
16	*TestUCMadminA*	Must	5	5	18
17	*TestUCMadminR*	Reject	6	6	13
18	*TestClientA*	Must	1	1	1
19	*TestCoverRecursionA*	Must	3	5	13
20	*TestCoverCondA*	Must	2	2	7
		TOTAL :	**89**	**108**	**256**

Tableau 1: *Résultats des tests appliqués à la spécification*

Un total de 89 tests résultèrent en 108 exécutions différentes qui se terminèrent toutes tel que prévu, après quelques corrections apportées à la spécification. Ceci nous assure un certain niveau de confiance quant à la validité de la spécification par rapport aux scénarios et aux exigences informelles. Le nombre d'exécutions est plutôt restreint car LOLA emploie des techniques de réduction en cour d'exécution, tout en conservant l'équivalence observationnelle, ce qui est suffisant dans notre cas.

4.3 Mesure de la couverture

Les ensembles de tests #0 à #17 assurent donc une bonne couverture fonctionnelle de notre spécification. Cependant, qu'en est-il de sa structure sous-jacente, c'est-à-dire le système de transitions étiquetées (LTS)? Afin de mesurer la couverture structurelle, nous avons développé un outil simple qui permet d'insérer des sondes dans une spécification, afin de vérifier si certains points stratégiques sont atteints par les tests. Ces points se retrouvent en général devant chaque **`stop`**, **`exit`**, et instantiation de processus. Ainsi, des commentaires particuliers `(*_PROBE_*)` sont ajoutés à la spécification, et ils seront traduits automatiquement en portes internes avec un identifiant unique (par exemple : `Probe!P_34;`). Notre processus exemple, *Req_ChangeAdmin*, en contient quatre.

En parcourant l'ensemble des traces générées par LOLA lors de l'exécution des tests, il est possible de savoir si une sonde est atteinte ou non par un ou plusieurs tests. Nous avons inséré un total 40 sondes dans notre spécification, et nous avons compilé le nombre de fois où chacune d'entre elles a été atteinte par nos tests. Le tableau 2 résume un sous-ensemble des résultats obtenus. Six sondes n'ont pas été couvertes par les tests #0 à #17. Trois tests supplémentaires (#18 à #20) ont été ajoutés pour rejoindre ces sondes. Quelques-unes n'avaient pas été atteintes parce que certains processus LOTOS subdivisaient, via les conditions et types de données utilisés, quelques chemins décrits dans les UCM.

La dernière colonne du tableau 1 indique 256 exécutions différentes pour ces 89 tests, ce qui assura la couverture structurelle de la spécification, comme nous le montre le tableau 2. Nous devons mentionner que cette approche basée sur les tests fut très pragmatique et efficace dans notre cas. La qualité des tests générés nous a aussi agréablement surpris. Comme les 21 ensembles de tests (1220 lignes commentées) s'exécutent en moins de 5 secondes sur un Pentium 150MHz., nous avons pu faire

aisément des tests de régression tout en modifiant et corrigeant la spécification à maintes reprises.

Sonde #	Ligne #	Tests #0 à #17	Tests #0 à #20	Sonde #	Ligne #	Tests #0 à #17	Tests #0 à #20
0	744	**0**	1	**20**	1062	**0**	2
1	751	101	106	**21**	1066	5	5
		...				...	
19	1049	5	5	**39**	1255	6	6

Tableau 2: *Couverture de la spécification décrite par les 40 sondes*

5. Discussion et travaux futurs

5.1 Retour sur l'approche proposée

En rétrospective, cette approche nous a permis de comprendre, de documenter et de spécifier d'une façon rigoureuse un système communicant et réactif relativement complexe. Nous croyons qu'elle se distingue des approches traditionnelles sous plusieurs aspects :

- **Séparation de l'architecture et des fonctionnalités lors de la conception** : les scénarios peuvent être développés avec ou sans l'architecture sous-jacente. Aussi, nous pouvons évaluer des architectures sans nous référer aux scénarios qui vont les traverser.
- **Combinaison du comportement et de l'architecture** : le comportement (scénarios) et l'architecture d'un système peuvent aussi être regroupés sous une seule vue bidimensionnelle, à un haut niveau d'abstraction. Cette superposition aide le concepteur dans son raisonnement et son allocation des responsabilités.
- **Séparation causalité / messages** : les scénarios sont décrits à un niveau d'abstraction plus élevé que celui des messages, ce qui leur donne un niveau d'indépendance supérieur face à l'architecture.
- **Réutilisation de scénarios** : les scénarios sont considérés comme des entités de première classe. En ce sens, ils ne sont pas centrés sur les composantes, et donc plus facilement réutilisables. Par exemple, le scénario de modification de l'administrateur (cf. figure 4) pourrait aisément être superposé sur l'architecture de la figure 3(a), menant ainsi à un découpage fonctionnel différent et donc à une toute autre spécification.
- **Génération de tests et validation** : les scénarios servent à générer des tests de façon rigoureuse afin de valider le prototype par rapport aux scénarios, et donc par rapport aux exigences, tout en obtenant une bonne couverture fonctionnelle.
- **Retraçage (*traceability*)** : les noms d'activités et d'événements, de même que l'architecture choisie, sont cohérents à partir des exigences jusqu'à la spécification et les tests, en passant par les UCM. Ceci facilite grandement le retraçage d'un niveau à l'autre, surtout lors des diagnostiques.
- **Documentation de la conception** : elle se fait au fur et à mesure que nous avançons dans le cycle. Bien souvent, les concepteurs ne documentent leur conception que s'ils en sont obligés, et nous voyons cette approche comme étant incitative.

5.2 Autres approches reliées

Nous mentionnons quelques commentaires sur six approches qui nous semblent parmi celles les plus liées à la nôtre. Toutes débutent leur processus à partir d'un ensemble d'exigences informelles.

- **[ABBD 96]** : prédécesseur de notre approche, la méthodologie présentée dans cet

article formalise, en LOTOS, un ensemble de scénarios qui doivent préalablement être composés en un seul UCM global. Les scénarios sont les seules entités considérées, sans aucune référence à l'architecture. Un langage de description (TMDL) existe cependant pour décrire les scénarios, et la description peut alors être compilée automatiquement en LOTOS (sans types de données).

- **[BB 97]** : cette méthodologie utilise aussi les UCM comme notation pour les scénarios, mais la transformation se fait manuellement vers une architecture ROOM [SGW 94]. Chaque UCM est converti en un MSC [ITU 96] de haut niveau et un ensemble de MSC de base. Ces derniers sont par la suite reliés à une structure ROOM, où le comportement des composantes est décrit à l'aide de machines à états finis. Il n'y a aucun modèle de validation formelle pour l'instant.
- **[Tuok 96]** : les scénarios sont aussi représentés sous forme de séquences causales d'événement internes et externes. L'approche présente un processus de conception itératif (en spirale) où les scénarios sont formellement décrits, et par la suite intégrés de façon informelle afin de générer un prototype LOTOS exécutable. Le processus nécessite toutefois la présence d'une architecture stable.
- **[SDV 96]** : les scénarios, orientés vers les composantes, décrivent des systèmes temps-réels avec des événement observables, en considérant les contraintes de temps. Cette méthode permet la synthèse et la génération automatique de spécifications formelles, sous forme d'automates temporisés, qui peuvent par la suite servir de modèle pour la vérification. Ils offrent aussi des techniques de détection d'incohérences et d'aide à la complétude.
- **[RKW 95]** : un scénario représente un usage du système. Cette approche, appelée *Usage Oriented Requirements Engineering* (UORE), complète l'analyse orientée scénario (UCDA) avec une étape de synthèse. Cependant, celle-ci considère qu'un seul acteur à la fois peut être impliqué dans chaque scénario, ce qui simplifie le problème. Les scénarios sont décrits à l'aide d'une syntaxe formelle, et le processus de développement est encore une fois centré vers les composantes.
- **[MC 96]** : Cette approche, appelée *Rigorous Object-Oriented Analysis* (ROOA), permet de passer informellement d'un modèle OMT à un prototype LOTOS. Un scénario inclut une série d'événements externes avec le résultat auquel il faut s'attendre. L'approche prône le prototypage et la validation formelle. Elle aussi met de nouveau l'accent sur les composantes (objets) et les échanges de message.

5.3 Travaux futurs

Les approches présentées à la section précédente mettent en évidence quelques étapes manquantes à notre cycle de conception. Nous considérons un nouveau cycle où certaines lacunes seraient comblées, spécialement au niveau de la cohérence et de la complétude des scénarios, et au niveau des protocoles (ou patrons de messages) à employer pour concrétiser les relations causales entre responsabilités lors de la synthèse de la spécification. Voici donc quelques points que nous aimerions voir inclus dans notre approche :

- **Extensibilité dans un processus de conception itératif** : nous cherchons à réduire les répercussions sur la spécification dues à l'ajout ou la modification de scénario. Nous visons de ce fait la génération rapide de prototypes.
- **Utilisation cachée de LOTOS** : nous aimerions cacher la présence de LOTOS au concepteur pour lui permettre de se concentrer sur les quatre documents où ses décisions sont requises, c'est-à-dire la génération du dictionnaire de données, des scénarios, de l'architecture et des protocoles à utiliser.
- **Langages de description** : l'automatisation du processus passe par des langages de description pour les étapes initiales. Déjà, TMDL [ABBL 96] fut utilisé pour décrire les UCM, mais il n'est plus adapté aux besoins actuels, surtout au niveau

des données et de l'architecture.

- **Vérification de la cohérence et de la complétude des scénarios** : par le biais d'un dictionnaire de données, nous devrions pouvoir détecter des scénarios manquants et les contradictions entre scénarios. Nous étudierons les travaux de [SDV 96] à cet effet.
- **Synthèse automatisée** : possiblement l'étape la plus difficile, qui cherche à produire des automates, distribuant ainsi le comportement des scénarios sur une topologie de composantes. Les protocoles et patrons de messages doivent être considérés lors de la synthèse. Une simplification des travaux de [SDV 96] nous semble pertinente pour cette étape.
- **Génération de tests automatisée** : en assumant une description formelle des scénarios, il semble possible de dériver un ensemble optimisé de tests visant à parcourir tous les chemins d'un UCM, indépendamment de l'architecture, afin de fournir une couverture fonctionnelle.
- **Réutilisation des scénarios** : les scénarios décrits en UCM peuvent être réutilisés sur des architectures alternatives d'un même système. Nous croyons qu'ils peuvent aussi l'être pour d'autres applications, à la manière des patrons de conception (ou *design patterns*). Nous cherchons d'ailleurs à réutiliser les UCM produits pour le serveur de communication pour groupes à *GPRS*, le service d'échange de données par paquets basé sur GSM (*Global System for Mobile Communications*).

Notre but ultime sera donc d'automatiser au maximum ce processus afin de produire des scénarios et des prototypes le plus rapidement possible, avec un minimum d'interventions du concepteur, tout en cachant la complexité inhérente à LOTOS.

6. Conclusion

La conception de systèmes communicants et réactifs nécessite des méthodes systématiques pour passer des exigences à la première spécification. Nous avons présenté une approche rigoureuse qui permet de passer d'un ensemble de scénarios, tirés des exigences informelles et décrits avec des Use Case Maps, à une spécification LOTOS où le comportement global est distribué sur une architecture de composantes interreliées.

Les scénarios représentent des séquences causales d'activités, décrites au-delà des échanges de messages entre composantes. Des tests peuvent en être tirés pour être ensuite appliqués à la spécification, dans le but de la valider par rapport aux exigences. Grâce à l'insertion de sondes, il est aussi possible de déterminer la couverture structurelle offerte par la suite de tests. La cohérence entre les noms utilisés au niveau des exigences, des scénarios, de la spécification et des tests, permet un retraçage direct entre les différents niveaux de conception. Les scénarios ainsi décrits peuvent être réutilisés pour sélectionner une architecture parmi différentes alternatives. Ils permettent aussi d'inciter à documenter la conception de façon systématique.

Nous avons illustré une partie de cette approche à l'aide d'un exemple, un serveur de communication pour groupe (GCS), et en particulier l'une de ses neufs fonctionnalités principales : la modification de l'administrateur. Les résultats des tests appliqués à la spécification furent aussi mentionnés. Nous avons survolé des approches similaires à la nôtre, et exprimé enfin quelques directions de recherche prometteuses.

Nous tenons à remercier le Conseil de recherche en sciences naturelles et en génie (CRSNG—Canada), le Fonds pour la formation de chercheurs et l'aide à la recherche (FCAR—Québec) et le Telecommunication Research Institute of Ontario (TRIO—

Ontario) pour leur appui financier. Merci à M. Woodside pour avoir fourni une description informelle de l'exemple. Nous remercions enfin H. Ben Fredj, P. Forhan, B. Ghribi et B. Stepien pour leurs pertinents commentaires, et plus particulièrement J. Sincennes avec qui la spécification LOTOS a été originalement conçue.

7. Références

[Amy 94] D. Amyot, *Formalization of Timethreads Using LOTOS*. Thèse de maîtrise, Département d'informatique, Université d'Ottawa, Ottawa, Canada, 1994.

[Amy 97] D. Amyot, *Group Communication Server: A Scenario-Based Design Exercise*. Rapport technique, Département d'informatique, Université d'Ottawa, Ottawa, Canada, 1997 (*à paraître*).

[ABBD 96] D. Amyot, F. Bordeleau, R.J.A. Buhr et L. Logrippo, "Formal support for design techniques: a Timethreads-LOTOS approach". Dans: G. von Bochman, R. Dssouli et O. Rafiq (éd.), *FORTE VIII, 8th International Conference on Formal Description Techniques*, Chapman & Hall, 57-72, 1996.

[BLV 95] T. Bolognesi, J. van de Lagemaat et C. Vissers, *LOTOSphere: Software Development with LOTOS*. Kluwer Academic Publishers, Hollande, 1995.

[BB 97] F. Bordeleau et R.J.A. Buhr, "The UCM-ROOM Design Method: from Use Case Maps to Communicating State Machines". Dans: *Conference on the Engineering of Computer-Based Systems*, Monterey, mars 1997.

[BC 95] R.J.A. Buhr et R.S. Casselman, *Use Case Maps for Object-Oriented Systems*, Prentice-Hall, USA, 1995.

[CKM 93] A. Cavalli, S. Kim et P. Maigron, "Improving Conformance Testing for LOTOS". Dans: R.L. Tenney, P.D. Amer et M.Ü. Uyar (éd.), *FORTE VI, 6th International Conference on Formal Description Techniques*, North-Holland, 367-381, octobre 1993.

[ISO 88] ISO, Information Processing Systems, Open Systems Interconnection, *LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, IS 8807*, 1988.

[ITU 96] ITU, *Recommendation Z. 120: Message Sequence Chart (MSC)*. Geneva, 1996.

[JCJO 93] I. Jacobson, M. Christerson, P. Jonsson et G. Övergaard, *Object-Oriented Software Engineering, A Use Case Driven Approach*. Addison-Wesley, ACM Press, 1993.

[MC 96] A.M.D. Moreira et R.G. Clark, "Adding rigour to object-oriented analysis". Dans: *Software Engineering Journal*, IEE, 270-280, septembre 1996.

[PMI 94] D.L. Parnas, J. Madey et M. Iglewski, "Precise Documentation of Well-Structured Programs". Dans: *IEEE Transactions on Software Engineering*, volume 20, numéro 12, 948-976, décembre 1994.

[QAP 95] J. Quemada, A. Azcorra et S. Pavón, "The LOTOSphere Design Methodology". Dans: [BLV 95], 29-58, 1995.

[PL 90] S. Pavón et M. Lanas, "The Testing Functionalities of LOLA". Dans: *FORTE III, 3rd International Conference on Formal Description Techniques*, Madrid, novembre 1990.

[RKW 95] B. Regnell, K. Kimbler et A. Wesslén, "Improving the Use Case Driven Approach to Requirements Engineering". Dans: *Proceedings of Second International Symposium on Requirements Engineering*, York, U.K., mars 1995

[SGW 94] B. Selic, G. Gullekson et P.T. Ward, *Real-Time Object-Oriented Modeling*, Wiley & Sons, 1994.

[SDV 96] S. Somé, R. Dssouli et J. Vaucher, "Un cadre pour l'ingénierie des exigences avec des scénarios". Dans: A. Bennani, R. Dssouli, A. Benkiran et O. Rafiq (éds), *CFIP 96, Ingénierie des protocoles*, ENSIAS, Rabat, Maroc, 1996.

[Tuok 96] R. Tuok, *Modeling and Derivation of Scenarios for a Mobile Telephony System in LOTOS*. Thèse de maîtrise, Département d'informatique, Université d'Ottawa, Ottawa, Canada, 1996.