Modélisation par Composants des Interfaces Réseaux dans les Simulateurs de Réseaux Sans Fil

Ryad Ben-El-Kezadri

Thèse de Doctorat de l'Université de Paris VI
Pierre et Marie Curie

Spécialité

Systèmes informatiques

présentée par

M. Ryad BEN-EL-KEZADRI

pour obtenir le titre de

Docteur de l'Université Pierre et Marie Curie

Modélisation par Composants des Interfaces Réseaux dans les Simulateurs de Réseaux Sans Fil

Soutenue le 14/12/2007 devant le jury composé de

Rapporteur	M.	André-Luc Beylot	Prof à l'Ecole Nationale Supérieure d'Electronique, d'Electrotechnique, d'Informatique, d'Hydraulique, et des Télécommunications, France
Rapporteur	M.	Walid Dabbous	Directeur de recherche à l'Institut National de Recherche en Informatique et en Automatique, France
Examinatrice	Mme	Monique Becker	Prof à l'Institut National des Télécommunications, France
Examinateur	M.	Serge Fdida	Prof à l'Université Pierre et Marie Curie, France
Directeur	M.	Guy Pujolle	Prof à l'Université Pierre et Marie Curie, France
Directeur	M.	Farouk Kamoun	Prof à l'Ecole Nationale des Sciences de l'Informatique, Tunisie

A ma famille

Remerciements

Je tiens tout d'abord à remercier les membres de ce jury :

- Mme Monique Becker qui m'a fait l'honneur de présider mon jury de thèse,
- Mr Andre-Luc Beylot et Mr Walid Dabbous pour avoir accepté d'être rapporteurs de ma thèse,
- Mr Serge Fdida qui a eu l'amabilité de participer à ce jury,
- Mr Guy Pujolle et Mr Farouk Kamoun pour m'avoir permis de réaliser ce travail et pour leur soutien permanent.

Je remercie également :

- les membres du Laboratoire Informatique de Paris 6 et du laboratoire Cristal avec qui j'ai pu travaillé (dans l'ordre chronologique) : Davor Males, Hugues Lecarpentier, Olish, Wafa Berrayana, Mohamed Ayadi, Meriem Ben Ghorbel, Harabi Hamden, Badr ElMontacer, Jalel Barhoumi, Firas Toumi, Nabil Zouaoui et Marouane Ommezine,
- mes anciens encadrants et tout ceux que j'ai pu rencontrer au cours de mes années d'études,
- mes amis proches,
- ma petite et grande famille.

Résumé

Ces dernières années ont été marquées par l'intégration dans les interfaces de réseaux sans fil de fonctionnalités de plus en plus nombreuses : qualité de service, routage, sécurité, mobilité, etc. Ces interfaces évoluent aujourd'hui vers des systèmes autonomes adaptatifs.

Le problème de la simulation dans ce contexte est que l'écart technologique entre ces équipements et les moutures de simulation disponibles est devenu trop grand.

Les systèmes en composants offrant un meilleur potentiel d'adaptation et d'évolution que les structures monolithiques classiques, nous proposons l'application du principe de modularisation aux cartes réseaux dans les simulateurs.

Cette thèse est une réflexion autour de la flexibilité et de la réutilisation des composants dans les interfaces réseaux des simulateurs sans fil. Nous avons recours pour ce faire aux méthodes d'ingénierie logicielle FODA (Feature Oriented Domain Analysis) et FORM (Feature Oriented Reuse Method) pour définir l'ensemble des fonctionnalités possibles d'une interface réseau et les mapper dans une structure logicielle. La structure ainsi définie met en œuvre plusieurs styles d'architecture dont les machines virtuelles, les systèmes à évènements et les processus communicants. Une suite d'outils permettant la comparaison des simulateurs est également proposée dans le cadre de la validation d'un tel système.

Nos contributions sont les suivantes. (i) La conception et l'implémentation d'une suite d'outils dédiée à la comparaison et à la validation des simulateurs grâce à la mesure et à la certification par rapport aux spécifications. La solution se base sur un format de trace standard très détaillé baptisé 'langage neutre'. (ii) La définition d'un procédé de mapping des préoccupations utilisateurs vers une architecture de simulation. Celui s'appuie sur les méthodes FODA et FORM. (iii) La définition d'un prototype de carte réseau basé sur cette méthodologie.

Mots clefs : Réseaux sans fil, interface réseau, simulateur, modélisation, architecture logicielle, flexibilité, réutilisabilité des composants.

Abstract

These last years were marked by the integration into wireless network interface cards of an increasing number of functions: quality of service, routing, security, mobility, etc. These devices are now evolving towards autonomous and adaptive systems.

The problem of network simulation in this context is that the technological gap between these devices and available simulators has become too large.

Since modular systems offer better adaptation and evolution potentials than traditional monolithic structures, we propose to apply the modularization principle to the network interface cards of simulators.

This thesis is a reflection about flexibility and component reuse in the wireless network interfaces of simulators. We make use of the FODA (Feature Oriented Domain Analysis) and FORM (Feature Oriented Reuse Method) software engineering methods to define the functions of a network interface and their mapping into a unified software structure. The structure thus defined implements several architectural styles such as virtual machines, event systems and communicating processes. A toolset allowing the comparison of simulators is also proposed to validate this structure.

We contribute to (i) the design of a toolset dedicated to simulators comparison and verification through measurement and validation against the specifications. The solution is based on a detailed and standard trace file format we named ‘neutral language’. (ii) the mapping of users concerns to a simulation architecture. This process is based on the FODA and FORM method. (iii) the design of a prototype network card based on this methodology.

Key words : Wireless networks, network interface card, simulator, modeling, software architecture, flexibility, component reuse

Table des Matières

I. INTRODUCTION 13

1. YAVISTA, UNE SUITE D'OUTILS LOGICIEL POUR LA VALIDATION ET LA COMPARAISON DES SIMULATEURS MANET 17

1. INTRODUCTION 17
2. LES RAISONS DE LA NOUVELLE ARCHITECTURE YAVISTA 18
3. LE LANGAGE NEUTRE YAVISTA 19
3.1. Définition du langage neutre 21
3.2. Implémentation du langage neutre 23
4. LA SUITE LOGICIELLE YAVISTA 25
4.1. Le séqueuceur 25
4.1.1. Les propriétés du séqueuceur 25
4.1.2. Travaux similaires 25
4.2. Le grapheur 26
4.2.1. Les propriétés du grapheur 26
4.2.2. Travaux similaires 27
4.3. Le certifieur 28
4.3.1. Spécifications formelles du protocole d'accès DCF en langage neutre 28
4.3.2. Le moteur ELYSE 29
4.3.2.1. Propriétés du moteur ELYSE 31
4.3.3. Travaux similaires 31
5. LE PROBLEME DE CORRECTION DES SIMULATEURS 32
5.1. Etat de l'art des caractéristiques de NS-2 et Glomosim 32
5.2. Etude de cas : nombre de collisions observé sous NS-2 et Glomosim 33
5.2.1. Sous NS-2 33
5.2.2. Sous Glomosim 34
6. CONCLUSION 36
RÉFÉRENCES 37
NETOGRAPHIE 38

2. FLEXIBILITE ET REUTILISABILITE DES SYSTEMES 39

1. INTRODUCTION 39
2. MODULARISATION DES LIGNES DE PRODUITS 41
2.1. Module, modularité et modularisation 41
2.2. Architecture des lignes de produits 43
2.3. Principes d'organisation des architectures 44
2.3.1. Architecture centralisée 44
2.3.2. Architecture en machine virtuelle 45
2.3.3. Architecture appel et retour 45
2.3.4. Architecture à composants indépendants 46
2.3.5. Architecture par flot de données 47
3. LA MODULARITE LOGICIELLE 47
3.1. Définition du module 47
3.1.1. Cohésion 48
3.1.2. Couplage 48
3.2. Implémentation d'un module 49
3.3. Mesures de modularité et de structure 49
3.3.1. Métriques de morphologie 50

3.3.2. Métriques d'impureté de l'arbre ... 51
3.3.3. Métriques de flots d'information ... 51
3.3.4. Métriques de complexité ... 52
3.3.5. Métriques de cohésion ... 53
3.3.6. Métriques de couplage ... 53
3.3.7. Métriques de modularité générale ... 53
4. LA GENERALITE ... 54
4.1. Les caractéristiques ... 54
4.2. Introduction et fixation de la variabilité dans les lignes de produits ... 55
4.3. Introduction et fixation de la variabilité dans les systèmes adaptables ... 57
4.3.1. Réflexion structurelle ... 58
4.3.2. Réflexion comportementale ... 58
4.3.3. Réflexion architecturale ... 59
4.4. Architecture des systèmes adaptables ... 59
4.5. Mesure d'adaptation ... 60
5. CONCLUSION ... 62
RÉFÉRENCES ... 62

3. FLEXIBILITE ET REUTILISABILITE DES INTERFACES RESEAUX DANS LES SIMULATEURS SANS FIL ... 65

1. INTRODUCTION ... 65
2. AVANTAGES ET INCONVENIENTS DE LA MODULARISATION DANS LE DOMAINE DE LA SIMULATION SANS FIL ... 66
2.1. Facteurs limitant ... 66
2.2. Facteurs motivants ... 67
3. MODULARISATION DES INTERFACES RESEAUX ... 67
3.1. Considérations relatives à la mesure de la modularité des simulateurs ... 68
3.2. Considérations relatives à la mesure de la généralité des simulateurs ... 68
3.2.1. Caractéristiques liées à l'accès au canal ... 71
3.2.2. Caractéristiques liées à la file d'attente ... 71
3.2.3. Caractéristiques liées au contrôle des flux ... 72
3.2.4. Caractéristiques liées à la mise en forme des flux ... 73
3.2.5. Caractéristiques liées au contrôle des ressources ... 73
3.2.6. Caractéristiques liées à l'adaptation à l'environnement ... 74
3.2.7. Caractéristiques liées à l'orientation des données ... 74
3.2.8. Caractéristiques liées à l'écoute du médium ... 75
4. MESURES DE GENERALITE ET DE MODULARITE DES SIMULATEURS RESEAUX ... 76
4.1. NS ... 76
4.1.1. Mesures de généralité ... 76
4.1.2. Mesures de modularité ... 79
4.1.2.1. Mesures de morphologie et d'impureté ... 79
4.1.2.2. Mesures de flots d'information, de couplage et de complexité ... 79
4.2. Glomosim ... 80
4.2.1. Mesures de généralité ... 81
4.2.2. Mesures de modularité ... 83
4.2.2.1. Mesures de morphologie et d'impureté ... 83
4.2.2.2. Mesures de flots d'information, de couplage et de complexité ... 83
4.3. J-Sim ... 83
4.3.1. Mesures de généralité ... 84
4.3.2. Mesures de modularité ... 86
4.3.2.1. Mesures de morphologie et d'impureté ... 86
4.3.2.2. Mesures de flots d'information, de couplage et de complexité ... 86
4.4. YANS ... 87
4.4.1. Mesures de généralité ... 89
4.4.2. Mesures de modularité ... 93

4.4.2.1. Mesures de morphologie et d'impureté 93
4.4.2.2. Mesures de flots d'information, de couplage et de complexité 93
4.5. Analyse des résultats *95*
4.5.1. Mesures de généralité 95
4.5.2. Mesures de modularité 96
5. CONCLUSION 97
REFERENCES 97
NETOGRAPHIE 100

4. COW, UN MODELE D'INTERFACE RESEAU FLEXIBLE ET REUTILISABLE 101

1. INTRODUCTION 101
2. CHOIX DE CONCEPTION COMPOSANT 101
2.1. Le développement orienté composant *101*
2.1.1. Les composants logiciels 101
2.1.2. Ingénierie des logiciels à base de composants 102
2.2. La plateforme de simulation *103*
2.2.1. Les plateformes génériques à base de composants 103
2.2.2. Les simulateurs réseaux traditionnels 103
2.2.3. Les simulateur réseaux à base de composants 103
2.2.4. Le simulateur J-Sim 104
3. PRESENTATION DU MODELE COW 104
3.1. Développement des composants réutilisables *104*
3.1.1. Vue en modules 106
3.1.2. Vue en processus 108
3.1.3. Vue en sous modules 110
3.1.3.1. Vue en sous modules de 802.11 111
3.1.3.2. Vue en sous modules de TSMA 113
3.2. Développement des produits spécifiques *115*
3.2.1. Sélection des caractéristiques 115
3.2.2. Adaptation du système 115
3.2.2.1. Service adaptable 115
3.2.2.2. Monitoring de l'environnement 116
3.2.2.3. Logique adaptative 116
3.2.3. Composition des modules 117
3.2.3.1. Interconnexion des composants 117
3.2.3.2. Vérification des interconnexions 117
4. MESURES DE GENERALITE ET DE MODULARITE 117
4.1. Mesures de généralité *117*
4.2. Mesures de modularité *120*
4.2.1. Mesures de morphologie et d'impureté 120
4.2.2. Mesures de flots d'information, de couplage et de complexité 121
4.3. Analyse des résultats *122*
4.3.1. Mesures de généralité 122
4.3.2. Mesures de modularité 122
5. CONCLUSION 123
REFERENCES 124
NETOGRAPHIE 124

5. LES PARADIGMES DE SIMULATION DU SIMULATEUR YAVISTA 127

1. INTRODUCTION 127
2. LE MODELE YAVISTA 127
2.1. L'architecture conceptuelle de Yavista *128*
2.2. Le modèle d'accès au canal Yavista *129*
3. LES OBJECTIFS DU SIMULATEUR YAVISTA 129

3.1. Simulation parallèle *129*
3.2. Sélection temps réel des caractéristiques *129*
3.3. Réutilisation des éléments de simulation dans les cartes réseaux *130*
3.4. Validation complète avec ELYSEE *130*
4. CONCLUSION 131
RÉFÉRENCES 132

CONCLUSION 133

1. SYNTHESE 133
1.1. La validation des simulateurs *133*
1.2. La flexibilité et la réutilisabilité dans les simulateurs sans fil *134*
2. PERSPECTIVES 135
2.1. L'aide à la recherche avec la suite Yavista *135*
2.2. La recherche avec le simulateur Yavista et ELYSE *135*

BIBLIOGRAPHIE PERSONNELLE 137

CONFERENCES INTERNATIONALES 137
JOURNAUX 137

A. ANNEXE A : IDENTIFICATION DES CARACTERISTIQUES DE NS-2.30 ET GLOMOSIM 2.03 139

1. INTERPRETATIONS DU STANDARD 139
1.1. Accès au canal lorsque le medium est libre *139*
1.2. Utilisation de EIFS si erreur PLCP *140*
2. SIMPLIFICATIONS DU SYSTEME 141
2.1. Comportement des stations au démarrage *141*
2.2. Capture RTS *142*
2.3. Décrementation du backoff *143*
2.4. Modèle de capture *147*
3. ERREURS DE MODELISATION 149
3.1. Gestion du backoff quand les files sont vides *149*
3.2. Durée IFS *150*
3.3. Resynchronisation EIFS *151*
3.4. Perturbation des échanges de trames *152*
3.5. Remise à zéro EIFS *153*
RÉFÉRENCES 154

B. ANNEXE B : INSTRUMENTATION DE 802.11 DANS NS-2.29 ET GLOMOSIM 2.03 155

1. NS-2 155
1.1. Processus de décodage *155*
1.2. Processus de transmission *156*
2. GLOMOSIM 157
NETOGRAPHIE 159

C. ANNEXE C : CONTRATS DE COW 161

LISTE DES FIGURES 163

LISTE DES TABLEAUX 166

Introduction

Le domaine de la simulation des réseaux semble aujourd'hui vivre un tournant. Jusqu'alors les évolutions de simulateurs étaient réalisées sous forme de contributions externes par extension des modèles originaux fournis dans les distributions. Le problème de ce modèle de développement est que l'écart technologique entre les modèles de base fournis dans les simulateurs et les systèmes réels est devenu trop grand. Le problème affecte tout particulièrement les technologies sans fil.

Celles-ci utilisent effectivement des concepts inédits (mobilité, gestion de l'énergie, adaptation des débits, de la puissance, réseau auto formant), des techniques d'accès au réseau compliquées (gestion du canal partagé) et des mécanismes traditionnellement pris en charge au niveau réseau (sécurité, qualité de service, routage, etc). Les réseaux de nouvelles générations (réseaux 3G, réseaux ad-hoc, réseaux ambiants) se positionnent maintenant comme des réseaux à services ajoutés dotés de fonctions d'auto-configuration et d'auto-gestion.

Face au renouvellement incessant des standards, les interfaces et les pilotes réseaux sont maintenant conçus pour fournir des services généraux et évolutifs. Le driver Madwifi supporte à ce titre les normes 802.11, 802.11h, 802.11e et 802.11i.

Les simulateurs classiques (NS-2, Glomosim) ont jusqu'alors fait preuve d'une longévité remarquable. Celle-ci s'explique en partie par leur structure minimaliste et monolithique facilement extensible. La fracture technologique observée avec les systèmes réels place cependant aujourd'hui les utilisateurs de simulateurs dans une impasse.

Pour ne pas devenir obsolète, les outils de simulation devront s'adapter et offrir des services et une flexibilité comparables aux systèmes modélisés. Les simulateurs se transformeront dans cette optique en outils complets et précis capables de simuler les interactions entre réseaux hétérogènes ou l'adaptation aux conditions environnementales. Les couches du modèle OSI s'apparenteront dès lors à des logiciels à part entière dotés de fonctionnalités inspirées des systèmes réels: mise à jour du code, reconfiguration des interfaces (ajout d'une fonction de correction d'erreur ou de QoS, modification de l'accès au médium, etc).

Dans cette optique, le développement de nouvelles moutures de simulation supportant des modèles à la fois plus complets et extensibles semble la voie la plus crédible. Le problème est que dans un simulateur complexe, les contraintes de flexibilité sont très fortes. L'ajout a posteriori d'une fonctionnalité même mineure peut ainsi nécessiter un effort d'implémentation plusieurs fois supérieur aux prévisions du fait des problèmes d'intégration. A titre d'exemple, l'intégration d'une nouvelle méthode d'accès au canal peut induire des modifications profondes dans la gestion des fonctionnalités de management ; celles-ci pouvant

encore se propager à d'autres niveaux de la couche liaison.

Dans ce mémoire, nous proposons une méthode de conception des simulateurs assimilant les simulateurs à des lignes de produits paramétrables et plaçant l'utilisateur au centre de la stratégie de développement. Notre méthode dérive des techniques FODA et FORM et de la spécification en composants issues de l'ingénierie logicielle. Grâce à la spécification des composants et à la définition d'architectures de lignes de produits, les composants peuvent être réutilisés pour différents produits ou familles de produits, même s'ils sont développés par des organisations complètement différentes avec des technologies elles aussi différentes. La modification de l'accès au canal est ici obtenue par simple remplacement de composant - l'architecture du produit prévoyant ce type de modification dès l'origine. Les méthodes orientées composants offrent de surcroît un potentiel de réutilisabilité maximal car elles se basent sur l'assemblage d'entités précompilées indépendantes possédant de fait des contraintes très fortes au niveau de la définition des interfaces et des interactions.

L'objectif de cette thèse est de proposer une expérience créative, un point de vue et une méthodologie pour concevoir une interface réseau flexible et réutilisable dans un simulateur. Toutes les méthodes et techniques proposées dans ce mémoire placent l'utilisateur et le développeur au centre de la problématique : nécessité de comprendre, nécessité d'analyser, nécessité de pouvoir manipuler, etc. La Figure i.1 montre l'impact des contraintes et des besoins formulés par ces utilisateurs sur l'architecture des simulateurs MANET de nouvelle génération.

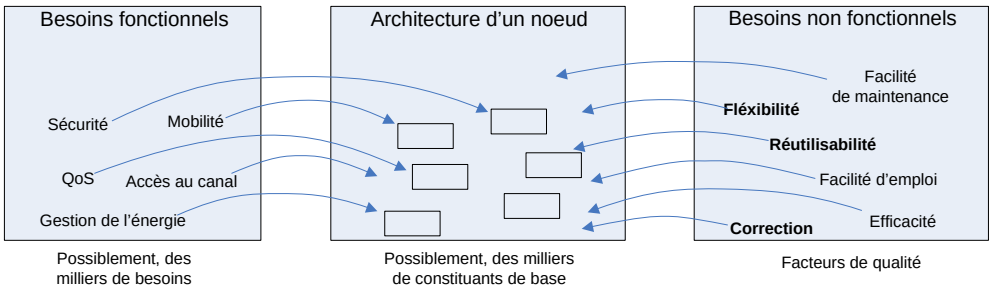


Figure i.1 Impact des besoins fonctionnels et non fonctionnels sur l'architecture des simulateurs

La mise en place d'un tel système ne va pas sans lever un certain nombre de questions et de problèmes auxquels nous tâcherons de répondre au cours de ce mémoire, tant au point de vue pratique - ingénierie logicielle - que théorique - principe de modélisation métier (réseau) et des préoccupations utilisateurs.
Les contributions de cette thèse sont :

- La conception et l'implémentation d'une suite d'outils baptisée YAVISTA (Yet Another Toolset to Validate the Simulator Activity) dédiée à la comparaison et à la validation des simulateurs grâce à la mesure et à la certification par rapport aux spécifications. L'architecture du système se base sur le concept de code neutre.
- la définition d'un procédé de mapping des préoccupations utilisateurs vers une architecture de simulation. Celui se base sur les méthodes FODA (Feature Oriented Domain Analysis) et FORM (Feature Oriented Reuse Method).
- La définition d'un prototype de carte réseau baptisée CoW (Component WiFi) basé sur cette

méthodologie.

Le premier chapitre de ce mémoire propose une solution au problème de correction des simulateurs MANET. Nous proposons dans ce cadre une méthode permettant la comparaison et la validation massive des simulateurs. Par comparaison massive, nous entendons la comparaison des traces d'exécution produites par différents simulateurs, différentes versions de simulateurs ou même éventuellement des cartes réseaux réelles. L'objectif avoué de ce chapitre est de proposer des moyens et des outils permettant de s'assurer de la correction d'un protocole sans recours à des moyens de validation externes : comparaison à des modèles analytiques, aux mesures empiriques ou aux simulateurs. CoW se voulant 'un simulateur conçu pour être validé', cette introduction montre in fine le lien invisible unissant le modèle de simulation CoW à son modèle de validation, YAVISTA.

Les deux chapitres suivants sont dédiés à l'état de l'art sur la flexibilité et la réutilisabilité.

La première partie de l'état de l'art concerne la flexibilité et la réutilisabilité dans les systèmes. Nous expliquons l'intérêt et les méthodes pour créer des logiciels généraux et modulaires. Rapportée au domaine d'étude, cette analyse prépare la critique des simulateurs réseaux existants et la mesure de notre conception.

La seconde partie de l'état de l'art offre un état de l'art de la flexibilité et de la réutilisabilité dans les simulateurs réseaux existants à l'aide de la méthode FODA. Nous étudions et mesurons en particulier les simulateurs réseaux NS-2, J-Sim, Glomosim et YANS.

Le chapitre quatre est relatif à la conception de CoW avec FORM. Nous présentons la méthode de conception, l'architecture et les modules implémentés. Comme nous le verrons, des composants spécifiques sont développés pour la prise en compte des fonctions de gestion, de contrôle et de stockage des paquets. Il en résulte une architecture hautement fonctionnelle, proche de la norme originale. Des produits sont proposés pour les technologies 802.11 et TSMA.

Le dernier chapitre propose une vue étendue des concepts développés dans ce mémoire au travers de l'architecture conceptuelle ultra flexible du simulateur Yavista.

Chapitre 1

YAVISTA, une Suite d'Outils Logiciel pour la Validation et la Comparaison des Simulateurs MANET

1. Introduction

La comparaison et la validation des simulateurs est un domaine difficile et peu traité, faute d'outil d'analyse adéquat. Le manque de travaux et de visibilité inspire souvent chez l'utilisateur un sentiment de méfiance vis-à-vis de la simulation. Aujourd'hui, la communauté attend des nouveaux standards de simulation, la conception d'outils facilitant l'accès aux résultats de simulation et leur validation. Le problème qui se pose actuellement est la dépendance exclusive des outils d'analyse vis-à-vis des simulateurs. Sans standardisation dans le domaine, l'effort resterait dispersé et la comparaison et l'analyse des simulateurs difficiles.

Ce problème affecte tout particulièrement les réseaux MANET (Mobile Ad-hoc Network) car le protocole 802.11 est compliqué et les simulateurs contiennent un grand nombre de simplifications difficiles à évaluer sans outil adapté. Pour pallier à ces problèmes, nous avons mis au point une solution clef en main baptisée YAVISTA (Yet Another Toolset To Validate the Simulator Activity) permettant la compréhension, la comparaison et la certification des simulateurs MANET. YAVISTA se base sur l'instrumentation des implémentations et l'utilisation d'un format de trace standard. Le système propose trois types d'analyse : l'affichage de l'activité des stations sous forme de frise temporelle avec YAVISTA Sequencer, la visualisation des métriques de simulation avec YAVISTA Grapher et la validation d'une implémentation contre les spécifications du standard 802.11 avec YAVISTA Certifier. Le plan de ce chapitre est le suivant. Nous présentons d'abord les raisons ayant motivées la définition de la nouvelle architecture de simulation YAVISTA. Nous décrivons ensuite les différents éléments de la solution[1] en détaillant successivement notre format de trace standard, dit 'langage neutre', YAVISTA Sequencer, YAVISTA Grapher et YAVISTA Certifier. Nous proposons enfin un état de l'art des différents problèmes pouvant affecter les simulateurs. Nous appliquons à ce titre YAVISTA sur NS-2 et Glomosim.

[1] YAVISTA étant un système très complet, seule une partie de ses fonctionnalités est présentée dans ce chapitre. Pour plus d'informations sur le logiciel, prière de visiter yavista.sourceforge.net

2. Les Raisons de la Nouvelle Architecture YAVISTA

La comparaison des simulateurs MANET est un sujet sensible aux conclusions contrastées. Bien que reconnue comme une méthode de validation puissante [BAG99], la comparaison des simulateurs a jusqu'alors apporté plus de questions que de réponses. Une compilation des différents cas d'études questionnant la crédibilité des simulateurs MANET est à ce titre fournie dans [AND06]. Dans [TAK01], les auteurs concluent à la similarité des modèles de NS-2 et Glomosim. A contrario, [CAV02] et [RED06] mettent en avant d'importantes différences entre les résultats de simulation. [CAV02] explique le phénomène par l'existence probable de défauts cachés dans l'implémentation des couches PHY et MAC de NS-2, Glomosim et Opnet. [RED06] fournit une tentative d'explication des différences observées entre GTNetS, NS-2 et Glomosim ainsi qu'une liste de recommandations pour paramétrer identiquement les simulateurs.

L'obstacle auquel se confrontent ces travaux, est la comparaison de résultats incomplets, dans des formats différents avec des outils d'analyse eux aussi différents. Les simulateurs évoluent de sorte dans des domaines cloisonnés, chaque simulateur possédant sa propre suite d'outils d'analyse (voir Figure 1.1).

Les conséquences sont triples :

1. Les méthodes actuelles ne permettent pas d'identifier raisonnablement l'ensemble des caractéristiques des simulateurs,
2. L'explication et la justification de ces différences au travers d'analyses de code se montrent généralement très lourdes, le code incriminé exhibant souvent des aspects très techniques,
3. L'impact de ces différences sur les résultats est lui-même difficilement mesurable, faute de pouvoir dégager un comportement de simulation standard général.

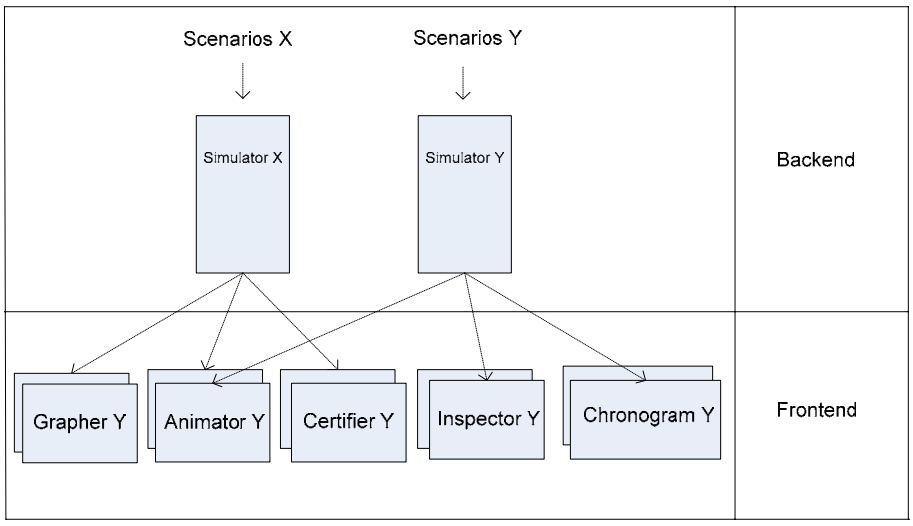


Figure 1.1 Architecture des simulateurs

Ces observations motivent la généralisation des audits de simulation à l'ensemble du modèle OSI et la mise en place d'un framework étendu permettant la validation et l'analyse des résultats.

Le framework proposé dans ce mémoire s'inspire de la structure des compilateurs qui propose autant de 'backends' que de langages sources à traduire en langage intermédiaire et de 'frontends' que de langages cibles disponibles. L'utilisation d'un langage intermédiaire unique permet la déclinaison d'un seul logiciel d'analyse pour l'ensemble des simulateurs.

La particularité de Yavista est de faire de ce langage intermédiaire l'élément principal de l'architecture de simulation.

La suite d'outil Yavista propose une implémentation partielle de cette architecture pour les réseaux MANET avec au niveau du 'middleend' le portage d'un langage neutre dans {NS-2.29, NS-2.30 [Net-NS-2], Glomosim 2.03 [Net-Glomosim]} et au niveau du frontend un grapheur, un certifieur, et un générateur de chronogramme. La Figure 1.2 replace chacun de ces éléments dans le contexte général.

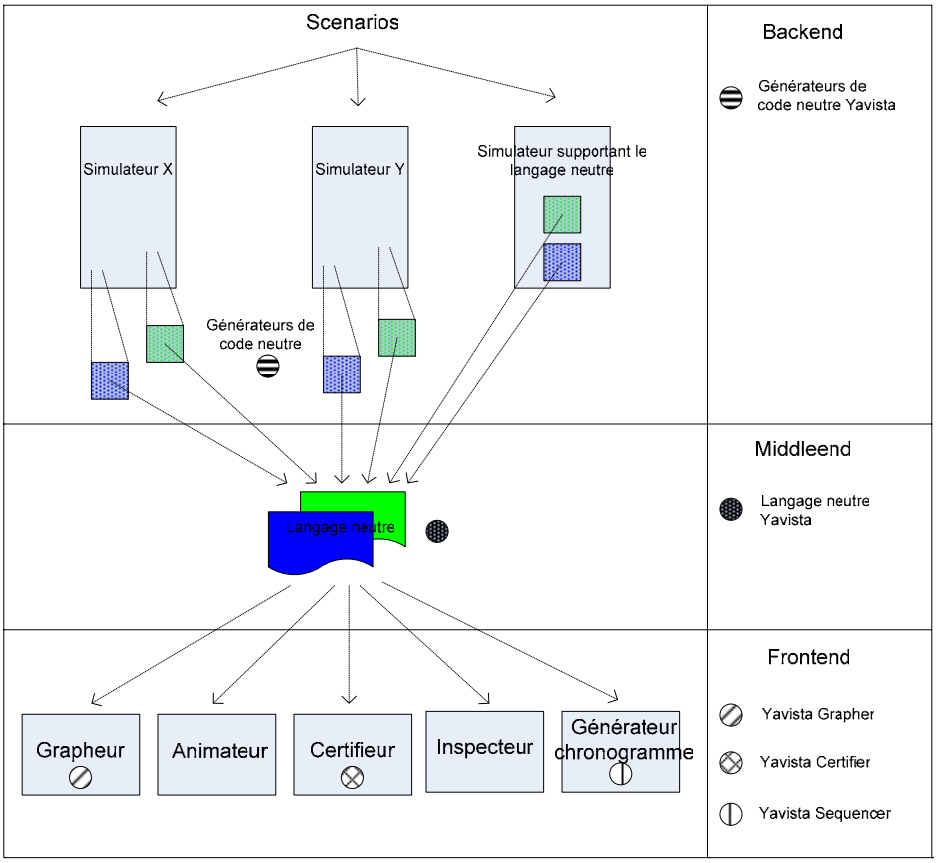


Figure 1.2 Architecture de standardisation des simulateurs

3. Le Langage Neutre YAVISTA

Contrairement aux solutions traditionnelles, YAVISTA [Net-Yavista] se base sur un format de trace standard très complet, dit langage neutre, définissant une interface commune à l'ensemble des simulateurs (voir Figure 1.3). Ces traces sont ensuite traitées pour mesurer, valider et comparer le comportement des simulateurs.

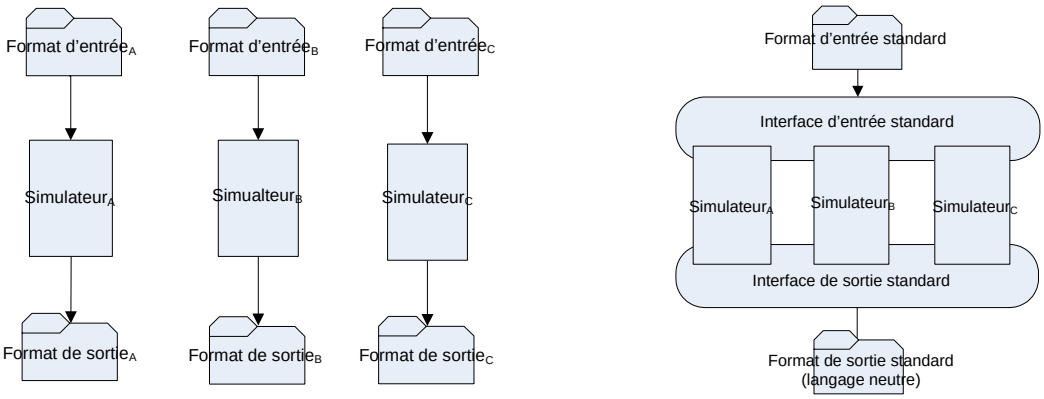


Figure 1.3 Méthode de comparaison des simulateurs traditionnelle (à gauche) et YAVISTA (à droite)

L'avantage des méthodes de validation basées sur l'analyse des traces est d'être particulièrement adaptées aux trois étapes du processus de validation et de correction [SAR98] : validité du modèle conceptuel, vérification du modèle implémenté et validité opérationnelle (voir Figure 1.4).

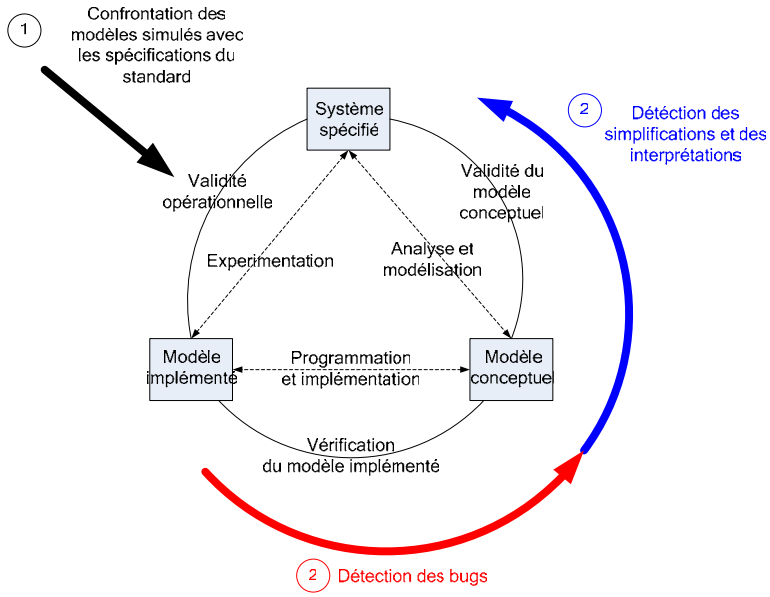


Figure 1.4 Approche de comparaison sous YAVISTA

Dans le contexte MANET, le problème de validation des simulateurs, outre leur propre instrumentation, provient de la difficulté à instrumenter les systèmes réels (ceux-ci pouvant de surcroît interpréter les spécifications). Yavista contourne ce problème, en retrouvant l'ensemble des simplifications, des interprétations et des bugs des simulateurs à partir de la confrontation des traces des modèles implémentés avec le comportement attendu du système (voir Figure 1.4).

3.1. Définition du langage neutre

Nous appelons langage neutre, le langage le plus simple à comprendre pour un humain, à valider par une machine et capable de décrire l'ensemble des actions $a_1, a_2, ..., a_n$ réalisées par un protocole réseau. Nous appelons code neutre, une trace d'exécution écrite en langage neutre.

Basiquement, le langage neutre comprend un ensemble de mots standards $\{<a_1>,<a_2>, ...,<a_n>\}$ capables de décrire l'ensemble des actions $a_1, a_2, ..., a_n$. Chaque action possède un nom et un ensemble d'attributs indiquant par exemple la durée et l'instant où elle commence. Les traces décrivant l'ensemble de l'exécution du protocole, celles-ci contiennent, et le comportement normal des simulateurs, et la manifestation des caractéristiques des simulateurs – une caractéristique étant définie comme l'expression d'une différence de comportement entre les simulateurs.

La propriété fondamentale du code neutre est d'offrir une interface standard entre le frontend et le backend. Il est de fait impossible d'identifier le simulateur ayant généré la trace T, si T est exempte de caractéristiques.

Les éléments du langage neutre YAVISTA proposés pour la couche MAC des simulateurs MANET sont présentés dans les Tables 1.1, 1.2 et 1.3. Ce langage a été porté dans E = { Glomosim 2.03, NS-2.29, NS-2.30 } par instrumentation du code. Le langage est concrètement implémenté en XML. L'utilisation du langage XML est justifiée par la richesse de l'API XML. Les applications directes de cette API dans le middleend sont la fusion et la segmentation des traces, la validation lexicale contre une DTD, etc.

Table 1.1 Description des éléments XML

Nom de l'évènement	Description de l'évènement
CS_PHY	Activité physique sur le canal
CS_NAV	Activité logicielle sur le canal
CS_DEFER	Attente d'intertrame IFS
CS_BACKOFF	Période de backoff
MAC_ACK	Transmission d'un ACK
MAC_RTS	Transmission d'un RTS
MAC_CTS	Transmission d'un CTS
MAC_DATA	Transmission de donnée
TX_WF	Attente d'une réponse pour un CTS, un ACK ou des données
RECV_ACK	Réception d'un ACK
RECV_RTS	Réception d'un RTS
RECV_CTS	Réception d'un CTS
RECV_DATA	Réception de données
RECV_NOT_DEST	Réception de données non destinées à la station
DROP_ERROR	Réception de données erronées
RESUME_BO	Reprise de backoff
PAUSE_BO	Pause de backoff
START_BO	Tirage de backoff
TOMEDIUMDELAY	Temps entre le moment où la couche PHY reçoit le paquet et le moment ou le premier bit du paquet est transmis
CONFIRM_PKT	Statut de l'opération d'envoi de données
CONFIRM_CTS	Statut de l'opération d'envoi de RTS

Table 1.2 Description des attributs XML

Nom de l'attribut	Description de l'attribut
Instant (instant)	Instant d'occurrence de l'évènement
Durée (duration)	Durée de l'évènement
Type de l'évènement (evtType)	Type de l'évènement (utilisé uniquement pour personnaliser l'affichage des évènements)
Identifiant du nœud (nid)	Identifiant du nœud ayant généré l'évènement
Nombre de slot de backoff tiré (totalslots)	Nombre de slots de backoff tirés
Nombre de slot de backoff attendu (pastslots)	Nombre de slots de backoff attendus
Fenêtre de contention (cw)	Fenêtre de contention
Source (source)	Identifiant du noeud source du message
Destination (destination)	Identifiant du noeud destination du message
Débit de transmission physique (rate)	Débit de transmission physique
Taille du paquet (size_packet)	Taille du paquet (sans l'entête physique)
Type de l'application (type)	Type de l'application ayant généré dans le message
Numéro de séquence MAC (seq_mac)	Numéro de séquence MAC
Numéro de séquence applicatif (seq_app)	Numéro de séquence applicatif
Offset de désynchronisation slot (offset)	Fraction de slot déjà attendue (pour les modèles de décrémentation de backoff continus)
STA Short Retry Count (ssrc)	MAC SSRC
STA Long Retry Count (slrc)	MAC SLRC
Statut de confirmation (status)	Indique si le paquet a été correctement reçu, doit être retransmis ou jeté
Drapeau RTS/CTS/DATA/ACK (rtsex)	Indique si le paquet appartient à un échange RTS/CTS/DATA/ACK
Drapeau de capture (captured)	Indique si le paquet reçu a été capturé
Drapeau de manque de puissance (lowpower)	Indique si le paquet reçu manque de puissance
Drapeau d'activité en transmission (tx_active)	Indique si le paquet reçu a été corrompu par une transmission
Drapeau de collision (collided)	Indique si le paquet reçu est entré en collision
Drapeau d'expiration (expired)	Indique si un intertrame ou un backoff se termine pour cause d'expiration ou d'interruption

Table 1.3 Format des éléments XML

Nom de l'évènement	instant	duration	evtType	nid	totalslots	pastslots	cw	source	destination	rate	size_packet	type	seq_mac	seq_app	offset	ssrc	slrc	status	rtsex	captured	lowpower	tx_active	collided	expired
CS_PHY	×	×	×	×																				
CS_NAV	×	×	×	×																				
CS_DEFER	×	×	×	×																				×
CS_BACKOFF	×	×	×	×											×									×
MAC_ACK	×	×	×	×				×	×	×														
MAC_RTS	×	×	×	×				×	×	×														
MAC_CTS	×	×	×	×				×	×	×														
MAC_DATA	×	×	×	×				×	×	×	×	×	×	×										
TX_WF	×	×	×	×																				
RECV_ACK	×	×	×	×				×	×											×				
RECV_RTS	×	×	×	×				×	×											×				
RECV_CTS	×	×	×	×				×	×											×				
RECV_DATA	×	×	×	×				×	×											×				
RECV_NOT_DEST	×	×	×	×				×	×											×				
DROP_ERROR	×	×	×	×				×													×	×	×	
RESUME_BO	×		×	×	×	×	×																	
PAUSE_BO	×		×	×	×	×	×																	
START_BO	×		×	×	×	×	×																	
TOMEDIUMDELAY	×	×	×	×																				
CONFIRM_PKT	×		×	×												×	×	×	×					
CONFIRM_CTS	×		×	×												×	×	×	×					

3.2. Implémentation du langage neutre

YAVISTA repose sur une instrumentation stricte du langage neutre. A titre d'exemple, l'évènement *MAC_ACK* doit commencer quand le premier bit de l'entête PLCP de l'acquittement est transmis sur le médium et se terminer à la transmission du dernier bit. Le coût de l'instrumentation est la nécessité de combler la distance entre le langage de programmation et le langage neutre à l'aide d'un patching de code complexe. Du fait du manque d'uniformisation des modèles dans les simulateurs, il peut en effet exister une certaine distance entre le modèle de simulation et le langage neutre. Le cas se présente pour les modèles simplifiés ou lorsque les informations nécessaires à l'émission du code neutre sont dispersées sur les couches adjacentes. Les structures de données du modèle de simulation ne réalisent pas alors le jeu complet d'actions $a_1, a_2, \ldots, a_n$ mais un ensemble éloigné moins structurant.

La Figure 1.5 illustre le rôle des générateurs de code neutre. Ceux-ci ont pour objectif de générer du code dans un cadre de référence – le langage neutre – sans altérer le système implanté. L'interprétation du code généré permet une détection plus simple – que par analyse du source[2] –des effets de bord introduits lors des phases de modélisation et d'implémentation du standard.

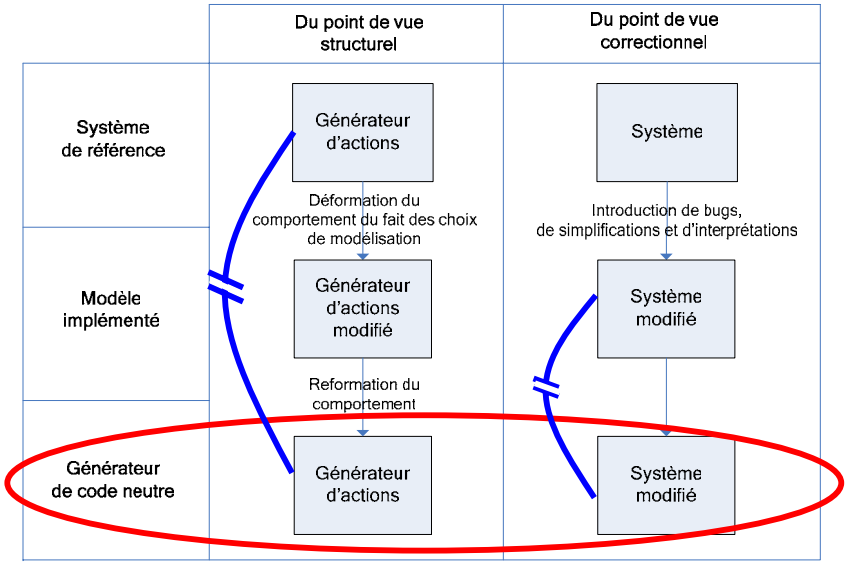


Figure 1.5 Rôle des générateurs de code neutre

Malgré l'effort d'implémentation nécessaire, la méthode est susceptible de se généraliser car elle permet de :

- fournir des audits de simulation complets : Dans [BEK07], la méthode a permis l'observation et la catégorisation de 11 comportements différents entre la couche MAC 802.11 de NS 2.30 et Glomosim 2.03.
- concrétiser des modèles abstraits dans une représentation physiquement définie et compréhensible à tous. Le code neutre est en ce sens d'avantage représentatif du fonctionnement d'une norme que le code source des simulateurs.

[2] Voir annexe B pour une représentation du code source de NS-2.29 et Glomosim 2.03.

La Figure 1.6 montre comment est implémentée l'interface de sortie standard dans les simulateurs MANET. Celle-ci se compose d'un module dédié à la couche PHY, l'autre à la couche MAC. Du fait du couplage entre les couches MAC et PHY et de la répartition des fonctionnalités de bas niveau dans les deux couches, l'implémentation des modules est distribuée sur toute l'interface radio. Le module MAC est lui-même doté d'un module de dissection des paquets pour classifier les flux en provenance des couches supérieures.

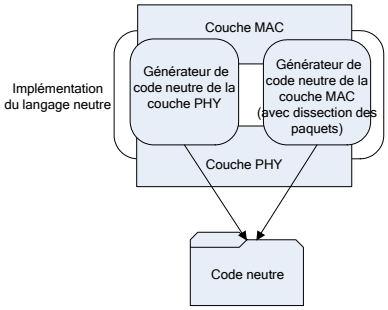


Figure 1.6 Implémentation de l'interface de sortie standard

L'instrumentation en code neutre se décompose en deux phases :

- La phase de modélisation permet de dégager les différents processus implémentés, les états des processus et leur flot de contrôle. Le flot de contrôle est représenté à l'aide d'un diagramme StateChart [HAR87] spécifiant les transitions, les conditions sur les transitions, les actions déclenchées lors des changements d'états et leurs évènements déclencheurs (voir Figure 1.7).
- L'implémentation du code neutre consiste à rajouter les actions StateChart nécessaires à l'émission du code neutre, c'est-à-dire à l'instanciation des éléments $<a_1>$, $<a_2>$, ..., $<a_n>$. Le travail nécessite de resynthétiser les actions a_1, a_2, ..., a_n à partir du modèle implémenté, en le complétant éventuellement si les informations nécessaires à l'opération manquent. Les détails de la procédure pour NS-2.29 et Glomosim 2.03 sont décrits dans [Net-Yavista].

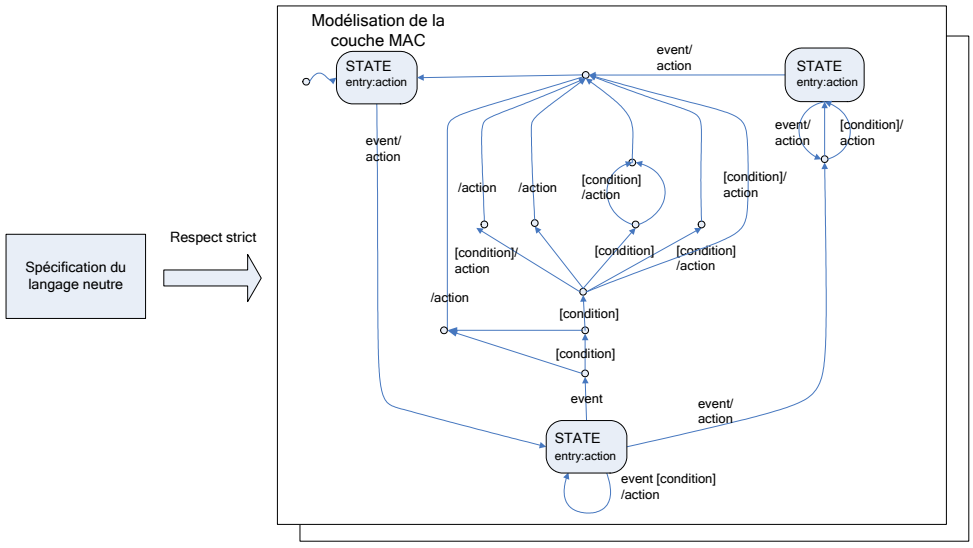


Figure 1.7 Méthode d'implémentation d'un module de génération de code neutre

4. La Suite Logicielle Yavista

Comme le montre la Figure 1.8, l'architecture de YAVISTA comprend au total, un générateur de chronogramme (le séquenceur), un traceur de graphe (le grapheur) et un certifieur.

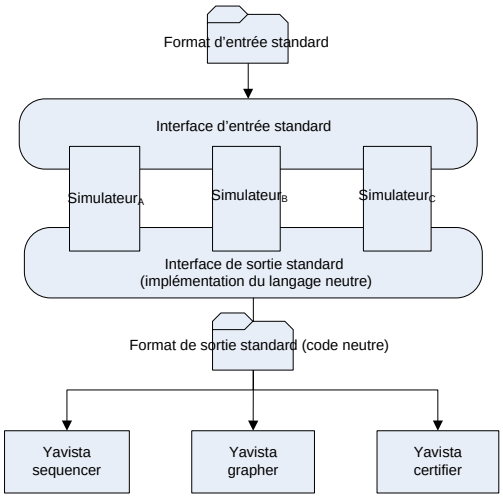


Figure 1.8 Architecture de YAVISTA

4.1. Le séquenceur

Le séquenceur fournit des frises temporelles décrivant les échanges de trames sur le médium. Ce type de représentation permet de contrôler la correction des échanges de trames entre les stations. L'interprétation du code neutre consiste en sa transformation au format graphique vectoriel SVG et à son affichage dans l'éditeur graphique libre Inkscape [Net-Inkscape].

4.1.1. Les propriétés du séquenceur

En plus des informations de niveau MAC, les frises détaillent les relations de bas niveau existant entre les couches MAC et PHY telles que les informations de capture (attribut *captured*), de collision (attribut *collided*) et de puissance (attribut *lowpower*). La Figure 1.9 montre une collision entre les nœuds *A* et *B*. L'évènement *DROP_ERROR TX* au niveau des noeuds *A* et *B* indique le rejet des paquets reçus pendant la transmission. Au niveau du nœud *C*, seul le paquet de *A* est capturé ; l'autre est perdu (collision). Les deux paquets reçus par le nœud *D* sont perdus du fait du manque de puissance (*DROP_ERROR POWER*).

4.1.2. Travaux similaires

Le premier générateur de frises temporelles a été proposé dans [DHO03]. Nam [EST00] intègre également un module graphant les occurrences d'évènements mais celui-ci est limité aux protocoles de haut niveau. L'inconvénient de ces outils est le manque de prise en compte des détails et le seul support des traces NS-2.

Figure 1.9 Frise temporelle générée avec YAVISTA séquenceur

4.2. Le grapheur

Le grapheur permet d'extraire les résultats de simulation des traces et de les afficher à l'écran. L'interprétation du code neutre est réalisée par un programme comptant le nombre d'occurrences d'un type donné (paquet reçu, collision provoquée, collision subie, capture provoquée, etc) dans les traces. La liste des nœuds impliqués dans la simulation est directement extraite du code neutre et affichée sous forme statique à l'écran. L'interface du grapheur se base sur l'API libre Jung [0'M04] permettant de sélectionner les éléments à comparer (nœud, groupe de noeuds, lien).

4.2.1. Les propriétés du grapheur

L'interface du grapheur (voir Figures 1.10 et 1.11) permet la sélection rapide des éléments de simulation à comparer (nœud, groupe de noeuds, lien) et des métriques, des unité de mesure et de la période d'échantillonnage à utiliser. L'outil offre également des fonctions de supervision réseau à l'aide d'un mécanisme d'alertes indiquant les principales différences entre les simulations à l'étude.

L'aide à la navigation inclut le déplacement de la vue dans l'environnement et la redisposition automatique et intelligente des éléments de simulation. Les différents niveaux de détail (niveau radio, niveau applicatif) permettent la localisation des nœuds responsables ou victimes des collisions et l'affichage des chemins de données empruntés sur le réseau par les paquets applicatifs, de routage, etc.

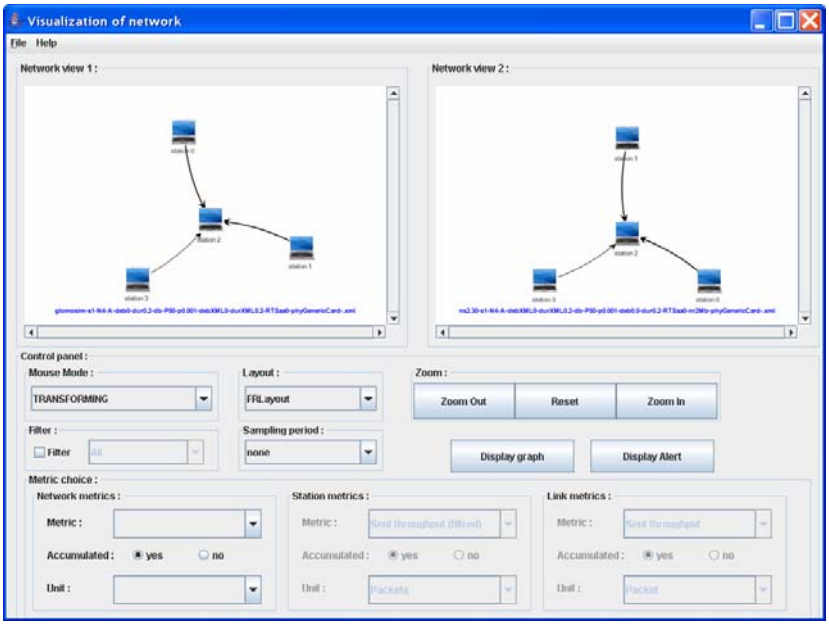


Figure 1.10 Visualisation des éléments du réseau avec YAVISTA grapheur

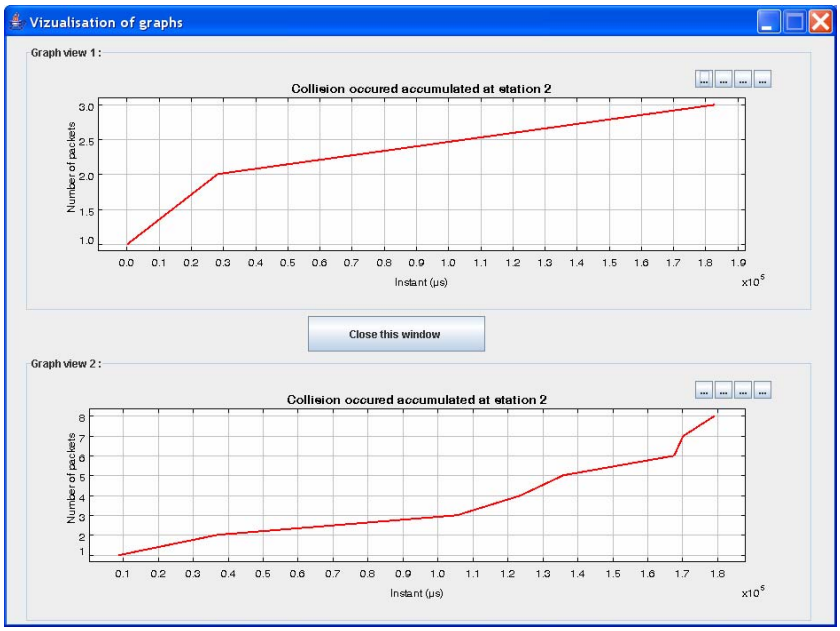


Figure 1.11 Comparaison de courbes avec YAVISTA grapheur

4.2.2. Travaux similaires

Dans le domaine de l'analyse des résultats, tracegraph [NOW03] permet l'affichage d'un large choix de métriques paquet. ns2measure [CIC06] permet en plus l'analyse statistique des résultats de simulation et le choix de métriques non orientées paquet (fenêtre de contention 802.11, etc).

4.3. Le certifieur

Le certifieur permet de mesurer le degré de conformité du code neutre par rapport aux spécifications des standards. Le degré de conformité d'une trace se mesure en pourcentage d'instructions neutres erronées. Les tests de certification YAVISTA font appel à des scénarios clefs impliquant un nombre minimal de stations et des topologies prédéfinies pour tester la validité des implémentations MAC des simulateurs.

L'approche que nous proposons avec Apache SCXML a déjà été empruntée par le projet de recherche [GAR06]. Celui-ci utilise, à notre façon, des automates d'états SCXML pour guider l'inspection des traces d'exécution. L'idée de base consiste à découper chaque séquence de messages en sous phases (macro état) et associer un service de validation à chaque macro état. Un service de validation teste la présence et la correction des éléments du langage neutre attendus dans la trace lors de la validation.

4.3.1. Spécifications formelles du protocole d'accès DCF en langage neutre

Le certifieur est conçu sous la forme d'automates à états programmés en code neutre. L'utilisation du code neutre garantit la compréhensibilité des règles par un humain et leur maintenance. Les automates du certifieur sont modélisés en deux niveaux :

- Le haut niveau reproduit les séquences de messages spécifiées dans le standard. Les Figures 1.12 et 1.13 décrivent respectivement un exemple de séquence DATA/ACK au niveau de l'émetteur et du récepteur. Une séquence en émission décrit l'ensemble des macros états attendus lors d'une opération d'émission : attente de transmission, transmission, attente de confirmation, attente de backoff.
- Le bas niveau décrit l'implémentation (ou service de validation) des différents macros états identifiés dans le haut niveau. Différentes interprétations du standard peuvent mener à différentes implémentations de bas niveau.

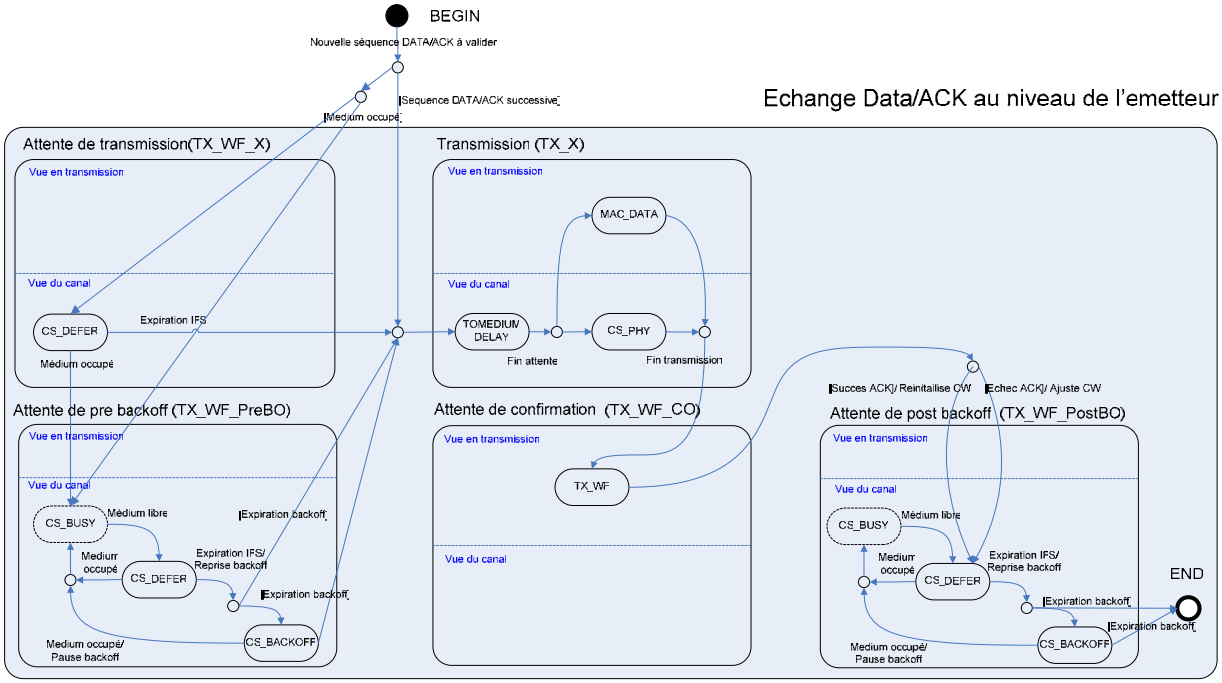


Figure 1.12 Description de l'automate validant la séquence DATA/ACK côté émetteur (émission de Data)

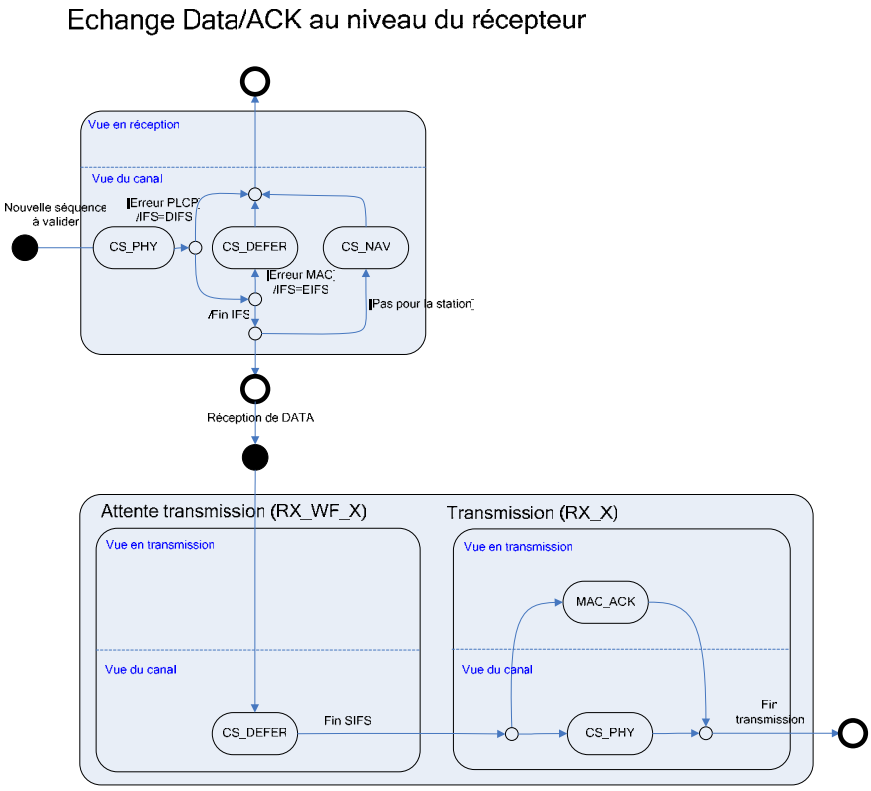


Figure 1.13 Description de l'automate validant la séquence DATA/ACK côté récepteur (émission de ACK)

4.3.2. Le moteur ELYSE

ELYSE (nEutral Langage support for hYper SErtifcation) est une implémentation concrète du système de validation du certifieur. ELYSE se base sur le format SCXML (State Chart XML) [Net-SCXML] et le moteur libre Commons SCXML de Apache [Net-ApacheSCXML]. Ces deux techniques dotent ELYSE des capacités d'expressivité nécessaires pour implémenter les tests conditionnels représentés aux Figures 1.12 et 1.13.

Chaque automate est implémenté dans le certifieur dans le format SCXML. L'implémentation de haut niveau de la séquence DATA/ACK est décrite à la Figure 1.14. Celle-ci indique comment s'enchaînent les appels des différents services de validation en fonction des valeurs retournés par chacun de ces services lors du traitement d'une séquence DATA/ACK. A chaque service est attachée une implémentation dans un fichier SCXML. L'implémentation SCXML de bas niveau du service *TX_WF_X* est donnée à la Figure 1.15.

```
<?xml version="1.0"?>
<scxml xmlns="http://www.w3.org/2005/07/scxml"
       version="1.0"
       xmlns:my="http://my.custom-actions.domain/CUSTOM"
       initialstate="BEGIN">
   <state id="BEGIN" src="BEGIN.xml">
      <transition event="successive" target="TX_X"/>
      <transition event="busy"       target="TX_WF_PreBO"/>
      <transition event="idle"       target="TX_WF_X"/>
   </state>
   <state id="TX_WF_X" src="TX_WF_X.xml">
      <transition event="end"         target="TX_X"/>
      <transition event="interrupted" target="TX_WF_PreBO"/>
   </state>
   <state id="TX_X" src="TX_X.xml">
      <transition event="end" target="TX_WF_CO"/>
   </state>
   <state id="TX_WF_PreBO" src="TX_WF_PreBO.xml">
      <transition event="end" target="TX_X"/>
   </state>
   <state id="TX_WF_CO" src="TX_WF_CO.xml">
      <transition event="end" target="TX_WF_PostBO"/>
   </state>
   <state id="TX_WF_PostBO" src="TX_WF_PostBO.xml">
      <transition event="end" target="END"/>
   </state>
   <state id="END" src="END.xml"/>
</scxml>
```

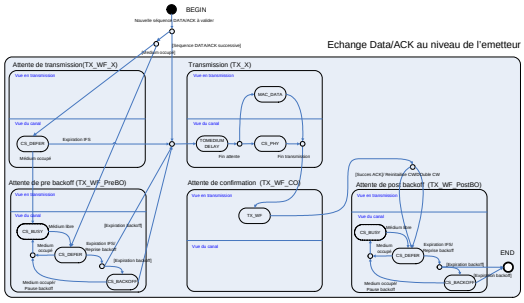
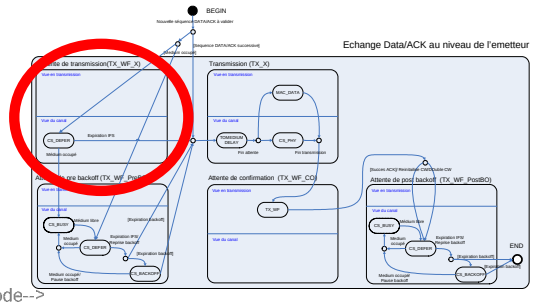


Figure 1.14 Implémentation de la séquence DATA/ACK en émission (haut niveau)




```
<?xml version="1.0"?>
<scxml xmlns="http://www.w3.org/2005/07/scxml"
       version="1.0"
       xmlns:my="http://my.custom-actions.domain/CUSTOM"
       initialstate="CS_DEFER">
   <state id="CS_DEFER">
     <onentry>
       <!--Custom action to extract CS_DEFER parameters from neutral code-->
       <my:extract name="Get CS_DEFER parameters"/>

       <!--Assign CS_DEFER parameters in Apache variables-->
       <var name="instant"  expr="cs_defer.getInstant()"/>
       <var name="duration" expr="cs_defer.getDuration()"/>
       <var name="expired"  expr="cs_defer.getExpired()"/>

       <!--CS_DEFER control procedure-->
       <if cond="instant&lt; 0">
            <log expr="Error: CS_DEFER not found"/>
       <else/>
            <log expr="CS_DEFER found at :'+ instant +', expired:' + expired"/>
             <if cond="expired==1">
                <send event="end"/>
            <else/>
                <send event="interrupted"/>
             </if>
       </if>
     </onentry>
   </state>
</scxml>
```

Figure 1.15 Implémentation de bas niveau de l'état d'attente de transmission (fichier TX_WF_X_implem.xml)

L'interprétation du code neutre est réalisée en deux phases :

1. L'initialisation. Cette phase consiste à :
   - identifier le début d'une nouvelle séquence à traiter
   - reconnaître la séquence de messages à utiliser pour la validation (DATA, DATA/ACK, etc)
2. La validation. Cette phase consiste à :
   - vérifier s'il existe dans l'automate d'état de la séquence une suite de transition générant les mêmes instructions que celles sous test. Les instructions à tester figurent dans l'automate du certifieur sous forme d'états et d'actions. Une partie des instructions neutres des traces matérialise les conditions environnementales (énergie sur le medium, etc) –c'est-à-dire les évènements et les conditions guidant les transitions. Les règles du bas et du haut niveau guident le programme de certification dans l'analyse de la trace.

Pour chaque station à tester, le certifieur instancie en interne une instance de validation contenant les fonctions d'initialisation côté émetteur et récepteur.

4.3.2.1. Propriétés du moteur ELYSE

Le moteur ELYSE est doté des propriétés remarquables suivantes :

- Désynchronisation des opérations côté émetteur et récepteur : l'exécution des instances de validation côté émetteur et récepteur est désynchronisée. Des règles auxiliaires peuvent cependant tester la correction de la synchronisation des stations émettrice et réceptrice.
- Totalement scripté : Toutes les règles de validation sont exprimées en SCXML. Les modules compilés d'ELYSE gèrent simplement le temps et les instances de validation.
- Détection d'erreurs multiples : L'utilisation des deux niveaux permet de continuer l'analyse d'une séquence même après la détection d'une erreur par le bas niveau, en inférant l'état suivant dans le haut niveau.
- Interaction avec le séquenceur : Le certifieur marque les frises temporelles du séquenceur pour permettre à l'utilisateur de visualiser les erreurs.
- Extensibilité : L'extensibilité du modèle se traduit par la possibilité
  - d'ajouter des automates pour chaque séquence de messages spécifiée dans les standards
  - de proposer plusieurs alternatives de validation pour les fonctions sujettes à interprétation (en proposant plusieurs implémentations de services de validation).
  - de définir différents niveaux de certification via l'ajout des tests supplémentaires sur l'exécution de certaines actions lors des transitions. A titre d'exemple, l'automate côté récepteur est capable de tester le problème de resynchronisation d'EIFS découvert dans [BEK07] grâce à l'action *Fin IFS* du premier service de validation (voir Figure 1.13).

4.3.3. Travaux similaires

Les seuls outils de validation formelle disponibles actuellement pour la simulation sont Check and Simulate [SOB04] et Verisim [BHA00]. Check and Simulate [SOB04] s'inspire des techniques de Model Checking pour vérifier la correction du simulateur J-Sim. Les différences principales avec YAVISTA sont

l'expression du modèle en LTL (Linear Temporal Logic) [PNU97], l'intégration du modèle à J-Sim et l'application des concepts à un niveau protocolaire différent (ARQ). Bien que le concept semble prometteur, l'effort de recherche semble essentiellement porté sur l'intégration du système au simulateur.

Verisim [Bha00] a été conçu pour valider le protocole AODV sous NS-2. Les règles de validation sont exprimées en MEDL (Meta Event Description Language), une extension de LTL augmentant l'expressivité du langage originel pour par exemple pouvoir compter les occurrences d'évènements. L'avantage du certifieur YAVISTA sur Verisim est d'exploiter une structure d'automates à deux niveaux permettant de continuer naturellement la validation des traces en cas d'erreur.

De façon générale, notre solution se distingue par son lien naturel avec les spécifications des standards, son indépendance vis-à-vis des simulateurs à tester, l'expressivité du langage neutre et la puissance d'expression de SCXML. ELYSE hérite à ce titre de tous les attributs des approches programmatiques : évaluation des expressions arithmétiques SCXML, utilisation de variables SCXML et modularité SCXML. La modularité d'ELYSE se base à ce titre sur le concept novateur de composition de services de validation.

5. Le Problème de Correction des Simulateurs

5.1. Etat de l'art des caractéristiques de NS-2 et Glomosim

L'approche pédagogique de validation Yavista se base sur la comparaison des frises d'exécution des simulateurs et l'arbitrage par les spécifications du standard 802.11. [BEK07] recense 11 caractéristiques propres à NS-2 et Glomosim. Les caractéristiques se rapportant généralement à des problèmes techniques de très bas niveau (voir annexe A), nous proposons simplement ici un résumé des observations des deux simulateurs. Les caractéristiques observées sur NS-2 et Glomosim sont reportées sur la couronne périphérique du diagramme de la Figure 1.16. Nous classons les caractéristiques en trois types :

- les simplifications : ce sont des choix de conception volontaires visant à la simplification du système à l'étude,
- les bugs : ce sont des défauts logiciels non prévus à la conception,
- les interprétations de la norme : ce sont des points extrêmement sensibles de la norme pouvant faire l'objet de requêtes d'interprétation auprès du comité 802.11 pour obtenir unanimité.

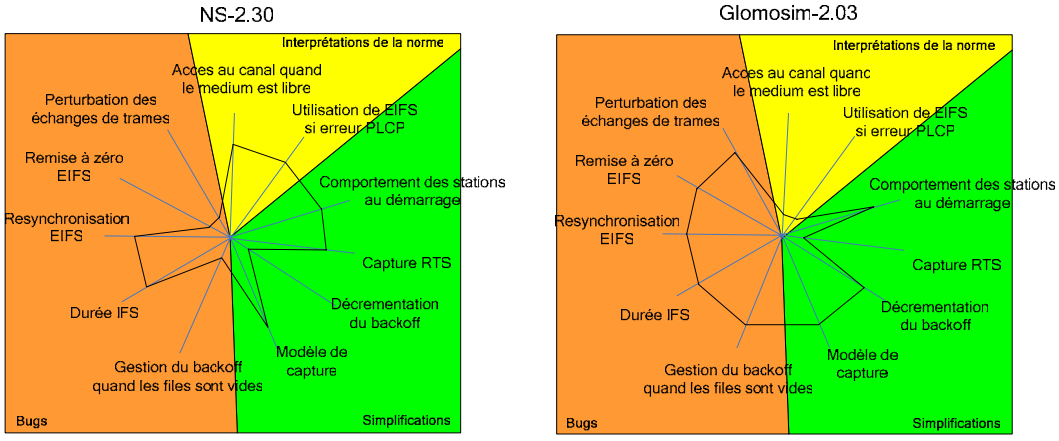


Figure 1.16 Caractéristiques de NS-2.30 et Glomosim 2.03

La Figure 1.16 montre que comparé l'un à l'autre, Glomosim est particulièrement buggé et que NS-2 interprète particulièrement mal la norme.

5.2. Etude de cas : nombre de collisions observé sous NS-2 et Glomosim

Pour illustrer, les possibilités d'audit de Yavista, nous détaillons dans cette section, la caractéristique la plus intéressante des simulateurs, à savoir la *décrémentation du backoff* (voir [BEK07]). L'intérêt de cette caractéristique est sa liaison avec la probabilité de collisions utilisée dans les modèles mathématiques [XIA03] de mesure de la saturation des réseaux 802.11. Ces modèles mathématiques étant validés contre des simulateurs, le choix du simulateur peut donc intervenir indirectement sur ces modèles.

La décrémentation du backoff met en jeu le temps de propagation sur le medium et le temps de passage de l'état de réception à l'état d'émission dans la couche physique. La façon dont un simulateur prend en compte ces éléments peut donner lieu à deux modèles de décrémentation différents : discret comme NS-2 ou continu comme Glomosim.

5.2.1. Sous NS-2

Sous NS-2, lorsqu'une trame est transmise sur le medium, toutes les stations du réseau décrémentent leur backoff du même nombre de slots. La preuve du lemme est géométrique : soit B, la station ayant transmis la dernière trame, C la station ayant transmis l'avant dernière trame et A une station quelconque. Soit N_A et N_B, le nombre de slots de backoff restant de A et B avant la transmission de B ($N_B < N_A$). Les temps de propagation t_{AC}, t_{AB} et t_{BC} entre les différentes stations sont représentés à la Figure 1.17. A, B et C sont chacune à portée respective.

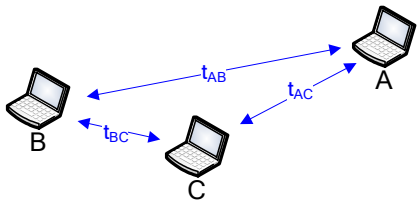


Figure 1.17 Temps de propagation sur le réseau

Un simple raisonnement géométrique montre que si B attend $T_B = N_B * SlotTime$ avant de transmettre, la station A considérera le medium libre durant la période :

$$T_A = N_B * SlotTime + t_{CB} + t_{AB} - t_{AC}$$ [3]

Si $t_{CB} + t_{AB} - t_{AC} < 0$, la station A attend moins de N_B slots. En conséquence, NS-2 décrémente N_A de N_B-1 slots seulement. Sinon, N_A est décrémenté de N_B slots. La preuve du lemme repose sur le fait que $t = t_{CB} + t_{AB} - t_{AC}$ est toujours positif. Malheureusement, lorsque les A, B et C sont alignés, ou lorsque les rôles de B et C sont confondus, le manque de précision de l'unité flottante FPU dans le calcul de T_A à l'interruption du timer de backoff biaise le calcul de t. Dans ce cas, le nombre de slots à décrémenter (N_B-*1* ou N_B) est aléatoire et le nombre de collision/capture sous estimé par rapport à la théorie (en théorie une collision/capture survient si A et B transmettent dans le même slot, c'est à dire si ($|N_A - N_B| = 0$). La Figure 1.18 illustre ce phénomène lorsque A, B et C sont alignés.

[3] NS ne prend pas en compte le temps de passage de l'état de réception à l'état d'émission dans la couche physique

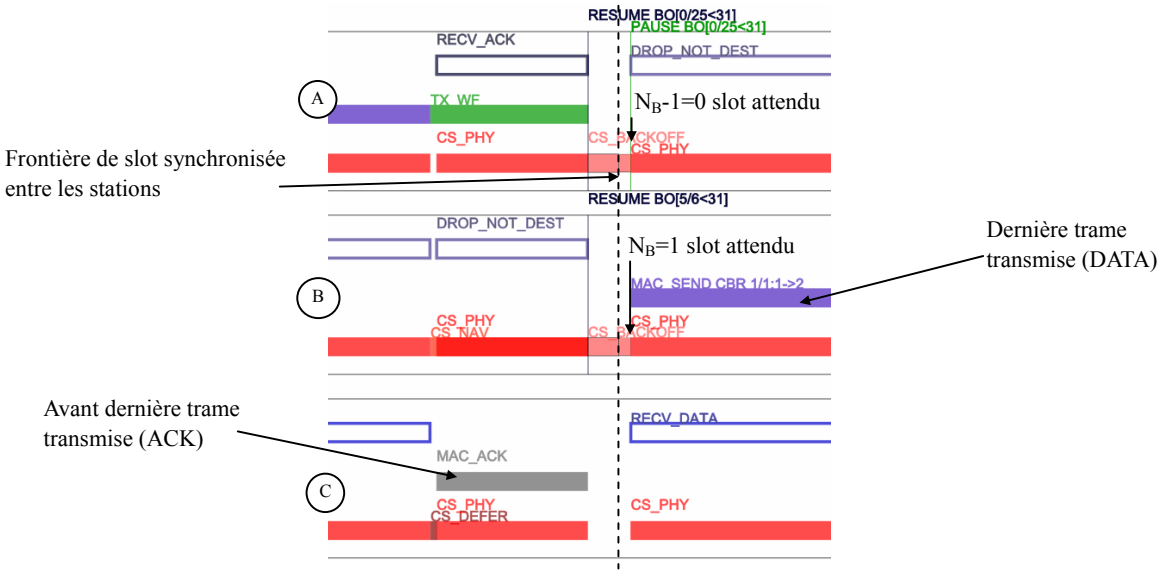


Figure 1.18 Décrémentation du backoff dans une chaîne de trois noeuds sous NS-2

5.2.2. Sous Glomosim

Sous Glomosim, deux éléments majeurs régissent la décrémentation du backoff :

- il existe une période de temps irréversible Δ de 5 µsec entre le moment où une station décide de transmettre un RTS ou un paquet de données[4] et le moment où le premier bit de la trame est effectivement transmis sur le medium
- quand une trame est détectée sur le medium, chaque station décrémente son backoff de la durée pendant laquelle le médium a été perçu libre. Un raisonnement géométrique montre que si la station B attend T_B=S_B avant de transmettre[5], la station A perçoit le medium libre pendant:

$$T_A = S_B + t_{CB} + t_{AB} - t_{AC} + \Delta$$

Sachant que T_A et T_B sont réels, le backoff des stations n'est pas synchronisé et expire à des instants approximativement multiples de Δ. Une collision/capture survient entre les stations A et B si S_A et S_B expirent dans un intervalle de Δ. De fait, Si A et B s'apprêtent à transmettre dans le même slot, la probabilité de collision/capture entre les stations A et B vaut approximativement:

$$P = \frac{2\Delta}{SlotTime} = 0{,}5$$

Le résultat est illustré à la Figure 1.19.

[4] appartenant à une séquence DATA/ACK ou DATA
[5] S_B est la valeur de backoff en seconde

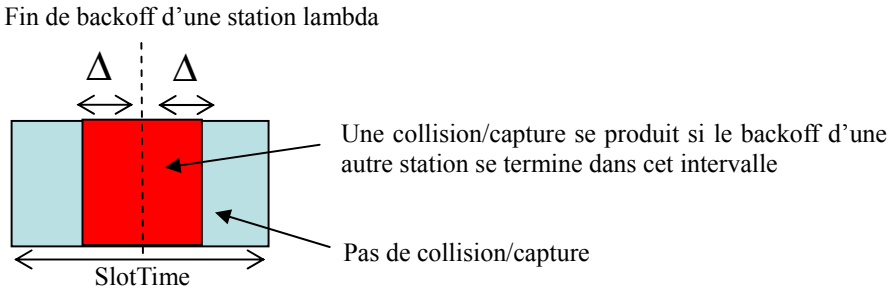


Figure 1.19 Illustration du cas de capture/collision sous Glomosim

A cause de la désynchronisation des backoffs, ce modèle produit en moyenne deux fois moins de collisions/captures que NS-2 quand le medium atteint sa capacité limite. La Figure 1.20 montre l'évolution des backoffs des stations *A*, *B* et *C* entre la transmission d'un ACK et la trame de données suivante. La désynchronisation se manifeste par le fait que les frontières de slots des différentes stations ne correspondent pas exactement.

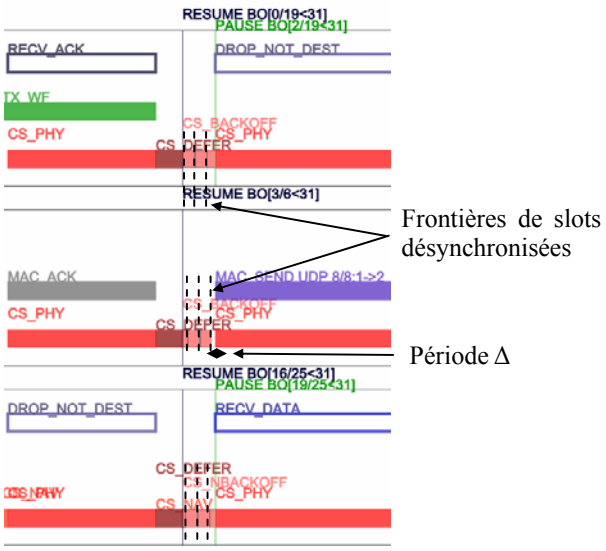


Figure 1.20 Décrémentation du backoff sous Glomosim

La différence théorique de 100% entre NS-2.30 et Glomosim 2.03 est confirmée par les résultats expérimentaux de la Figure 1.21. Les graphes montrent la probabilité de collision des paquets. Ils sont obtenus en moyennant les résultats de 9 simulations (3 topologies différentes[6] pour 3 configurations différentes des générateurs de nombres aléatoires). Une collision est comptée à chaque fois qu'un paquet transmis ne reçoit pas d'acquittement. Les intervalles de confiance sont calculés à 95%. Les Macs PDUs ont pour taille 76 octets. Les résultats théoriques de [TAY01], validés contre le simulateur de Bianchi, sont également reportés sur les figures[7].

[6] Une topologie correspond à un placement aléatoire des nœuds dans une zone à $1m^2$. Du fait de la proximité des stations dans cette scène, le rapport à puissance des paquets transmis simultanément est généralement proche de 1. De sorte, le modèle de capture des deux simulateurs agit de même et conclut à la collision et la perte des deux paquets [Bek07].

[7] La formule de la probabilité de collision utilisée est $p=\frac{1}{2}\left(1+\frac{4}{g}-\sqrt{1+\left(\frac{4}{g}\right)^2}\right)$ où $g=\frac{W}{n-1}$ où W matérialise la taille de la fenêtre de contention et n le nombre de station

Les expérience reproduisent les conditions décrites dans [TAY01]: les noeuds sont tous à portée, il n'y a pas de bruit, le medium atteint sa capacité limite, la charge UDP est équilibrée, les simulations durent 10 secondes. Seules 5 secondes de régime permanent sont conservées.

A la différence de la seconde expérience (Figure 1.21 à droite), la première expérience (Figure 1.21 à gauche) neutralise complètement l'effet du modèle de capture en associant la même puissance à tous les paquets reçus.

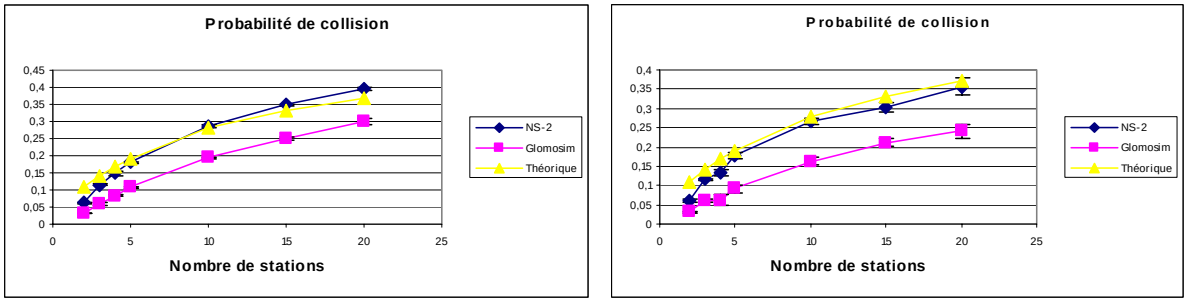


Figure 1.21 Comparaison de la probabilité de collisions des paquets sous NS-2, Glomosim et la théorie. Neutralisation du modèle de capture, dans une zone de $1m^2$ à gauche & modèle de capture actif, dans une zone de $1m^2$ à droite

Un constat intéressant dans les deux cas est la convergence relative du rapport des probabilités de collisions observées sous les deux simulateurs. La raison est que la valeur théorique *P* calculée sous Glomosim, n'est plus complètement valable lorsque le nombre de nœuds, devant théoriquement entrés en collision à la fin d'un backoff, dépasse 2.

6. Conclusion

La suite logicielle YAVISTA propose un système de comparaison et de validation des simulateurs inédit basé sur le concept de code neutre et sur une technique de redressement des modèles de simulation. Le langage neutre fournit une description structurée des communications réseaux et différents niveaux d'analyse des résultats.

Pour les plateformes propriétaires ou spécialisées, l'adoption d'un formalisme standard leur permettrait d'être comparées aux simulateurs historiques. Pour le programmeur, YAVISTA est une réponse adaptée au besoin de validation des protocoles expérimentaux, pour lesquels il n'existe ni modèle analytique, ni modèle de simulation, ni système réel comparable.

Une autre conséquence de l'utilisation d'un tel standard serait l'émergence d'un business model dans le domaine de la validation avec l'apparition d'auditeurs externes spécialisés dans le contrôle des protocoles. La démarche aboutirait à la mise en place de middleend et de frontend complets et à la banalisation des simulateurs.

L'utilisation d'un langage déclaratif standard est justifiée par le fait que les modèles implémentés dans les simulateurs possèdent peu de points communs et qu'il n'existe pas a priori de modèle canon implémentant le standard. Les modèles de simulation n'exhibant pas à ce jour de propriétés descriptives claires – telles qu'il en existe pour les modèles analytiques –, le langage neutre est particulièrement adapté à la description des modèles de simulation.

Techniquement, le certifieur se distingue des autres outils de validation formelle disponibles pour les simulateurs réseaux, Check and Simulate et Verisim, par l'expressivité et la modularité du moteur ELYSE – celles-ci apportant en retour simplicité et efficacité.

Les techniques de développement présentées dans ce chapitre pourraient sembler étrangères au thème du mémoire. Il n'en ai rien : ELYSE est une très bonne introduction au simulateur YAVISTA car il en est pur artefact. Tout deux se basent en effet sur la même découpe des fonctions d'accès au canal. La suite de ce mémoire présente les fondements permettant la compréhension et la modélisation de leur architecture.

Références

[AND06] T.R. Andel, A. Yasinsac, "On the Credibility of Manet Simulations", *Computer*, 39(7), pp. 48-54, Juillet 2006

[BAG99] R. Bagrodia, M. Takai, "Position Paper on Validation of Network Simulation Models", *DARPA/NIST Network Simulation Validation Workshop*, Mai 1999

[BEK07] R. Ben-El-Kezadri, F. Kamoun, "YAVISTA: A Graphical Tool for Comparing MANET Simulators", Journal of Computers (JCP, ISSN1796-203X), en cours de révision

[BHA00] K. Bhargavan, C. A. Gunter, M. Kim, I. Lee, D. Obradovic, O. Sokolsky, M.Viswanathan, "Verisim: Formal analysis of network simulations", dans *Proc. of the 2000 ACM SIGSOFT international symposium on Software testing and analysis (ISSTA'00)*, pp. 2–13, Portland, OR, Etats-Unis , Aôut 2000

[CAV02] D. Cavin, Y. Sasson, A. Schiper, "On the Accuracy of MANET Simulators", dans *Proc. of the 2nd ACM international workshop on Principles of mobile computing (POMC'02)*, pp. 38–43, Toulouse, France, Octobre 2002

[CIC06] C. Cicconetti, E. Mingozzi, G. Stea, "An integrated framework for enabling effective data collection and statistical analysis with ns-2", dans *Proc. of the 2006 Workshop on Ns-2: the IP network simulator (WNS2'06)*, Pise, Italie, Octobre 2006

[DHO03] D. Dhoutaut, Q. Vo, I. Guérin Lassous, "Global visualization of experiments in ad hoc networks" , Rapport Technique 4933, INRIA Rhône-Alpes Research Unit, INRIA, Montbonnot-St-Martin, France, 2003
Disponible :http://lifc.univ-fcomte.fr/~dhoutaut/chronogrammes_ns/ chronogrammes_fr.html

[EST00] D. Estrin, M. Handley, S. McCanne, Y. Xu, H. Yu, "Network Visualization with Nam, the VINT Network Animator", *Computer*, 33(11), pp. 63–68, Novembre 2000
Disponible : www.isi.edu/nsnam/nam

[GAR06] K. Gary, S. Kokoori, B. David, M. Otoom, M. B. Blake, K. Cleary , "An Architecture Validation Toolset for Ensuring Patient Safety in an Open Source Software Toolkit for Image-Guided Surgery Applications", *IJ - 2006 MICCAI Open Science Workshop*, 2006

[HAR87] D. Harel, "Statecharts: A visual formalism for complex systems", *Science of Computer Programming*, 8(3), pp. 231–274, Juin 1987

[KUK05] S. Kurkowski T. Camp, N. Mushell M. Colagrosso, "A Visualization and Analysis Tool for NS-2 Wireless Simulations: iNSpect", dans *Proc. of the 13th Annual Meeting of the IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS'05)*, pp. 503–506, Atlanta, GA, Etats-Unis, Septembre 2005

[NOW03] Nowak K., Małek J., Trace Graph - Data presentation system for Network Simulator NS-2, dans *Proc. of 24th International Systems Architecture and Technology (ISAT'03*), Szklarska Poreba, Pologne, Septembre 2003

Disponible: http://www.tracegraph.com/

[O'M04] J. O'Madadhain, D. Fisher, "JUNG: the Java Universal Networks/Graph API", *UCI ISR Research Forum*, Poster Session, Irvine, CA, Etats-Unis, Juin 2004

[PNU97] A. Pnueli, "The Temporal Logic of Programs", Rapport Technique CS97-14, Weizmann Institute Of Science, Israel, 1997

[RED06] D. Reddy, G. F. Riley, B. Larish, Y. Chen, "Measuring and Explaining Differences in Wireless Simulation Models", dans *Proc. of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation (MASCOTS'06),* pp. 275–282, Monterey, CA, Etats-Unis, Septembre 2006

[SOB04] A. Sobeih, M. Viswanathan, J.C Hou, "Check and simulate: a case for incorporating model checking in network simulation", dans *Proc of Second ACM and IEEE International Conference on Formal Methods and Models for Co-Design, 2004. (MEMOCODE'04)*, pp. 25–37, San Diego, CA, Etats-Unis; Juin 2004

[SAR98] R. G. Sargent, "Verifying and validating simulation models", dans *Proc. of the 30th Winter Simulation Conference (WSC'98)*, pp. 121–130, Washington, DC, Etats-Unis, Decembre 1998

[TAK01] M. Takai, J. Martin, and R. Bagrodia, "Effects of wireless physical layer modeling in mobile ad hoc networks", dans *Proc. of the 2nd ACM international symposium on Mobile ad hoc networking & computing (MobiHoc '01)*, pp. 87–94, New York, NY, Etats-Unis, Octobre 2001

[TAY01] Y. C. Tay, K. C. Chua, "A capacity analysis for the IEEE 802.11 MAC protocol", Wireless Networks, 7 (2), pp. 159 – 171, Kluwer Academic Publishers Hingham, MA, Etats-Unis, Mars/Avril 2001

[XIA03] Y. Xiao, J. Rosdahl, "Performance analysis and enhancement for the current and future IEEE 802.11 MAC protocols", *ACM SIGMOBILE Mobile Computing and Communications Review*, Special Issue: Wireless home networks, 7(2), pp. 6–19, Avril 2003

Netographie

[Net-NS-2] NS-2.29, Disponible: http://www.isi.edu/nsnam
Dernière visite le 31/08/06

[Net-Glomosim] Glomosim 2.03, Disponible: http://pcl.cs.ucla.edu/projects/glomosim/
Dernière visite le 31/08/06

[Net-Yavista] The YAVISTA framework, Disponible: yavista.sourceforge.net
Dernière visite le 15 juin 2007

[Net-Inkscape] Inkscape, Disponible: http://www.inkscape.org/
Dernière visite le 15 juin 2007

[Net-ApacheSCXML] Apache SCXML Commons,
Disponible: http://jakarta.apache.org/commons/scxml/index.html
Dernière visite le 15 juin 2007

[Net-SCXML] SCXML, State Chart XML (SCXML): State Machine Notation for Control Abstraction, W3C Working Draft 21 February 2007, 2007
Disponible: http://www.w3.org/TR/scxml/
Dernière visite le 15 juin 2007

Chapitre 2

Flexibilité et Réutilisabilité des Systèmes

1. Introduction

Dans le domaine de la simulation, les implémentations actuelles d'interfaces sans fil présentent une flexibilité et une réutilisabilité relativement faibles.

En terme de flexibilité, l'ajout ou la modification de certaines fonctionnalités comme les fonctions de management, l'insertion d'une fonction ARQ ou de fragmentation peuvent remettre en cause la totalité de conception la couche MAC.

En terme de réutilisabilité, les évolutions de protocoles sont généralement réalisées par copie de fragment de code ou par héritage. La réutilisation se limite souvent à la récupération des modules de file d'attente.

La raison est que la couche liaison des simulateurs réseaux traditionnels (NS-2, Glomosim, J-Sim) est développée dans une optique monolithique. Les simulateurs présentent en effet une architecture classique en couche/sous couche, chaque sous couche étant implémentée dans un fichier (cf chapitre 3).

Flexibilité et réutilisabilité sont deux facteurs de qualité logiciel permettant d'apprécier un logiciel. Ils intéressent en premier lieu les développeurs informatique soucieux de la maintenance et de la révision de leurs produits, les utilisateurs étant préoccupés, pour leur part, par l'efficacité et la correction de leur programme [IEEE90]. La définition des deux facteurs est standardisée par l'IEEE depuis 1990 [IEEE90] :

- La flexibilité est la facilité à modifier un système ou sous système logiciel en vue de l'utiliser dans un environnement pour lequel il n'a pas été conçu spécifiquement.
- La réutilisabilté est le degré auquel un sous système logiciel peut être utilisé dans plus d'un système.

Le problème de la flexibilité et la réutilisabilité est que ces concepts n'étant pas directement matérialisables, il faut, pour pouvoir les mesurer, établir une correspondance associant ces facteurs à des critères de qualité interne. Des correspondances sont à ce titre proposées dans [McC77] et [BOE83] entre autre.

Table 2.1 Modèle d'assurance qualité logicielle de McCall

	Produit en phase opérationnelle					Produit en phase de maintenance			Produit en phase de révision		
Critère de Qualité \ Facteur de qualité	Facilité d'emploi	Intégrité	Efficacité	Correction	Fiabilité	Facilité de maintenance	Testabilité	Flexibilité	Facilité de réutilisation	Portabilité	Interopérabilité
Facilité d'utilisation	★										
Facilité d'apprentissage	★										
Facilité de compréhension	★										
Volume d'E/S	★										
Taux d'E/S	★										
Contrôle d'accès		★									
Surveillance accès		★									
Efficacité de stockage			★								
Efficacité d'exécution			★								
Traçabilité				★							
Complétude				★							
Précision					★						
Tolérance aux fautes					★						
Consistance				★	★	★					
Simplicité					★	★	★				
Concision						★					
Instrumentation							★				
Capacité d'extension								★			
Généralité								★	★		
Auto description						★		★	★	★	
Modularité						★		★	★	★	★
Indépendance machine									★	★	
Indépendance système									★	★	
Communication standardisée											★
Données standardisées											★

Dans cette étude, nous étudions principalement, les critères de généralité et de modularité ceux-ci étant, comme le montre la table de correspondance de Mac Call [McC77] (voir Table 2.1), les principaux contributeurs à la flexibilité et la réutilisabilité d'un simulateur. Au même titre que la flexibilité et la réutilisabilité, la définition de ces critères est standardisée:

- La généralité est définie comme [IEEE90] le degré auquel un système[8] ou un sous système peut réaliser une vaste panoplie de fonctions.
- La modularité [IEEE90] en génie logiciel est définie comme le niveau de décomposition d'un système en sous entités discrètes de sorte qu'un changement apporté à l'une de ces entités a un impact

[8] Ensemble de sous systèmes matériels ou logiciels organisés pour accomplir une fonction spécifique ou un ensemble de fonctions [IEEE90].

minimal sur les autres entités.

Un produit flexible et réutilisable possède souvent plusieurs versions cousines, toutes dérivant d'une même souche de base. La modularité et la généralité sont des concepts ambigus car on les applique autant pour qualifier les produits que la souche dont ils dérivent. La modularité de la souche de base est une question fondamentale, car d'elle, dépend la facilité à dériver les différentes versions du produit. La généralité de la souche de base[9] conditionne, elle, le nombre de variantes qu'il est possible de décliner à partir de la ligne de produits.

Ce chapitre commence de fait par décrire en quoi concerne l'activité de modularisation et comment celle-ci se rapporte aux lignes de produits. Nous présentons ensuite les critères de modularité et de généralité et proposons conjointement des méthodes de mesure à la fois applicables au niveau du produit et de la ligne de produits.

2. Modularisation des Lignes de Produits

2.1. Module, modularité et modularisation

La définition du principe de modularité a évolué avec les méthodes de rationalisation des constructions apparues au début du 20ème siècle. Le concept a d'abord été appliqué à la construction des logements préfabriqués et à l'assemblage d'ensembles tels qu'une cuisine, un salon ou une chambre à coucher. En mécanique, les blocs de construction sont prédéfinis. Ils sont de nature physique et leur interface est définie par la géométrie. L'attribut primitif des assemblages obtenus à partir de ces éléments est la possibilité de créer de la variété en combinant ou en échangeant les blocs constitutifs.

Les chercheurs ont plus tard ajouté une dimension de service fonctionnel aux unités d'assemblage. Ce principe suppose qu'une implémentation soit associée au service à réaliser et que les interactions entre modules soient correctement gérées.

Le module doit posséder à ce titre une part de fonctionnalité suffisante et essentielle à l'intérieur de son produit; une résistance pourrait, sinon, être considérée comme le module d'un ordinateur.

Du fait des interactions entre les modules, la géométrie n'est plus le caractère essentiel des constructions modulaires. En électronique et en thermodynamique, les échanges énergétiques et électriques doivent notamment être considérés. En informatique, le module n'a plus de consistance physique et d'interfaces matérielles.

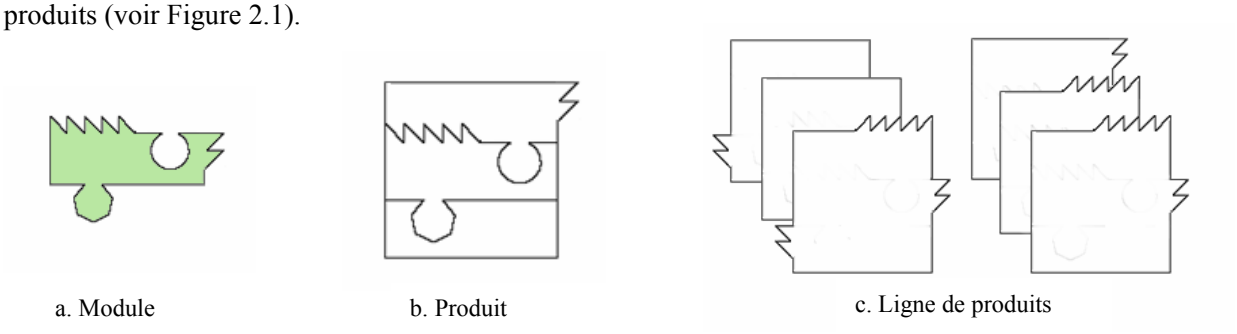
Dans cette nouvelle perspective, les principes de permutation et de combinaison exigent

- Une variété suffisante de modules réalisant la même fonction,
- La spécification d'interfaces et d'interactions standards pour permettre de connecter les différents modules existants.

Pour créer des variantes de produits par combinaison et permutation de modules, le point de vue doit

[9] partant du postulat que cette souche existe

dépasser le cadre du module individuel. Le module est toujours intégré dans un produit et dans une ligne de produits (voir Figure 2.1).



a. Module b. Produit c. Ligne de produits

Figure 2.1 Module, produit et ligne de produits

Indépendamment du domaine d'application, le module, la modularité et la modularisation peuvent se définir ainsi [MIL98]:

Un **module** est une unité essentielle et à fonctionnalité propre relativement à son produit d'appartenance. Par rapport à la définition de sa ligne de produits, un module a des interfaces et des interactions standards permettant les combinaisons et les permutations.

La **modularité** peut être envisagée sous plusieurs points de vue : du point de vue i) du module uniquement, ii) du produit dans lequel il est instancié ou iii) de la ligne de produits dans laquelle il est défini. Pour [MIL98], le principe de modularité est rattaché aux lignes de produits et celui-ci se matérialise par des contraintes strictes sur l'architecture et la répartition des fonctionnalités. Une architecture modulaire est une architecture composée d'unité à fonctionnalité propre (les modules) avec des interfaces et des interactions en accord avec la définition de la ligne de produits. Remplacer un module par un autre aboutit à la création d'une nouvelle variante du produit.

La **modularisation** est l'activité de structuration en modules. La modularisation est une activité découlant de l'analyse d'un domaine d'application donné et aboutissant à la création de ligne de produits.

Ces concepts sont représentés à la Figure 2.2.

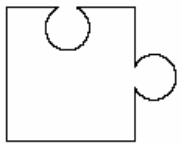
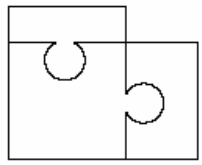
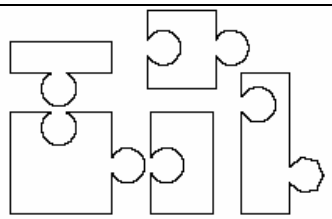
Du point de vue du module	**Du point de vue du produit**	**Du point de vue de la ligne de produit**
		
• Unité fonctionnelle auto contenue	• Unité fonctionnelle auto contenue • Interface et interactions bien définies	• Unité fonctionnelle auto contenue • Interface et interactions standardisées • Combinaison et permutation comme principes pour créer de la variété

Figure 2.2 Principe de modularité du point de vue du module, du produit et de la ligne de produits

2.2. Architecture des lignes de produits

[SVA00] définit une ligne de produits logiciels comme une architecture de ligne de produits munie, d'un ensemble de composants[10], de leurs implémentations et de plusieurs produits logiciels (les variantes).

Une architecture de ligne de produits est une architecture logicielle standard résultant de la modularisation d'un domaine d'application donné. Elle comprend des composants, des connecteurs et certaines contraintes additionnelles (besoins non fonctionnels tels que la qualité de service ou la sécurité, etc). Cette définition se conforme à [BAS98] qui définit l'architecture logicielle d'un système comme le ou les structures d'un système, ceci comprenant des composants[11], les propriétés externes visibles de ces composants (services fournis, besoins non fonctionnels, etc) et leurs relations entre eux.

Une instance de composant (ou module) est développée à partir d'un framework orienté objet constitué de classe abstraites représentant à la fois une abstraction du module et une partie du domaine à implémenter. Le module couche MAC 802.11 dans un simulateur réseau pourrait par exemple être une dérivation d'un framework[12] *sous couche MAC* décrivant les services communs des différents protocoles MAC possiblement implémentable.

Les variantes de produits sont obtenues en fixant les points de variétés ─ c'est-à-dire les endroits dans la conception ou l'implémentation où la variété peut être générée ─ d'une ligne de produits.

Les points de variété peuvent être introduits dans l'architecture via plusieurs mécanismes. En permettant par exemple la substitution d'un module par un autre. Un mécanisme de configuration pourrait ensuite, pour fixer la variété, choisir le module adapté au produit souhaité.

La Figure 2.3 montre comment obtenir de la variété à partir de la composition de frameworks représentant chacun une partie du domaine à implémenter: couche de convergence LLC, file d'attente, couche MAC.

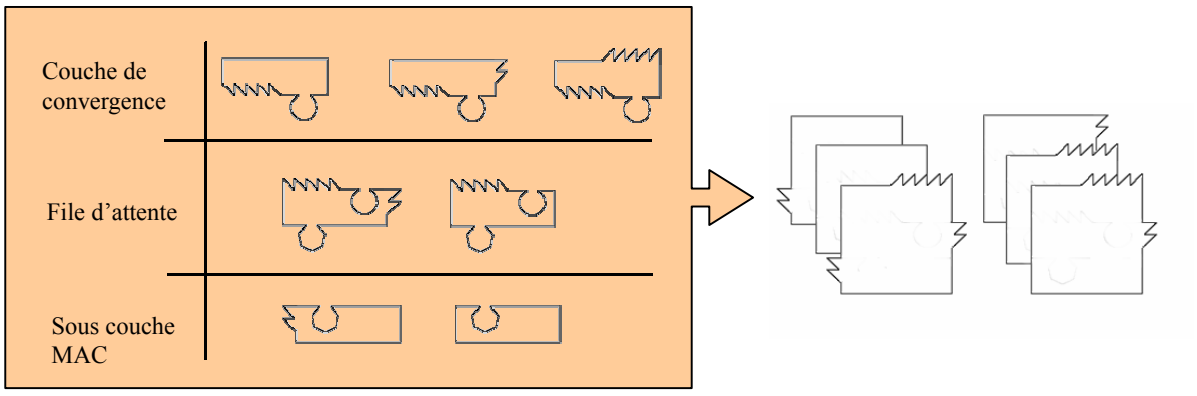


Figure 2.3 Ligne de produits à trois points de variété : couche LLC, file d'attente, sous couche MAC

[11] Du point de vue de l'architecture de la ligne de produits, les détails des composants et en particulier leurs implémentations sont omis. On entend ici par composant la partie visible des instances composants/modules c'est-à-dire leurs interfaces et leurs interactions avec les autres composants.
[12] Du point de vue de l'architecture de la ligne de produits, framework et composant sont synonymes.

Une ligne de produits évolue en fonction des besoins exprimés par les clients [SVA99]. Si une nouvelle famille de produits doit être introduite et que les points de variété introduits dans l'architecture ne permettent pas de satisfaire ces besoins, une nouvelle famille de produits avec une nouvelle architecture doit être créée. Sinon, les produits souhaités sont dérivés en fixant les points de variété existants.

2.3. Principes d'organisation des architectures

La construction des systèmes modulaires se base généralement sur des styles d'architecture. Un style d'architecture est un patron de construction réutilisable permettant de simplifier le développement des applications. Un style d'architecture est déterminé par [BAS98] :

- Un ensemble de type de modules (module de stockage de données, module de calcul) effectuant des fonctions données à l'exécution, etc.
- Une topologie indiquant le positionnement des composants et leurs relations en cours d'exécution.
- Un ensemble de contraintes sémantiques (une base de données accessible en lecture uniquement).
- Un ensemble de connecteurs (appel de sous routine, sockets) qui permettent la communication entre les composants.

Cette section présente les cinq principaux styles de structuration des architectures logicielles : architecture centralisée, architecture en machine virtuelle, architecture à appel et retour, architecture à composants indépendants et architecture par flot de données. Un même système peut utiliser plusieurs styles à la fois. Une partie du système peut par exemple utiliser une architecture centralisée et le reste une architecture en couche. On parle alors d'architectures à style hétérogène [BAS98].

2.3.1. Architecture centralisée

Une structure centralisée met en jeu deux types de composants : une base de données centralisant les informations et une collection de composants qui traitent, stockent et accèdent aux informations centralisées. Chaque client possède son propre processus de contrôle.

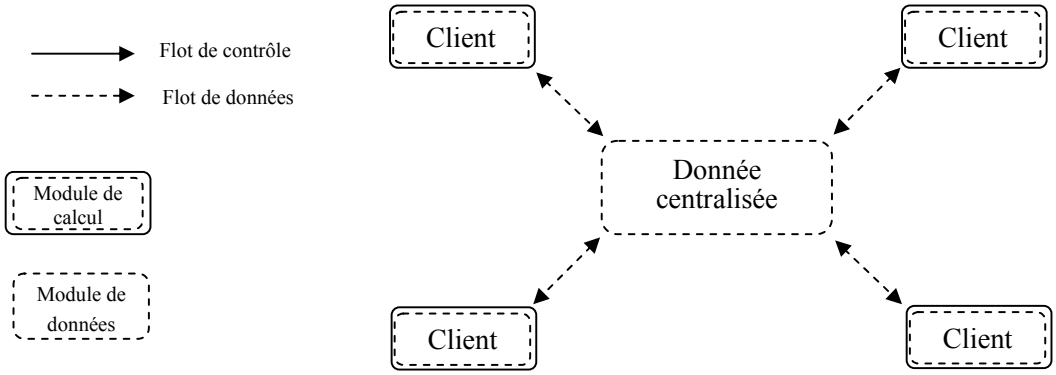


Figure 2.4 Exemple d'architecture centralisée

La Figure 2.4 montre une architecture centralisée avec un composant passif central accessible par plusieurs clients simultanément. Les données (flot de données) transitent entre le composant central et les clients. Le composant central peut également être actif en émettant par exemple des requêtes de notifications (flot de contrôle) vers les clients à chaque fois qu'une donnée est modifiée. Ce type d'architecture est idéal du point de vue de l'intégration de nouveaux clients, ceux-ci étant relativement indépendamment et l'accès aux données étant standardisé.

2.3.2. Architecture en machine virtuelle

La machine virtuelle est constituée d'au moins deux entités : un interpréteur et un programme. L'interpréteur contient un processus de contrôle décodant et exécutant une à une les instructions du programme passif. A chaque instruction du programme correspond une action codée dans l'interpréteur. L'interpréteur *I* peut lui-même être exécuté par un interpréteur *I'*. Dans ce cas les actions codées dans *I* peuvent être modifiées par *I'* ce qui rajoute de l'adaptabilité au système. Ce style offre des avantages de portabilité puisque pour exécuter un programme sur différentes plateformes une traduction de la machine virtuelle suffit.

2.3.3. Architecture appel et retour

C'est le style dominant. Plusieurs déclinaisons existent. Nous citons en particulier :

- *Architecture avec programme principal et sous routines* : Le but est de diviser le programme en petites parties pour permettre son adaptation. Il n'existe généralement qu'un processus de contrôle et chaque routine de la hiérarchie obtient le contrôle (plus des données optionnelles) de la routine appelante puis invoque les routines de niveaux inférieurs (voir Figure 2.5).

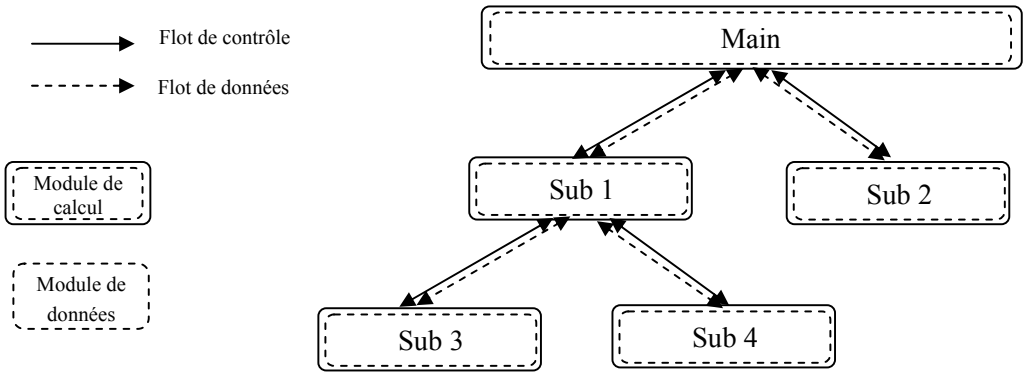


Figure 2.5 Exemple d'architecture avec programme principal et sous routines

- *Système en couches* : Ce style consiste à structurer le système en différentes couches de 0 à *n*. Dans une structuration stricte, chaque couche *i* s'appuie sur celle qui lui est immédiatement inférieure $i-1$ et exploite la notion d'abstraction (voir Figure 2.6). Il en découle des qualités d'adaptabilité et de portabilité. Néanmoins, dans une structuration moins stricte, une couche *i* peut appeler une couche inférieure d'indice quelconque *k*. Ceci permet d'améliorer les performances mais nuit généralement à la portabilité du système.

  Les systèmes d'exploitation et le modèle OSI sont basés sur une architecture en couches.

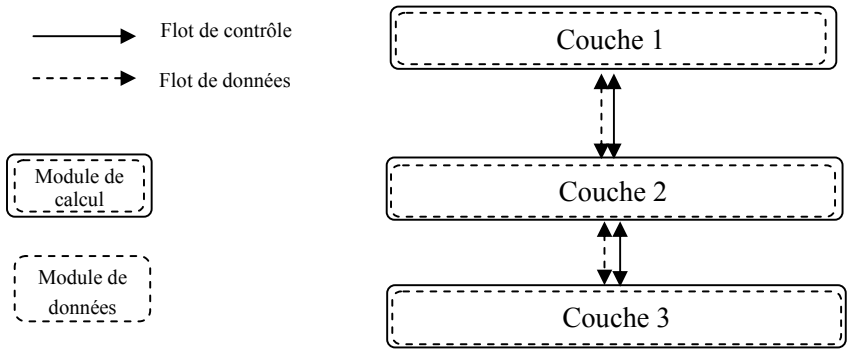


Figure 2.6 Exemple d'architecture en couches

- *Système orienté objet* : C'est la version moderne du style appel et retour. Le paradigme objet se base sur la séparation des concepts entre les données privées de l'objet et les primitives d'accès publiques aux données. L'objet encapsule le secret des opérations et seule l'interface publique de l'objet est accessible au reste de l'environnement (voir Figure 2.7). Ceci augmente la réutilisation des modules objets et l'adaptabilité du système. Lorsque les objets s'assimilent à des boites noires fournissant des services à d'autres objets, on parle alors d'architecture client serveur basée sur appels (par opposition aux architectures client serveur à processus indépendants).

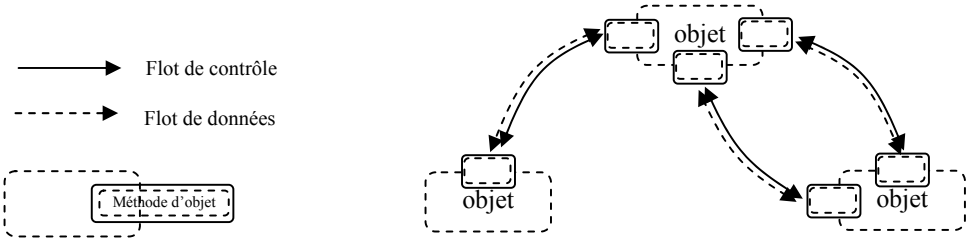


Figure 2.7 Exemple d'architecture orientée objet

2.3.4. Architecture à composants indépendants

Ces architectures consistent d'un grand nombre de processus indépendants possédant leur propre processus de contrôle. Les modules s'échangent des messages mais n'ont pas de contrôle direct les uns sur les autres.

- Dans les *systèmes à événements* certains composants diffusent des événements. Les composants du système souhaitant réagir à ces événements peuvent s'enregistrer auprès de ces sources. La différence fondamentale avec la structuration orientée objet est qu'un composant ne connaît pas à priori les autres composants pouvant être affectés par ses événements et n'agit pas directement sur lui pour lui passer l'information.
- L'autre sous style de cette classe est appelé *processus communicants*. Ce sont des systèmes classiques à base de plusieurs processeurs. Les processeurs étant généralement répartis sur le réseau, ce style offre des propriétés de mise à l'échelle intéressante. Les systèmes client serveur sont un exemple bien connu de ce type de système.

2.3.5. Architecture par flot de données

L'*architecture par flot de données* est une variation du style orienté objet. Elle est utilisée lorsque chaque composant possède un ensemble d'entrées et de sorties et qu'un problème peut être découpé en tâches séquentielles indépendantes. Pour qu'un système puisse être ainsi composé, le type des données supportées en entrée et en sortie doit être unifié. Une variante très connue de ce style est l'architecture *pipe-and-filter*. Les modules de traitement sont appelés *filtre*. Un filtre lit le flot de données sur ses entrées, effectue un traitement local, puis produit un flot de données sur ses sorties. L'avantage de cette technique réside dans la simplicité de composition des filtres. De plus, du fait du découplage total des filtres, la maintenance de ces systèmes est très simple et la réutilisabilité des filtres importante. Le terme *pipe* se réfère à la capacité des nœuds de transformation à s'exécuter en parallèle.
Le graphe résultant d'une telle composition peut être une chaîne simple, un arbre (voir Figure 2.8) ou même un graphe. L'écriture des commandes UNIX se base sur ce principe. Le type des données circulant dans un pipe UNIX est la ligne de texte.

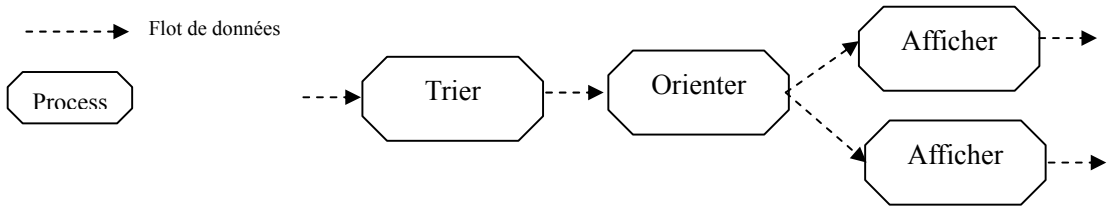


Figure 2.8 Exemple d'architecture par flot de donnée

3. La Modularité Logicielle

En génie logiciel, la modularité est historiquement rattachée à la notion de produit et non à la ligne de produits. La modularisation d'un logiciel obéit à une stratégie 'diviser pour régner'. Ce principe simplifie l'implémentation et permet ainsi de traiter des problèmes plus complexes, c'est-à-dire nécessitant un nombre toujours plus grand d'opérations élémentaires.

3.1. Définition du module

Un module est une entité résultant de la modularisation d'un problème d'ingénierie logicielle. Ces entités s'expriment sous la forme d'une fonction, d'un groupe de fonctions, d'une classe ou d'un composant.

Les premiers travaux relatifs à la modularité logicielle datent des années 70 [PAR72a], [PAR72b], [STE74]. [YOU79] définit alors un module dans un programme comme une suite d'instructions, délimitées par des éléments frontières plus ou moins opaques et possédant un identifiant agrégé (nom de fichier, de fonction, de classe). Le terme encapsulation désigne le regroupement de ces instructions dans les éléments frontières. La littérature ne fournit pas pour autant de consensus réel autour de la notion de module, car celle-ci étant exploitée dans de nombreux langages de programmation, des définitions propriétaires spécifiques à chaque langage sont apparues [SZY02].

Les différents éléments d'une structure modulaire, exhibent des relations qui peuvent être classées en deux types : relations de couplage et de cohésion.

3.1.1. Cohésion

Yourdon et Constantine [YOU79] définissent la cohésion d'un module comme le degré de participation des divers éléments d'un module dans l'exécution d'une tâche.

[MYE78] propose sept degrés de cohésion pour définir les différents niveaux de participation possible entre deux éléments *x* et *y* d'un module. Ceux-ci sont représentés dans la Table 2.2 en allant du degré le plus faible au plus fort.

Table 2.2 Catégories de cohésion

Niveau de cohésion	Description de la relation entre les éléments	Qualificatif de la relation
0	Les éléments du module exécutent des fonctionnalités différentes sans aucun lien entre elles	Cohésion de coïncidence (Pas de cohésion)
1	Les éléments du module exécutent des fonctionnalités différentes mais de même type (gestion des entrées/sorties ou gestion des fautes)	Cohésion logique
2	Les éléments du module exécutent des fonctionnalités différentes mais à des instants proches	Cohésion temporelle
3	Les éléments du module exécutent des fonctions différentes mais dans un ordre bien spécifié	Cohésion procédurale
4	Les éléments du module exécutent des fonctionnalités différentes mais accèdent aux mêmes données	Cohésion communicationnelle
5	Les éléments du module exécutent des fonctionnalités différentes mais les résultats produits par les uns sont utilisés en entrée par les autres	Cohésion séquentielle
6	Les éléments du module participent tous à la même fonctionnalité	Cohésion fonctionnelle

3.1.2. Couplage

Yourdon et Constantine [YOU79] définissent le couplage comme le degré d'interdépendance entre les modules. Comme pour la cohésion, [MYE78] définit plusieurs niveaux de couplage pour décrire les différents types de relations possibles entretenues par une paire de modules *A B*. La Table 2.3 présente les sept niveaux de couplage proposés en les classant du degré le plus faible au plus fort.

Table 2.3 Catégories de couplage

Niveau de couplage	Description de la relation entre les modules	Qualificatif de la relation
0	A et B sont totalement indépendants	Pas de couplage
1	A et B s'échangent des paramètres scalaires	Couplage de données
2	A et B s'échangent des paramètres de type 'enregistrement'	Couplage de signature
3	A passe des paramètres de contrôle à B	Couplage de contrôle
4	A et B communiquent via un medium externe tel qu'un fichier	Couplage externe
5	A et B référencent les mêmes variables globales	Couplage commun
6	A modifie des données ou des instructions dans B	Couplage de contenu

Bien que très répandue, l'approche de Yourdon et Constantine présente un certain nombre d'inconvénients limitant son automatisation et sa généralisation [FEN96] :

- Le champ d'action de la méthode se limite principalement à l'étude des langages procéduraux. En effet selon cette méthode, les types abstraits de données et les objets [BUD02] dans les langages modulaires exhibent un niveau de cohésion intermédiaire (cohésion communicationnelle). D'autres

approches [MAC87] associent au contraire à ce type d'entité le niveau de cohésion maximal. Le même problème se pose aux classes dans les langages orientés objets. Une extension de cette approche adaptée à ce type de langage est proposée par Eder et al. dans [EDE94].

- A l'intérieur d'un module, des groupes d'éléments peuvent séparément montrer une cohésion temporelle et communicationnelle. Dans ce cas, on attribue de coutume le caractère le moins désirable au module (cohésion temporelle). Les nuances sont ainsi difficilement prises en compte.
- La métrique de cohésion définie par Myers est sujette au critère de subjectivité puisqu'elle dépend de la fonctionnalité initialement affectée au module et de son expression par le développeur.

Vu ces inconvénients, l'approche de Yourdon et Constantine sert d'avantage de règle pour guider la conception et illustrer les principes de modules et de modularité que pour mesurer une architecture logicielle. Voila qui explique pourquoi nous l'introduisons ici et non dans la section de mesure de la modularité à venir.

3.2. Implémentation d'un module

Un module peut être implémenté aux moyens de diverses techniques : regroupement de procédures dans les langage procéduraux, module intrinsèque dans les langage modulaires, objet ou regroupement d'objets dans les langages orientés objets ou composants logiciel avec les méthodes de développement orientées composants. Chacune de ces techniques impose au code source des contraintes particulières favorisant ou non le critère de modularité. La plupart des langages sont laxistes dans le sens où ils autorisent le programmeur d'un langage structuré comme Pascal à utiliser des instructions de type GOTO, BREAK et EXIT permettant d'interrompre un traitement pour effectuer un branchement à un autre point du programme ceci multipliant de façon générale les points d'entrée et de sortie dans le programme.

La modularité est un critère semi interne: bien qu'elle ne dépende ni de l'observateur ni de la machine, elle reste étroitement liée au style de programmation utilisé (langage objet, procédural, etc). A titre d'exemple, les langages modulaires et orientés objets mettent au premier plan les techniques d'abstraction [BUD02] permettant de cacher certaines informations d'implémentation au reste du système et de ne divulguer que le nécessaire. Le module possède alors la faculté de diviser un espace de nommage en deux parties. Une partie publique (fonctions) accessible de la totalité du système et une partie privée (fonctions et données) accessible depuis le module seulement. Les systèmes développés dans le respect de ces concepts présentent généralement une modularité supérieure.

3.3. Mesures de modularité et de structure

Du fait des limites des travaux de Yourdon et Constantine, des approches rigoureuses ([CHI91], CHI94], [HIT95], [BIE95], [HEN96], [LEE95], [BRI93] et BRI94]) ont été proposées pour formaliser les métriques originelles, et ce, particulièrement dans le cadre des systèmes orientés objets. Plusieurs analyses montrent l'existence de liens entre les facteurs de qualité observables et les métriques de critères de qualité interne proposées [BAR99].

La structure d'un programme peut être capturée à l'aide des métriques du critère de structure. En fonction des modèles de qualité logicielle – certains parlant de structure plutôt que de modularité – une distinction peut être établie entre ces deux critères.

- Le critère de modularité se base sur une analyse individuelle ou par paire des modules. Il est mesuré à l'aide des métriques de couplage et de cohésion. La définition de ces métriques varie selon le style de programmation. On s'accorde néanmoins à dire que la modularité est optimale lorsque :
  - le couplage entre les modules est minimal,
  - la cohésion à l'intérieur des modules est maximale.

  Ces métriques s'appliquant individuellement sur chaque module ou couple de modules, la généralisation aux produits et à la ligne de produits sous jacente peut être obtenue en moyennant les différentes valeurs obtenues.
- Le critère de structure d'une architecture inclut outre les métriques mesurant la modularité, d'autres métriques permettant l'analyse de la structure des flots de contrôle (mesures de complexité) et de données (mesures de morphologie, d'impureté de l'arbre et de flots d'information).

Nous présentons en premier lieu les métriques de structure, la mesure des flots de données permettant d'évaluer la conception dans son ensemble. Nous présentons ensuite les métriques de modularité utilisées pour évaluer la pertinence d'une composition modulaire[13].

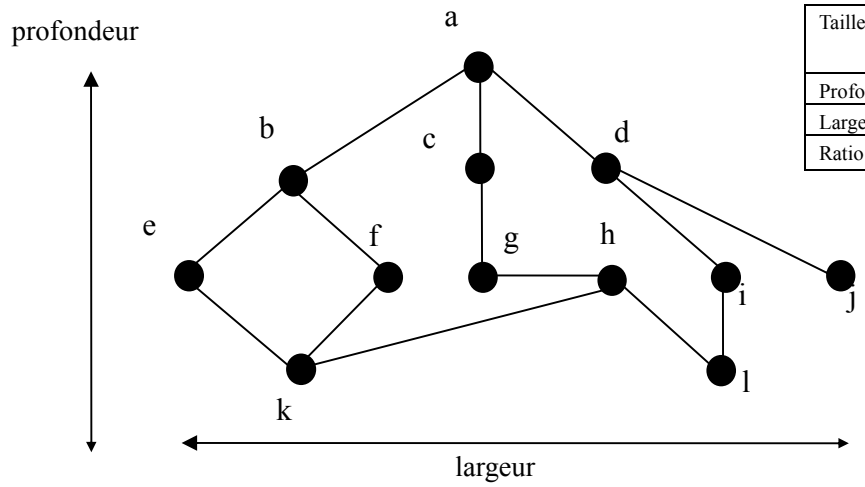
3.3.1. Métriques de morphologie

La morphologie se rapporte à la forme du graphe de dépendances des modules schématisant l'organisation d'un programme. Un arc relie deux modules si l'un fait appel ou passe des données à l'autre. Il existe plusieurs métriques de morphologie.

- La taille représente le nombre total de modules ou d'arcs représentés ou la somme des deux.
- La profondeur représente le plus long chemin existant entre le module de plus haut niveau du programme et une feuille.
- La largeur représente le nombre maximal de nœuds présents à un niveau du programme.
- Le ratio arc à nœud représente la densité du programme en terme de connectivité.

Un exemple est fourni à la Figure 2.9.

[13] Pour plus d'informations sur les différentes métriques de modularité disponibles, le lecteur pourra se référer à [BRI96] et [BRI97].



Taille en arcs	15
en noeuds	12
Profondeur	3
Largeur	6
Ratio arcs à noeuds	1.25

Figure 2.9 Mesures de la morphologie. A gauche le graphe de dépendances du programme, à droite les métriques de morphologie.

Certains graphes cycliques dans les architectures orientées objets peuvent prêter à de multiples représentations, chacune correspondant à un point de vue. La mesure de la largeur et de la profondeur du système n'est alors plus possible. Une solution consiste à abstraire dans des modules composites de haut niveau les modules fortement couplés entre eux et prendre les mesures sur le graphe abstrait de haut niveau.

3.3.2. Métriques d'impureté de l'arbre

La qualité de la conception d'un programme peut être mesurée à partir de la forme de son graphe de dépendances. Une dépendance entre deux modules est marquée par un arc. Un graphe de dépendances dont la forme est proche d'un arbre (absence de boucle) indique une qualité d'architecture supérieure.
La métrique d'impureté de l'arbre *m(G)* mesure la distance entre le graphe d'une architecture et celle d'un arbre. La formule est donnée par :

$$m(G) = 2\frac{(e-n+1)}{(n-1)(n-2)}$$

Avec :

e = nombre de connexions

n = nombre de noeuds

m(G) varie entre 0 et 1. Plus la valeur de *m(G)* est proche de 0, plus la conception du système est bonne. Une valeur de 1 indique un graphe complet. La valeur d'impureté du graphe de la Figure 2.9 vaut 0.088.

3.3.3. Métriques de flots d'information

Un programme peut être représenté par son graphe d'appel des modules. Le graphe d'appel possède un noeud racine correspondant au nœud de plus haut niveau et représentant une abstraction de l'ensemble du système.

Henry et Kafura [HEN81] puis Shepperd [SHE93] ont proposé une approche pour mesurer le niveau total d'information échangée entre les modules individuels et le reste du système lors des appels de modules. La métrique repose sur la définition de flots locaux directs et indirects et de flots globaux.
Un flot local direct existe dans les deux cas suivants :

1. un module en invoque un autre et lui passe de l'information,
2. le module invoqué retourne un résultat au module appelant.

Un flot local indirect existe si les informations retournées par le module appelé sont ensuite transmises à un module tiers.
Un flot global existe si de l'information est échangée entre deux modules via une structure globale de données. Nous pouvons alors définir le fan-in et le fan-out d'un module.

- Le fan-in d'un module *M* est le nombre de flots locaux se terminant en *M* plus le nombre de structures globales de données récupérées par *M*.
- Le fan-out d'un module *M* est le nombre de flots locaux émanant de *M* plus le nombre de structures globales de données mises à jour par *M*.

Plus généralement, le fan-in d'un module *M* se comprend comme le nombre de modules qui utilisent *M* et le fan-out de *M* comme le nombre de modules qu'utilise *M*. Une bonne conception maximise le fan-in pour factoriser le code dans des modules réutilisables et maintient le fan-out sous la valeur sept [ESA95].
La mesure de flot d'information *IF4* se calcule alors comme :

$$IF4(M) = (fan\text{-}in(M) * fan\text{-}out(M))^2$$

3.3.4. Métriques de complexité

La mesure de complexité cyclomatique de McCabe capture la complexité du graphe de flot de contrôle d'un programme. Un graphe de flot *F* est une représentation des différents branchements et boucles d'un programme à l'aide d'arcs orientés (branchements) et de nœuds (instructions). La complexité cyclomatique mesure le nombre de chemins indépendants de *F*. Si *F* contient *e* arcs et *n* nœuds, la complexité *v(F)* vaut :

$$v(F) = e - n + 2$$

Un exemple est fourni à la Figure 2.10. La complexité cyclomatique est essentiellement utilisée pour mesurer le facteur de qualité de maintenabilité et plus particulièrement la testabilité. Au dessus de 10, *v(F)* indique un composant extrêmement difficile à maintenir et tester.

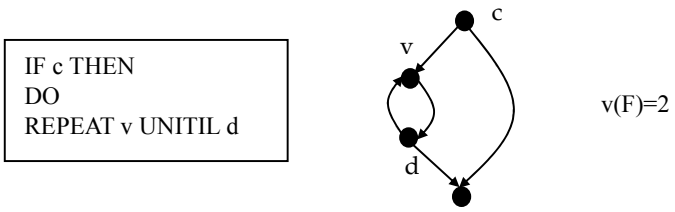


Figure 2.10 Complexité cyclomatique d'un graphe de flot. A gauche le programme original, au centre le graphe de flot, à droite la mesure de complexité

3.3.5. Métriques de cohésion

Métrique proposée par Chidamber et Kemerer [CHI91], [CHI94] : La métrique *LCOM2* (Lack of Cohesion in Methods) est définie comme le nombre de paires de méthodes n'ayant aucun attribut commun (*P*) moins le nombre de paires de méthodes similaires (*Q*). Deux méthodes sont similaires si elles utilisent des attributs de la classe en commun. Du fait de la soustraction de *Q*, *LCOM2* est une mesure inverse de la cohésion, d'où son nom.

$$LCOM2 = \begin{cases} P-Q & si\ P>Q \\ 0 & sinon \end{cases}$$

Métrique proposée par Bieman et Kang [BIE95] : La métrique *TCC* (Tight Class Cohesion) est sensée résoudre les faiblesses de *LCOM2*[14] [BRI97]. *TCC* mesure le pourcentage des paires de méthodes publiques connectées dans une classe. Deux méthodes m_1 et m_2 sont connectées et on note $cau(m_1, m_2)$ dans les deux cas suivants :

- m_1 et m_2 sont similaires (elles utilisent des attributs de la classe en commun)
- une méthode invoque l'autre directement ou indirectement via l'appel d'autres fonctions

$$TCC(c) = \frac{\left|\{m1, m2\}\right| \ | \ m_1, m_2 \in M_I(c) \cap M_{pub}(c) \wedge m_1 \neq m_2 \wedge cau(m_1, m_2)}{\left(\left| M_I(c) \cap M_{pub}(c) \right| \left| M_I(c) \cap M_{pub}(c) - 1 \right| / 2 \right)}$$

avec

$M_I(c)$ ensemble des méthodes implémentées dans la classe *c*

$M_{pub}(c)$ ensemble des méthodes publiques de la classe *c*

Le numérateur compte le nombre de connexions effectives entre les méthodes de la classe et le dénominateur le nombre maximal de connexions possibles dans la classe.

3.3.6. Métriques de couplage

La métrique CBO (Coupling Between Object) [CHI94] indique le nombre de collaborations d'une classe avec une autre. Une collaboration peut être une relation d'héritage, la définition d'un attribut, l'affectation d'un attribut, un appel de méthode ou un passage de paramètre ou un type de retour dans la signature d'une fonction.

3.3.7. Métriques de modularité générale

Ces mesures intuitives proposées par [HAU89] calculent les ratios :

$$M1 = \frac{\text{nombre de modules}}{\text{nombre de procédures}}$$

$$M2 = \frac{\text{nombre de modules}}{\text{nombre de variables}}$$

[14] Des classes possédant des structures similaires peuvent exhiber avec LCOM2 des résultats différents

4. La Généralité

La généralité est réalisée en introduisant des points de variété dans la ligne de produits. C'est en dérivant ces points de variété vers des implémentations particulières que sont réalisés les différents produits de la ligne de produits.

Les points communs et les points de variations entre les différents produits d'un même domaine peuvent s'exprimer en terme de caractéristiques [KAN90]. Une caractéristique est un aspect, une qualité ou une fonction importante directement visible pour l'utilisateur ou les autres systèmes interagissant avec le produit. Une caractéristique peut être composée d'autres caractéristiques. A titre d'exemple, les *fonctions de management* sous 802.11 peuvent être décomposées en trois : *fonction de synchronisation*, *fonction d'authentification* et *fonction d'association*.

4.1. Les caractéristiques

Bien que les produits d'un même domaine partagent ensemble un grand nombre de caractéristiques, ils possèdent tous des caractéristiques spécifiques les rendant différents les uns des autres. Ils existent plusieurs types de caractéristiques [KAN90], [VAN01].

- les caractéristiques obligatoires : elles existent dans tous les produits
- les caractéristiques optionnelles : elles ne sont pas présentes dans tous les produits
- les caractéristiques alternatives : elles spécialisent une caractéristique plus générale

Le modèle des caractéristiques capture les informations pertinentes pour l'utilisateur telles que les services fournis par l'application et des informations objectives permettant de guider la sélection du produit (voir Figure 2.11).

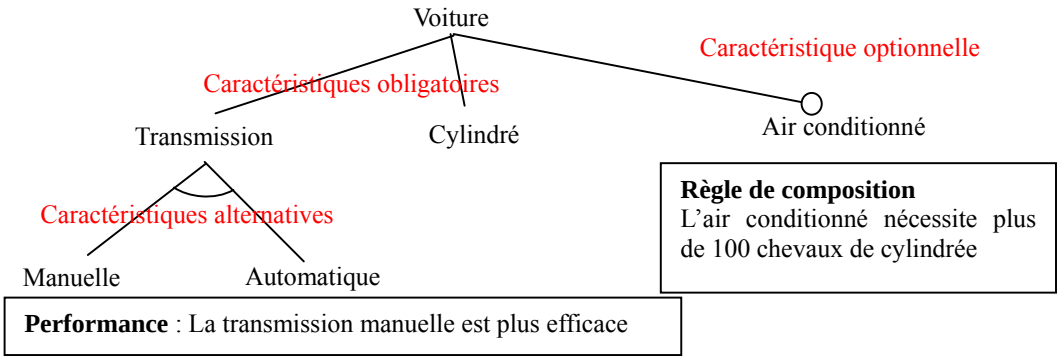


Figure 2.11 Exemple montrant les caractéristiques pertinentes d'une automobile

La liste complète des caractéristiques souhaitées de la ligne de produits est obtenue après étude du domaine en question c'est-à-dire de la documentation et des diverses implémentations existantes.

L'intérêt d'introduire une caractéristique optionnelle ou alternative dans un produit est double:

1. Retarder à plus tard les prises de décisions relatives aux besoins spécifiques de chaque produit. Dans ce cas, les besoins sont connus à l'avance.
2. Prendre en compte des besoins non prévus initialement. Ces besoins peuvent correspondre aux nécessités de [KET04] :

- faire évoluer le logiciel en ajoutant de nouveaux services
- parfaire l'implémentation pour améliorer ses performances
- corriger le logiciel si certaines de ses parties contiennent des défauts
- adapter le logiciel à son environnement.

Les deux cas exigent le développement d'une souche logicielle générique adaptable en différents produits.

4.2. Introduction et fixation de la variabilité dans les lignes de produits

Les points de variété représentent les endroits du programme où les décisions de conception sont prises. Aux ensembles de points de variété sont attachés des ensembles de caractéristiques variantes pouvant les fixer. La fixation peut intervenir :

- Initialement, lors de la dérivation de l'architecture d'un produit : Il s'agit ici de fixer un ensemble de points de variété avec une caractéristique variante au moment où l'architecture du produit est dérivée de l'architecture générale de la ligne de produit
- A la compilation : on cherche ici à se débarrasser des fragments de code source non utilisés dans le produit final
- A l'édition des liens : la phase d'édition des liens intervient entre les phases de compilation et d'exécution. La variabilité permet de diminuer la taille de l'exécutable en sélectionnant les fragments de code binaire à utiliser dans le produit.
- A l'exécution : les points de variété sont fixés interactivement par l'utilisateur ou automatiquement par le logiciel. Cette technique est moins performante en terme de vitesse d'exécution mais elle permet en revanche au logiciel de s'adapter en temps réel à son environnement.

Chaque étape dispose de mécanismes propres pour fixer la variété. [SVA05] propose une vue générale de ces différentes techniques (voir Table 2.4).

Table 2.4 Techniques de fixation de la variabilité dans les lignes de produits

Entités logicielles impliquées	Dérivation de l'architecture du produit	Compilation	Edition des liens	Exécution
Framework	1. Réorganisation de l'architecture 2.Framework multiple 3.Framework optionnel		8. Remplacement binaire par directives d'édition des liens 9. Remplacement binaire physique	10. Architecture centrée infrastructure
Composants	4. Spécialisation multiple de composants 5. Spécialisation optionnelle de composants	6. Superposition de fragments de code	8. Remplacement binaire par directives d'édition des liens 9. Remplacement binaire physique	11. Spécialisation de composants à l'exécution 12. Implémentation multiple de composants
Lignes de code		6. Superposition de fragments de code 7. Condition sur constante		13. Condition sur variable

Ces différentes techniques sont exposées à la suite :

1. **Réorganisation de l'architecture** : Même si les produits d'une ligne de produits partagent les mêmes concepts, la gestion interne des flots de données et de contrôle peuvent différer. Il faut alors réorganiser l'architecture de la ligne en une architecture concrète. La reconception est réalisée à l'aide d'un langage de description d'architecture (ADL). Elle peut affecter l'ordre dans lequel sont connectés les modules et même la façon dont ceux-ci se connectent ─ ie leurs interfaces et contrats (voir chapitre 4).
2. **Frameworks multiples** : un framework peut être remplacé par un autre implémentant une autre interface et même un domaine différent. On parle alors de frameworks multiples. Il faut dans ce cas introduire ultérieurement dans les modules utilisant le framework multiple un système de sélection d'interfaces adapté (basé sur la spécialisation multiple de composants).
3. **Framework optionnel** : certains frameworks peuvent être présents dans certains produits, absents dans d'autre. La variabilité peut être réalisée ou du côté de l'appelé en réalisant un module vide disposant d'une interface conventionnelle mais retournant des valeurs nulles, ou du côté de l'appelant en implémentant une technique compensant l'absence de paire.
4. **Spécialisation multiple de composant**s : lorsqu'un composant *A* communique avec un composant *B*, en fonction des implémentations de *B*, différents modes de fonctionnement de *A* peuvent être sollicités. La variabilité se base ici sur une conception du framework de *A* divisant l'interface et l'implémentation en sous classes, chaque sous classe implémentant un mode de fonctionnement donné.
5. **Spécialisation optionnelle de composants** : des fonctionnalités du composant peuvent être optionnelles. La variabilité est si possible réalisée dans ce cas en implémentant la partie optionnelle du composant dans une classe vide à l'intérieur du framework.
6. **Superposition de fragments de code :** cette technique permet d'ajouter, juste avant la compilation, des fragments de code aux points spécifiés dans le programme. La superposition peut être réalisée en utilisant la programmation par aspects (AOP) [KIC96] par exemple. Les points de variété peuvent également être également fixés lors de l'exécution. Le programme utilise alors en interne une technique similaire à la condition sur variable.
7. **Condition sur constante** : Il s'agit ici d'utiliser des directives de compilation paramétrées par une constante pour supprimer le code non utilisé. Un programme C compilable sur système AIX et Linux peut par exemple sélectionner les lignes de code appropriées à la plateforme cible en utilisant des directives ifdefs.
8. **Remplacement binaire par directives d'édition de lien** : la méthode consiste ici à choisir le code binaire (typiquement des bibliothèques) à l'aide des directives de compilation.
9. **Remplacement binaire physique** : Le code binaire est remplacé directement en remplaçant un fichier exécutable par un autre par exemple. Cette technique est utilisée pour faciliter la maintenance d'un logiciel après sa livraison.
10. **Architecture centrée infrastructure**: il s'agit ici d'ajouter un niveau d'abstraction supplémentaire entre les composants manipulables au moment de l'exécution. Les composants n'ont plus directement

accès les uns aux autres mais communiquent à travers de ports d'entrée sortie. La réorganisation et la permutation des composants sont ainsi grandement facilitées. La connexion des composants peut être réalisée à l'aide d'un langage de script au moment de charger le programme

11. **Spécialisation de composants à l'exécution** : Cette technique permet d'adapter un composant à son environnement d'exécution. Une technique peut consister à déléguer les traitements d'un objet *A* dans un objet *B*. Pour ce faire *A* possède généralement un pointeur vers *B* et invoque les méthodes de *B* lorsqu'un traitement doit être réalisé. La variabilité correspond à la possibilité de remplacer l'objet *B* par un objet *C* implémentant le même service autrement.
12. **Implémentation multiple de composants** : Un framework peut exécuter plusieurs composants implémentant le même domaine en parallèle. La technique consiste à insérer dans le framework un mécanisme de communication permettant le multiplexage de ses différentes implémentations.
13. **Condition sur variable** : cette technique très répandue consiste à paramétrer les opérations en insérant dans le code des tests sur variables pour diriger le flot d'exécution du programme.

4.3. Introduction et fixation de la variabilité dans les systèmes adaptables

Un système adaptable est un système dont la variabilité peut être introduite et fixée dynamiquement à l'exécution. Un système adaptatif[15] est capable de s'adapter lui-même en fonction de l'évolution de l'environnement. Pour prendre en compte des besoins non prévus initialement, particulièrement lorsque le code n'est pas préparé pour de telles adaptations, l'introduction de points de variété à la conception s'avère insuffisante [HIR05]. La prise en compte de ces besoins utilise des techniques spécifiques permises par le langage. Celles-ci permettent d'introduire et de fixer la variabilité dynamiquement. Il existe plusieurs formes de support de ce type [HIR05]:

- Les primitives pour remplacer les références, le chargement dynamique des classes, etc,
- Les protocoles meta objets (MOP) qui sont des APIs permettant la consultation et la réécriture du code,
- La programmation par aspects (AOP) qui permet la superposition de structure et de comportement [KIC96].

Ces techniques mettent plus ou moins intensivement en jeu le principe de réflexion [KOJ03], c'est à dire la capacité d'un système à raisonner et à agir sur lui-même. La réflexion est une technique d'adaptation utilisée depuis de longues années dans le développement des systèmes flexibles [KEY04]. Les systèmes réflexifs ont la propriété de pouvoir utiliser leur propre représentation pour étendre et adapter leur comportement. Ils incorporent des structures représentant leurs propres aspects, et grâce à ces structures, ils accèdent et manipulent leurs comportements et structures internes.
Les systèmes réflexifs les plus simples réunissent, en général, deux niveaux:

- Le niveau de base représente le système concret, responsable de fournir les fonctionnalités de base attendues du système.
- Le méta-niveau constitue une représentation abstraite du système.

[15] Dans ce rapport, les termes adaptatif et auto-adaptatif sont synonymes

Dans un système réflexif, tous les éléments adaptables du méta-niveau sont des points de variété potentiels. La fixation de ces points de variété correspond à la modification, l'ajout ou à la suppression de l'un de ces éléments à l'aide d'un protocole Meta Objet (MOP). Les adaptations réalisées au méta-niveau sont directement réfléchies au niveau de base.

Initialement, deux types de réflexion ont été introduits : la réflexion structurelle et la réflexion comportementale. La notion de réflexion a été ensuite étendue aux aspects architecturaux des systèmes, ce qui a donné naissance à la réflexion architecturale.

4.3.1. Réflexion structurelle

Elle concerne la structure des objets [ALT01]. La réflexion structurelle permet en général l'extension de la structure des objets comme l'ajout d'un attribut mais pas de leur comportement.

L'approche la plus courante pour mettre en oeuvre la réflexion structurelle consiste à utiliser les méta-classes. Une méta-classe est en quelque sorte le type d'une classe. Toute classe est alors une instance d'une méta-classe. Les méta-classes permettent de concrétiser les classes, et de les rendre manipulables en exécution. Une classe englobe la structure de ses objets. En rendant les classes manipulables, on permet la manipulation de la structure des objets. La Figure 2.12 montre une illustration simple des trois niveaux d'un système réflexif basé sur les méta-classes.

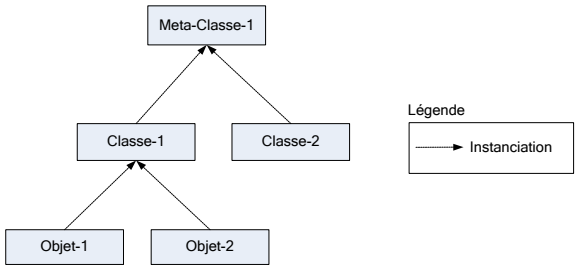


Figure 2.12 Utilisation des méta-classes

La classe *"java.lang.Class"* de Java est un exemple de méta-classe. Elle permet, par exemple, de retourner des informations décrivant les classes (attributs, méthodes, super-classes, etc) et d'instancier des classes dynamiquement à l'exécution.

4.3.2. Réflexion comportementale

Elle concerne l'exécution des objets [ALT01]. Elle permet d'un coté de fournir une description du comportement des objets, et d'un autre coté, de donner la possibilité aux utilisateurs d'intervenir sur le déroulement de l'exécution du système. En général, la réflexion comportementale peut être réalisée par l'association d'un méta-objet à chaque objet de base. Ce méta-objet permet de contrôler et de modifier le comportement de l'objet auquel il est associé. La Figure 2.13 montre le lien entre un objet et son méta-objet.

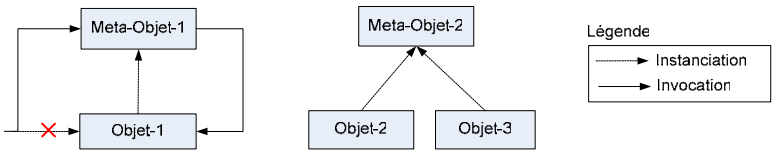


Figure 2.13 Utilisation des méta-objets

Un méta-objet permet de contrôler la sémantique d'un objet particulier. Pour cela, lorsqu'un appel de méthode a lieu sur un objet qui possède un méta-objet, l'appel est intercepté et remplacé par un appel à une méthode du méta-objet. Le méta-objet peut alors faire le traitement souhaité : il peut traiter lui-même l'appel, ou le déléguer tel quel à l'objet de base. Il peut aussi transformer le nom de la méthode ou la valeur des arguments, réaliser des pré et post traitements, etc.

4.3.3. Réflexion architecturale

Au lieu de s'appliquer à la structure des objets ou à leur comportement, la réflexion architecturale s'intéresse plutôt aux interconnexions entre les objets.

L'architecture d'une application est représentée au méta-niveau par un graphe typé. La Figure 2.14 montre un exemple de graphe. Les noeuds du graphe représentent les composants. Chaque noeud est caractérisé par l'interface du composant qu'il représente, et étiqueté par son implémentation. Les arcs du graphe représentent les connecteurs entre les composants. Un arc peut avoir des propriétés de reconfiguration qui le caractérisent. Une propriété exprime, par exemple, la capacité du connecteur de changer son protocole de communication.

Le stockage et la gestion du graphe de configuration est de la responsabilité d'un élément spécifique du méta-niveau : le gestionnaire de configuration.

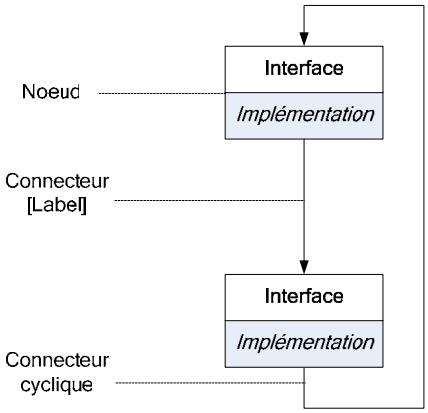


Figure 2.14 Représentation du méta niveau d'une architecture logiciel à l'aide d'un graphe

4.4. Architecture des systèmes adaptables

Les systèmes adaptables sont généralement constitués de trois éléments principaux (voir Figure 2.15) [CRE04]:

- La partie *Service* représente le système à adapter dynamiquement. Le service est implémenté via un ensemble de modules interconnectés.
- La partie *Monitoring* est chargée d'évaluer en continu l'ensemble constitué du service et de son contexte. Certains auteurs définissent le contexte comme étant la situation de l'utilisateur ou de son environnement. D'autres le considèrent comme l'environnement de l'application. De la façon la plus générale, le contexte est un environnement d'exécution qui subit un changement constant [AYE05].
- La partie *Contrôle* ou *Système de reconfiguration* [Ket04] génère les ordres d'adaptation selon une logique propre [DOW04]. La logique adaptative peut être elle même modifiée et cette modification peut provenir de l'intérieur ou de l'extérieur du système. Le système de reconfiguration contrôle le méta niveau des systèmes réflexifs ou le système lui-même dans les systèmes non réflexifs. Dans un système adaptatif, l'ordre d'adapter le système ne provient pas de l'extérieur mais du système lui-même. L'ordre d'adaptation consiste à choisir parmi les caractéristiques variantes disponibles, la caractéristique idéale pour fixer les points de variété. Les systèmes adaptatifs et adaptables peuvent posséder des points de variété ouverts, c'est à dire acceptant l'ajout à posteriori de nouvelles caractéristiques variantes. Dans ce cas, un mécanisme doit exister pour mettre à jour l'ensemble des caractéristiques variantes disponibles.

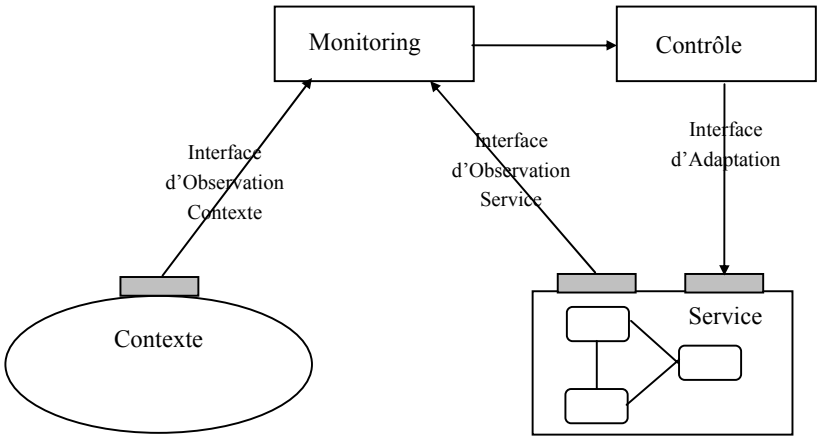


Figure 2.15 Architecture logique d'un système adaptable

Les systèmes de reconfiguration sont confrontés à plusieurs aspects importants [KET04] dont 1) le transfert d'état entre les modules lors de leur remplacement, 2) la préservation des communications en cours lors des remplacements, 3) la facilité d'utilisation de la solution d'adaptation et 4) l'automatisation des actions adaptatives.

4.5. Mesure d'adaptation

L'adaptation n'étant pas un critère fonctionnel, peu de travaux proposent des mesures d'adaptation des lignes de produits [HOE03], [SUB01]. [SUB01] propose deux métriques pour calculer, au niveau architectural, le niveau d'adaptabilité d'un produit et de la ligne de produits dont il dérive. Les deux

métriques s'appuient la notion d'index d'adaptabilité d'un élément (EAI). Un élément de l'architecture est soit un module, soit un connecteur. Un élément non adaptatif (c'est-à-dire sans point de variété fixable dynamiquement) a un EAI de 0. Un élément adaptatif (c'est-à-dire disposant d'un ou plusieurs points de variété fixables dynamiquement) possède un EAI de 1.

Métrique d'index d'adaptabilité de l'architecture (AAI) :

$$AAI = \frac{\sum_{\text{Tous les élements de l'architecture}} EAI}{\text{Le nombre d' éléments}}$$

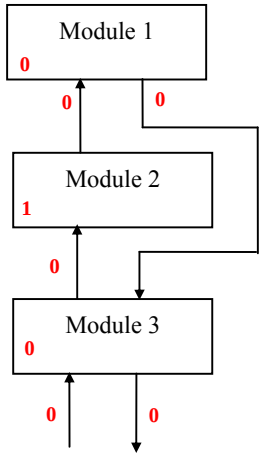
La mesure AAI reflète l'adaptabilité d'un produit.

Métrique d'index d'adaptabilité logiciel (SAI) :

$$SAI = \frac{\sum_{\text{Tous les produits}} AAI}{\text{Le nombre de produits}}$$

La mesure SAI reflète l'adaptabilité de la ligne de produit dont le produit est issu.

La Figure 2.16 montre un exemple de calcul d'AAI. Le produit est matérialisé par une architecture concrète. Le produit considéré contient 8 éléments (3 modules et 5 connecteurs) mais qu'un élément adaptatif : le module 2. Son AAI vaut donc 0.125. Si la ligne de produits comporte un second produit à 8 éléments mais qu'aucun n'est adaptatif, le SAI vaudra 0.0625 ((0.125+0)/2).



Nombre d'éléments	3 modules
	5 connecteurs
Nombre d'éléments adaptatifs	1 module
	0 connecteur
AAI	$\frac{1}{3+5} = \frac{1}{8} = 0.125$

Figure 2.16 Exemple de calcul d'index d'adaptabilité de l'architecture AAI

Etant donné qu'il existe souvent plusieurs caractéristiques adaptatives (ou dimensions d'adaptabilité) dans un logiciel (par exemple la sécurité et le traitement des données), [SUB01] raffine la notion d'EAI en permettant à cette valeur de varier entre 0 et 1. Un composant dont l'adaptabilité est mesurée en terme de sécurité et de traitement des données possède, selon la nouvelle définition, un EAI de 0,5 si une seule des deux dimensions considérées est adaptative.

Nos travaux apportent trois modifications aux définitions originales de [SUB01].

- Ils transposent le domaine d'application des trois métriques (les systèmes adaptatifs) aux systèmes adaptables. Dans ce cadre, un élément non adaptable (c'est-à-dire sans point de variété) possède un EAI de 0. Un élément adaptable (c'est-à-dire disposant d'un ou plusieurs points de variété) possède un

EAI de 1.

- Dans les simulateurs réseaux, les connecteurs n'étant pas particulièrement adaptables ceux-ci ne sont pas considérés dans le calcul de l'AAI et du SAI.
- Dans certains cas, il est possible, lorsque l'adaptation individuelle des composants ne suffit pas pour réaliser la caractéristique souhaitée, de modifier l'architecture du simulateur. Pour que la mesure d'adaptation reflète l'effet de cette technique, nous considérons adaptable, tous les composants faisant partie d'un ensemble susceptible d'être modifié à ce titre[16].

5. Conclusion

La flexibilité et la réutilisabilité sont deux notions complexes faisant intervenir des aspects logiciels n'ayant pas tous atteint maturité. Ce chapitre offre dans ce contexte, une vue synthétique du domaine et le panorama nécessaire à l'évaluation objective d'un logiciel flexible à éléments réutilisables. Nous avons opté dans ce sens pour une approche théorique présentant les différentes métriques produit statiques permettant l'évaluation d'une conception logicielle. La mise en œuvre de ces principes fait de plus en plus appel, aujourd'hui, à des outils de reenginering dotés de système de représentation graphique et de modules d'inspection complémentaires [BAR99]. Dans le chapitre suivant, nous appliquons une solution issue du domaine libre pour mesurer la structure des principaux simulateurs existants. Nous proposons également une modélisation FODA des interfaces réseaux, transposable aux simulateurs.

Références

[ALT01] E. Althammer, "Reflection Patterns in the Context of Object and Component Technology", PhD. Thesis, University of Konstanz, Konstanz, Allemagne, 2001

[AYE05] D. Ayed, "Déploiement sensible au contexte d'applications à base de composants", Thèse de doctorat, Université d'Évry Val d'Essonne, Evry, France, 2005

[BAR99] H. Bar et al, "The FAMOOS Object-Oriented Reengineering Handbook", 1999

[BAS98] L. Bass, P. Clements, R. Kazman, "Software Architecture in Practice", Addison-Wesley, 1998

[BIE95] J.M. Bieman, B.-K. Kang, "Cohesion and Reuse in an Object-Oriented System", dans *Proc. of the ACM Symposium on Software Reusability (SSR'94)*, pp. 259–262, Seattle, WA, Etats-Unis, Avril 1995

[BOE83] P.E. Presson, J. Tsai, T.P. Bowen, J.V. Post, R. Schmidt, "Software Interoperability and Reusability Guidebook for Software Quality Measurement", Boeing Aerospace Company, Rapport technique, RADC T-43-174, Vol II, Rome Air development center, Air Force Systems Command, Griffiss Air Force Base, New York, NY, Etats-Unis, 1983

[BRI93] L. Briand, S. Morasca, V. Basili, "Measuring and Assessing Maintainability at the End of High-Level Design", dans *Proc. of the Conference on Software Maintenance*, Montreal, QC, Canada, Septembre 1993

[BRI94] L. Briand, S. Morasca, V. Basili, "Defining and Validating High-Level Design Metrics", Rapport Technique CS-TR 3301, University of Maryland, Baltimore, MD, Etats-Unis, 1994

[BRI96] L. C. Briand, J. W. Daly, J. Wüst, "A Unified Framework for Coupling Measurement in Object-Oriented Systems", Rapport Technique ISERN-96-14, Fraunhofer Institute for Experimental Software

[16] Se reporter à la section 4.4.1 du simulateur YANS au chapitre 3 pour un exemple d'application

Engineering, Kaiserslautern, Allemagne, 1996

[BRI97] L. C. Briand, J. W. Daly, J. Wüst, "A Unified Framework for Cohesion Measurement in Object-Oriented Systems", Rapport Technique ISERN-97-05, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Allemagne, 1997

[BUD02] T. Budd, *"An Introduction to Object-Oriented Programming"*, Third Edition, Addison-Wesley, 2002

[CHI91] S.R. Chidamber, C.F. Kemerer, "Towards a Metrics Suite for Object Oriented design", dans *Proc. of the Conference on Object-Oriented Programming: Systems, Languages and Applications (OOPSLA'91)*, Phoenix, AZ, Etats-Unis, Octobre 1991. Publié dans SIGPLAN Notices, 26 (11), pp. 197–211, 1991

[CHI94] S.R. Chidamber, C.F. Kemerer, "A Metrics Suite for Object Oriented Design", dans *IEEE Transactions on Software Engineering*, 20(6), 476–493, 1994

[CRE04] M. Cremene, M. Riveill, C. Martel, C. Loghin, C. Miron, "Adaptation dynamique de services", dans *1ière Conférence Francophone sur le Déploiement et la (Re)Configuration de Logiciels (DECOR'04)* (Communication avec actes), pp. 53–64, Grenoble, France, Octobre 2004

[DOW04] J. Dowling, "The Decentralised Coordination of Self-Adaptive Components for Autonomic Distributed Systems", PhD. Thesis, University of Dublin, Trinity College, Dublin, Irlande, 2004

[EDE94] J. Eder, G. Kappel, M. Schrefl, "Coupling and Cohesion in Object-Oriented Systems", Rapport Technique, University of Klagenfurt, Klagenfurt, Autriche, 1994.

[ESA95] ESA Board for Software Standardisation and Control (BSSC), "Guide to the software architectural design phase", Issue 1 Revision 1, Agence Spatiale Européenne, France, Mars 1995

[FEN96] N. E. Fenton, S. L. Pfleeger*, "Software Metrics : A rigorous & Practical Approach"*, Second Edition, International Thomson Computer Press, 1996

[HAU89] H.-L. Hausen, "Yet another model of software quality and productivity", *Measurement for Software Control and Assurance*, Littlewood, Elsevier, 1989

[HEN81] S. Henry, K. Kafura, "Software structure metrics based on information flow", *IEEE Transactions on Software Engineering,* 7(5), pp. 510–518, 1981

[HEN96] B. Henderson-Sellers, *"Software Metrics"*, Prentice Hall, Hemel Hempstead, Royaume-Uni, 1996

[HIR05] R. Hirschfeld, R. Lämmel, "Reflective designs", *IEE Proceedings Software*, 152(1), pp. 38–51, 2005

[HIT95] M. Hitz, B. Montazeri, "Measuring Coupling and Cohesion in Object-Oriented systems", dans *Proc. of the Third International Symposium on Applied Corporate Computing (*ISACC*'95)*, pp. 25–27**,** Monterrey, Mexique, Octobre 1995

[HOE03] A. van der Hoek, E. Dincel, N. Medvidovic , "Using Service Utilization Metrics to Assess the Structure of Product Line Architectures" dans *Proc. of the Nine International Software Metrics Symposium (METRICS'03)*, pp. 298–308, Sydney, Australie, Septembre 2003

[IEEE90] "IEEE Standard Glossary of Software Engineering Terminology", IEEE Std 610.12-1990, IEEE, 1990

[KAN90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, "Feature-Oriented Domain Analysis (FODA) Feasibility Study", Rapport Technique CMU/SEI-90-TR-21 ESD-90-TR-22, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA, Etats-Unis, 1990

[KEE04] J. Keeney, "Completely Unanticipated Dynamic Adaptation of Software", PhD. Thesis,

University of Dublin, Trinity College, Dublin, Irlande, 2004

[KET04] A. KETFI, “Une approche générique pour la reconfiguration dynamique des applications à base de composants logiciels”, Thèse de doctorat, Université Joseph Fourier, Grenoble, France, 2004

[KIC96] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. V. Lopes, J.-M. Loingtier, J. Irwin, “Aspect-Oriented Programming”, dans *Proc. of the 11th European Conference on Object-Oriented Programming (ECOOP '97)*, pp. 220–242, Jyväskylä, Finlande, Juin 1997

[KOJ03] S. Kojarski,_ K. Lieberherr,_ D. H. Lorentz,_ R. Hirschfeld, “Aspectual Reflection”, dans *AOSD SPLAT Workshop (SPLAT’03)*, Boston, MA, Etats-Unis, Mars 2003

[LEE95] Y.-S. Lee, B.-S. Liang, S.-F. Wu, F.-J. Wang, “Measuring the Coupling and Cohesion of an Object-Oriented Program Based on Information Flow”, dans *Proc. of the International Conference on Software Quality (ICSQ'95)*, pp. 81–90, Maribor, Slovenie, Novembre 1995

[MAC87] A. Macro, J. Buxton, *“The Craft of Software Engineering”*, Addison-Wesley, 1987

[McC77] J.A. Mc Call, P.K. Richards, G.F. Walters, “Factors in Software Quality”, Volume I, II and III, US Rome Air Development Center, Reports NTIS AD/A-049 014, 015, 055, National Technical Information Services, US Department, 1977

[MIL98] T. D. Miller, P. E. E. Pedersen, “Defining Modules, Modularity and Modularization, Evolution of the Concept in a Historical Perspective”, dans *Proc. of the 13th IPS Research Seminar*; Aalborg University, Fuglsoe, Danemark, 1998

[MYE78] G. J. Myers, *“Composite/Structured Design”*, Van Nostrand Reinhold, 1978

[PAR72a] D. L. Parnas, "A Technique for Software Module Specification with Examples", *Communications of the ACM*, 15(5), pp. 330–336, 1972

[PAR72b] D. L. Parnas, "On the Criteria to be Used in Decomposing Systems into Modules", *Communications of the ACM*, 15(12), pp. 1053–1058, 1972

[SHE93] M. J. Shepperd, D. Ince, *“Derivation and validation of software metrics”*, Oxford Clarendon Press, 1993

[STE74] W. Stevens, G. Meyers, L. L. Constantine, “Structured design”, *IBM Systems Journal*, 13(2), 1974

[SUB01] N. Subramanian, L. Chung , “Metrics for Software Adaptability”, dans *Proc. of the Software Quality Management Conference (SQM’01)* , pp. 95–108, Loughborough, Royaume-Uni, Avril 2001

[SVA99] M. Svahneberg et J. Bostch, “Evolution in Software Product Lines : Two Cases”, *Journal of Software Maintenance:Research and practices*, 11(6), pp. 391–422, 1999

[SVA00] M. Svahneberg, J. Bostch, “Issues concerning variability in Software product lines”, dans *Proc. of the Third International Workshop on Software Architectures for Product Families (IW-SAPF’03)*, pp. 50–60, Las Palmas de Gran Canaria, Espagne, Mars 2000, Publié dans *Lecture Notes in Computer Science*, Vol. 1951, pp. 146–157, 2000

[SVA05] M. Svahnberg J. Van Gurp, J. Bosch, “A Taxonomy of Variability Realization Techniques”, *Software–Practice & Experience*, 35(8), pp. 705–754, John Wiley & Sons, 2005

[SZY02] C. Szyperski, *“Component Software”*, Second Edition, Addison-Wesley, 2002

[VAN01] J. Van Gurp, J. Bosch, M. Svahnberg, “On the Notion of Variability in Software Product Lines”, dans *Proc. of the Working IEEE/IFIP Conference on Software Architecture (WICSA'01)*, Vol. 00, pp. 45–54, Amsterdam, Pays-bas, Aôut 2001

[YOU79] E. Yourdon, L. L. Constantine, *“Structured design”,* Prentice Hall, 1979

Chapitre 3

Flexibilité et Réutilisabilité des Interfaces Réseaux dans les Simulateurs Sans Fil

1. Introduction

Comme nous l'avons vu précédemment, les systèmes généraux et modulaires se basent le plus souvent sur la décomposition en sous systèmes. Les critères de modularité et de généralité permettent d'évaluer la pertinence de la modularisation.

Dans le domaine des réseaux, le principe de modularisation a essentiellement été appliqué à la conception des systèmes de communication. Les découpes en couches des modèles OSI et TCP/IP sont des exemples de modularisation de systèmes réseaux.

Les besoins en flexibilité et réutilisabilité ne cessant de croître, la modularisation s'applique aujourd'hui aux implémentations de couches elles-mêmes. Plusieurs implémentations de protocoles composites on été proposées, particulièrement au niveau des couches supérieures du modèle OSI. X-Kernel [HUT91], [0'M92], Netscript [SIL98], Horus [REN96], [REN95] et Cactus [WON01] figurent parmi les initiatives les plus connues. Bien que ces systèmes ne se rapportent pas spécifiquement aux interfaces réseaux, ceux-ci ont montré la viabilité du principe et introduit des concepts pionniers: décomposition en modules, architecture par flots de données[17] [0'M92], décomposition en types abstraits, architecture en couches, utilisation de contrats [REN95], décomposition en composants, architecture à composants indépendants [SIL98], décomposition en modules, superposition d'une architecture centralisée et d'un système à évènement, abstraction des messages [WON01].

Au niveau des couches basses, même si le concept d'interfaces réseaux modulaires ouvertes suscite moins d'engouement, l'intérêt pour ces architectures grandit. Celui-ci se traduit aujourd'hui par:

- la publication de spécifications modulaires [IEE99], [IEE02],
- des solutions logicielles matérielles mixtes programmables ou adaptatives à base de composants [FAR00], [BIA98], [CIE89]. [CIE89] énonce en particulier des principes de conception essentiels: utilisation de machines à états finis, séparation des automates en réception et en émission, architecture matérielle câblée réutilisable.
- le développement de systèmes d'exploitation ouverts basés sur les composants tels que TinyOS [GAY05].

[17] A noter qu'à la différence d'un système pipe and filter pur, pour des raisons de performances, 1) un processus est associé à chaque message entrant dans l'architecture 2) ce processus est recyclé dans tous les micro-protocoles (modules) traversés par le message

Cette évolution profite des phénomènes suivants :

- La réorganisation générale des systèmes due à la nécessité de faire converger les réseaux informatique et télécoms et faciliter la communication entre les couches réseau (cross-layering),
- L'arrivée à maturité de puissantes techniques de programmation radio (radio reconfigurable, radio logicielle) permettant par exemple la programmation des algorithmes de la couche MAC [HUN06],
- La mise au premier plan des méthodes de génie logiciel (spécification, composants, middleware programmable).

Contrairement aux implémentations réelles, les simulateurs semblent pourtant conserver la paternité stricte au modèle OSI. La question posée dans ce mémoire est de savoir si le principe de modularisation peut s'appliquer au domaine de la simulation des réseaux et ce dans un cadre de recherche. Pour y répondre, nous commençons par citer l'ensemble des avantages et inconvénients posés par cette pratique. Nous présentons ensuite les travaux existants en simulation des réseaux. Nous mesurons enfin la modularité et la généralité de ces implémentations.

2. Avantages et Inconvénients de la Modularisation dans le Domaine de la Simulation Sans Fil

2.1. Facteurs limitants

Nous distinguons quatre facteurs limitant la modularisation de la couche liaison.

1. La perception de l'approche dans le domaine de la recherche : La modularisation est une démarche d'ingénierie représentant d'avantage un moyen pour la recherche en réseaux qu'une fin.
2. Le développement d'un simulateur à grain fin dans le cadre d'un projet de recherche :
    1. Les simulateurs classiques sont plus flexibles que les structures modulaires car ils présentent une implémentation minimaliste anarchiquement extensible. Bien que cette thèse aille à l'encontre du principe de généralité des architectures modulaires, elle est vérifiée par l'emprise du simulateur NS-2 sur le domaine de la simulation. L'extensibilité des simulateurs à grain fin étant plus contraignante, l'acceptation de ces architectures par la communauté est de facto limitée.
    2. Les structures modulaires sont conçues pour les problèmes complexes. Il s'agit de modéliser les détails, donc d'implémentations lourdes.
3. Les performances d'un simulateur à grain fin : Les performances des structures modulaires sont inférieures aux modèles classiques du fait de l'overhead en communication entre les modules.
4. La précision des résultats de simulation : la comparaison entre mesures empiriques et résultats de simulation montre d'importantes différences. [TAK01a], [TAK01b], [CAV02] les expliquent principalement par le manque de précision des modèles de la couche physique et du canal radio. La couche physique constitue de fait un goulet d'étranglement limitant les perspectives de développement des couches supérieures.

2.2. Facteurs motivants

Pour apporter des solutions de prototypage aux ingénieurs, les simulateurs réseaux doivent nécessairement évoluer. La modularisation des interfaces sans fil pourrait apporter, à ce titre, des solutions aux problèmes suivants:

1. La complexité des protocoles informatiques :
   - Les modèles ne sont pas complets : Les couches sont devenues si complexes qu'elles ne peuvent plus être modélisées complètement. A titre d'exemple, les implémentations de 802.11 dans les simulateurs historiques n'intègrent ni le *management*, ni la *gestion du débit*. Les fonctions optionnelles proposées par la norme sont également omises.
   - Les modèles ne sont pas totalement corrects : du fait des libertés prises lors du processus de simplification du système à la conception, les modèles proposés contiennent des erreurs [RED06], [BEN07a]. [SAR98] affirme à ce titre que les méthodes de conception du génie logiciel doivent être utilisées pour développer des implémentations de modèle correctes.
2. Le potentiel de réutilisabilité : le code source des simulateurs est généralement étendu grâce au mécanisme d'héritage ou par ajout de fonctions. Les conceptions modulaires offrent une réutilisabilité supérieure car le code binaire est utilisé et intégré à la manière d'une boîte noire. Le tout est de savoir s'il est préférable de dériver deux technologies différentes d'une même ligne de produits ou de les implémenter séparément.
3. Le potentiel d'adaptabilité : Les technologies sans fil mettent de plus en plus en avant des concepts généraux (radio logicielle, auto-configuration, adaptation à l'environnement). Les structures modulaires sont particulièrement adaptées à ce contexte vu leur potentiel d'adaptation et de maintenance. La modularité fournit dans cette perspective le cadre nécessaire à l'automatisation de cette adaptabilité. L'adaptation d'une structure modulaire pourrait consister au remplacement en temps réel d'un module par un équivalent ou à l'assemblage et à la validation ad-hoc d'une composition.
4. Par beaucoup d'aspects (composant, etc), le logiciel est capable d'émuler le fonctionnement du matériel. Les solutions matérielles étant lourdes à mettre en œuvre, le domaine de la simulation constitue un terrain idéal pour proposer des solutions innovantes dans ce domaine.

3. Modularisation des Interfaces Réseaux

Malgré certains de ses inconvénients, l'approche de modularisation a été empruntée dans plusieurs projets. Des implémentations à grain moyen sont proposées pour NS-2 dans [Net-Zigbee], [Net-UMTS], [Net-Bluetooth] pour modéliser respectivement les technologies 802.15.4, UMTS et 802.15.2. [PAQ06] propose la rationalisation du simulateur NS-2 à l'aide d'une structure de nœud plus générale capable d'intégrer des modèles développés séparément. Une architecture générique dédiée aux technologies UMTS et 802.11 est également proposée dans [BER04]. Son prototype SDL (Specification and Description Language) est évalué sous le simulateur MOSEPS (Modular Object-oriented Software and Environment for Protocol Simulation) développé en interne. L'approche est mise en pratique dans

[BER05] pour les réseaux multi modes large bande. Dans cette approche, les couches sont composées et paramétrées en temps réel par les plans de *management*. Chaque couche est composée d'une partie générale et d'une partie spécifique. Les fonctions génériques et spécifiques définies sont fournies dans une liste, indiquant également le lieu d'implémentation de chaque fonction.

Pour évaluer l'intérêt de l'approche de modularisation dans les simulateurs de paquets, nous comparons ici les implémentations traditionnelles NS-2 [Net-NS-2], Glomosim [Net-Glomo], J-Sim [Net-J-Sim] avec le simulateur YANS [Net-YANS] à la base du futur NS-3. La mouture originale de YANS offre à la fois une implémentation modulaire et générique des technologies 802.11a, b et e.

3.1. Considérations relatives à la mesure de la modularité des simulateurs

La mesure de la modularité se base sur l'analyse de l'architecture du simulateur et du code des différents modules utilisés dans le système.

- L'analyse de l'architecture est réalisée manuellement. Elle permet de dériver :
  - les métriques de morphologie, à savoir la taille, la largeur, la profondeur et le ratio arcs à noeuds du simulateur.
  - la métrique d'impureté de l'arbre
- L'analyse du code est réalisée à l'aide de l'outil CCCC [LIT01][18]. CCCC a été choisi pour sa consistance avec notre choix de mesures (cf chapitre 2). C'est de plus un outil éprouvé et capable d'analyser les systèmes écrits en langages objet (C++ et java) et procéduraux (langage C).

  Nous utilisons en particulier pour les simulateurs objets NS (C++), YANS (C++) et J-Sim (Java) les métriques suivantes :
  - LOC (Line of Code): nombre de lignes de code (différente d'une ligne vide)
  - Fan-in : mesure de flots d'information[19]
  - Fan-out : mesure de flots d'information[3]
  - IF4 (Information Flow): mesure de flots d'information
  - CBO (Coupling Between Object) : métrique de couplage
  - MVG : mesure de complexité cyclomatique de McCabe

  Glomosim étant implémenté en C, les métriques objets et de flots d'information ne sont pas disponibles. Seules les métriques LOC et MVG sont en effet rendues par CCCC.

  Le code est mesuré classe par classe. Seules les classes relatives à l'architecture sont considérées. Pour les simulateurs objets, nous listons également les classes parentes quand celles-ci contiennent du code de traitement spécialisé.

3.2. Considérations relatives à la mesure de la généralité des simulateurs

Le développement d'un modèle définissant les caractéristiques des différentes technologies réseaux est l'étape préliminaire pour comparer la généralité des différents simulateurs existants, et ce

[18] D'autres outils libres d'analyse utilisant des métriques diverses et variées existent [Net-JDep], [Net-DepF], [Net-CKJM], [Net-Vil], [Net-Mula] et [Net-SiSS]

[19] Il semblerait que les valeurs de Fan-in et de Fan-out aient été inversées sous CCCC. Dans la suite de ce rapport, cette interversion est corrigée pour correspondre à la définition de [ESA95]

indépendamment des technologies modélisées.

Le modèle des caractéristiques doit être simple et compréhensible à l'utilisateur. Le développement d'un modèle générique est de plus soumis à trois problèmes techniques majeurs :

- Les relations de dépendances entre caractéristiques[20] : certains standards présentent des caractéristiques fortement dépendantes. C'est le cas de la norme 802.11 dans laquelle la gestion des acquittements est intégrée dans l'accès au canal DCF. La séparation des caractéristiques selon le principe de généricité pourrait donc sembler enfreindre l'intégrité d'un standard. Nous pensons toutefois que dans le domaine de la simulation, certaines fonctions étant naturellement omises ou modifiées, le problème d'interdépendance ne doit pas empêcher de séparer certaines caractéristiques même à priori dépendantes[21].
- La différentiation des caractéristiques : Certaines normes définissent des fonctionnalités similaires ou quasi similaires dans plusieurs couches. Une solution pour distinguer ces fonctionnalités consisterait à associer les caractéristiques à la couche où elles sont définies. Le problème de cette solution est de rendre les caractéristiques dépendantes de l'implémentation. Pour résoudre ce problème, nous avons choisi de regrouper les fonctionnalités par thèmes en ramifiant les fonctions similaires existantes en autant de caractéristiques identifiables.
- La spécialisation des caractéristiques : Certaines normes mettent en jeu des algorithmes très spécifiques. Pour pallier à ce problème, un certain niveau d'abstraction dans la définition du modèle doit être observé. L'effet négatif est de pénaliser les simulateurs techniques.

La suite de ce chapitre présente une revue des caractéristiques possibles d'une interface réseau dans un simulateur[22]. L'analyse est menée indépendamment de toute technologie et sans considération pour l'architecture des technologies étudiées. Ce travail couvre principalement les fonctionnalités d'accès au réseau des modèles OSI [UIT94], UMTS [3GP96] et 802.11 [IEE99] et englobe donc la norme 802.11. La modélisation du domaine et des caractéristiques est réalisée selon le formalisme FODA (Feature Oriented Domain Analysis) [KAN90]. L'avantage de la méthode FODA sur les autres méthodes d'analyse de domaine est d'être particulièrement adaptée à la construction d'éléments réutilisables [FER99].

La Figure 3.1 situe le domaine d'un système sans fil dans le domaine réseau dans son ensemble. La communication inter couche est formalisée à l'aide d'une couche transversale complètement abstraite [SRI05].

[20] Problème communément appelé « interaction des caractéristiques »
[21] Il reste toujours possible de spécifier les dépendances inter caractéristiques à posteriori produit par produit
[22] A noter cependant que les fonctions physiques de très bas niveau (modulation, etc) ne sont pas listées.

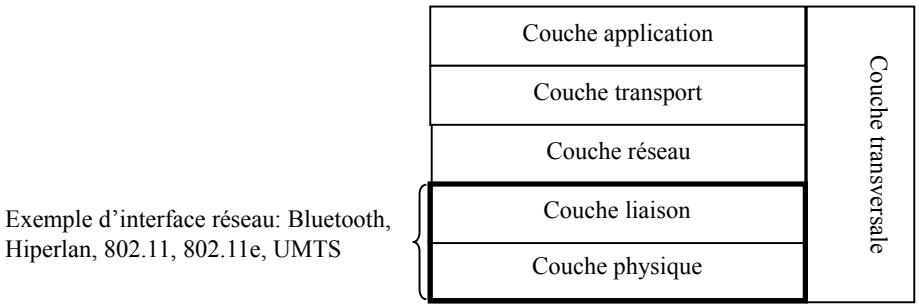


Figure 3.1 Diagramme de structure représentant la couche liaison dans le modèle TCP/IP augmenté d'un mécanisme de communication inter couche

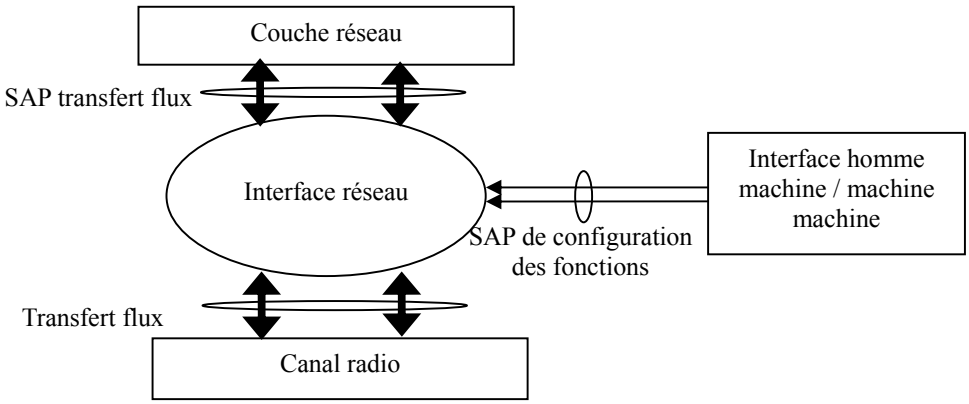


Figure 3.2 Diagramme de contexte représentant les principaux flux d'informations avec les autres entités du système

Les fonctions sont considérées le plus indépendamment possible, chacune présentant une interface de service avec l'extérieur du système si une interaction est nécessaire pour configurer la fonction (voir Figure 3.2). A titre d'exemple, la qualité de service est un service fourni au reste du système nécessitant la configuration de plusieurs fonctions dont l'accès au canal et les files d'attente notamment. Ces fonctions sont configurées via le SAP de configuration.

Les Figures 3.3 à 3.11 listent les différentes caractéristiques possibles d'une *interface réseau*. Les caractéristiques optionnelles sont représentées par des cercles et les alternatives par des arcs.

Dans notre perspective, seuls l'*accès au canal*, la *file d'attente* et certaines fonctions d'*orientation de flux* et d'*écoute du médium* figurent comme caractéristiques essentielles dans un simulateur : les autres sont des caractéristiques optionnelles et ne sont généralement implémentées que dans les simulateurs spécialisés.

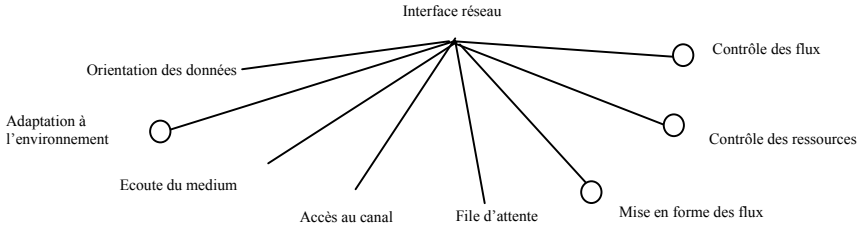


Figure 3.3 Caractéristiques définissant le fonctionnement d'une carte réseau

Chacune des fonctions peut être ramifiée en sous caractéristiques. Les sections suivantes décrivent les

ramifications de chacune des caractéristiques présentées à la Figure 3.3. En gras figurent les caractéristiques associées à une dimension d'adaptation pour le calcul des index d'adaptation des simulateurs (voir la section 4.5 du chapitre 2).

3.2.1. Caractéristiques liées à l'accès au canal

La Figure 3.4 représente quelques-unes des techniques d'*accès au canal* existantes.

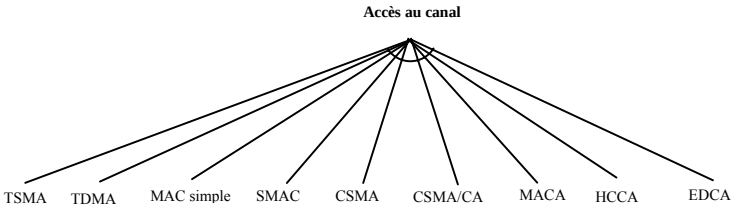


Figure 3.4 Caractéristiques définissant la méthode d'accès au canal

Les accès *TDMA*, *TSMA* [CHL94], *MACA* [KAR90] et *CSMA* figurent aujourd'hui comme des techniques relativement délaissées de l'industrie grand public.

- *MACA* (Multiple Access with Collision Avoidance) est un CSMA modifié n'utilisant que la détection virtuelle de porteuse (pas de détection physique)
- *TSMA* (Time-Spread Multiple-Access) est un protocole sloté dédié au réseau multi sauts se basant sur un algorithme mathématique garantissant aux stations un accès sans contention au medium.
- *TDMA* (Time Division Multiple Access) est une technologie slotée dédiée aux réseaux à 1 saut et garantissant aux stations un accès sans contention au medium.

Les accès *CSMA/CA*, *HCCA* et *EDCA* sont en revanche intégrés aux produits 802.11 et 802.11e. *SMAC* définit un accès au canal pour les réseaux de senseurs [YE04]. *Mac-simple* définit une couche MAC dédiée au réseau ad-hoc.

3.2.2. Caractéristiques liées à la file d'attente

Comme le montre la Figure 3.5, les caractéristiques de *file d'attente* définissent à la fois la *discipline de service* et la *gestion des buffers*.

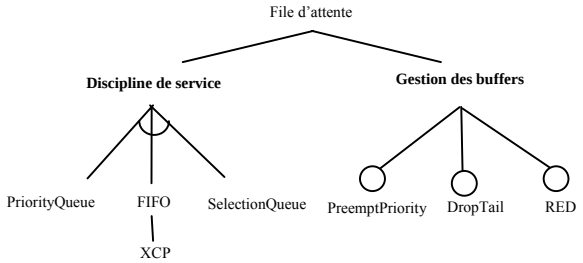


Figure 3.5 Caractéristiques définissant la gestion des files d'attente

La *discipline de service* définit l'algorithme effectuant la sélection du prochain paquet à servir dans la file. Les techniques d'ordonnancement les plus répandues dans les simulateurs sont les suivantes :

- *FIFO* (First In First Out) est une file à capacité illimitée traitant les paquets dans leur ordre d'arrivée.
- *XCP* est un exemple de file FIFO. Les files XCP ont été développées dans le cadre du protocole de

transport XCP (eXplicit Congestion Protocol). Elles mettent à jour les entêtes de transport des paquets avec des informations relatives à la congestion locale, ce afin de notifier aux entités de bout en bout l'état du réseau.

- *PriorityQueue* place les paquets de haute priorité (paquets émis par les protocoles de routage, etc) en position privilégiée dans la file d'attente.
- *SelectionQueue* : Nous appelons SelectionQueue les files d'attente proposant à l'entité gérant l'accès au canal, la possibilité sélectionner un paquet selon une sémantique donnée (choix d'un flux ou d'une priorité). Ce type de file intègre généralement un classifieur permettant d'aiguiller les paquets dans différentes files.

La technique d'ordonnancement peut être combinée à une technique de *gestion des buffers* de type *RED* (Random Early Detection), *DropTail* ou *PremptPriority*.

- *RED* est un algorithme de gestion de buffer éliminant des paquets de la file d'attente quand son remplissage a atteint un certain niveau.
- *DropTail* est une technique, limitant la taille des files FIFO. Lorsque la file est pleine ou bien les nouveaux paquets sont jetés ou bien les plus anciens sont éliminés pour permettre au nouveau de rentrer.
- *PreemptPriority* est une technique de gestion de buffers basée sur les files prioritaires. Le mode préemptif jette les paquets moins prioritaires à l'arrivée de paquets plus prioritaires lorsque la capacité totale de la file est atteinte.

3.2.3. Caractéristiques liées au contrôle des flux

Comme le montre la Figure 3.6, les fonctions de *contrôle de flux* définissent les traitements réalisés sur les communications pour corriger ou détecter une erreur dans un flux, ou détecter une erreur de séquencement dans un flux.

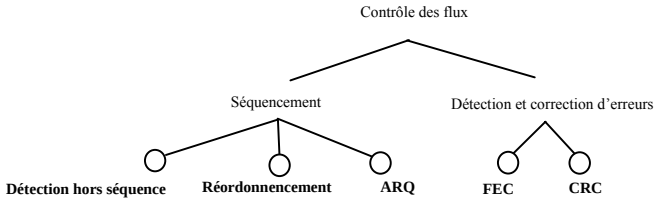


Figure 3.6 Caractéristiques définissant les mécanismes de séquencement et de détection et de correction d'erreurs

On distingue parmi les fonctions de *détection et de correction d'erreurs* les méthodes de :

- *détection d'erreurs* Cyclic Redundancy Code (CRC) : elles permettent au récepteur d'un message de vérifier que les données transmises ne contiennent pas d'erreurs,
- *correction d'erreurs* FEC (Forward Error Coding) : elles permettent la correction des erreurs détectées à la réception dans les paquets à l'aide de bits de redondance.

Le contrôle de séquence est réalisé en insérant dans les paquets un numéro de séquence, ce afin de détecter les paquets déséquencés (doublons inclus) et éventuellement de les réordonner. La reprise sur erreur *ARQ* (Automatic Repeat-reQuest) permet la retransmission des paquets qui n'ont pas été

correctement reçus lorsque l'émetteur ou le récepteur détecte une erreur dans la transmission. Le contrôle de séquence peut être accompagné de mécanismes limitant l'émission à l'aide d'une fenêtre coulissante.

3.2.4. Caractéristiques liées à la mise en forme des flux

Les fonctions de *mise en forme des flux* définissent les traitements à réaliser sur les données lors des communications. Celles-ci peuvent concerner la *compression*, la *fragmentation* ou le *chiffrement* (voir Figure 3.7).

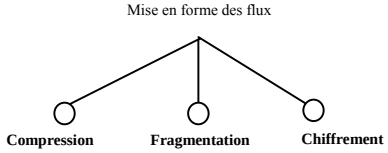


Figure 3.7 Caractéristiques définissant la compression, la fragmentation et le chiffrement

3.2.5. Caractéristiques liées au contrôle des ressources

Le *contrôle des ressources* gère les transitions entre les différents états opérationnels des fonctions de *Management* ou de *mode veille* (voir Figure 3.8).

Le *mode veille* pilote le passage entre les niveaux d'activité radio du noeud. On distingue ici des algorithmes spécifiques :

- *PSM* (Power Saving Mode) est l'algorithme proposé dans le standard 802.11,
- *Periodic listen & sleep* et son amélioration *Periodic listen & sleep avec adaptive listening* ont été proposés dans le cadre du développement de la couche MAC pour les réseaux de senseurs SMAC [YE04].

Le *Management* définit le niveau opérationnel du nœud ou des flux: hors du réseau, paging, synchronisé, authentifié, connecté, etc.

- La caractéristique de *connexion* est propre aux technologies fonctionnant en mode connecté c'est-à-dire conservant dans les nœuds un état définissant leur niveau opérationnel : non authentifié, authentifié, connecté.
- La *découverte du voisinage* concerne essentiellement la découverte automatique des paramètres réseaux via les fonctions de scanning ou de paging nécessaires à la mise en place d'une connexion.
- La *réservation des liens* consiste à contrôler l'admission des flux dans le réseau et à paramétrer les fonctions d'ordonnancement des files d'attente.

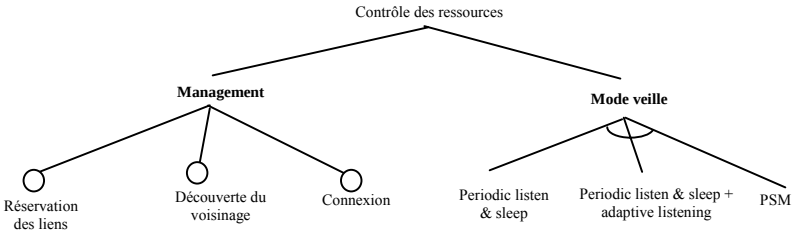


Figure 3.8 Caractéristiques définissant les fonctions de contrôle des ressources

3.2.6. Caractéristiques liées à l'adaptation à l'environnement

L'adaptation aux conditions environnementales concerne l'*adaptation au canal* et la gestion des *handovers* dus à la mobilité (voir Figure 3.9).

L'*adaptation au canal* indique si un contrôle de *puissance*, de *contention* ou de *débit* est effectué en temps réel.

Nous distinguons en particulier dans le cadre de l'*adaptation de débit* les algorithmes *ARF* (Automatic Rate Fallback) [KAM97] et *AARF* (Adaptive ARF) [LAC04].

- *ARF* est l'algorithme implémenté dans les premières cartes 802.11 WaveLAN. ARF se base sur la réception ou non d'un acquittement pour augmenter ou diminuer le débit physique de la carte.
- Une version plus stable de cet algorithme, *AARF*, est proposée dans le driver Madwifi dédié aux cartes 802.11a/b/g. Contrairement à ARF dans lequel les seuils de prises de décisions d'adaptation sont statiques, AARF adapte continuellement ces seuils.

La gestion de la contention consiste à ralentir les émetteurs à l'aide d'un algorithme basé sur la charge du canal.

- *BEB* est l'algorithme Binary Exponential Backoff utilisé dans la norme 802.11
- *NBA* [TAI05] (Neighboorhood Backoff Algorithm) est un algorithme de backoff tenant compte du nombre de voisins de chaque nœud.

La gestion des handovers définit la politique à appliquer lors des changements de cellule. La décision de *déclenchement d'un handover* et le choix de la nouvelle cellule peut se baser sur :

- les mesures de charge (*load balancing*),
- les mesures de rapport signal à bruit (*SNR*).

La *gestion des buffers* dans le réseau concerne le reroutage des paquets des stations en cours de handovers.

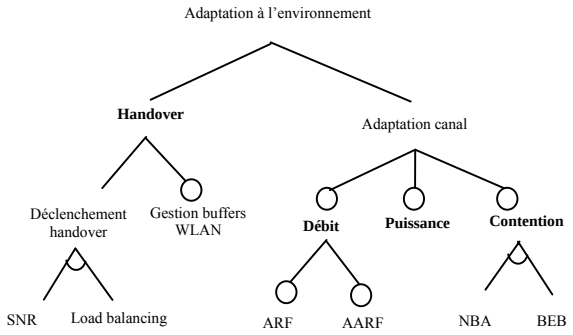


Figure 3.9 Caractéristiques définissant la gestion de l'adaptation

3.2.7. Caractéristiques liées à l'orientation des données

L'orientation des trames dans le réseau met principalement en jeu les notions de *projection de flux* et d'*intégration au réseau filaire* (voir Figure 3.10).

L'*orientation des données* marque principalement la différence entre les technologies issues des domaines informatiques et télécoms.

- Dans les technologies informatiques, l'orientation des données à l'intérieur de la couche liaison consiste simplement à relayer vers la couche physique les données provenant des couches supérieures et vice-versa.
- Dans les technologies télécoms, par contre, une hiérarchie de flux peut être définie et les flux provenant des couches supérieures successivement multiplexés à l'intérieur de cette hiérarchie.

L'*intégration au réseau filaire* consiste à rediriger les trames de données parvenant à une station vers la partie filaire du système.

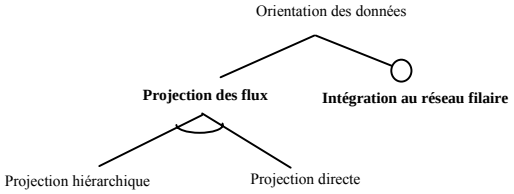


Figure 3.10 Caractéristiques définissant le mapping des flux réseaux sur les flux physiques

3.2.8. Caractéristiques liées à l'écoute du médium

L'*écoute du medium* permet à une station de savoir si le lien est libre et si les paquets reçus peuvent être correctement décodés. Comme le montre la Figure 3.11, celle-ci englobe plusieurs sous caractéristiques.

- La *détection virtuelle* de porteuse permet à une station tierce de détecter, en analysant les trames circulant sur le medium, les réservations de canal effectuées par deux stations souhaitant s'accaparer le médium pour communiquer.
- La *détection physique* consiste à détecter si de l'énergie circule sur le médium physique. Deux méthodes existent. CS (Carrier Sense) indique la détection d'un signal de type connu possiblement démodulable (peu importe son niveau d'énergie). ED (Energy Detection) indique une activité énergétique dont la puissance dépasse un certain seuil. Les deux méthodes peuvent être utilisées conjointement.
- Le *modèle d'acceptation* des paquets détermine si un paquet reçu est correctement décodé en présence d'interférences. Trois méthodes existent : la première, ET, se base uniquement sur le niveau énergétique du signal et ignore la présence d'interférences. SNRT teste le rapport signal à bruit à chaque fois qu'arrivent de nouvelles interférences au cours de la réception d'un signal potentiellement décodable (c'est-à-dire à niveau énergétique suffisant) pour savoir si le paquet pourra être décodé sans erreur. La dernière, BER, teste à chaque fois que le niveau de bruit change, si la dernière portion reçue de signal a été corrompue par les interférences.
- Le *modèle de capture* détermine si un paquet potentiellement décodable, fraîchement reçu, peut devenir le nouveau paquet en cours de décodage après que le matériel (exemple la couche physique) ai déjà permis le décodage d'un précédent paquet et son assemblage dans les couches supérieures (exemple la couche MAC). Plusieurs modèles de capture existent [WAR01], mais les simulateurs n'implémentent généralement qu'un modèle de base comparant les puissances des paquets potentiellement décodables (cf [BEK07a]).

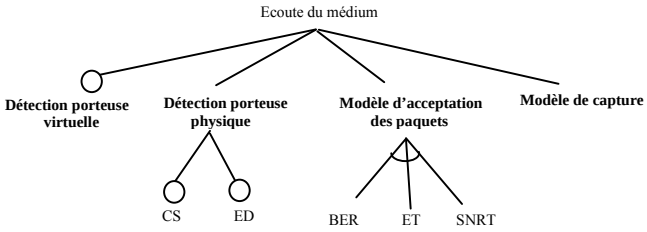


Figure 3.11 Caractéristiques définissant le séquencement et les fonctions de détection de porteuse

4. Mesures de Généralité et de Modularité des Simulateurs Réseaux

4.1. NS

NS-2 [Net-NS-2] repose sur un modèle à grain épais (voir Figure 3.12). Le modèle s'appuie essentiellement sur l'architecture orientée objet. Les modules possèdent une interface de communication générique rudimentaire composée de deux fonctions. La première, *recv(args),* permet la réception des paquets dans les sens montant et descendant. La seconde, *handle(evt),* permet d'envoyer au module un événement pour déclencher un traitement interne.

La version de NS-2 à l'étude est la version 2.29. Les stations et points d'accès ne sont pas différentiés. L'architecture d'un point d'accès est donc la même qu'une station mobile.

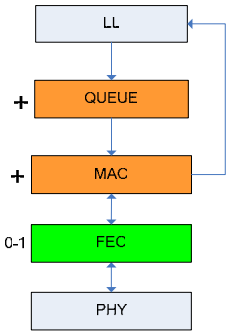


Figure 3.12 Diagramme en couches de NS

NS-2 intègre 4 implémentations de couche MAC (mac-simple, mac-tdma, mac-802.11, mac-smac) pour les réseaux informatiques sans fil et plusieurs types de files d'attente (XCP, DropTail, PriorityQueue, RED) plus ou moins couplées avec le protocole de routage, les protocoles de transport ou les applications utilisés. La couche LL et la couche physique sont communes à tous les réseaux. Le composant FEC est un composant optionnel implémentant la fonction de correction d'erreur. Sous NS, un nœud pouvant se comporter comme une station de base, l'intégration de telles bases au réseau filaire permet de simuler des réseaux multicellulaires. L'interconnexion câblée de réseaux ad-hoc est également permise. Ces deux types d'interconnexion sont toutefois pris en charge au niveau IP.

4.1.1. Mesures de généralité

La Figure 3.13 décrit l'ensemble des caractéristiques présentes dans les implémentations disponibles de réseaux locaux sans fil sous NS.

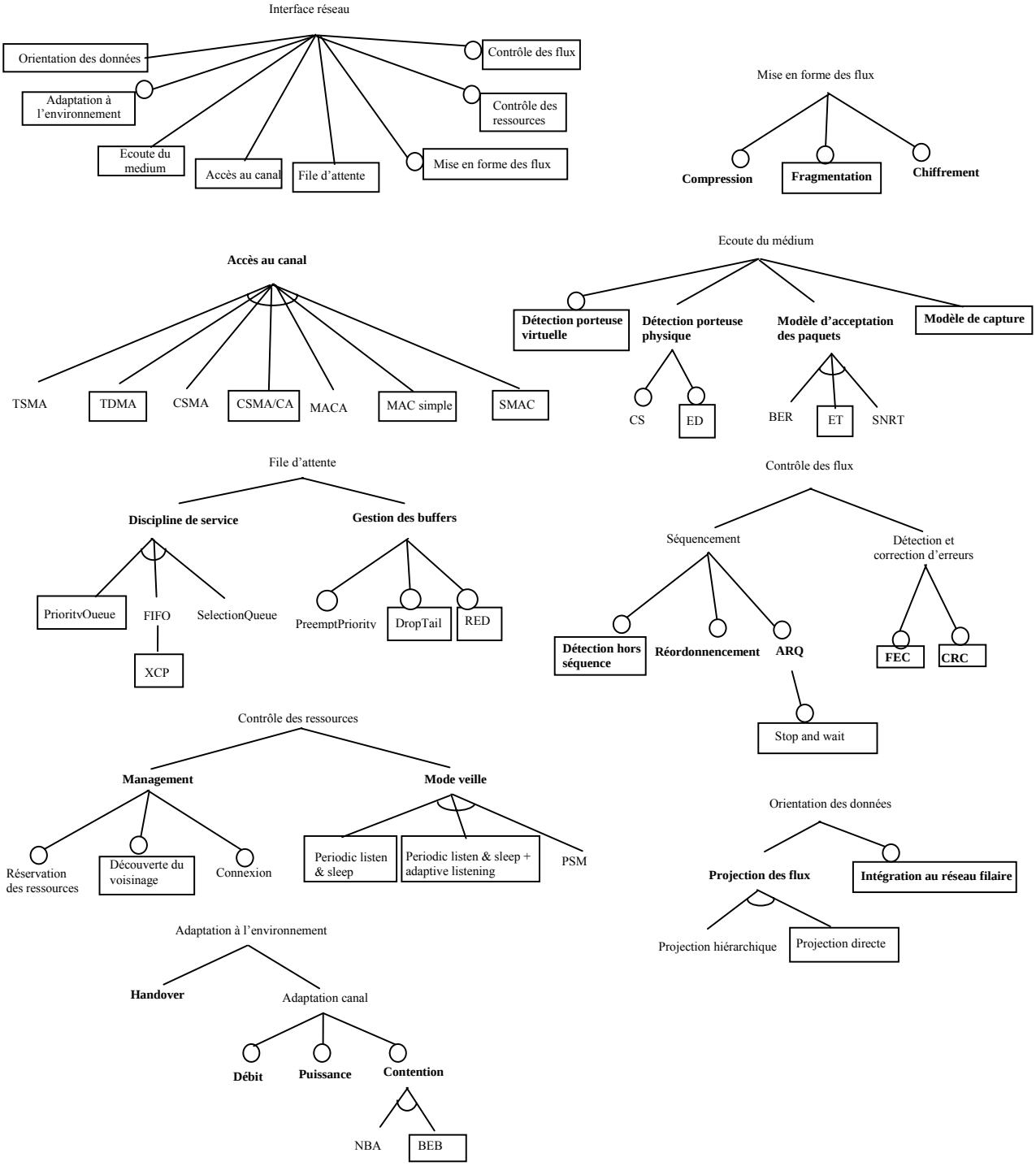


Figure 3.13 Caractéristiques disponibles dans NS

Comme nous pouvons le voir à la Table 3.1, il existe un fort couplage entre la caractéristique d'*accès au canal* et les autres caractéristique implémentées, puisqu'il existe sauf pour les caractéristiques *de file*

d'attente et de *FEC*, aucun autre aspect alternatif ou optionnel transversal aux méthodes d'accès. Ceci s'explique par le fait que NS-2 est un simulateur à grain épais contenant un composant principal très spécialisé, le composant MAC, regroupant autour de l'accès au canal, la majorité des fonctionnalités. Cette analyse nous permet de distinguer, in fine dans NS, 4 produits principaux (TDMA, 802.11, MAC simple ou SMAC) chacun implémentant une technique MAC différente.

Table 3.1 Dépendances inter caractéristiques obligatoires et exclusions sous NS[23]

		Accès au canal			
		TDMA	CSMA/CA	MAC simple	SMAC
Contrôle des ressources	Periodic listen & sleep	⊘	⊘	⊘	a
	Periodic listen & sleep + adaptive listening	⊘	⊘	⊘	
	Découverte du voisinage	⊘	⊘	⊘	×
Contrôle des flux	CRC	×	×	×	×
	FEC	o	o	o	o
	Stop and wait	⊘	×	⊘	×
	Détection hors séquence	⊘	×	⊘	⊘
Mise en forme de flux	Fragmentation	⊘	⊘	⊘	o
Adaptivité à l'environnement	BEB	⊘	×	⊘	⊘
Orientation des flux	Intégration au réseau filaire	×	×	×	×
	Projection directe	×	×	×	×
Ecoute du medium	Détection de porteuse virtuelle	⊘	×	⊘	×
	Détection de porteuse physique ED	⊘	×	⊘	×
	Modèle de capture	⊘	×	×	×
	ET	×	×	×	×
File d'attente	RED	a	a	a	a
	DropTail				
	XCP	a	a	a	a
	PriorityQueue				

La Table 3.2 propose un mapping entre les caractéristiques et les composants qui les hébergent. Il montre également les différentes caractéristiques adaptables de chaque produit[24]. Une fois le mapping réalisé, le niveau d'adaptation de chaque composant peut être calculé selon la formule présentée à la section 4.5 du chapitre 2. Les EAI des composants adaptables sont fournis à la Table 3.3. La valeur des autres éléments est 0. Une fois la valeur d'EAI de tous les éléments de l'architecture disponible, et connaissant le nombre de composants de l'architecture (5 composants), l'AAI de chaque produit et le SAI de la ligne de produits peuvent être calculés. Ceux-ci sont fournis dans la Table 3.3.

[23] Le caractère « ⊘ » indique une caractéristique ne se réalisant jamais à l'exécution car non implémentée, « × » une caractéristique se réalisant toujours, « o » une caractéristique optionnelle et « a » une caractéristique alternative.
[24] La représentation des caractéristiques de la Table 3.2 est proche de la notation introduite [Kan03] sauf que ne sont mentionnées ici que les caractéristiques adaptables.

Table 3.2 Présentation du mapping caractéristiques/composants et des caractéristiques adaptables des produits Mac simple, SMAC, TDMA et 802.11

LL																						
QUEUE	×	×																				
MAC			×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×			
PHY																				×	×	
FEC																						×
	Discipline de service	Gestion des buffers	Accès au canal	Management	Mode veille	Projection des flux	Intégration au réseau filaire	Détection porteuse virtuelle	Compression	Fragmentation	Chiffrement	Handover	Réordonnancement	Détection hors séquence	ARQ	CRC	Modèle de capture	Adaptation contention	Adaptation débit	Adaptation puissance	Détection porteuse physique	Modèle acceptation paquets	FEC
Mac simple	×	×																				×
SMAC	×	×			×					×												×
TDMA	×	×																				×
802.11	×	×																				×

Table 3.3 Calculs des EAI, AAI et SAI sous NS

	EAI(LL)	**EAI(Queue)**	**EAI(MAC)**	**EAI(PHY)**	**EAI(FEC)**	**AAI**	**SAI**
Mac-simple	0	1	0	0	1	40%	41%
SMAC	0	1	2/18	0	1	42%	
TDMA	0	1	0	0	1	40%	
802.11	0	1	0	0	1	40%	

4.1.2. Mesures de modularité

4.1.2.1. Mesures de morphologie et d'impureté

Les mesures de morphologie et d'impureté sont fournies à la Table 3.4.

Table 3.4 Mesures de morphologie et d'impureté de NS

Taille en arcs	5
en noeuds	5
Profondeur	5
Largeur	1
Ratio arcs à noeuds	1
Nombre de dépendances	5
Impureté de l'arbre	0.17

4.1.2.2. Mesures de flots d'information, de couplage et de complexité

Les mesures de flots d'information, de couplage et de complexité sont fournies à la Table 3.5.

Table 3.5 Mesures de flots d'information, de couplage et de complexité de NS

		Parent	Instanciable	LOC	MVG	CBO	Fan-in	Fan-out	IF4
LL	LL	N/A	Oui	166	51	14	4	10	1600
QUEUE	XCPQueue	DropTail	Oui	399	73	8	2	6	144
	PriQueue	DropTail	Oui	103	18	11	5	6	900
	DropTail	Queue	Oui	107	29	5	2	3	36
	REDQueue	Queue	Oui	629	150	11	3	8	576
	Queue	N/A	Non	248	45	28	22	6	17424
	CMUPriQueue	N/A	Oui	271	51	7	1	6	36
MAC	MacTdma	Mac	Oui	323	53	14	4	10	1600
	Mac802_11	Mac	Oui	1047	209	25	9	16	20736
	MacSimple	Mac	Oui	172	27	11	4	7	784
	SMAC	Mac	Oui	1654	401	24	10	14	19600
	Mac	N/A	Non	155	38	26	14	12	28224
FEC	FECModel	N/A	Oui	66	13	3	0	0	0
PHY	WirelessPhy	Phy	Oui	392	118	14	7	7	2401
	Phy	N/A	Non	109	28	17	10	7	4900

4.2. Glomosim

Glomosim [Net-Glomo] repose comme NS sur un modèle à grain épais (voir Figure 3.14). Sous Glomosim, les modules ne possédant pas de temporisateurs propres, la gestion des évènements est déléguée au moteur de simulation étroitement intégré aux composants réseaux. De fait, Glomosim apparaît essentiellement comme une architecture avec programme principal et sous routine dans laquelle le programme principal – c'est-à-dire le moteur de simulation – appelle des routines en cascade dans les composants de simulation. Glomosim s'inspire fortement de l'architecture en couches avec la définition de services génériques clairs pour permettre la communication entre les couches du modèle OSI.

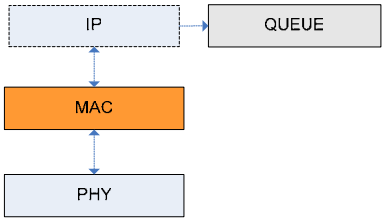


Figure 3.14 Diagramme en couches de Glomosim

Le simulateur intègre 4 implémentations de couche MAC pour les réseaux informatiques ad-hoc (802.11, CSMA, MACA et TSMA). L'intégration au réseau filaire se limite à l'interconnexion d'un ou plusieurs réseaux ad-hoc à travers un réseau câblé. L'interconnexion est toutefois prise en charge an niveau IP et un routage statique doit être utilisé. Le module de file d'attente et la couche physique sont communs à tous les systèmes. Deux instances de couches physiques peuvent être utilisées indépendamment de la technologie de niveau 2. La première prend en compte la puissance cumulée des interférences dues aux transmissions parasites. La seconde ne considère que la puissance du dernier paquet reçu.

4.2.1. Mesures de généralité

La Figure 3.15 décrit les caractéristiques disponibles sous Glomosim.

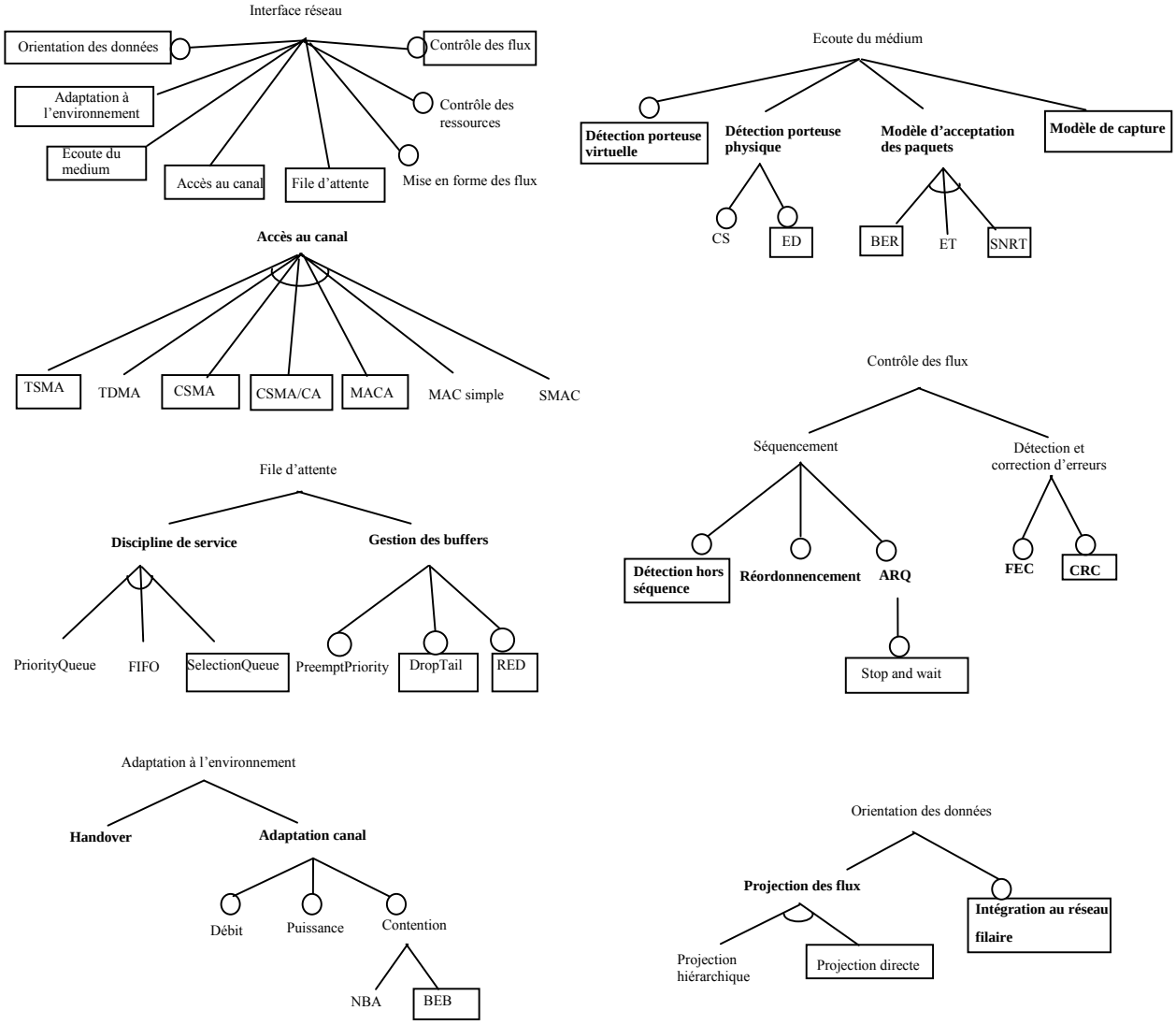


Figure 3.15 Caractéristiques disponibles dans Glomosim

Comme le montre la Table 3.6, il existe dans Glomosim un fort couplage entre la caractéristique d'*accès au canal* et les autres caractéristiques implémentées, les seules caractéristiques adaptables indépendamment de l'accès au canal étant *le modèle d'acceptation des paquets* et la *gestion des buffers*. La variabilité est implémentée à ce niveau grâce à un mécanisme de paramétrage générique de la structure de file d'attente proposée. Le couplage existant entre les caractéristiques s'explique par le fait que Glomosim repose, comme NS, sur une architecture à grain épais avec un composant principal, le composant MAC implémentant la quasi-totalité des fonctionnalités. Ce constat nous amène à définir 4 produits (MACA, CSMA, 802.11, TSMA) identifiables par leur technique d'accès au canal.

Table 3.6 Dépendances inter caractéristiques obligatoires et exclusions sous Glomosim[25]

		Accès au canal			
		MACA	CSMA	CSMA/CA	TSMA
Contrôle des flux	Stop and wait	⦸	⦸	×	×
	CRC	⦸	⦸	×	⦸
	Détection hors séquence	⦸	⦸	×	×
Adaptivité à l'environnement	BEB	×	×	×	⦸
Ecoute du médium	Détection porteuse physique ED	⦸	×	×	⦸
	Détection porteuse virtuelle	×	⦸	×	⦸
	Modèle de capture	×	×	×	×
	BER	a	a	a	a
	SNRT				
File d'attente	RED	o	o	o	o
	DropTail	×	×	×	×
	SelectionQueue	×	×	×	×
Orientation des données	Projection directe	×	×	×	×
	Intégration au réseau filaire	×	×	×	×

La Table 3.7 propose le mapping entre les caractéristiques et les composants qui les hébergent et montre les différentes caractéristiques adaptables de chaque produit.

Table 3.7 Présentation du mapping caractéristiques/composants et des caractéristiques adaptables des produits MACA, CSMA, 802.11 et TSMA

QUEUE	×	×																					
MAC			×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×				
PHY																				×	×	×	×
	Discipline de service	Gestion des buffers	Accès au canal	Management	Mode veille	Projection des flux	Intégration au réseau filaire	Détection porteuse virtuelle	Compression	Fragmentation	Chiffrement	Handover	Réordonnancement	Détection hors séquence	ARQ	CRC	Adaptation contention	Adaptation puissance	Adaptation débit	Modèle acceptation paquets	Détection porteuse physique	Modèle de capture	FEC
MACA		×																		×			
CSMA		×																		×			
802.11		×																		×			
TSMA		×																		×			

Les EAI des composants adaptables sont déduits du tableau précédent. Les résultats sont fournis à la Table 3.8. Connaissant, l'EAI de chaque composant et le nombre total de composants définis dans l'architecture (3 composants), l'AAI de chaque produit et le SAI de la ligne de produits peuvent être calculés. Les valeurs obtenues sont présentées à la Table 3.8.

[25] Le caractère « ⦸ » indique une caractéristique ne se réalisant jamais à l'exécution car non implémentée, « × » une caractéristique se réalisant toujours, « o » une caractéristique optionnelle et « a » une caractéristique alternative.

Table 3.8 Calculs des EAI, AAI et SAI sous Glomosim

	EAI(Queue)	EAI(MAC)	EAI(PHY)	AAI	SAI
Mac-simple	0,5	0	0,25	25%	25%
SMAC	0,5	0	0,25	25%	
TDMA	0,5	0	0,25	25%	
802.11	0,5	0	0,25	25%	

4.2.2. Mesures de modularité

4.2.2.1. Mesures de morphologie et d'impureté

Les mesures de morphologie et d'impureté sont fournies à la Table 3.9. La mesure de morphologie prend en compte le module IP.

Table 3.9 Mesures de morphologie et d'impureté de Glomosim

Taille en arcs	3
en noeuds	4
Profondeur	3
Largeur	2
Ratio arcs à noeuds	1
Nombre de dépendances	3
Impureté de l'arbre	0

4.2.2.2. Mesures de flots d'information, de couplage et de complexité

Glomosim étant implémenté en C, les composants de l'architecture ne possède pas de relation d'héritage. Toutefois, les fichiers csma.pc, maca.pc tsma.pc et 802_11.pc pouvant être assimilés à des spécialisations de mac.pc, nous avons étendu dans la Table 3.10 la signification des colonnes Parent et Instanciable au langage C, dans le respect de la logique et de l'étymologie.

Les mesures de flots d'information, de couplage et de complexité sont fournies à la Table 3.10.

Table 3.10 Mesures de complexité de Glomosim

		Parent	Instanciable	LOC	MVG
QUEUE	fifoqueue	N/A	Oui	326	43
MAC	csma	Mac	Oui	267	32
	maca	Mac	Oui	546	54
	tsma	Mac	Oui	688	78
	802_11	Mac	Oui	1074	126
	mac	N/A	Non	280	54
PHY	radio_nonoise	Radio	Oui	376	30
	radio_accnoise	Radio	Oui	505	58
	radio	N/A	Non	323	44

4.3. J-Sim

J-Sim [Net-J-Sim] est un simulateur à grain épais à base de composants. Chaque composant est contrôlé par un processus léger et dispose de ports d'entrée sortie, qui une fois connectés par l'utilisateur, permettent l'envoi et la réception de messages. L'interconnexion des composants réalise une architecture à composants indépendants à base de processus communicants. Des interfaces claires définissent la liste des messages échangeables entre chaque composant.

De toutes les technologies WLAN, J-Sim ne modélise que le mode ad-hoc de Wi-Fi. L'implémentation de 802.11 est un portage de la version 2.23a de NS. Celui-ci intègre, en plus, un mécanisme gérant l'accumulation du bruit proche de celui de Glomosim ainsi qu'un mécanisme de gestion de l'énergie de type PSM pour permettre aux stations en mode économie d'énergie de pouvoir recevoir leurs paquets. Comme Glomosim, l'intégration au réseau filaire est prise en charge au niveau IP et se limite au raccord des réseaux ad-hoc par voie câblée. Du fait de sa parenté avec NS-2, J-Sim reproduit la même architecture (voir Figure 3.16). Plusieurs types de file d'attente sont disponibles (FIFO, DropTail, PriorityQueue, PreemptPriorityQueue, RED, etc).

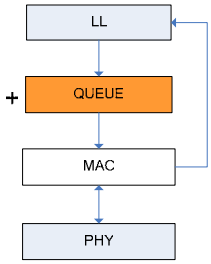
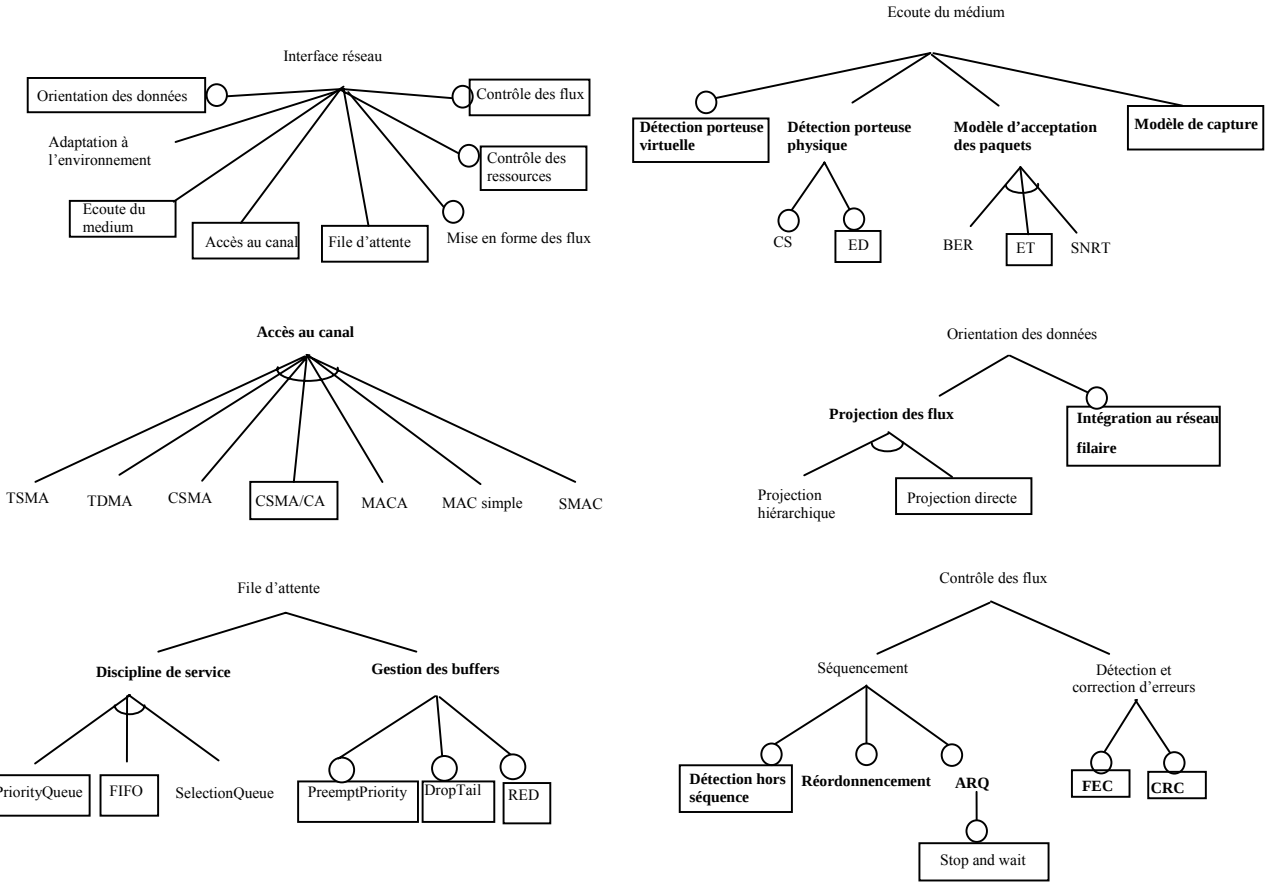


Figure 3.16 Diagramme en couches de J-Sim

4.3.1. Mesures de généralité

La Figure 3.17 décrit les caractéristiques disponibles sous J-Sim.



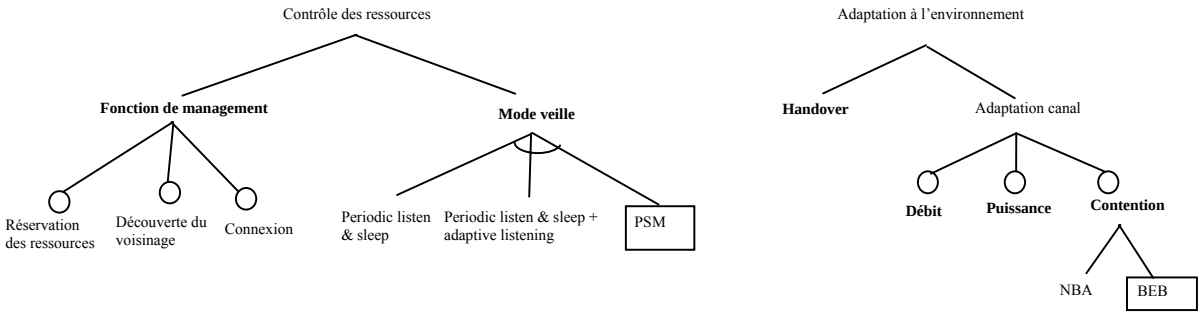


Figure 3.17 Caractéristiques disponibles dans J-Sim

Comme le montre la Table 3.11, il existe un fort couplage entre la caractéristique d'*accès au canal* et les autres caractéristiques implémentées, les seules caractéristiques pouvant être adaptées étant le *mode veille* et la *file d'attente*. Dans J-Sim, comme dans NS-2 et Glomosim, la majorité des fonctionnalités sont effectivement fixées dans le composant MAC sans ne plus pouvoir être modifiées.

Table 3.11 Dépendances inter caractéristiques obligatoires et exclusions sous J-Sim[26]

		Accès au canal
		CSMA/CA
Contrôle des ressources	PSM	o
Contrôle des flux	CRC	×
	Stop and wait	×
	Détection hors séquence	×
Adaptation à l'environnement	BEB	×
Orientation des flux	Projection directe	×
	Intégration au réseau filaire	×
Ecoute du medium	Détection de porteuse virtuelle	×
	Détection de porteuse physique ED	×
	Modèle de capture	×
	ET	×
File d'attente	PreemptPriority	a
	RED	
	DropTail	
	FIFO	a
	PriorityQueue	

La Table 3.12 propose le mapping caractéristiques/composants et montre les différentes caractéristiques adaptables du produit 802.11 sous J-Sim.

[26] Le caractère « ⊘ » indique une caractéristique ne se réalisant jamais à l'exécution car non implémentée, « × » une caractéristique se réalisant toujours, « o » une caractéristique optionnelle et « a » une caractéristique alternative.

Table 3.12 Présentation du mapping caractéristiques/composants et des caractéristiques adaptables du produit 802.11 sous J-Sim

	Discipline de service	Gestion des buffers	Accès au canal	Management	Mode veille	Projection des flux	Intégration au réseau filaire	Détection porteuse virtuelle	Compression	Fragmentation	Chiffrement	Handover	Réordonnancement	Détection hors séquence	ARQ	CRC	Modèle de capture	Adaptation contention	Adaptation débit	Adaptation puissance	Détection porteuse physique	Modèle d'acceptation paquet	FEC
LL																							
QUEUE	×	×																					
MAC			×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×			
PHY																					×	×	×
802.11	×	×			×																		

Une fois le mapping réalisé, le niveau d'adaptation de chaque composant peut être calculé selon la formule présentée à la section 4.5 du chapitre 2. Les EAI des composants adaptables sont fournis à la Table 3.13. La valeur des autres éléments est 0. Une fois la valeur d'EAI de tous les éléments de l'architecture disponible, et connaissant le nombre de composants de l'architecture (4 composants), l'AAI de chaque produit et le SAI de la ligne de produits peuvent être calculés. Ceux-ci sont donnés à la Table 3.13.

Table 3.13 Calculs des EAI, AAI et SAI sous J-Sim

	EAI(LL)	EAI(Queue)	EAI(MAC)	EAI(PHY)	AAI	SAI
802.11	0	1	1/18	0	26%	26%

4.3.2. Mesures de modularité

4.3.2.1. Mesures de morphologie et d'impureté

Les mesures de morphologie et d'impureté de J-Sim sont fournies à la Table 3.14.

Table 3.14 Mesures de modularité de morphologie et d'impureté de J-Sim

Taille en arcs	4
en noeuds	4
Profondeur	4
Largeur	1
Ratio arcs à noeuds	1
Nombre de dépendances	4
Impureté de l'arbre	0,33

4.3.2.2. Mesures de flots d'information, de couplage et de complexité

Les mesures de flots d'information, de couplage et de complexité de J-Sim sont présentées à la Table 3.15.

Table 3.15 Mesures de flots d'information, de couplage et de complexité de J-Sim

		Parent	Instanciable	LOC	MVG	CBO	Fan-in	Fan-out	IF4
LL	LL	N/A	Oui	101	15	4	0	4	0
QUEUE	FIFO	Queue	Oui	116	29	5	0	0	0
	PreemptPriorityQueue	Queue	Oui	74	37	3	0	3	0
	PriorityQueue	Queue	Oui	225	60	10	1	9	81
	Queue	ActiveQueue	Non	79	0	46	38	8	92416
	ActiveQueue	N/A	Non	104	14	6	2	4	64
MAC	Mac802_11	N/A	Oui	1604	402	37	11	26	81796
PHY	WirelessPhy	N/A	Oui	462	95	10	0	10	0

4.4. YANS

Une implémentation hautement modulaire du standard 802.11 est proposée par l'INRIA dans le projet YANS [Net-YANS] à la base du futur mode Wi-Fi de NS-3[27]. YANS fonctionne sous NS-2 et s'appuie essentiellement sur une architecture orientée objet. Des interfaces claires définissent la liste des messages échangeables entre chaque module. A la manière des systèmes à évènements, un mécanisme d'écoute astucieux (Listener, en rouge sur les Figures 3.18 et 3.19) permet à un composant de s'enregistrer temporairement auprès d'un autre pour recevoir des notifications d'évènements.

Comme nous pouvons le constater, la structure globale de YANS possède quelques similarités avec l'implémentation du driver Madwifi avec un partitionnement MacLow/MacHigh assez proche de la découpe HAL/driver Madwifi sous Linux et l'intégration de certaines constantes DCF (comme les seuils SSRC et SLRC) dans les files d'attente. YANS propose une implémentation des protocoles 802.11a/b/e. Trois architectures sont définies en fonction du mode simulé[28].

- Une architecture commune est définie pour les STAs ad-hoc, les STAs et les APs non QoS (voir Figure 3.18).
- Deux architectures sont définies, une côté QAP, une côté QSTA pour la simulation des modes EDCA et HCCA de 802.11e (voir Figure 3.19).

Le composant MAC High réalise les fonctions de gestion (association, synchronisation, authentification, etc) et assure le relais des données. Le composant MAC Low gère les fonctions de bas niveau : génération des trames de contrôle et timing SIFS. Le composant MACRxMiddle gère la défragmentation et l'élimination des doublons.

L'implémentation de l'accès au médium dépend de la technologie :

- En mode 802.11, les STAs ad-hoc, les STAs et les APs non QoS intègrent une station DCF constituée d'un composant DCATXOP, d'un composant DCF et d'une file d'attente MacQueue80211e (voir Figure 3.18) de type FIFO. Le rôle du composant DCF est d'obtenir l'accès au canal pour transmettre les trames de données. Il maintient pour cela la gestion du backoff, la progression de la fenêtre de contention, le choix de l'intertrame DIFS/EIFS et le NAV. Le composant DCATXOP fragmente les

[27] La version du projet étudié est celle du 07/09/2005.
[28] Le mode infrastructure ne semble pas encore fonctionnel. L'architecture des STA et des AP devrait se rapprocher de celle du mode adhoc.

paquets si nécessaire, fournit à MAC Low les données à transmettre et met à jour le composant DCF lors de la réception des ACK et des CTS.

- En mode 802.11e, le trafic en entrée est soit dirigé dans les AC du mode EDCA soit dans les files gérant les flots HCCA (voir Figure 3.19). Les AC BE, BK, VI et VO représentent respectivement le système de files d'attente à priorité EDCF avec les catégories d'accès best effort, arrière plan, voix et vidéo. Une AC est composée, à la manière d'une station ad-hoc, d'un composant DCF, d'une file MacQueue80211e et d'un EDCATXOP. Le composant EDCATXOP prend en compte les spécificités du mode d'accès EDCA pour contrôler le composant DCF.

  Plus spécifiquement, côté QAP, le QAPscheduler réalise le contrôle d'admission, alloue les supertrames (envoi des beacons), les périodes CAP du mode HCCA et les TXOP à chaque station (envoi des paquets CFPoll). Le QAPscheduler possède également un pointeur sur la couche physique pour récupérer les paramètres (débit physique, durée des intertrames, etc) nécessaires à son fonctionnement.

Trois instances de couche PHY sont disponibles.

- PHY-ET est la plus simple. Le paquet est correctement décodé si sa puissance est supérieure au seuil de réception et si aucun paquet ne parvient à la couche PHY durant sa réception.
- PHY-SNR agit comme PHY-ET mais vérifie en plus à la fin de la réception du paquet que le SNIR (rapport signal à interférence) est supérieur à un seuil.
- PHY-BER est l'instance la plus compliquée. Elle recalcule, en tenant code du codage utilisé pour transmettre le paquet, la probabilité d'erreur sur chaque portion du paquet à décoder à chaque fois que le niveau d'interférence varie. Le paquet est jeté s'il contient une erreur[29].

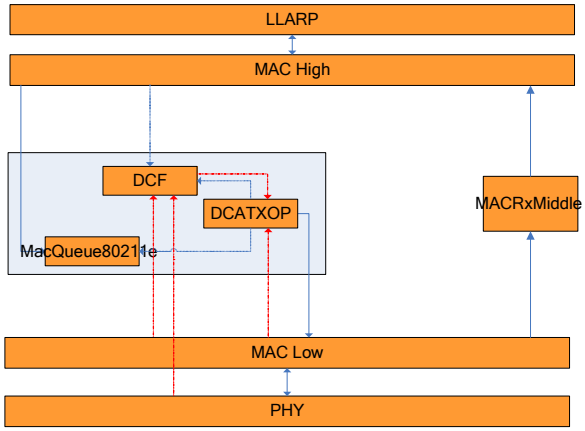
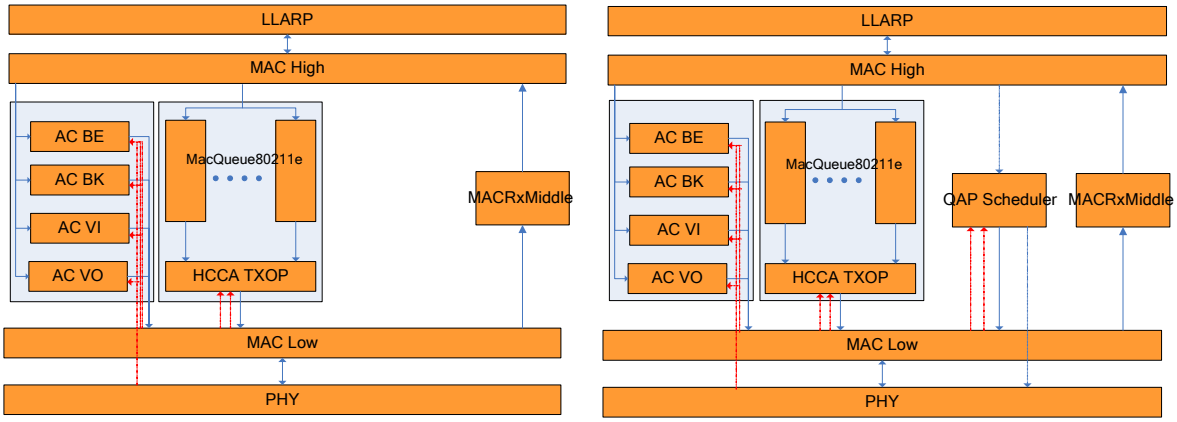


Figure 3.18 Architecture des stations ad-hoc des NQAPs et NQSTAs (stations non QoS)

[29] ou un nombre d'erreurs trop important pour être corrigé par la FEC, si celle-ci est active

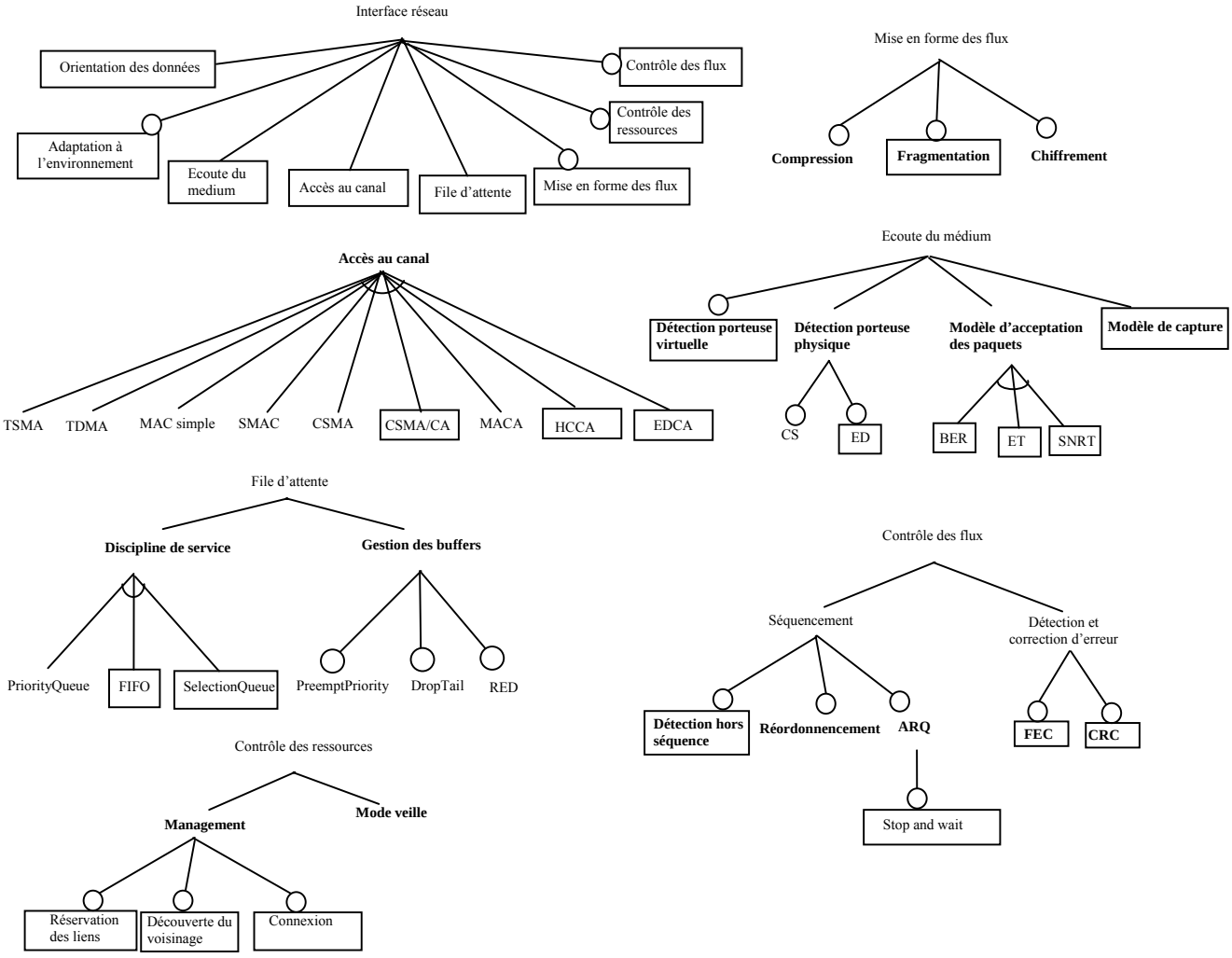


(a) Architecture des QSTAs (b) Architecture des QAPs

Figure 3.19 Architecture à qualité de service

4.4.1. Mesures de généralité

La Figure 3.20 décrit les caractéristiques disponibles sous YANS.



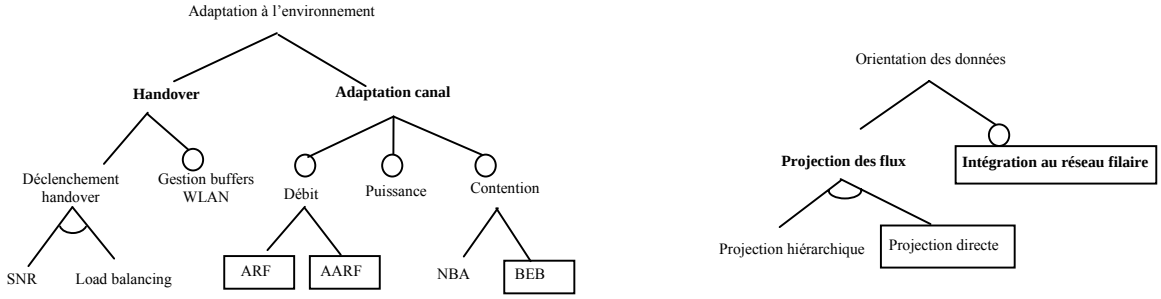


Figure 3.20 Caractéristiques disponibles sous YANS

Comme le montre la Table 3.16, il existe une corrélation entre le choix d'un composant MacHigh et celui des éléments DCATXOP, EDCATXOP, HCCATXOP et QAPScheduler. Cette corrélation est cependant limitée à ces 5 éléments architecturaux.

Table 3.16 Corrélation entre les composants Mac High et les autres éléments de l'architecture

	MACHighadhoc	**MACHighNQSTA**	**MACHighNQAP**	**MACHighQSTA**	**MACHighQAP**
DCATXOP	×	×	×		
EDCATXOP				×	×
HCCATXOP				×	×
QAPScheduler					×

Dans YANS, il est donc possible d'adapter l'architecture du simulateur en changeant le type du composant MacHigh. Pour que la mesure d'adaptation reflète l'effet de cette technique, nous considérons optionnelles donc adaptables, toutes les caractéristiques implémentées dans les composants dont la présence est corrélée à l'architecture choisie.

Pour mettre en valeur les possibilités de réutilisation des composants non corrélés entre eux, nous avons introduit, un niveau conceptuel intermédiaire, baptisé *Architecture*, entre la ligne de produits et les variantes disponibles. Ce formalisme est inspiré de la référence [SVA99].

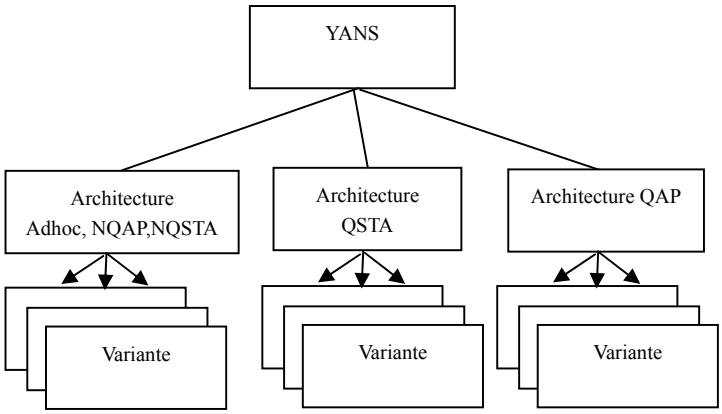


Figure 3.21 Hiérarchie de la ligne de produit

Ainsi, la caractéristique *Management* est considérée adaptable (voir Table 3.18) car sa sous caractéristique, *réservation des liens*, bien que fixée dans les trois architectures (voir Table 3.17), est implémentée dans l'architecture des QAP et QSTA mais pas dans les autres.

Table 3.17 Dépendances inter caractéristiques obligatoires et exclusions sous YANS[30]

		Architecture		
		Ad-hoc,NQSTA,NQAP	QAP	QSTA
Contrôle des ressources	Réservation des liens	⊘	×	×
	Découverte du voisinage	×[31]	×	×
	Connexion	×[32]	×	×
Contrôle des flux	Détection hors séquence	×	×	×
	FEC	o	o	o
	Stop and wait	×	o	o
	CRC	×	×	×
Mise en forme de flux	Fragmentation	o	⊘	⊘
Orientation des flux	Intégration au réseau filaire	×	×	×
	Projection directe	×	×	×
Ecoute du medium	Détection de porteuse virtuelle	×	×	×
	Détection de porteuse physique ED	×	×	×
	Mode de capture	×	×	×
	SNRT	a	a	a
	ET			
	BER			
Adaptation à l'environnement	AARF	a,o	a,o	a,o
	ARF			
	BEB	×	×[33]	×[34]
File d'attente	FIFO	×	⊘	⊘
	SelectionQueue	⊘	×	×
Accès au canal	CSMA/CA	×	⊘	⊘
	EDCA	⊘	a	a
	HCCA	⊘		

[30] Le caractère « ⊘ » indique une caractéristique ne se réalisant jamais à l'exécution car non implémentée, « × » une caractéristique se réalisant toujours, « o » une caractéristique optionnelle et « a » une caractéristique alternative.
[31] NQAP et NQSTA seulement.
[32] NQAP et NQSTA seulement.
[33] Ne s'exécute pas pendant le mode HCCA
[34] Ne s'exécute pas pendant le mode HCCA

Table 3.18 Présentation du mapping caractéristiques/composants et des caractéristiques adaptables sous YANS

	Accès au canal	ARQ	Chiffrement	Discipline de service	Gestion des buffers	Réordonnancement	CRC	Détection porteuse virtuelle	Intégration au réseau filaire	Fragmentation	Détection porteuse physique	Mode le capture	Modèle acceptation paquets	Management	Compression	Adaptation débit	Adaptation contention	Adaptation puissance	Détection hors séquence	Projection des flux	Mode veille	Handover	FEC
MACRxMiddle			×							×					×				×				
Phy											×	×	×										×
MacHighAdhoc				×					×					×						×	×	×	
MacHighQAP				×					×					×						×	×	×	
MacHighQSTA				×					×					×						×	×	×	
MacHighNQAP				×					×					×						×	×	×	
MacHighNQSTA				×					×					×						×	×	×	
MacLow							×	×															
LLARP																							
MacQueue80211e				×	×																		
DCATXOP	×	×	×							×					×	×							
EDCATXOP	×	×	×							×					×	×							
HCCATXOP	×	×	×							×					×	×							
QAPScheduler	×																						
DCF	×																×	×					
Caractéristiques adaptables	×	×		×						×			×	×		×							×

Les valeurs de EAI, AAI et SAI sont déduites de la Table 3.18. Celles-ci sont présentées à la Table 3.19.

Table 3.19 Calculs des EAI, AAI et SAI sous YANS

	EAI(MACRxMiddle)	**EAI(Phy)**	**EAI(MacHigh)**	**EAI(MacLow)**	**EAI(LLARP)**	**EAI(MacQueue80211e)**	**EAI(DCATXOP)**	**EAI(EDCATXOP)**	**EAI(HCCATXOP)**	**EAI(QAPScheduler)**	**EAI(DCF)**	**AAI**	**SAI**
Architecture adhoc, NQSTA,NQAP	1/3	1/2	1/3	0	0	1/2	2/3				1/3	33%	38%
Architecture QSTA	1/3	1/2	1/3	0	0	1/2		2/3	2/3		1/3	37%	
Architecture QAP	1/3	1/2	1/3	0	0	1/2		2/3	2/3	1	1/3	43%	

4.4.2. Mesures de modularité

4.4.2.1. Mesures de morphologie et d'impureté

La Table 3.20 contient les mesures de morphologie et d'impureté de YANS. Certains modules sont regroupés dans des modules composites comme indiqués aux Figures 3.18 et 3.19.

Table 3.20 Mesures de morphologie et d'impureté de YANS

	Architecture		
	Adhoc,NQSTA,NQAP	QSTA	QAP
Taille en arcs	10	13	18
en noeuds	6	7	8
Profondeur	5	5	5
Largeur	2	3	4
Ratio arcs à noeuds	1.5	1.8	2.25
Nombre de dépendances	7	9	12
Impureté de l'arbre	0.2	0.2	0,24

4.4.2.2. Mesures de flots d'information, de couplage et de complexité

Les Tables 3.21 à 3.25 montrent les résultats obtenus pour les stations ad-hoc et les stations/points d'accès QoS/non QoS.

Table 3.21 Mesures de flots d'information, de couplage et de complexité des stations ad hoc

		Parent	**Instanciable**	**LOC**	**MVG**	**CBO**	**Fan-in**	**Fan-out**	**IF4**
LL	LLArp	N/A	Oui	209	25	8	2	6	144
MAC	MacLow	N/A	Oui	611	83	11	2	9	324
	MacHighAdhoc	MacHigh	Oui	63	1	6	0	6	0
	MacHigh	N/A	Non	16	0	7	5	2	100
	MacRxMiddle	N/A	Oui	160	32	5	1	4	16
	Dcf	N/A	Oui	361	38	18	9	9	6561
	DcaTxop	N/A	Oui	296	42	7	2	5	100
	MacQueue80211e	N/A	Oui	111	14	13	10	3	900
PHY	PhyEt	Phy	Oui	74	6	4	0	4	0
	PhySnrt	Phy	Oui	127	12	5	0	5	0
	PhyBer	Phy	Oui	214	27	10	1	8	64
	Phy80211	N/A	Non	471	65	15	5	10	2500

Table 3.22 Mesures de flots d'information, de couplage et de complexité des NQSTA

		Parent	Instanciable	LOC	MVG	CBO	Fan-in	Fan-out	IF4
LL	LLArp	N/A	Oui	209	25	8	2	6	144
MAC	MacLow	N/A	Oui	611	83	11	2	9	324
	MacHighNqsta	MacHighSta	Oui	69	0	7	0	7	0
	MacHighSta	MacHigh	Non	235	28	7	2	5	100
	MacHigh	N/A	Non	16	0	7	5	2	100
	MacRxMiddle	N/A	Oui	160	32	5	1	4	16
	Dcf	N/A	Oui	361	38	18	9	9	6561
	DcaTxop	N/A	Oui	296	42	7	2	5	100
	MacQueue80211e	N/A	Oui	111	14	13	10	3	900
PHY	PhyEt	Phy	Oui	74	6	4	0	4	0
	PhySnrt	Phy	Oui	127	12	5	0	5	0
	PhyBer	Phy	Oui	214	27	10	1	8	64
	Phy80211	N/A	Non	471	65	15	5	10	2500

Table 3.23 Mesures de flots d'information, de couplage et de complexité des NQAP

		Parent	Instanciable	LOC	MVG	CBO	Fan-in	Fan-out	IF4
LL	LLArp	N/A	Oui	209	25	8	2	6	144
MAC	MacLow	N/A	Oui	611	83	11	2	9	324
	MacHighNqap	MacHighAp	Oui	134	3	8	0	8	0
	MacHighAp	MacHigh	Non	235	28	7	2	5	100
	MacHigh	N/A	Non	16	0	7	5	2	100
	MacRxMiddle	N/A	Oui	160	32	5	1	4	16
	Dcf	N/A	Oui	361	38	18	9	9	6561
	DcaTxop	N/A	Oui	296	42	7	2	5	100
	MacQueue80211e	N/A	Oui	111	14	13	10	3	900
PHY	PhyEt	Phy	Oui	74	6	4	0	4	0
	PhySnrt	Phy	Oui	127	12	5	0	5	0
	PhyBer	Phy	Oui	214	27	10	1	8	64
	Phy80211	N/A	Non	471	65	15	5	10	2500

Table 3.24 Mesures de flots d'information, de couplage et de complexité des QSTA

		Parent	Instanciable	LOC	MVG	CBO	Fan-in	Fan-out	IF4
LL	LLArp	N/A	Oui	209	25	8	2	6	144
MAC	MacLow	N/A	Oui	611	83	11	2	9	324
	MacHighQsta	MacHighSta	Oui	250	28	11	0	11	0
	MacHighSta	MacHigh	Non	235	28	7	2	5	100
	MacHigh	N/A	Non	16	0	7	5	2	100
	MacRxMiddle	N/A	Oui	160	32	5	1	4	16
	Dcf	N/A	Oui	361	38	18	9	9	6561
	EdcaTxop	N/A	Oui	296	42	7	2	5	100
	MacQueue80211e	N/A	Oui	111	14	13	10	3	900
PHY	PhyEt	Phy	Oui	74	6	4	0	4	0
	PhySnrt	Phy	Oui	127	12	5	0	5	0
	PhyBer	Phy	Oui	214	27	10	1	8	64
	Phy80211	N/A	Non	471	65	15	5	10	2500

Table 3.25 Mesures de flots d'information, de couplage et de complexité des QAP

		Parent	Instanciable	LOC	MVG	CBO	Fan-in	Fan-out	IF4
LL	LLArp	N/A	Oui	209	25	8	2	6	144
MAC	MacLow	N/A	Oui	611	83	11	2	9	324
	MacHighQap	MacHighAp	Oui	292	16	12	0	4	0
	MacHighAp	MacHigh	Non	235	28	7	2	5	100
	MacHigh	N/A	Non	16	0	7	5	2	100
	MacRxMiddle	N/A	Oui	160	32	5	1	4	16
	Dcf	N/A	Oui	361	38	18	9	9	6561
	EdcaTxop	N/A	Oui	296	42	7	2	5	100
	MacQueue80211e	N/A	Oui	111	14	13	10	3	900
PHY	PhyEt	Phy	Oui	74	6	4	0	4	0
	PhySnrt	Phy	Oui	127	12	5	0	5	0
	PhyBer	Phy	Oui	214	27	10	1	8	64
	Phy80211	N/A	Non	471	65	15	5	10	2500

4.5. Analyse des résultats

4.5.1. Mesures de généralité

Les Figures 3.22 a. et b. comparent les SAI et le nombre moyen de caractéristiques adaptables de chaque simulateur. YANS est le simulateur offrant la meilleure flexibilité. Ces résultats reflètent le fait que YANS possède le plus de caractéristiques adaptables. Le simulateur possède en contrepartie un plus grand nombre de composants, dont certains non adaptables, d'où une flexibilité moins bonne que celle attendue. J-Sim prend la deuxième place grâce à la flexibilité de son module de file d'attente qui offre à la fois le choix de la discipline de service et de la gestion des buffers. Glomosim, pénalisé par la faible adaptabilité de sa file d'attente, évite la dernière place grâce à sa couche physique flexible. NS-2 privé de son module de FEC obtient la dernière place[35]. A noter que tous les simulateurs, sauf YANS, pâtissent du manque de flexibilité de leurs modules MAC.

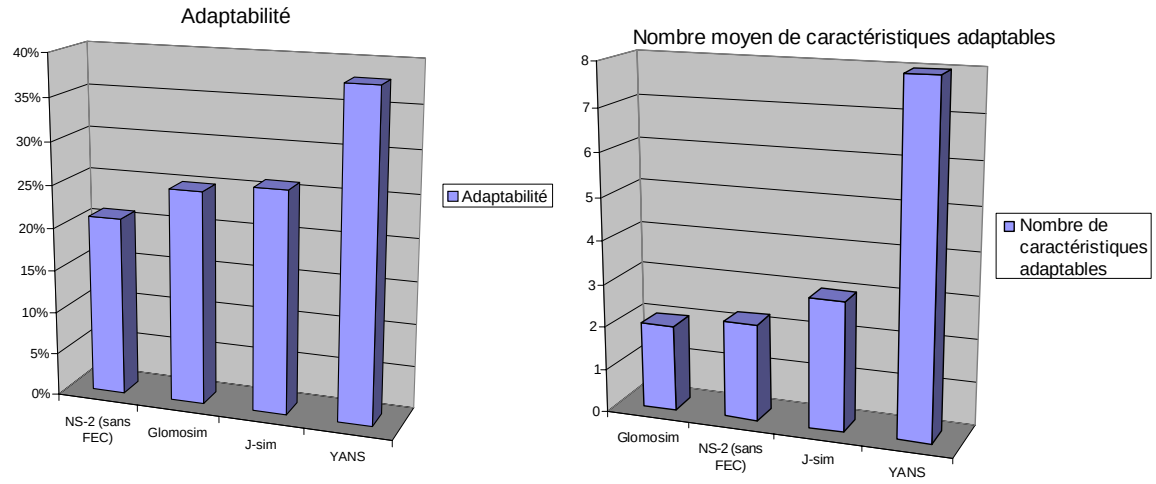


Figure 3.22 a. et b. Comparaison des mesures de généralité de Glomosim, J-Sim, NS-2 et YANS

[35] Avec son module de FEC, NS-2 gagnerait 20% de flexibilité et prendrait la première place. Dans les simulateurs comportant peu de modules, l'ajout d'un seul module peut sensiblement modifier la note. La mesure d'adaptabilité est donc moins représentative pour cette classe de simulateur.

4.5.2. Mesures de modularité

Les Figures 3.23 et 3.24 montrent le coût de la généralité en terme taille, d'impureté, de nombre de lignes de code, de complexité, de couplage et de flots d'information moyens. La moyenne est réalisée sur les produits/architectures de chaque simulateur.

Comme l'indique la Figure 3.23, YANS est le simulateur disposant du plus grand nombre de modules et de connectiques. A noter toutefois que la mesure d'impureté, qui rappelons le, reflète la complexité de la connectique, est du même ordre de grandeur que dans les autres simulateurs. Ces bonnes valeurs s'expliquent par le fait que les calculs sur YANS sont réalisés sur des modules composites cachant certaines dépendances internes, et que le décompte des dépendances ne considère pas les arcs multiples entre composants paires.

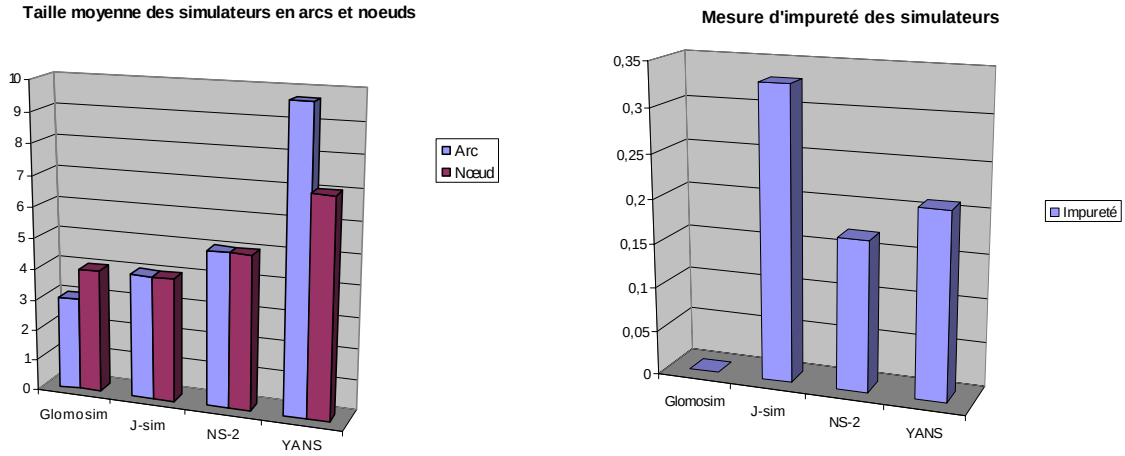


Figure 3.23 a. et b. Taille en arcs et nœuds et impureté moyennes de Glomosim, J-Sim, NS-2 et YANS

Comme le montre la Figure 3.24, Glomosim possède l'implémentation la plus légère. Le simulateur n'intègre pas en effet de couche LL et l'implémentation des files d'attente est très concise. NS-2 possède en revanche un module LL et un grand nombre de types de files ce qui augmente la taille de son code. Grâce à la simplicité de ses modèles, Glomosim possède de façon générale une complexité inférieure à NS. Le module 802.11 de Glomosim exhibe en particulier une complexité quasi deux fois inférieure à celle de NS-2. J-Sim se distingue particulièrement par sa valeur de flots d'information. Le responsable de ce pic est la couche MAC qui utilise un grand nombre de classes additionnelles externes pour réaliser la gestion de l'énergie. De façon assez surprenante, YANS possède une taille[36], une complexité et un couplage inférieurs à NS. Les mesures de couplage et de flots d'information apportent des informations particulièrement intéressantes puisque elles indiquent que, contrairement à ce qu'on pourrait attendre, l'architecture de YANS utilise un nombre restreint de classes annexes.

[36] La mesure de taille ne prend naturellement pas en compte le code des classes Listeners liées aux différents modules de l'architecture.

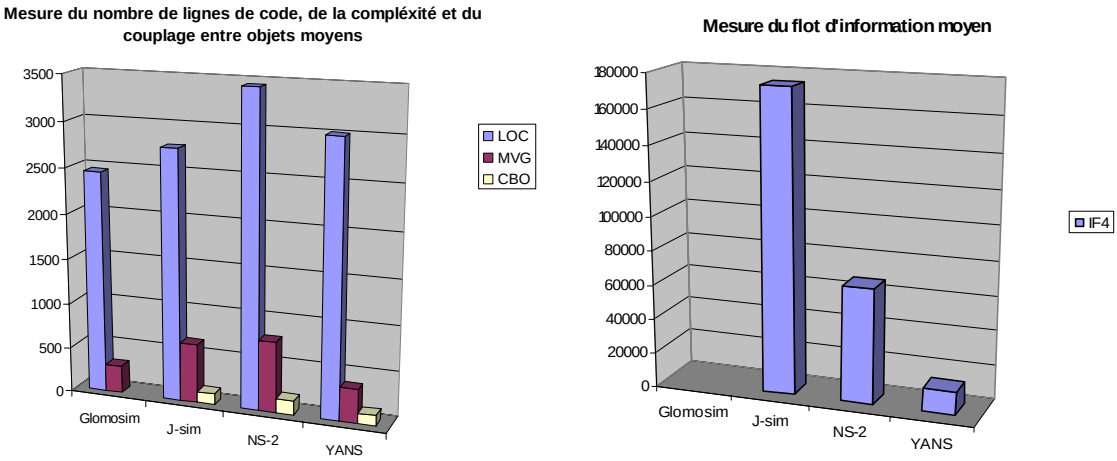


Figure 3.24 a. et b. Mesure du nombre de lignes de code, de complexité, de couplage entre objets et de flots d'information moyens de Glomosim, J-Sim, NS-2 et YANS

5. Conclusion

Dans ce chapitre consacré à la mesure de la flexibilité et de la réutilisabilité des simulateurs réseaux, nous avons à la fois présenté les simulateurs classiques à gros grain – NS, Glomosim, J-Sim – et le nouveau simulateur YANS à la base du futur 802.11 de NS-3. Plusieurs mesures de généralité et de modularité de ces logiciels sont proposées. La mesure de généralité, en particulier, s'appuie sur un modèle de caractéristiques générique définissant les principales fonctionnalités d'une interface réseau.

Les résultats de l'analyse montrent qu'un simulateur à grain moyen, comme YANS, possède une meilleure adaptabilité que les simulateurs traditionnels et ce sans surcoût en terme de complexité. Ces résultats encourageants montrent l'intérêt de l'approche de modularisation dans le domaine de la simulation des réseaux. Ceci a motivé la conception d'une nouvelle architecture d'interface réseau à grain moyen, baptisée CoW (Component WiFi). Nous la présentons au chapitre suivant.

Références

[3GP06] ETSI TS 25.301 V7.0.0 (03-2006) 3rd Generation Partnership Project, Technical Specification Group Radio Access Network, "Radio Interface Protocol Architecture", Mars 2006

[BEK07a] R. Ben-El-Kezadri, F. Kamoun, "YAVISTA: A Graphical Tool For Comparing the MANET Simulators", Journal of Computers (JCP, ISSN1796-203X), en cours de révision

[BEK07b] R. Ben-El-Kezadri, F. Kamoun, "Graphic Visualization of the 802.11 DCF Protocol Under the Ns-2 Simulator", dans *Proc. of the 40th Annual Simulation Symposium, part of the 2007 Spring Simulation Multiconference (ANSS'07)*, Norfolk, VA, Etats-Unis, Mars 2007

[BER04] L. Berlemann, A. Cassaigne, B. Walke, "Generic Protocol Functions for Design and Simulative Performance Evaluation of the Link-Layer for Re-configurable Wireless Systems" dans *Proc. of the 7th International Symposium on Wireless Personal Multimedia Communications (WPMC'04)*, pp. 5–5, Abano Terme, Italie, Septembre 2004

[BER05] L. Berlemann, R. Pabst, M. Schinnenburg, B. Walke, "A Flexible Protocol Stack for Multi-Mode Convergence in a Relay-based Wireless Network Architecture", dans *Proc. of IEEE Personal Indoor and Mobile Radio Conference 2005 (PIMRC05)*, Berlin, Allemagne, Septembre 2005.

[BHA95] N. T. Bhatti and R. D. Schlichting, "A System for Constructing Configurable High-Level Protocols", dans *Proc. of the conference on Applications, technologies, architectures, and protocols for computer communication (SIGCOMM'95)*, pp. 219–230, Cambridge, MA, Etats-Unis, Août 1995

[BIA98] G. Bianchi, A.T Campbell, "A programmable MAC", dans *Proc. of the 7th IEEE International Conference on Universal Personal Communications, (ICUPC '98)*, Vol.2, pp. 953–957, Florence, Italie, Octobre 1998

[CAV02] D Cavin,Y. Sasson, A. Schiper, "On the Accuracy of MANET Simulators", dans *Proc. of the 2nd ACM international workshop on Principles of mobile computing (POMC'02)*, pp. 38–43, Toulouse, France, Octobre 2002

[CHL94] I. Chlamtac, A. Farago, "Making transmission schedules immune to topology changes in multi-hop packet radio networks", dans *Transactions on Networking, IEEE/ACM*, 2(1), pp. 23–29, 1994

[CIE89] R. Cieslak, A. Fawaz, S. Sachs, P. Varaiya, J. Walrand, "The Programmable Network Prototyping System", *Computer*, pp. 67–76, Mai 1989

[ESA95] ESA Board for Software Standardisation and Control (BSSC), "Guide to the software architectural design phase", Issue 1 Revision 1, Agence Spatiale Européenne, France, Mars 1995

[FAR00] A. Faragó, A. D. Myers, V. R. Syrotiuk, G. V. Záruba, "Meta-MAC Protocols: Automatic Combination of MAC Protocols to Optimize Performance for Unknown Conditions", dans *IEEE Journal on Selected Aeras in Communications*, 18(9), pp. 1670–1681, Septembre 2000

[FER99] X. Ferré S. Vegas, **"**An Evaluation of Domain Analysis Methods", dans *Proc. of 4th International Workshop on Evaluation of Modeling Methods in Systems Analysis and Design (EMMSAD'99)*, Heidelberg, Allemagne , Juin 1999

[GAY05] D. Gay, P. Levis, D. Culler, "Software Design Patterns for TinyOS", dans *Proc. of the ACM SIGPLAN/SIGBED 2005 Conference on Languages, Compilers, and Tools for Embedded Systems (LCTES'05)*, Chicago, IL, Etats-Unis, Juin 2005

[HUN06] C. Hunter, J. Camp, P. Murphy, A. Sabharwal, C. Chris, "A Flexible Framework for Wireless Medium Access Protocols", dans *Conference Record of the Fortieth Asilomar conference on Signals, Systems and Computers, (ACSSC '06)*, pp. 2046 – 2050, Pacific Grove, CA, Etats-Unis, Octobre 2006
WARP: Wireless Open-Access Research Platform,
Disponible : http://warp.rice.edu/trac/wiki

[HUT91] N. C. Hutchinson, L. L. Peterson, "The X-Kernel: An Architecture for Implementing Network Protocols", *IEEE Transactions on Software Engineering*, 17(1), p.64–76, Janvier 1991

[IEE99] ANSI/IEEE Std, 802.11 1999 Edition (R2003), IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Network — Specific Requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 2003

[IEE02] IEEE Std 802.15.1 2002, IEEE Standard for Information technology— Telecommunications and information exchange between systems— Local and metropolitan area networks— Specific requirements — Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs) , 2002

[KAM97] A. Kamerman, L. Monteban,"WaveLAN 2: A High-performance Wireless LAN for the Unlicensed Band", *Bell Labs Technical Journal*, 1997

[KAN90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, A. S. Peterson, “Feature-Oriented Domain Analysis (FODA) Feasibility Study”, Rapport technique, CMU/SEI-90-TR-21 ESD-90-TR-22, Carnegie Mellon University, Software Engineering Institute, Etats-Unis, 1990

[KAN03] K.C. Kang, K. Lee., J. Lee, "Feature Oriented Product Line Software Engineering: Principles and Guidelines," *chapitre du livre “Domain Oriented Systems Development: Practices and Perspectives”*, Taylor and Francis, Londres, 2003.

[KAR90] P. Karn, “MACA - A New Channel Access Method for Packet Radio”, dans *Proc. of the ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, pp. 134–140, London, Ontario, Canada, 1990

[LAC04] M. Lacage, M. Hossein Manshaei, T. Turletti, “IEEE 802.11 Rate Adaptation: A Practical Approach”, dans *Proc of ACM International Symposium on Modeling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM’04)*, pp. 126–134, Venise, Italie, Octobre 2004.

[LIT01] Tim Littlefair, “An Investigation into the use of software code metrics in the industrial software development environment”, PhD. Thesis, Faculty of Communications, Health and Science, Edith Cowan University, Perth, Australie, Juin 2001

[0’M92] S. W. O'Malley , L. L. Peterson, “A dynamic network architecture”, *ACM Transactions on Computer Systems (TOCS)*, 10(2), pp.110–143, Mai 1992

[PAQ06] Laurent Paquereau and Bjarne E. Helvik, "A module-based wireless node for ns-2", dans *Proc. of the first Workshop on NS2: the IP network simulator (WNS2)*, Pise, Italie, 2006

[RED06] D. Reddy, G. F. Riley, B. Larish, Y. Chen, “Measuring and Explaining Differences in Wireless Simulation Models”, dans *Proc. of the 14th IEEE International Symposium on Modeling, Analysis, and Simulation (MASCOTS’06),* pp. 275–282, Monterey, CA, Etats-Unis, Septembre 2006

[REN96] R. van Renesse , K. P. Birman, S. Maffeis, “Horus: a flexible group communication system”, *Communications of the ACM*, 39(4), pp. 76–83, Avril 1996

[REN95] R. van Renesse , K. P. Birman , R. Friedman , M. Hayden , D. A. Karr, “A framework for protocol composition in Horus”, dans *Proc. of the fourteenth annual ACM symposium on Principles of distributed computing (PODC’95)*, pp. 80–89, Ottawa, ON, Canada, Aout 1995,

[SAR98] R. G. Sargent, “Verifying and validating simulation models”, *dans Proc. of the 30th Winter Simulation Conference (WSC’98)*, pp. 121–130, Washington, DC, Etats-Unis, Decembre 1998

[SIL98] S. da Silva, D. Florissi and Y. Yemini, “Composing active services in NetScript”, *DARPA Active Networks Workshop*, Tucson, AZ, Etats-Unis, Mars 1998

[SRI05] V. Srivastava and M. Motani, “Cross-layer design: a survey and the road ahead”, *IEEE Communications Magazine,* 43(12), pp. 112–119, Decembre 2005

[SVA99] M. Svahnberg, J. Bosch, “Evolution in Software Product Lines **:** Two cases”, *Journal of Software Maintenance: Research and Practice*, 11(6), pp. 391–422, Novembre 1999

[TAI05] M. Taifour, F. Nait-Abdesselam, D. Simplot-Ryl, “Neighbourhood backoff algorithm for optimizing bandwidth in single hop wireless ad-hoc networks”, dans *Proc of the 2005 International Conference on Wireless Networks, Communications and Mobile Computing (IWCMC’05),* Volume 1, pp. 336–341, Maui, Hawai, Juin 2005

[TAK01a] M. Takai, J. Martin, R. Bagrodia, “Effects of wireless physical layer modeling in mobile ad hoc networks”, dans *Proc. of the 2nd ACM international symposium on Mobile ad hoc networking & computing (MobiHoc ’01)*, pp 87–94, New York, NY, Etats-Unis, Octobre 2001

[TAK01b] M. Takai, R. Bagrodia, K. Tang, M. Gerla, “Efficient Wireless Network Simulations with

Detailed Propagation Models", *Wireless Networks*, 7(3), pp. 297–305, Mai 2001

[UIT94] "ITU-T Recommendation X.200 (1994) | ISO/IEC 7498-1:1994, *Information technology – Open Systems Interconnection – Basic Reference Model: The Basic Model*", 1994

[WAR01] C. Ware, J. F. Chicharo,T. A. Wysocki, "Simulation of Capture Behaviour in IEEE 802.11 Radio Modems", *Journal of Telecommunications and Information Technology*, 2(2), pp. 46–54, 2001

[WON01] G. T. Wong, M. A. Hiltunen, and R. D. Schlichting, "A configurable and extensible transport protocol", in *Proc of the Twentieth Annual IEEE Conference on Computer Communications (INFOCOM'01)*, pp. 319–328, Anchorage, AK, Etats-Unis, Avril 2001

[YE04] W. Ye, J. Heidemann, D. Estrin, "Medium Access Control With Coordinated Adaptive Sleeping for Wireless Sensor Networks", *IEEE/ACM Transactions on Networking*, 12(3), pp. 493–506, Juin 2004

Netographie

[Net-JDep] JDepend, Disponible**:** http://www.clarkware.com/software/JDepend.html
Dernière visite le 6 Mars 2007

[Net-DepF] Dependency Finder, Disponible: http://depfind.sourceforge.net/
Dernière visite le 6 Mars 2007

[Net-CKJM] CKJM Disponible: http://www.spinellis.gr/sw/ckjm/
Dernière visite le 6 Mars 2007

[Net-Vil], Vil, Disponible: http://www.1bot.com/index.html
Dernière visite le 6 Mars 2007

[Net-Mula], Mulato, Disponible http://imprint.dyndns.org/thiago/mulato/index.php?n=Main.HomePage
Dernière visite le 6 Mars 2007

[Net-SiSS] SiSSy, Disponible: http://sissy.fzi.de/SISSy/CMS/index_html
Dernière visite le 6 Mars 2007

[Net-NS-2] NS-2.29, Disponible: http://www.isi.edu/nsnam
Dernière visite le 31 août 2006

[Net-Glomosim] Glomosim 2.03, Disponible: http://pcl.cs.ucla.edu/projects/glomosim/
Dernière visite le 31 août 2006

[Net-YANS] YANS version du 07/09/2005, Disponible: http://spoutnik.inria.fr/ns-2-80211/
Dernière visite le 31 août 2006

[Net-J-Sim] J-Sim 1.3, Disponible: http://www.j-sim.org
Dernière visite le 31 août 2006

[Net-Zigbee] Interface Zigbee pour NS-2,
Disponible: http://ees2cy.engr.ccny.cuny.edu/zheng/pub/
Dernière visite le 6 Mars 2007

[Net-UMTS] Interface UMTS pour NS-2, Disponible: http://www.geocities.com/opahostil/
Dernière visite le 6 Mars 2007

[Net-Bluetooth] Interface Bluetooth pour NS-2, Disponible : http://bluehoc.sourceforge.net/
Dernière visite le 6 Mars 2007

Chapitre 4

CoW, un Modèle d'Interface Réseau Flexible et Réutilisable

1. Introduction

Ce chapitre présente le modèle d'une interface réseau à grain moyen flexible et réutilisable. Ce modèle, baptisé CoW (Component WiFi), diffère de YANS (cf chapitre précédent) par plusieurs aspects. En effet, si YANS possède des similarités avec les drivers MadWifi, les concepts de CoW sont tirés du modèle FODA et de l'ensemble des paradigmes de simulation Yavista. Le but de CoW est double :

- L'objectif premier est de modéliser les réseaux 802.11 en mode infrastructure dans le plus grand respect du standard
- Le modèle doit présenter un niveau de généricité et de flexibilité suffisant pour modéliser des technologies simples autres que 802.11. Il doit également présenter un niveau de fonctionnalité égal aux simulateurs traditionnels (NS-2, Glomosim, J-Sim).

CoW s'appuie sur la notion de composants logiciels et sur le modèle de composants du simulateur J-Sim. Dans ce chapitre, nous présentons successivement le développement orienté composant, le simulateur J-Sim et l'architecture de CoW. Nous expliquons à ce titre comment intégrer le modèle de caractéristiques développé au chapitre précédent et mettre en œuvre les techniques d'assemblage dynamiques de cartes réseaux. La dernière partie propose une évaluation de la généralité et de la modularité de CoW et la compare à YANS.

2. Choix de Conception Composant

2.1. Le développement orienté composant

Même si les objets ont apporté de grands bouleversements dans le domaine informatique dans les années 1990 et qu'ils restent très utilisés, les systèmes objets souffrent d'un certain nombre de lacunes au regard de leurs homologues à base de composants [OUS05]. Ces raisons ont motivé le recours aux composants et aux méthodes de développement à base de composants.

2.1.1. Les composants logiciels

L'approche par composants étant relativement nouvelle, il n'existe pas encore de consensus autour de la notion de composant [SYZ02], [BAC00]. Nous retiendrons dans ce mémoire la définition du Software

Engineering Institute [BAC00] de l'université de Carnegie pour qui un composant est :

- une implémentation opaque de fonctionnalités
- sujet à composition par des tiers
- conforme à un modèle de composant

Le premier point assimile le composant à une boite noire. Le second permet l'assemblage de composants développés par des sources indépendantes. Le modèle de composants spécifie les conventions et standards imposés aux développeurs de composants. La conformité au modèle de composants est l'une des propriétés distinguant les composants des autres formes de package logiciel (module, etc). Le modèle de composants est implémenté par un middleware qui représente le système sur lequel s'exécutent et communiquent les différents composants. Le modèle de composants impose des règles et des conventions sur les aspects suivants:

- Le typage des composants : le type d'un composant est représenté par la liste des interfaces qu'il implémente.
- Les schémas d'interactions entre les composants : ils définissent la nature des contrats entre composants et les possibilités d'interactions entre le middleware et les composants. La réalisation des propriétés non fonctionnelles peut également être spécifiée.
- Allocation des ressources : ces règles décrivent comment sont instanciés et déployés les composants sur le middleware.

La spécification du composant est donnée par le nom de ses interfaces et l'implémentation du composant. Une interface spécifie un sous ensemble de services offerts par le composant. Un contrat entre interfaces définit les services qui doivent être nécessairement offerts par chacune des interfaces pour que leurs composants puissent être connectés. Le contrat peut également définir des post et pré conditions régissant la manipulation des fonctions définies dans les interfaces ou mentionner un schéma d'interactions particulier pour orchestrer les appels de fonctions. Un contrat est explicite s'il possède sa propre spécification. Il est implicite si sa spécification est intégrée dans celle des interfaces. Dès qu'un contrat est scellé, les messages peuvent être échangés entre les ports des composants ; ces ports définissent les points dans l'implémentation où sont reçus et aiguillés les différents messages.

2.1.2. Ingénierie des logiciels à base de composants

L'ingénierie des logiciels à base de composants (*Component Based Software Engineering ou CBSE*) est une méthode de développement permettant de mieux rationaliser le développement des systèmes complexes. Elle distingue principalement deux cycles de vie : un pour le développement des composants et un autre pour le développement d'un système à base de composants [PRE00]:

- Lors du développement du composant, la réutilisabilité est l'enjeu principal : le code est conçu dans l'optique d'être réutilisé dans les différentes applications de la ligne de produits. La phase d'analyse du domaine permet d'identifier, d'analyser et de spécifier les besoins communs des diverses

applications du domaine en terme d'entité réutilisable. La phase de modélisation de l'architecture et des composants permet ensuite de dériver les différentes architectures du produit.

- Le développement d'un système à base de composants se focalise sur la sélection, l'adaptation et l'intégration des composants dans le produit en fonction de la configuration de caractéristiques souhaitée par l'utilisateur.

Dans la suite de ce mémoire nous utiliserons la méthode FORM (Feature-Oriented Reuse Method) [KAN98], [KAN02] extension de FODA particulièrement adaptée aux architectures de ligne de produits [MAT04], pour guider le développement de notre interface réseau.

2.2. La plateforme de simulation

Le choix de la plateforme est un élément déterminant dans la conception de CoW. Celui-ci peut se porter vers trois types de logiciels : les plateforme génériques à base de composants, les simulateurs réseaux traditionnels et les simulateurs à base de composants.

2.2.1. Les plateformes génériques à base de composants

Plusieurs middlewares permettant la construction d'applications à base de composants existent [BAR05]. Ceux-ci n'implémentent pas toutefois les bibliothèques de composants (UDP, TCP, IP, etc) nécessaires à la simulation des réseaux. La plateforme Ptolemy [Net-Pto] dédiée à la modélisation, à la simulation et à la conception de systèmes temps réel embarqués propose une bibliothèque réseau mais celle-ci ne couvre que les couches basses des réseaux de senseurs.

2.2.2. Les simulateurs réseaux traditionnels

Les simulateurs réseaux traditionnels offrent une interface homme-machine simple permettant d'invoquer les éléments architecturaux et de les interconnecter. La différence avec un simulateur à base de composants est que cette interconnexion n'est pas explicite pour l'utilisateur.

- Dans Glomosim, les niveaux du modèle en couches (IP, TCP, etc) sont préalablement définis et instanciés dans le moteur de simulation à l'initialisation de chaque nœud après lecture du fichier de configuration.
- Dans NS, l'architecture des noeuds n'est pas figée car décrite sous forme de scripts, mais ceux-ci sont généralement déportés dans des répertoires annexes.
- YANS propose, pour simplifier le mécanisme d'assemblage de NS-2, de définir les architectures possibles des noeuds mobiles dans le code source du simulateur dans une classe constructrice de cartes.

2.2.3. Les simulateur réseaux à base de composants

Les simulateurs réseaux Omnet++ [Net-Omnet] et J-Sim [Net-Jsim] implémentent tous deux partiellement un middleware utilisant une technologie composant. L'avantage d'Omnet++ est de posséder un ADL (Architecture Description Language) efficace baptisé NED (NEtwork Description) permettant la création d'architectures réseaux. NED permet également de définir les interfaces des composants. A contrario, J-Sim ne dispose pas d'ADL. La topologie de la simulation doit donc être saisie sous forme de

scripts TCL/Java. Bien que moins élégants qu'un ADL pour assembler un système, les scripts permettent les modifications d'architecture plus facilement à l'exécution. A noter que J-Sim permet comme Omnet++ la saisie des topologies à l'aide d'un navigateur/éditeur graphique et leur stockage dans des fichiers XML.

2.2.4. Le simulateur J-Sim

Omnet++ ne disposant pas encore d'implémentation de 802.11 à l'entame de ce mémoire, nous avons choisi J-Sim comme middleware pour CoW.

J-Sim repose sur une architecture à composants autonomes (ACA). Un composant sous J-Sim est assimilé à une puce électronique. Un composant dispose de ports d'entrée/sortie pour communiquer avec le monde extérieur via l'envoi et la réception de messages. L'assemblage d'une application consiste à interconnecter les ports des composants. Généralement, l'envoi des messages n'est pas bloquant. Dans ce cas, l'exécution du composant émetteur se poursuit après l'envoi, et la réception du message au niveau du récepteur déclenche le lancement d'un processus léger pour traiter la requête. Il existe également une primitive d'envoi bloquante. Dans ce cas, le composant émetteur attend la réponse du composant paire pour poursuivre l'exécution de son code.

Les contrats spécifient la nature des messages que les composants peuvent émettre et recevoir. Le contrat définit également les fonctions permettant la création de ces messages. Les composants doivent avoir connaissance des contrats à utiliser. Contrairement au modèle objet où l'appelant doit connaître le nom des services offerts par l'appelé, avant la phase de compilation, la vérification de la compatibilité des interactions entre composants n'est effectuée qu'à la phase d'exécution. Le modèle de composants de J-Sim permet également d'affecter des rôles - initiateur, répondeur - aux composants liés par un contrat mais ceux-ci ne sont pas pleinement exploités par les modèles réseaux.

3. Présentation du Modèle CoW

Cette section décrit la conception de CoW cycle par cycle : développement des composants réutilisables, puis réalisation des produits spécifiques.

3.1. Développement des composants réutilisables

La méthode FORM se base sur le modèle de caractéristiques établis avec la méthode FODA. Un récapitulatif du modèle proposé au chapitre précédent est à ce titre présenté à la Figure 4.1. En plus des caractéristiques fonctionnelles, la Figure 4.1 montre les caractéristiques de bas niveau FORM[37] permettant d'implémenter ces caractéristiques fonctionnelles. Le lien entre les caractéristiques de bas et haut niveaux est indiqué à l'aide de traits épais sur la Figure 4.1. Les *mécanismes d'accès* et *de management 802.11* décrivent en particulier comment réaliser le management et l'accès au canal dans les réseaux 802.11.

[37] Ces caractéristiques ne sont pas relatives aux capacités intrinsèques du produit mais aux techniques générales ou spécifiques aux réseaux sans fil utilisées pour implémenter les systèmes.

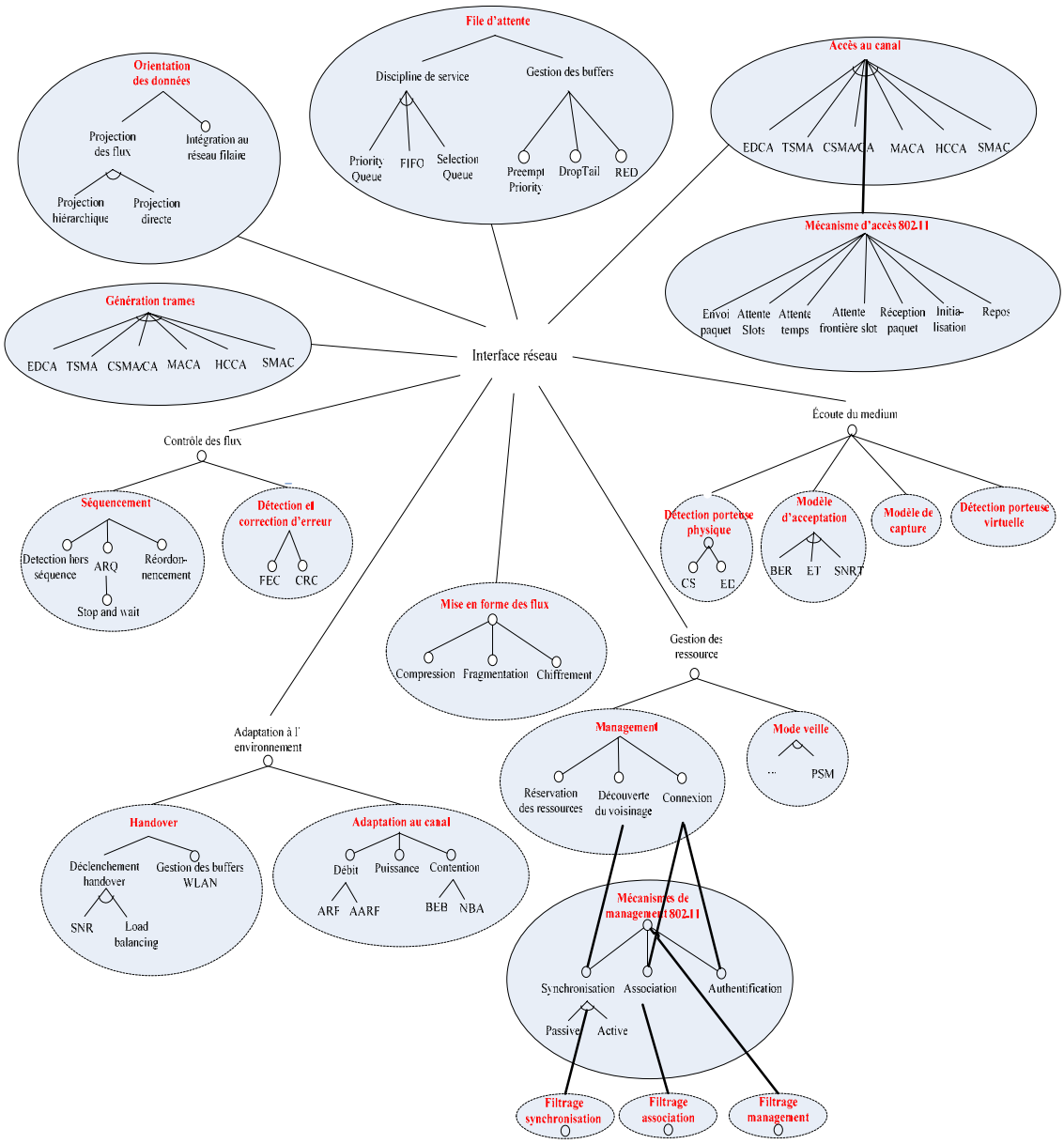


Figure 4.1 Modèle de caractéristiques FODA

Une fois les caractéristiques du domaine isolées, la méthode FORM nous permet de dériver les composants et trois vues architecturelles [BAS98] correspondant à trois visions différentes de l'architecture de la ligne de produits :

- La vue en modules définit l'architecture globale du système en regroupant les fonctions dans des modules chacun s'exécutant généralement sur un matériel différent. Le processus d'allocation des fonctions en matériel est guidé par le modèle de caractéristiques. Les caractéristiques fortement couplées peuvent être implantées dans les mêmes composants pour minimiser l'overhead en communication.

- La vue en processus représente le comportement dynamique des modules matériels comme le parallélisme des tâches à l'intérieur des composants. Pour ce faire, chaque module est raffiné en un ensemble de processus concurrents exécutant le code fonctionnel.
- La vue en sous modules alloue les caractéristiques associées à chaque module dans des sous modules réalisant généralement une unique caractéristique. Cette vue détaillée du système permet d'identifier les différentes entités réutilisables dans la ligne de produits.

Les éléments cœur d'une ligne de produits sont les composants implémentant les caractéristiques communes à tous les produits. Ces éléments doivent être adaptés et étendus pour générer un produit spécifique.

3.1.1. Vue en modules

Dans CoW, nous assimilons un module matériel FORM à un composant J-Sim. L'allocation des caractéristiques dans les modules de CoW est réalisée comme indiqué à la Figure 4.2. Les modules hachurés sur le schéma sont les éléments cœur de l'architecture en ligne de produits. Les caractéristiques couplées sont implémentées dans le même module. Comme le montre la Figure 4.2, CoW n'implémente qu'un sous arbre du modèle de caractéristiques défini au chapitre 3. Les caractéristiques – *Mode veille*, *Mise en forme des flux* et *Adaptation de la puissance* – ne sont pas, à titre d'exemple, prises en compte dans CoW.

La Figure 4.2 apporte des éléments d'information intéressants sur l'implémentation du management : les fonctions de bas niveau *Association*, *Synchronisation* et *Authentification* sont déléguées dans un composant séparé du reste du *Management* – dans la *SME* – à la manière de *Control* et *Access*. Les filtres permettant la réalisation des fonctions de mangement de bas niveau sont également déportés au niveau des modules *Multiplex*, *MPDU Preparation* et *Decode*. Le détail de leur implémentation est décrit à la section 3.2.2.

Le rôle de chaque module est le suivant :

1. Le bloc *Decode* agit principalement comme un filtre en amont du bloc *Access* : c'est pourquoi il intègre *Détection et correction d'erreur* qui filtre les trames corrompues et *filtrage association* qui jette les trames de données lorsque la station n'est pas associée. Ce bloc devant également notifier à *Access* les durées pendant lesquelles le canal est physiquement[38] ou virtuellement occupé, il implémente la *Détection de porteuse virtuelle*.
2. Le bloc *Access* maintient l'état du canal et interprète les ordres envoyés par le bloc *Control*. L'exécution des ordres dépend de l'état du canal. L'ensemble des ordres interprétables par *Access* est indiqué par la caractéristique *Mécanisme d'accès*. *Access* envoie également à *Control* des notifications indiquant le début d'une nouvelle séquence de messages.
3. Le bloc *Control* organise le séquencement des échanges sur le réseau en fonction des règles d'accès au canal. Des automates à états sont maintenus en réception et en émission. L'automate à utiliser à l'émission dépend de la séquence de paquet à transmettre (exemple : DATA, DATA/ACK, RTS/CTS/DATA/ACK). *Control* usine lui-même les trames de contrôle intervenant dans les séquences

[38] *Decode* tient cette information de la couche physique

de paquets. *Control* intègre la gestion du séquencement pour permettre l'exploitation du couplage naturel entre la caractéristique *ARQ* et l'accès au canal.

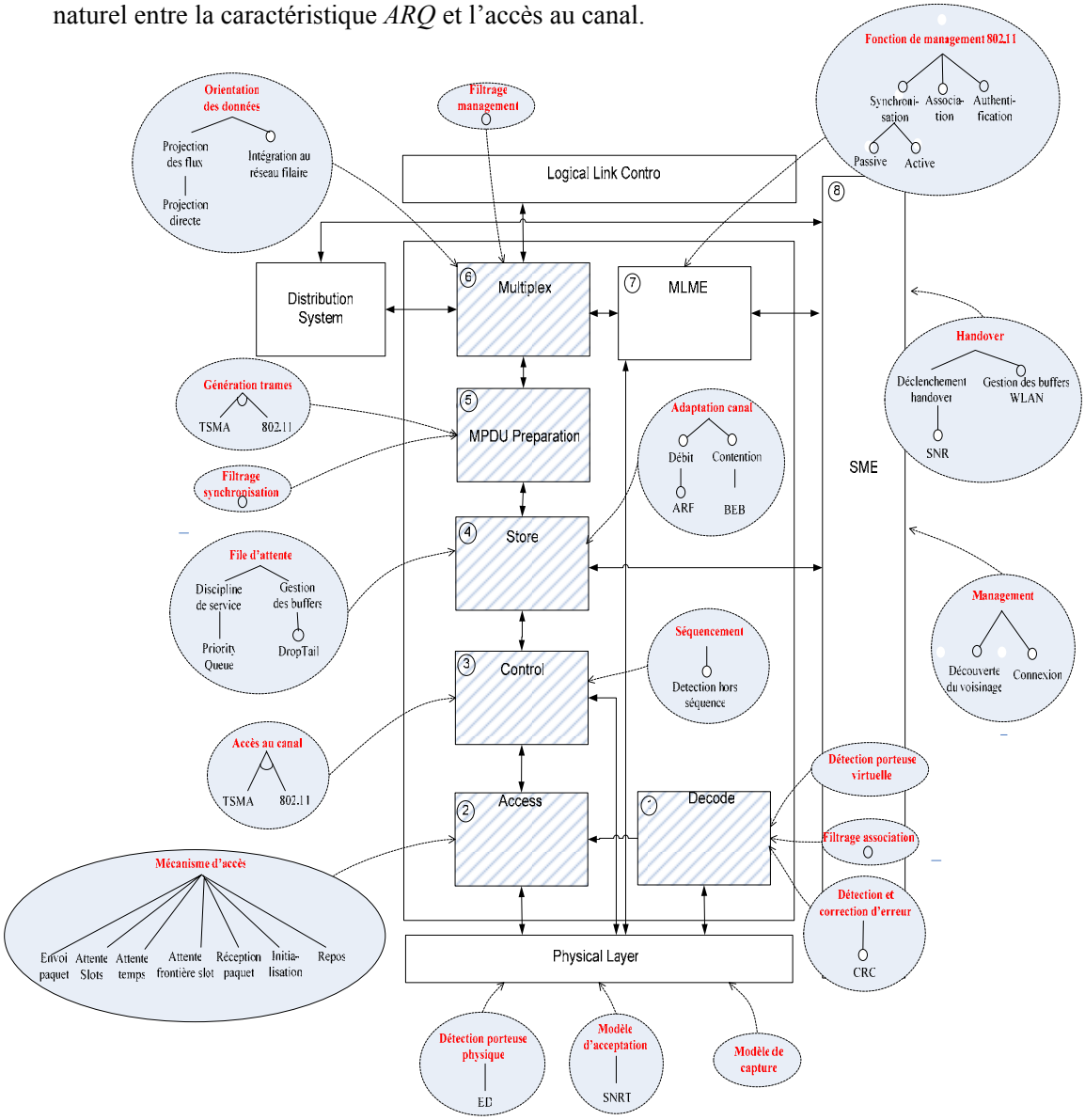


Figure 4.2 Implantation des caractéristiques dans CoW

4. Le bloc *Store* stocke spécifiquement les structures de données (paquets). L'adaptation de débit et de contention étant réalisée sur les paquets stockés dans les files, les caractéristiques *adaptation au canal* et *file d'attente* sont regroupées dans ce module.
5. Le bloc *MPDU Preparation* génère les trames aux formats spécifiques à partir des données reçues des couches supérieures ou de la *MLME* et les décapsule à la réception. *Filtrage synchronisation* jette les trames de données lorsque la station n'est pas synchronisée.
6. Le bloc *Multiplex* interface le *système de distribution*, *la couche de convergence* et la *MLME*. *Filtrage Management* aiguille les trames de management vers la *MLME*.
7. Le bloc *MLME* (Mac subLayer Management Entity) met à disposition de la *SME* les services

nécessaires spécifiques à la technologie de *Management* pour établir une connexion, découvrir le voisinage ou réserver les ressources nécessaires à la communication.

8. Le bloc *SME* (Station Management Entity) pilote la carte lorsque celle-ci est managée. La procédure de handover relançant la phase de découverte du voisinage, *handover* et *management* sont regroupées dans *SME*.

CoW se distingue des autres simulateurs par son architecture à style hétérogène (voir chapitre 2). Il existe donc dans CoW plusieurs architectures sous jacentes et autant de mécanismes de communication différents.

1. les composants faiblement couplés communiquant à l'aide de messages forment une architecture à composants indépendants. L'échange de messages permet un dialogue explicite entre les composants
2. l'utilisation de variables partagées entre les composants implique une architecture centralisée de type tableau noir. La communication par variables partagées matérialise un dialogue implicite.
3. la diffusion d'évènements permet à des composants de s'inscrire auprès de composants sources pour être avertis de certains évènements. Ce mécanisme est particulièrement utilisé entre les composants *Access* et *Control* pour notifier l'arrivée des paquets déclenchant l'exécution d'une séquence de messages.
4. la collaboration des composants *Control* et *Access* se base également sur une architecture en machine virtuelle. Les spécifications de protocoles implémentées dans le composant *Control* peuvent en effet être assimilées à un programme et les techniques d'accès de bas niveau de *Access* aux actions réalisées par ce programme.

L'ensemble de ces mécanismes de communication est supporté par FORM. Les contrats impliqués dans ces échanges sont décrits en annexe C.

3.1.2. Vue en processus

CoW implémente une vue en processus rudimentaire : tous les modules, sauf *Control*, implémentent un unique processus. La vue du module *Control* est décrite à la Figure 4.3.

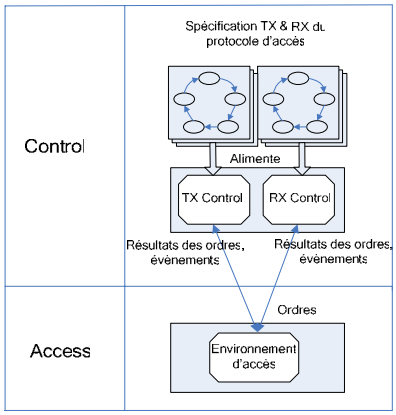


Figure 4.3 Vue en processus des modules *Control* et *Access* dans CoW

La particularité de *Control* est d'implémenter deux processus, l'un pour la coordination des séquences de messages côté émetteur (*TX Control)* et l'autre pour le côté récepteur (*RX Control)*.

- Coté émetteur, la spécification décrit généralement les phases, d'*attente d'accès au canal* (*TX_WF_X*), de *transmission des données* (*TX_X*), d'*attente d'acquittement* (*TX_WF_CO*) et de *backoff* (*TX_WF_PreBO* et *TX_WF_PostBO*).
- Côté récepteur, la spécification décrit le processus d'acquittement des données.

Les spécifications utilisées par l'émetteur et le récepteur dans CoW pour un échange DATA/ACK sous 802.11 sont fournies aux Figures 4.4 et 4.5. Le même procédé de modélisation est utilisé pour TSMA.

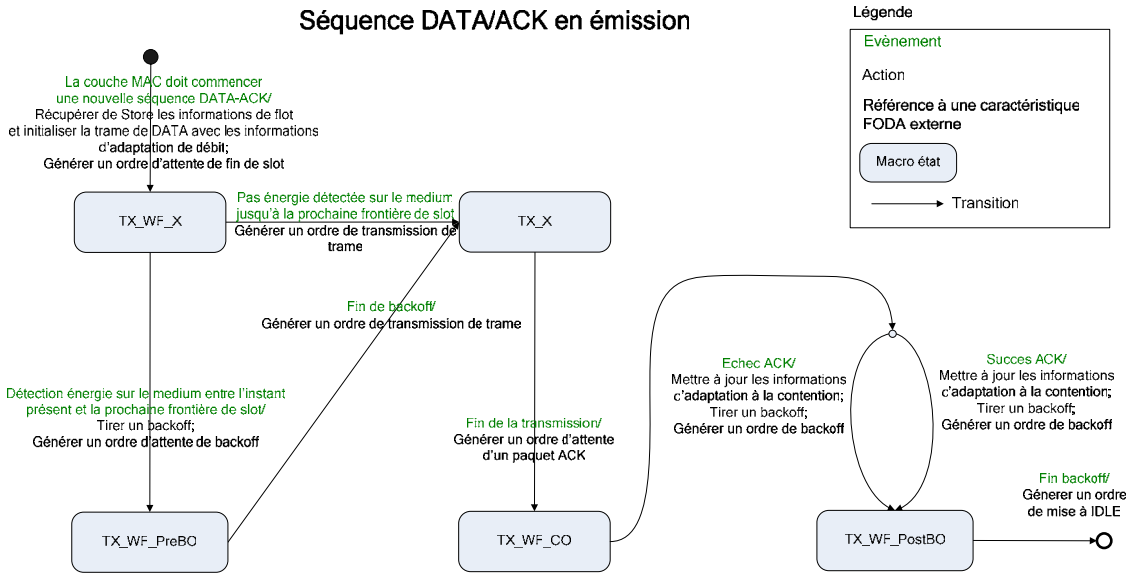


Figure 4.4 Implémentation de la séquence DATA/ACK en émission sous CoW

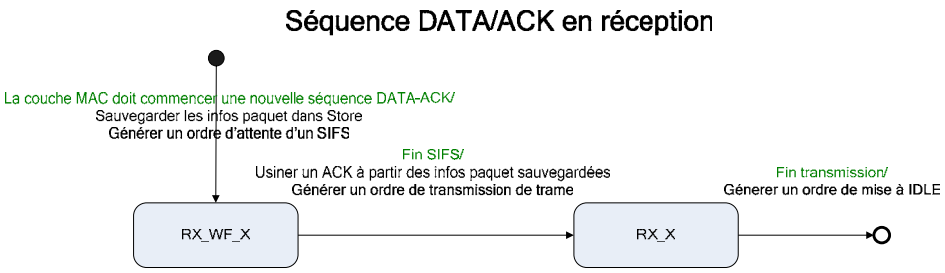


Figure 4.5 Implémentation de la séquence DATA/ACK en réception sous CoW

L'ensemble de services défini dans les spécifications est exécuté par le module *Access*. Celui-ci maintient l'état du canal et implémente les techniques d'accès de bas niveau spécifiques au protocole (envoi d'une trame, attente d'un intertrame, etc) côté émetteur et récepteur.

Le schéma de communication entre *Access* et *Control* est le suivant. Le module *Control* envoie les ordres à l'*environnement d'accès*. L'environnement réalise les commandes et renvoie les résultats de ces commandes. Lorsque l'*environnement d'accès* reçoit un paquet de DATA du medium, il émet un résultat de commande – non sollicité – contenant les DATA. Celles-ci sont directement extraites à la réception du message dans *Control* et envoyées dans le circuit de traitement des données.

Une vue de la solution est présentée à la Figure 4.6. Celle-ci s'appuie sur une architecture en machine virtuelle, le bas niveau mettant à disposition du haut niveau un ensemble de ressources programmables

(variables, timers, etc). L'utilisation judicieuse de ces ressources permet un meilleur mapping entre les actions $a_1, a_2, \ldots, a_n$ (cf chapitre 1) réalisées au niveau MAC et les structures de données les modélisant.

Dans CoW, en plus des états permettant la gestion des séquences de messages coté récepteur et émetteur, un état est maintenu dans le bas niveau pour maintenir la vue du canal. La gestion dans la file d'attente des compteurs de transmission (SLRC et SSRC) et de la fenêtre de contention augmente la flexibilité du modèle. Les ressources flots permettent la gestion de ces variables.

Quatre types de ressources sont donc définis au total sous CoW : ressources transmetteur, récepteur, canal et flot. La Figure 4.6 fournit une vue logique de la distribution de ces ressources dans le système. Les ressources sont spécifiées de sorte à fournir un bon mapping avec les actions du langage neutre. Ceci facilite l'instrumentation en code neutre (voir chapitre 1).

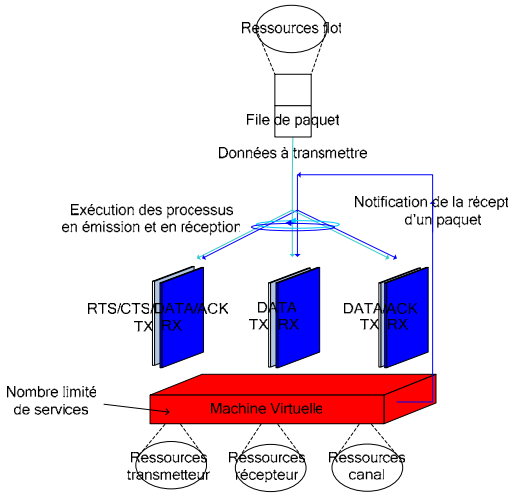


Figure 4.6 Implémentation du protocole 802.11 sous Yavista

Cette architecture est particulièrement adaptée à la spécification des différentes séquences de messages définies dans les standards. Elle offre en outre de grandes facilités de prototypage[39].

3.1.3. Vue en sous modules

Du fait de sa structure générique bien établie, CoW possède de bonnes propriétés pour implémenter un système réel ou simulé. Les deux vues en sous modules de CoW dérivent ainsi de portages de systèmes existants. La première vue implémente les modes infrastructure et ad-hoc de la norme 802.11. Elle est inspirée de l'implémentation SDL (Specification and Description Langage) des standards 802.11 et 802.11f [IEEE03] et de l'algorithme d'adaptation de débit ARF [KAM97]. La seconde implémente le modèle TSMA implémenté dans le simulateur Glomosim.

Sous FORM, toutes les techniques d'implémentation des points de variété étudiées au chapitre 2 peuvent être utilisées dans les sous modules [KAN98]. Un sous module peut par exemple introduire un point de variété en associant une classe abstraite à une caractéristique optionnelle ou alternative. La Table 4.1 décrit l'ensemble des caractéristiques variantes implémentées dans CoW et la méthode de réalisation associée.

[39] Le lecteur pourra comparer la complexité des spécifications de CoW avec celles de NS et Glomosim (voir annexe B).

Table 4.1 Caractéristiques variables de CoW

Liste des caractéristiques variables	Caractéristiques alternatives	Caractéristique optionnelle	Technique d'implémentation de la variabilité
Accès au canal	TSMA, 802.11	Non	Architecture centrée infrastructure
Génération des trames	TSMA, 802.11	Non	Architecture centrée infrastructure
Débit ARF	-	Oui	Condition sur variable
Découverte du voisinage	-	Oui	Architecture centrée infrastructure
Connexion	-	Oui	Architecture centrée infrastructure
Gestion des buffers WLAN	-	Oui	Condition sur variable
Synchronisation	Passive, Active	Non	Condition sur variable
Intégration au réseau filaire	-	Oui	Architecture centrée infrastructure

3.1.3.1. Vue en sous modules de 802.11

La vue de 802.11 est un portage brut de l'implémentation SDL (Specification and Description Language) proposée dans le standard 802.11[40]. Comme le montre la Figure 4.7, les machine à états décrites dans le standard 802.11 sont extraites de la norme, adaptées, et mappées dans les modules selon un schéma conforme au modèle de caractéristiques.

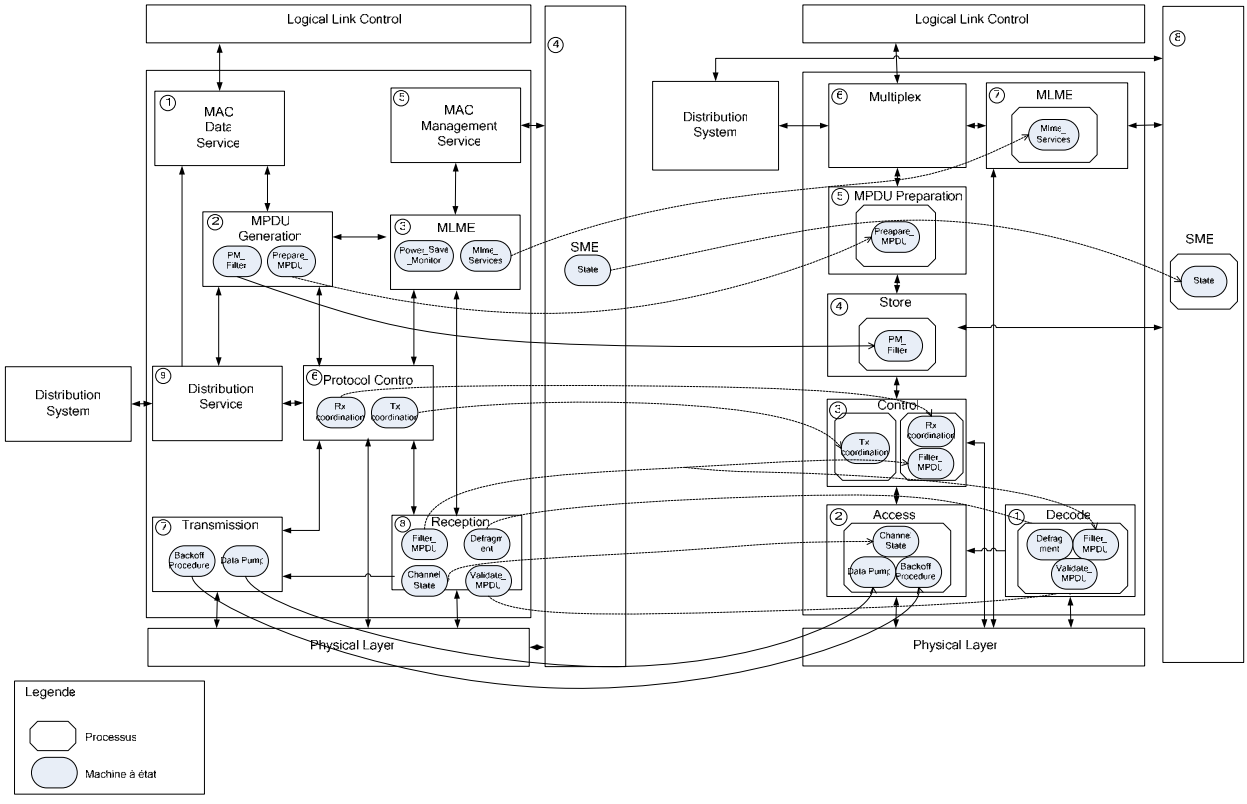


Figure 4.7 Mapping entre les processus de COW et les machines à états de la norme

[40] A noter que CoW ne teste l'état de la station (associé ou synchronisé) que dans les filtres intégrés à *MPDU Preparation* et *Decode*. Ces tests sont plus nombreux dans l'implémentation originale de la norme.

Ce schéma est obtenu en regroupant les machines à états réalisant la même caractéristique ensemble. Pour mapper 802.11 dans CoW, il faut donc ramifier les caractéristiques du modèle FODA pour exhiber la part exacte de chaque machine à états dans chaque caractéristique. Les sous caractéristiques identifiées s'expriment en terme de techniques propres à 802.11. La Table 4.2 montre la correspondance entre sous caractéristiques et les machines à états originales de 802.11. Une machine à états réalisant plusieurs caractéristiques est partagée entre plusieurs processus si ses caractéristiques ne sont pas colocalisées. C'est le cas de *Filter MPDU*.

Table 4.2 Mapping entres les sous caractéristiques de 802.11 et les machines à états de la norme

		Machines à états										
Caractéristiques	**Sous caractéristiques**	Prepare MPDU	PM_Filter	Tx Coordination	Rx Coordination	Backoff Procedure	Data Pump	MLME	Defragment	Channel State	Filter MPDU	Validate MPDU
Accès canal	Gestion séquence RX				×							
	Gestion Séquence TX			×								
Mécanisme d'accès	Attente du backoff					×						
	Perception des opportunités de transmission									×		
	Transmission binaire						×					
File d'attente	Stockage paquets DATA & MANAGEMENT		×									
Génération MPDU	Création paquets DATA & MANAGEMENT	×										
Séquencement	Détection hors séquence										×	
	Retransmission			×								
Détection et correction erreurs	Détection des erreurs binaires											×
Détection porteuse virtuelle	Détection de porteuse virtuelle										×	
Filtrage association	Filtrage si STA non associée ou AP non démarré								×			
Filtrage synchronisation	Filtrage si STA non synchronisée ou AP non démarré		×									
Fonction de management 802.11	Fonction de synchronisation							×				
	Fonction d'authentification							×				
	Fonction d'association							×				

L'implémentation des différentes caractéristiques FODA dans la machine virtuelle 802.11 de CoW est détaillée à la Figure 4.8. L'avantage de ce modèle est d'être très flexible grâce a l'utilisation d'un jeu de services réutilisables : *Attente de fin de slot*, *Attente d'IFS*, *Transmission*, *Attente de backoff*, *Attente de confirmation*, *Core*. Ce dernier service en particulier est auto-invoqué dans l'*environnement d'accès* à chaque réception de nouvelle trame. Le procédé d'implémentation de la machine virtuelle TSMA est le même.

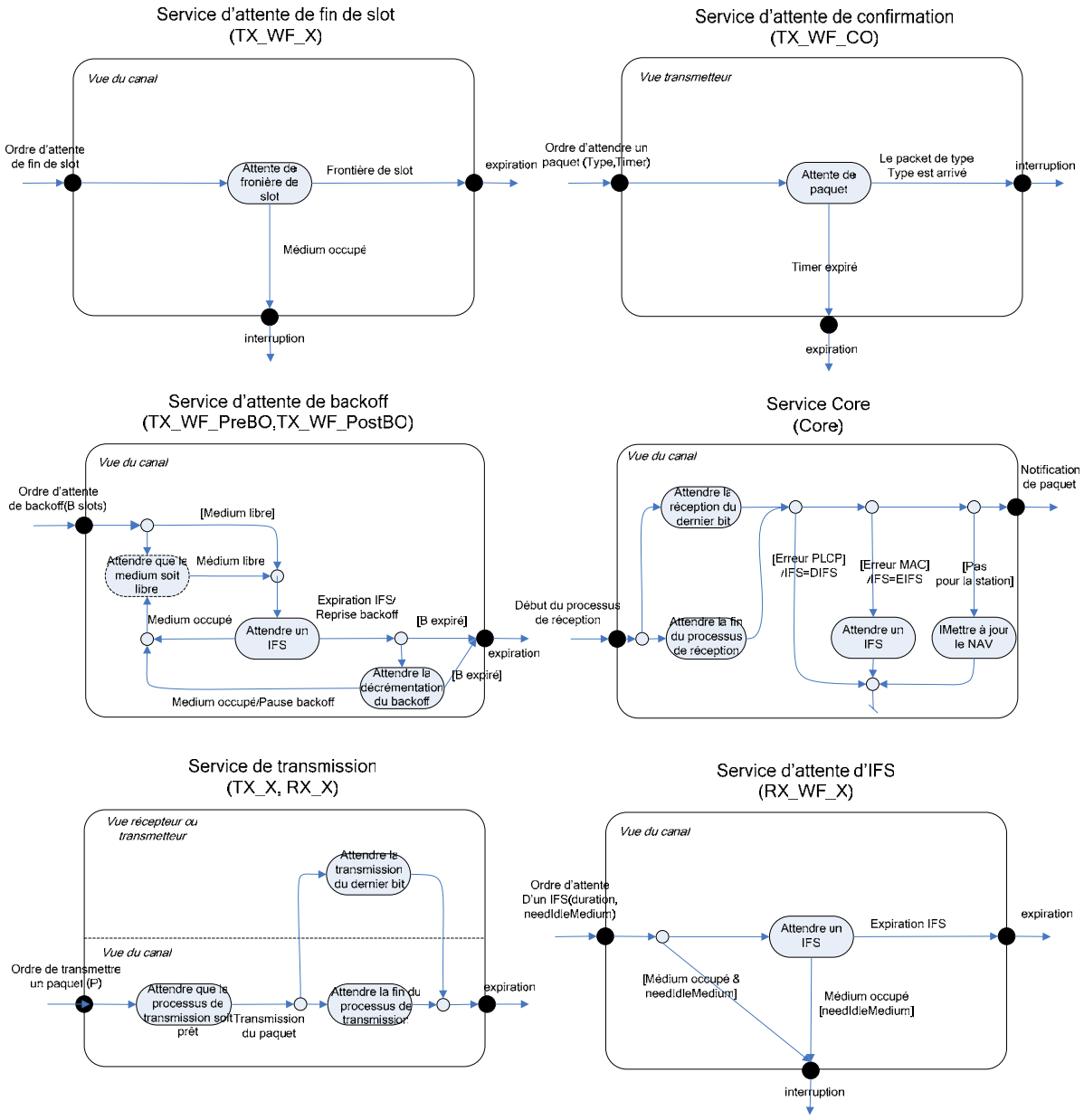


Figure 4.8 Implémentation de la machine virtuelle 802.11 de CoW

3.1.3.2. Vue en sous modules de TSMA

La spécification de la vue en sous modules de TSMA démontre la faisabilité de notre solution. Cette vue découle de l'étude de l'implémentation de TSMA de Glomosim. TSMA ne supportant ni le mode infrastructure, ni les connexions, la vue en sous modules n'intègre donc pas *de système de distribution* ni de *MLME* et de *SME*. La liste des slots que le protocole TSMA réserve pour chaque station est fournie statiquement à l'initialisation des simulateurs pour les deux implémentations. La vue en sous modules de TSMA est présentée à la

Figure **4.9**.

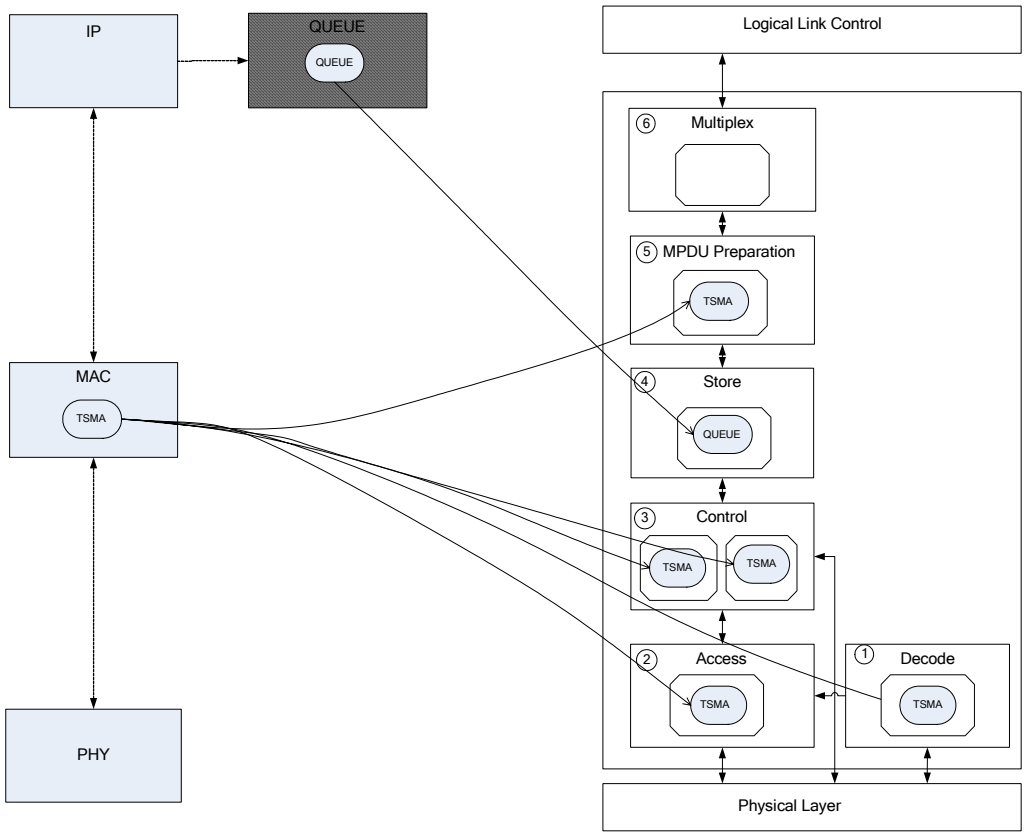


Figure 4.9 Mapping entre les processus de CoW et la conception TSMA de Glomosim

La Table 4.3 montre le mapping exact entre les sous caractéristiques et les modules de Glomosim.

Table 4.3 Mapping entres les sous caractéristiques de TSMA et Glomosim

		Modules Glomosim	
Caractéristiques	**Sous caractéristiques**	QUEUE	TSMA
Accès canal	Gestion séquence RX		×
	Gestion séquence TX		×
Mécanisme d'accès	Perception des opportunités de transmission (temporisation slot)		×
	Transmission binaire		×
File d'attente	Stockage paquet DATA	×	
Génération trames	Génération paquet DATA		×
Séquencement	Détection hors séquence		×
	Retransmission		×

3.2. Développement des produits spécifiques

La méthode FORM limite, aux yeux de l'utilisateur, le développement d'une application spécifique au choix d'une configuration de caractéristiques. Cette section décrit le processus de sélection des caractéristiques et les techniques d'adaptation des composants et de composition sous jacentes permettant la réalisation d'applications sous CoW. A noter que parmi les quatre considérations relatives à la reconfiguration des systèmes (voir la section 4.4 du chapitre 2), CoW traite l'automatisation grâce à une logique adaptative et la facilité d'utilisation grâce à un mécanisme de vérification des compositions. Précisons également que la conception de CoW en vue de la réutilisation se base sur l'abstraction des structures de messages, l'utilisation d'interfaces standardisées et d'un cœur d'architecture établi. Les composants *Decode*, *Store* et *Multiplex* sont ainsi intégralement réutilisés entre les vues TSMA et 802.11.

3.2.1. Sélection des caractéristiques

Les seules caractéristiques sélectionnables sous CoW sont les caractéristiques orientées utilisateurs décrites dans la Table 4.1. La sélection des caractéristiques est réalisée dans le script TCL/Java mettant en place le réseau à simuler. Les conditions sur variables permettent de configurer les composants, et les techniques centrées infrastructure, d'insérer un nouveau composant dans l'environnement.

3.2.2. Adaptation du système

3.2.2.1. Service adaptable

Certains conflits peuvent apparaître après la phase de sélection des composants. Pour résoudre ces conflits, une technique de *component wrapping* est souvent utilisée [BRO96]. Trois niveaux de wrapping existent : le *white box wrapping* examine les détails internes de chaque composant et effectue des modifications dans le code source pour éliminer les conflits. Le *grey box wrapping* fait appel à un langage d'extension ou à une API pour éliminer ou masquer les conflits. Enfin le *black box wrapping* permet l'introduction de pré et de post traitements dans les interfaces du composant pour éliminer les conflits.

Sous CoW, les caractéristiques variantes fixant la variabilité sont implémentées sous forme d'extension. Les extensions fournissent un service d'extension aux composants extensibles. Elles appartiennent aux composants à l'origine de conflits et sont insérées dans l'environnement par leur propriétaire.

Les points de variété sont les endroits prédéfinis dans le corps des composants où peuvent venir s'ancrer les extensions. Ceux-ci sont implémentés à l'aide d'une technique assimilable au *black box wrapping*. Pour ce faire, nous spécifions pour chaque composant extensible le niveau auquel effectuer les pré et post traitements. En pratique, les extensions réalisent le plus souvent des tâches de filtrage de messages. La Figure 4.10 montre comment insérer dynamiquement un filtre de paquets. Le formalisme utilisé est le langage SDL (Specification Description Language).

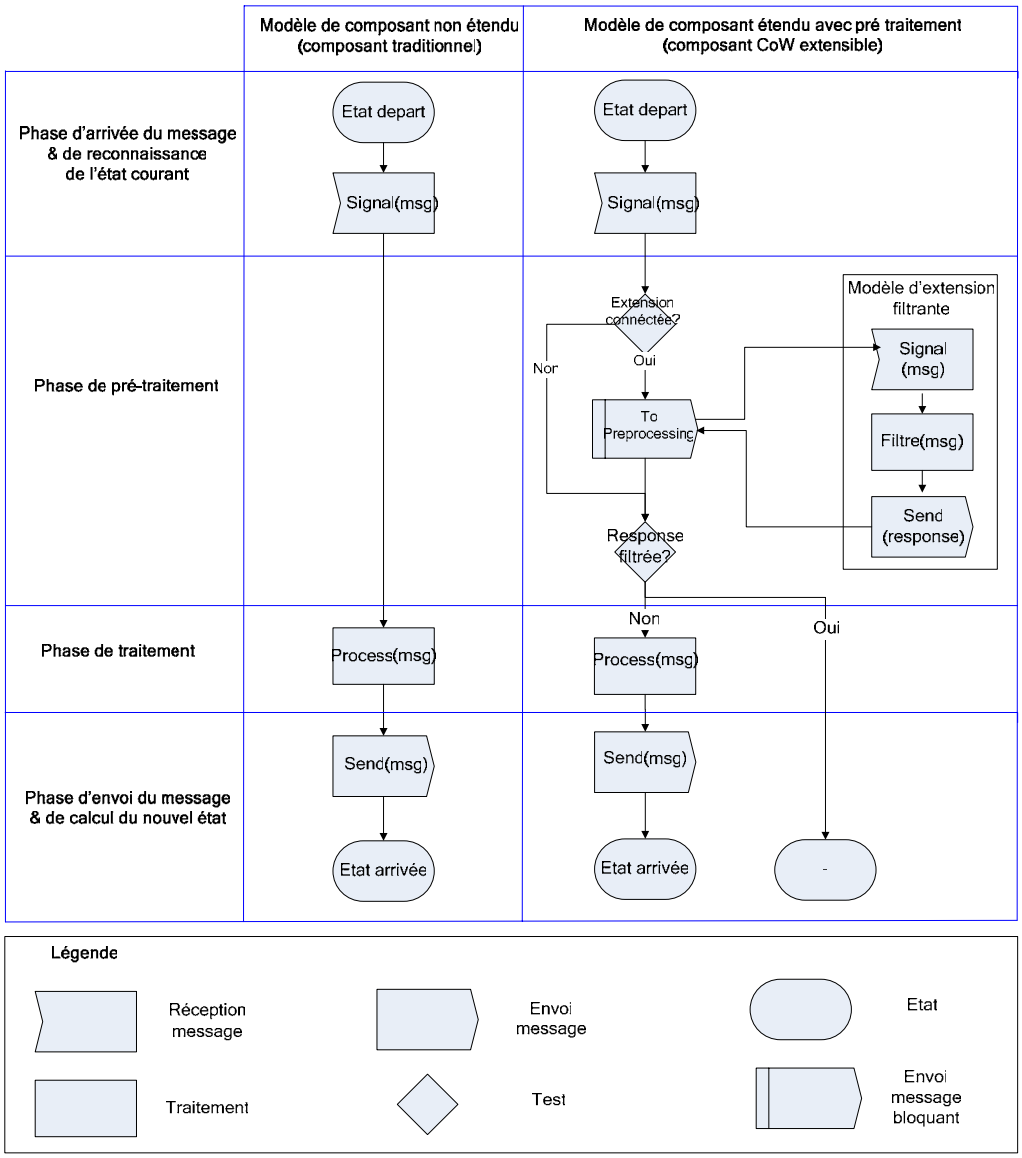


Figure 4.10 Adaptation des composants sous CoW

3.2.2.2. Monitoring de l'environnement

CoW implémente un contrôleur détectant les modifications de l'environnement (insertion ou suppression des composants/extensions). Au moment de leur insertion, les composants et les extensions doivent s'identifier auprès du contrôleur en lui fournissant leur nom, les services offerts et requis ainsi que les ports associés à la réalisation de ces services ; ces informations étant stockées sous forme d'entête dans les composants/extensions. L'ensemble des données recueillies est stocké dans une table du contrôleur.

3.2.2.3. Logique adaptative

La logique adaptative est contenue dans le contrôleur. Elle est figée et se limite à tester la compatibilité entre les services requis par un composant et les services fournis par les composants et

extensions présents dans l'environnement.

Ces mêmes tests permettent l'adaptation des éléments cœurs de l'architecture avec des composants optionnels.

3.2.3. Composition des modules

3.2.3.1. Interconnexion des composants

L'assemblage des composants est réalisé par le contrôleur lorsque un contrat peut être établi – c'est-à-dire lorsqu'un service offert satisfait un service requis. Le contrôleur connecte alors physiquement les ports de deux composants pairs.

3.2.3.2. Vérification des interconnexions

La vérification des interconnexions permet de tester la cohérence globale de l'application. Avant d'exécuter une interface réseau, le contrôleur s'assure au préalable que toutes les connexions nécessaires à son fonctionnement sont établies. Le caractère obligatoire d'une connexion dépend du type de contrat lié au service à réaliser. Il en existe quatre :

- Le contrat Contrôle/Données : Il identifie les contrats permettant l'échange de messages de contrôle et de données. Il fait souvent intervenir des transactions de type requête/réponse. La réception d'un tel message engendre le plus souvent au niveau des composants émetteur et récepteur un changement d'état. Le composant qui initie la requête initiale est le demandeur du service.
- Le contrat de Configuration : les messages de configuration sont des notifications générées pour mettre à jour les variables du composant. Elles ne provoquent de changements d'état ni à la réception ni à la transmission. Le composant qui reçoit la notification est le demandeur du service.
- Le contrat Client/Serveur : les messages client/serveur permettent la réalisation d'un service grâce à une transaction client/serveur. Ils sont bloquants, donc ne provoquent pas de changements d'état dans les composants. Le client est le demandeur du service.
- Le contrat d'Extension : les messages d'extension permettent de réaliser des transactions client/serveur entre un composant et ses extensions. Ils sont bloquants, et ne provoquent de changements d'état ni dans le composant ni dans l'extension. Le composant étendu est le demandeur du service.

Le contrat Client/Serveur étant bloquant, il est logiquement nécessaire. Nous considérons également le contrat Contrôle/Données nécessaire car il est responsable de la mise en place du chemin des données dans la carte. Les messages d'extension n'étant émis que si un contrat avec une extension est établi, aucune vérification n'est effectuée sur ce type. Le contrat de Configuration est considéré optionnel car nous pensons qu'un composant, même mal configuré, dispose encore de capacités de communication et de traitement intactes.

4. Mesures de Généralité et de Modularité

4.1. Mesures de généralité

La Figure 4.11 décrit les caractéristiques disponibles sous CoW.

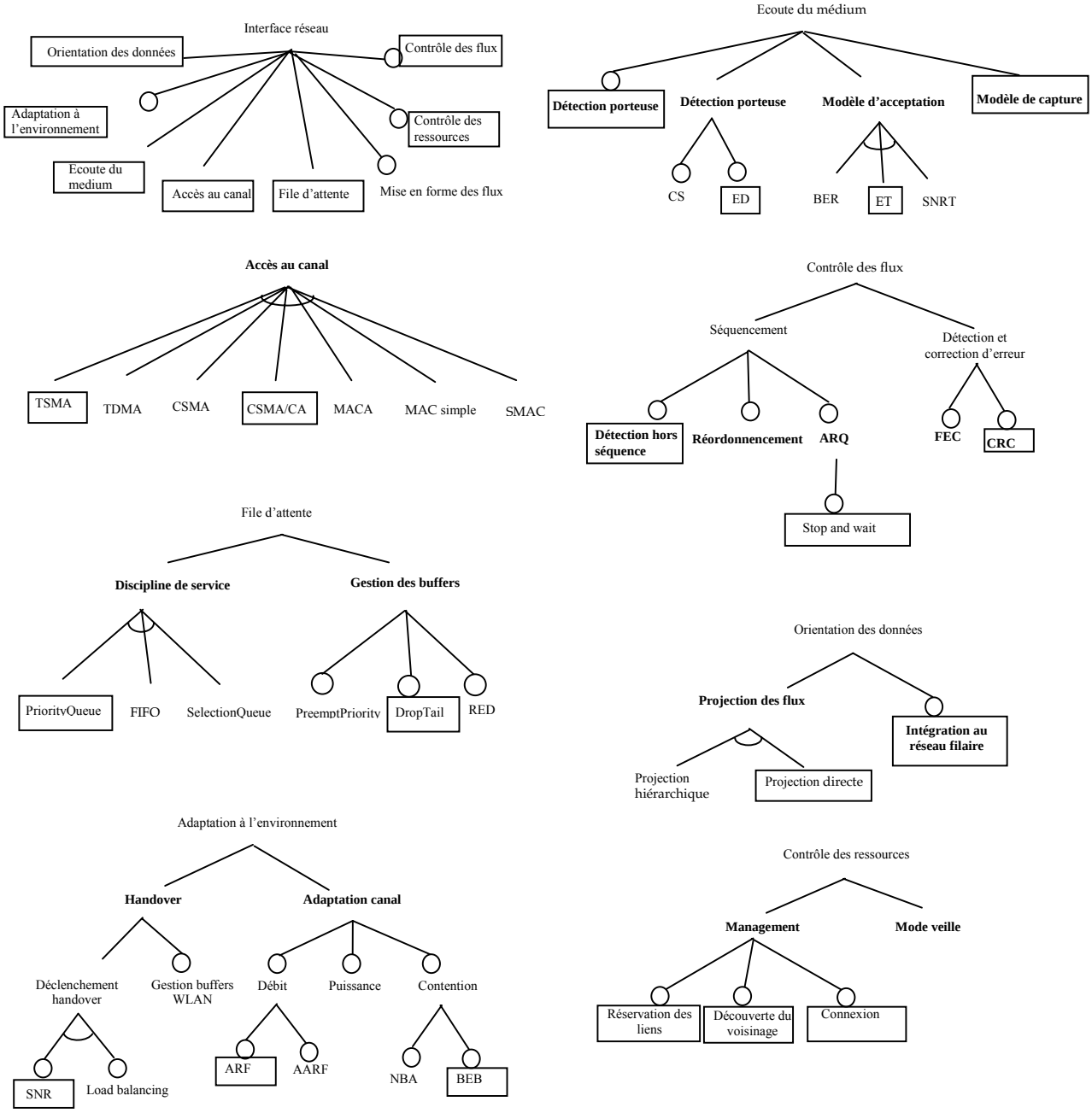


Figure 4.11 Caractéristiques disponibles dans CoW

Malgré l'architecture modulaire de CoW, la Table 4.4 montre un couplage important entre la caractéristique d'*accès au canal* et les autres caractéristiques implantées. Ceci s'explique par le fait que CoW est peu permissif et propose sauf le *contrôle des ressources*, *ARF* et *SNR* peu de fonctionnalités autres que celles des systèmes originaux modélisés (spécifications 802.11 et TSMA). Outre *ARF* et *SNR*, la seule véritable déviation par rapport aux systèmes modélisés est le caractère optionnel du *contrôle des ressources*. Choisir l'accès CSMA/CA sans la caractéristique optionnelle de *contrôle des ressources*

permet effectivement de simuler les réseaux 802.11 ad-hoc avec le même niveau de fonctionnalité que celui offert par les simulateurs traditionnels.

Table 4.4 Dépendances inter caractéristiques obligatoires et exclusions sous CoW[41]

		Accès au canal	
		CSMA/CA	TSMA
Contrôle des ressources	Réservations des liens	o	⊘
	Découvertes du voisinage	o	⊘
	Connexion	o	⊘
Contrôle des flux	CRC	×	×
	Stop and wait	×	×
	Détection hors séquence	×	×
Adaptation à l'environnement	BEB	×	⊘
	ARF	o	o
	SNR	o	⊘
Orientation des flux	Projection directe	×	×
	Intégration au réseau filaire	×	×
Ecoute du medium	Détection de porteuse virtuelle	×	⊘
	Détection de porteuse physique ED	×	×
	Modèle de capture	×	×
	ET	×	×
File d'attente	DropTail	×	×
	PriorityQueue	×	×

Compte tenu du grand nombre de caractéristiques optionnelles relatives à CSMA/CA et la réutilisation intégrale des composants *Decode*, *Store* et *Multiplex* dans les produits TSMA et CSMA/CA, nous introduisons, comme pour YANS, un niveau architectural supplémentaire entre la ligne de produits CoW et les variantes disponibles (voir Figure 4.12).

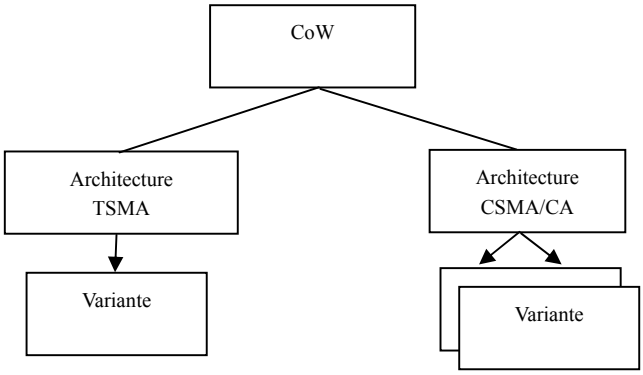


Figure 4.12 Hiérarchie de la ligne de produits CoW

Pour que la mesure d'adaptation reflète l'existence de plusieurs produits, nous considérons optionnelles donc adaptables, toutes les caractéristiques implémentées dans les composants dont la

[41] Le caractère « ⊘ » indique une caractéristique ne se réalisant jamais à l'exécution car non implémentée, « × » une caractéristique se réalisant toujours, « o » une caractéristique optionnelle et « a » une caractéristique alternative.

présence est corrélée à l'architecture choisie. La Table 4.5 propose le mapping caractéristiques/ composants et montre les différentes caractéristiques adaptables sous CoW.

Table 4.5 Présentation du mapping caractéristiques/composants et des caractéristiques adaptables sous CoW

	Discipline de service	Gestion des buffers	Accès au canal	Management	Mode veille	Projection des flux	Intégration au réseau filaire	Détection porteuse virtuelle	Compression	Fragmentation	Chiffrement	Handover	Réordonnancement	Détection hors séquence	ARQ	CRC	Modèle de capture	Adaptation contention	Adaptation débit	Adaptation puissance	Détection porteuse physique	Modèle d'acceptation paquet	FEC
SME				×	×							×											
MLME				×	×																		
Multiplex						×	×																
MPDU Prep									×	×	×												
Store	×	×																×	×	×			
Control			×										×	×	×								
Access			×																				
Decode								×								×							×
PHY																	×				×	×	
Caractéristiques adaptables			×	×			×					×							×				

Les valeurs de EAI, AAI et SAI sont déduites de la Table 4.5. Celles-ci sont présentées à la Table 4.6.

Table 4.6 Calculs des EAI, AAI et SAI sous CoW

	EAI(PHY)	**EAI(Decode)**	**EAI(Access)**	**EAI(Control)**	**EAI(Store)**	**EAI(MPDU Prep)**	**EAI(Multiplex)**	**EAI(MLME)**	**EAI(SME)**	**AAI**	**SAI**
Produit CSMA/CA	0	0	1	1/4	1/5	0	1/2	1/2	1/2	33%	31%
Produit TSMA	0	0	1	1/4	1/5	0	1/2			28%	

4.2. Mesures de modularité

4.2.1. Mesures de morphologie et d'impureté

La Table 4.7 contient les mesures de morphologie et d'impureté de CoW.

Table 4.7 Mesures de morphologie et d'impureté de CoW

	Architecture	
	TSMA	CSMA/CA
Taille en arcs	9	17
en noeuds	8	10
Profondeur	7	7
Largeur	2	3
Ratio arcs à noeuds	1.3	1.9
Nombre de dépendances	9	17
Impureté de l'arbre	0.1	0.22

4.2.2. Mesures de flots d'information, de couplage et de complexité

Les Tables 4.8 à 4.10 montrent les résultats obtenus pour les stations TSMA, 802.11 adhoc et 802.11 infrastructure.

Table 4.8 Mesures de flots d'information, de couplage et de complexité des stations TSMA

		Parent	Instanciable	LOC	MVG	CBO	Fan-in	Fan-out	IF4
LL	LL	N/A	Oui	123	15	5	5	0	0
MAC	Multiplex	N/A	Oui	156	22	8	8	0	0
	MPDU_Prep	N/A	Non	139	17	12	10	2	400
	MPDU_Prep_TSMA	MPDU_Prep	Oui	10	0	3	3	0	0
	Store	N/A	Oui	621	121	13	11	2	484
	Control_TSMA	N/A	Oui	450	65	17	19	0	0
	Access_TSMA	N/A	Oui	463	77	11	11	0	0
	Decode	N/A	Oui	216	47	8	8	0	0
PHY	WirelessPhyNonoise	N/A	Oui	634	102	15	0	17	0

Table 4.9 Mesures de flots d'information, de couplage et de complexité des stations 802.11 ad-hoc

		Parent	Instanciable	LOC	MVG	CBO	Fan-in	Fan-out	IF4
LL	LL	N/A	Oui	123	17	5	5	0	0
MAC	Multiplex	N/A	Oui	156	22	8	8	0	0
	MPDU_Prep	N/A	Non	139	17	12	10	2	400
	MPDU_Prep_802_11	MPDU_Prep	Oui	51	11	3	3	0	0
	Store	N/A	Oui	621	121	13	11	2	484
	Control_802_11	N/A	Oui	675	120	19	19	0	0
	Access_802_11	N/A	Oui	751	201	13	19	0	0
	Decode	N/A	Oui	216	47	8	8	0	0
PHY	WirelessPhyNonoise	N/A	Oui	634	102	15	0	17	0

Table 4.10 Mesures de flots d'information, de couplage et de complexité des stations 802.11 infrastructure

		Parent	Instanciable	LOC	MVG	CBO	Fan-in	Fan-out	IF4
LL	LL	N/A	Oui	123	17	5	5	0	0
MAC	SME	N/A	Oui	800	146	15	15	0	0
	MLME	N/A	Oui	1059	218	18	18	0	0
	Multiplex	N/A	Oui	156	22	8	8	0	0
	MPDU_Prep	N/A	Non	139	17	12	10	2	400
	MPDU_Prep_802_11	MPDU_Prep	Oui	51	11	3	3	0	0
	Store	N/A	Oui	621	121	13	11	2	484
	Control_802_11	N/A	Oui	675	120	19	19	0	0
	Access_802_11	N/A	Oui	751	201	13	19	0	0
	Decode	N/A	Oui	216	47	8	8	0	0
PHY	WirelessPhyNonoise	N/A	Oui	634	102	15	0	17	0

4.3. Analyse des résultats

4.3.1. Mesures de généralité

Les Figures 4.13 a. et b. comparent les SAI et le nombre moyen de caractéristiques adaptables de CoW et de YANS. Comme nous pouvons l'observer, CoW est moins flexible que YANS. Le simulateur dispose toutefois d'un grand potentiel vu le manque actuel d'adaptation de la couche physique.

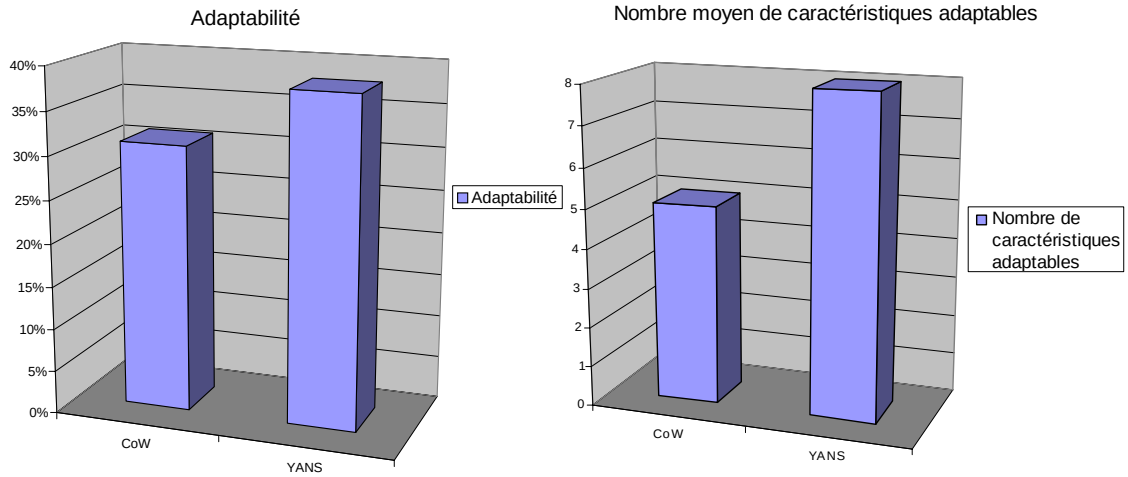


Figure 4.13 a. et b. Comparaison des mesures de généralité de CoW et YANS

4.3.2. Mesures de modularité

Les Figures 4.14 et 4.15 montrent le coût de la généralité en terme taille, d'impureté, de nombre de lignes de code, de complexité, de couplage et de flots d'information moyens. La moyenne est réalisée sur les produits/architectures de chaque simulateur.

Comme l'indique la Figure 4.14, CoW possède plus de modules et de connectiques que YANS, d'où une modularité plus fine[42]. Les résultats d'impureté, en faveur de CoW, reflètent l'utilisation d'une

[42] A noter que YANS dispose d'une structure hiérarchique et que seuls les modules de la hiérarchie supérieure sont comptés

colonne compacte – *Multiplex*, *MPDU Preparation*, *Store*, *Control, Access* – dans CoW.

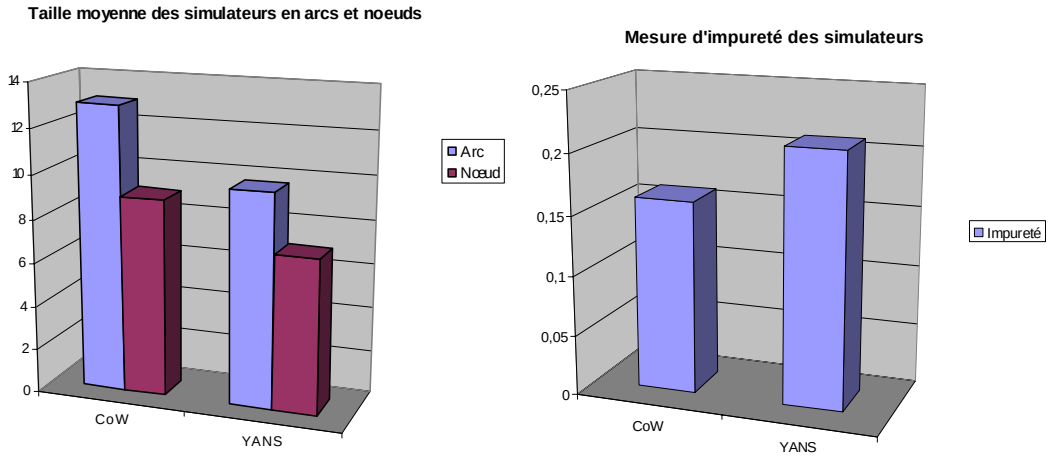


Figure 4.14 a. et b. Taille en arcs et nœuds et impureté moyennes de CoW et YANS

Comme le montre la Figure 4.15, CoW possède en contrepartie une implémentation plus lourde et une complexité plus grande s'expliquant principalement par la concision du code de YANS. La mesure de flots d'information exceptionnellement basse de CoW s'explique par la quasi absence d'héritage objet et l'utilisation des contrats et des composants J-Sim pour minimiser le fan-out et éviter les référencements inter objets/composants.

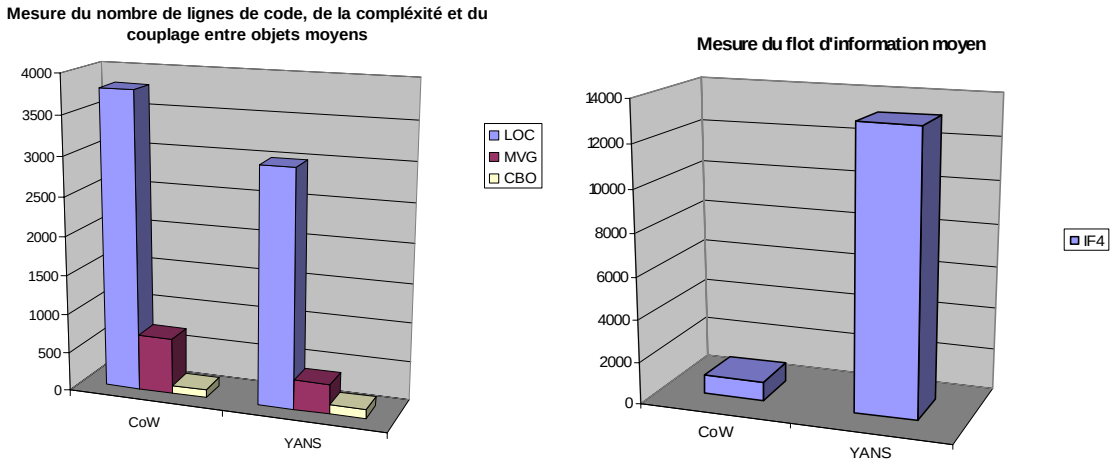


Figure 4.15 a. et b. Mesures du nombre de lignes de code, de complexité, de couplage entre objets et de flots d'information moyens de CoW et YANS

5. Conclusion

Ce chapitre a montré l'application de la méthode d'ingénierie FORM au modèle de caractéristiques FODA proposé pour la couche liaison des simulateurs sans fil. Le résultat est une architecture simple, compréhensible à tous. Les éléments principaux de la solution sont l'utilisation de composants à grain moyen paramétrable (les modules), la répartition des fonctions d'accès au canal en deux et la centralisation des informations flux dans un module central, *Store*, accessible aux autres modules de traitement des flots. Les mesures de généralité et de modularité de l'implémentation proposées montrent

un bon potentiel de flexibilité et de réutilisabilité, encourageant la poursuite du projet.

Pour clore ce mémoire, nous proposons dans le chapitre suivant une revue des principes phares ayant guidés la conception de CoW et ELYSE.

Références

[BAC00] F. Bachmann, L. Bass, C. Buhman, S. Comella-Dorda, F. Long, J. Robert, R. Seacord, K. Wallnau, "Volume II: Technical Concepts of Component-Based Software Engineering, 2nd Edition", Rapport Technique CMU/SEI-2000-TR-008 ESC-TR-2000-007, Carnegie Mellon University, Software Engineering Institute, Pittsburgh, PA, Etats-Unis, Mai 2000

[BAR05] Olivier Barais, "Construire et Maîtriser l'Evolution d'une Architecture Logicielle à base de Composants", Thèse de doctorat, Université des Sciences et Technologies de Lille, Lille, France, Novembre 2005

[BAS98] L. Bass, P. Clements, R. Kazman, *"Software Architecture in Practice"*, Addison-Wesley, 1998

[BRO96] A.W. Brown and K.C. Wallnau, "Engineering of Component-Based Systems", *Component-Based Software Engineering: Selected Papers from the Software Engineering Institute*, pp. 7–15, Wiley-IEEE CS Press, 1996

[IEEE03] ANSI/IEEE Std, 802.11F™ IEEE Trial-Use Recommended Practice for Multi-Vendor Access Point Interoperability via an Inter-Access Point Protocol Across Distribution Systems Supporting IEEE 802.11™ Operation, 2003

[KAM97] A. Kamerman, L. Monteban ,"WaveLAN 2: A High-performance Wireless LAN for the Unlicensed Band", *Bell Labs Technical Journal*, 1997

[KAN98] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, M. Huh, "FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures", *Annals of Software Engineering* , Springer Netherlands, 5(0), pp 143–168, Janvier 1998

[KAN02] K.C. Kang, Kwanwoo Lee., Jaejonn Lee, "Feature Oriented Product Line Software Engineering: Principles and Guidelines", Chapitre du livre *"Domain Oriented Systems Development: Practices and Perspectives"*, Taylor and Francis, Londres, 2002.

[MAT04] M. Matinlassi, "Comparison of software product line architecture design methods: COPA, FAST, FORM, KobrA and QADA", in *Proc of the 26th International Conference on Software Engineering, (ICSE'04)*, pp 127–136, Edinburgh, Ecosse, Royaume-Uni, Mai 2004

[OUS05], M. Oussalah, *"Ingénierie des composants, Concepts, techniques et outils"*, Vuibert Informatique, 2005

[PRE00] R. S. Pressman, D. Ince, *"Software engineering A practitioner's approach, European adaptation"*, Fifth Edition, Edition McGraw-Hill, 2000

[SYZ02] C. Syzperski, *"Component Software"*, Second Edition, Addison-Wesley, 2002

Netographie

[Net-Jsim] H. Tyan, "Design, realization and evaluation of a component-based compositional software architecture for network simulation", PhD. Thesis, Ohio State University, 2002
J-Sim, Disponible : http://www.j-sim.org/
Dernière visite le 31 août 2006

[Net-Omnet] A. Vargas, "The OMNeT++ Discrete Event Simulation System", *dans Proc. of the European Simulation Multiconference (ESM'01)*, Prague, République Tchèque, Juin 2001
Omnet++, Disponible : http://www.omnetpp.org/
Dernière visite le 31 août 2006

[Net-Pto] J. Davis, M. Goel, C. Hylands, B. Kienhuis, E.A. Lee, J. Liu, X. Liu, L. Muliadi, S. Neuendorffer, J. Reekie, N. Smyth, J. Tsay, and Y. Xiong. "Overview of the Ptolemy project", Rapport technique UCB/ERL M99/37, 1999
Ptolemy, Disponible : http://ptolemy.eecs.berkeley.edu/
Dernière visite le 31 août 2006

Chapitre 5

Les Paradigmes de Simulation du Simulateur YAVISTA

1. Introduction

Les principes de simulation Yavista définissent l'architecture et les concepts guides du moteur de validation ELYSE et du modèle d'interface réseau CoW. Contrairement à ses deux déclinaisons pratiques, le modèle conceptuel présenté ici fait abstraction des obstacles de réalisation. Nous supposons à ce titre qu'il s'exécute sur un middleware idéal mettant en oeuvre la technologie FORM. La mise en œuvre de l'accès au canal du modèle sur un middleware idéal [HUN06] (voir la section 1 du Chapitre 3) s'exécutant sur un système réel est notamment décrite dans [BEK08]. Les objectifs ultimes du modèle visent la levée des contraintes de correction, de performance, de flexibilité et de réutilisation posées sur les simulateurs.

Dans le domaine de la correction en particulier, les concepts du moteur étendu ELYSE^2 (ELyse is Yavista Simulator EnginE) généralisent la validation à l'ensemble des composants de l'architecture.

Le plan de ce chapitre est le suivant. Nous introduisons d'abord l'architecture du modèle Yavista. Nous présentons ensuite les objectifs stratégiques du projet en terme d'exécution parallèle, de flexibilité temps réel et de réutilisation. La dernière section décrit ELYSE^2.

2. Le Modèle YAVISTA

Le modèle Yavista se base sur une poignée de concepts simples garantissant la compréhensibilité du système, au risque même de surestimer ou sous estimer le couplage entre certaines caractéristiques.

- l'architecture YAVISTA est une architecture de référence [BAS98] utilisant des styles d'architectures variés et mappant le modèle de caractéristiques de référence YAVISTA décrit au chapitre 3.
- l'architecture YAVISTA dérive d'une mise en œuvre de la technologie FORM et offre trois niveaux de détail dans l'architecture : vue en modules, vue en processus, vue en sous modules
- la vue en modules est une architecture simple (voir Figure 5.1) dont les spécificités sont :
    - le déport des fonctionnalités de très bas niveau dans des composants d'implémentation séparés
    - l'utilisation d'une colonne compacte *MPDU Preparation*, *Store*, *Control*, permettant aux deux composants d'extrémité d'accéder aux informations flux du composant central *Store*
- la vue en processus met en jeu un nombre de processus proportionnel au nombre de flux maintenus. Les paquets des flux sont traités selon un mode de fonctionnement donné (DCF, etc). La gestion des

séquences de messages de chaque mode est réalisée à l'aide de deux processus l'un dédié à l'émission, l'autre à la réception. Les séquences de messages de chaque mode sont implémentées séparément côté émetteur et récepteur et modélisées à l'aide de diagrammes StateChart.

2.1. L'architecture conceptuelle de Yavista

L'apport de l'architecture Yavista est le support d'un vaste choix de caractéristiques alternatives dont les technologies télécoms à commutation de paquets. La particularité de ces technologies est l'utilisation de plusieurs canaux dédiés et partagés. Pour modéliser ces technologies, notre architecture exploite la caractéristique de *projection des flux* du composant *Multiplex*. Celui-ci aiguille les flux reçus vers les différents canaux de communication existants. Dans le sens inverse, à la réception, le composant *Decode* oriente les flux vers la pile de processus adaptée.

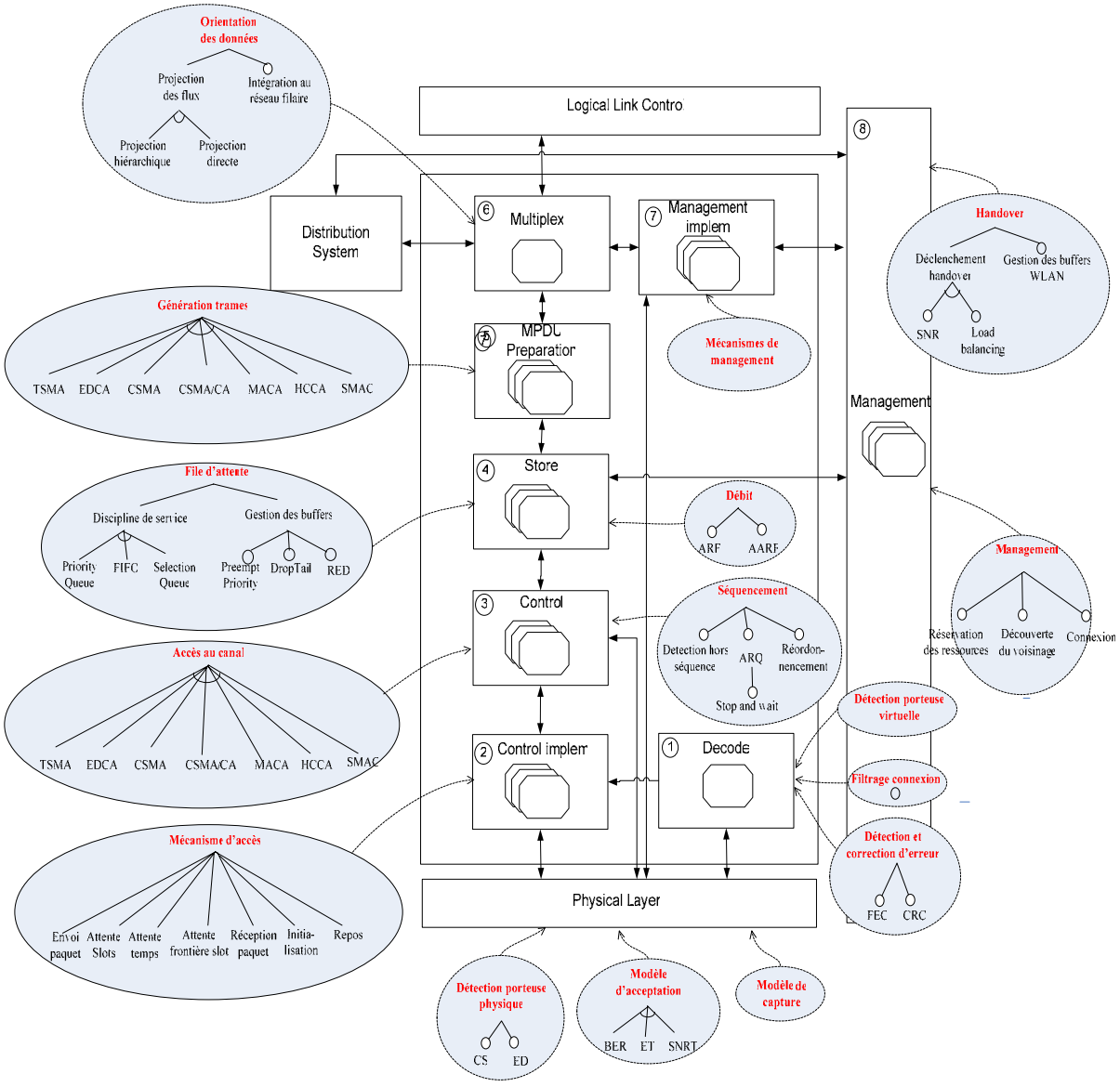


Figure 5.1 Mapping entres les processus et le modèle de caractéristiques Yavista

2.2. Le modèle d'accès au canal Yavista

L'architecture du modèle d'accès au canal Yavista est basée sur le concept de séparation des préoccupations issu du génie logiciel. Cette architecture est particulièrement adaptée à la modélisation des séquences de messages spécifiées dans les standards. Elle propose un module de haut niveau, *Control*, pour coordonner les séquences côté émetteur et récepteur et un module de bas niveau couplé, *Control Implem*, implémentant les techniques spécifiques d'accès au canal. Comme le montre la Figure 5.2, plusieurs utilisations du système sont possibles : systèmes à flot unique, à flots multiples à canaux physiques différentiés ou non.

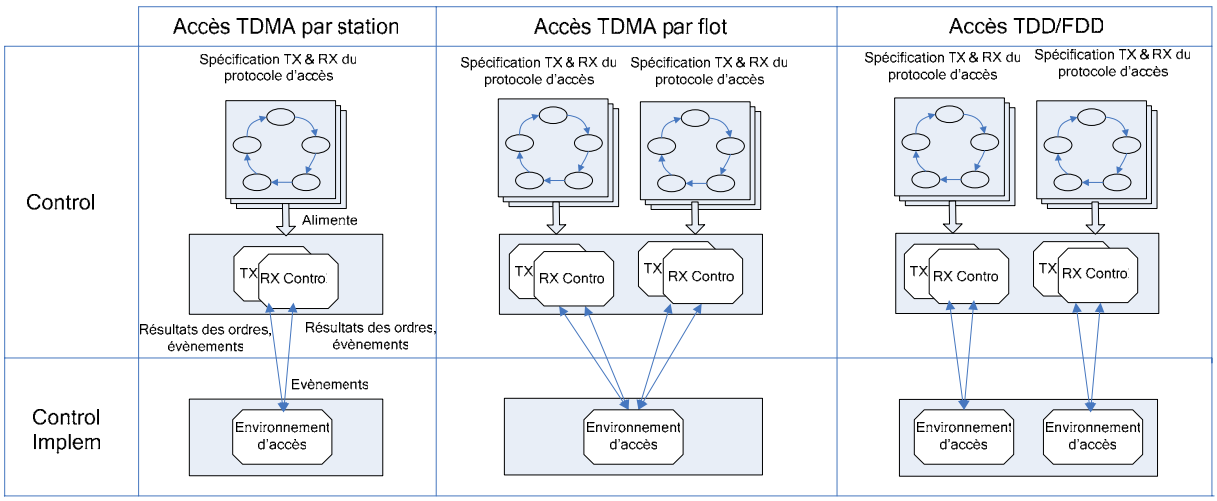


Figure 5.2 Architecture du simulateur Yavista pour plusieurs types d'accès au canal

3. Les Objectifs du Simulateur YAVISTA

3.1. Simulation parallèle

Yavista se voulant précis et complet, son implémentation exige des ressources mémoire et processeur importantes. Yavista est de plus limité car réservé à la simulation de la couche liaison des réseaux sans fil. Un modèle d'exécution parallèle dans la lignée du standard HLA (High Level Architecture) de collaboration des simulateurs [GOK06] permettra l'accélération de l'exécution et la collaboration avec d'autres simulateurs.

3.2. Sélection temps réel des caractéristiques

Les caractéristiques disponibles dans le simulateur Yavista se présentent sous la forme d'un arbre de caractéristiques. Du point de vue de l'utilisation, les avantages d'une interface de ce type sont les suivants :

- L'utilisateur configure simplement les nœuds du réseau en spécifiant la configuration de caractéristiques souhaitée parmi les caractéristiques FODA disponibles.
- Des systèmes externes tels que des agents peuvent reconfigurés dynamiquement les nœuds en changeant la configuration courante de caractéristiques. Le choix de la nouvelle configuration de

caractéristiques peut se baser sur les informations complémentaires accompagnant chaque caractéristique.

L'assemblage des noeuds du simulateur est guidé par les règles de composition des caractéristiques spécifiant les dépendances et exclusions mutuelles entre caractéristiques [BAT05].

Le mécanisme d'assemblage des interfaces à partir des sélections de caractéristiques est implémenté à l'aide d'un système réflexif dont le méta niveau est matérialisé par une configuration de caractéristiques et le niveau de base par les trois vues architecturelles FORM. Ce système intègre un mécanisme gérant les transferts d'état à chaque modification du niveau de base.

3.3. Réutilisation des éléments de simulation dans les cartes réseaux

Les fonctionnalités du simulateur (management, adaptation au canal, etc) pouvant faire l'objet d'une intégration avec les pilotes Wi-Fi généraux tels que MadWifi seront identifiées. Un modèle d'intégration permettra la greffe de ces éléments dans les drivers.

3.4. Validation complète avec ELYSEE

Le simulateur YAVISTA est un simulateur conçu pour être validé. ELYSEE est une extension du moteur ELYSE permettant la certification de l'ensemble des composants du modèle Yavista. ELYSEE reproduit pour ce faire une partie de l'architecture des stations et du modèle de caractéristiques Yavista.

Ainsi, comme le montre l'exemple de la Figure 5.3, sous ELYSEE,

- à chaque station est associée une instance de validation contenant elle-même autant d'instances que de processus à valider : RX, TX, processus de management, etc
- la structure en machine virtuelle du modèle Yavista est modélisée – comme sous ELYSE – à l'aide de références à des fichiers externes décrivant l'implémentation de chaque macro état.

Une propriété d'ELYSEE est que l'exécution des différents processus de validation est désynchronisée. Des règles de synchronisation additionnelles peuvent toutefois être ajoutées pour tester la collaboration des différents processus du réseau.

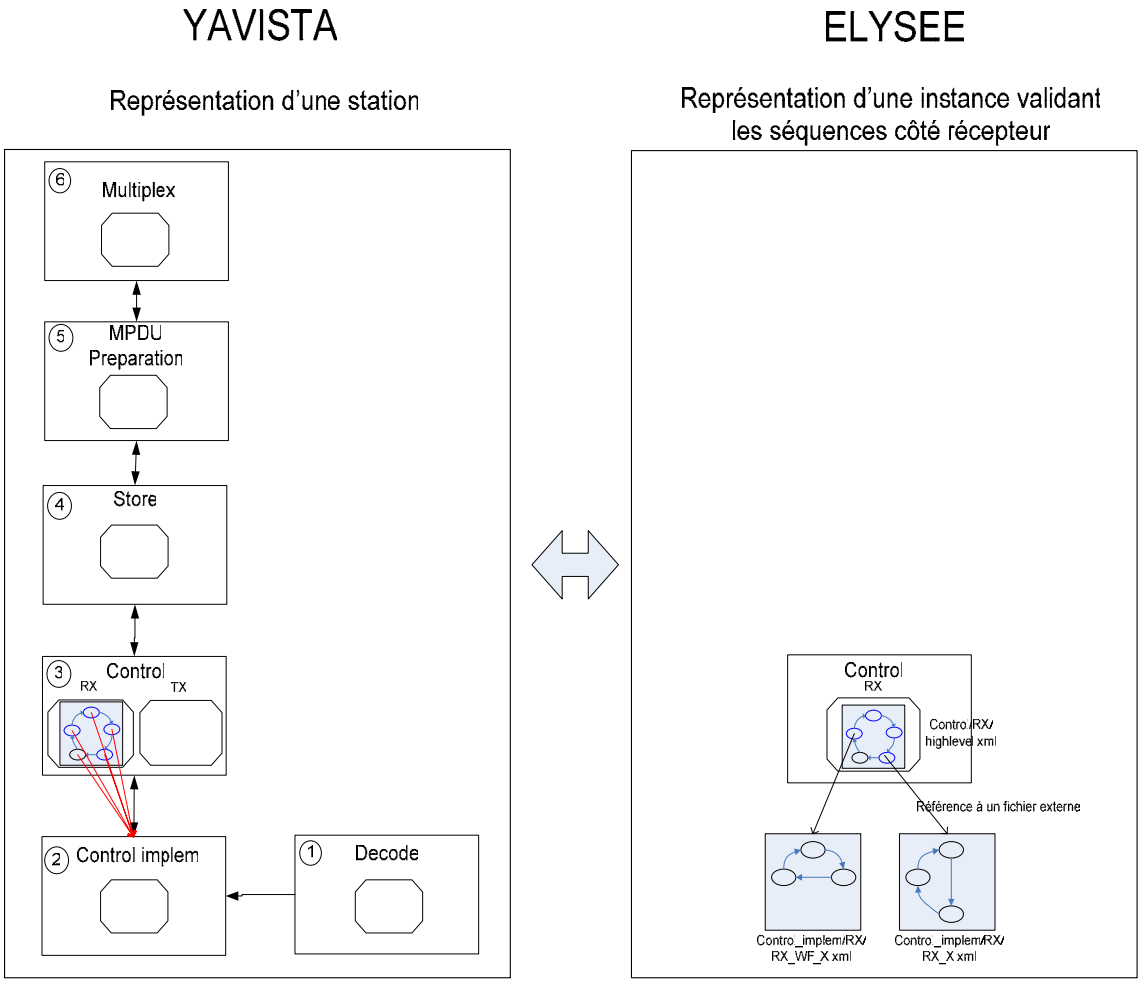


Figure 5.3 Parallèle entre stations Yavista et ELYSEE. Validation du processus RX de *Control*

4. Conclusion

Ce chapitre présente la structure d'accueil du modèle de caractéristiques FODA proposé au chapitre 3. Le modèle de simulation qui en découle définit les concepts phares à l'origine des réalisations proposées dans ce mémoire. Les différentes vues du système proposé avec la technologie FORM fournissent le niveau de flexibilité nécessaire à l'implémentation d'un grand ensemble de technologies de niveau liaison. La vue en modules propose une architecture simple, avec des modules à fonctionnalités bien définies. La vue en processus matérialise les différents circuits de traitement de l'information. Le système est lui-même causalement lié à un méta niveau permettant la modification du système par simple reconfiguration des caractéristiques. Ce type d'interface délimite ainsi clairement la frontière entre le domaine d'action des utilisateurs et celui des développeurs.

Références

[BAS98] L. Bass, P. Clements, R. Kazman, *"Software Architecture in Practice"*, Addison-Wesley, 1998

[BAT05] D. Batory, "Feature Models, Grammars, and Propositional Formulas", *Lecture Notes In Computer Science*, n° 3714, pp. 7-20, 2005

[BEK08] R. Ben-El-Kezadri, F. Kamoun, "A Flexible Channel Access Model for Wireless Network Interface Cards", dans *Proc of the 3rd International Conference on Communication Systems Software and Middleware (COMSWARE'08),* Bangalore, Inde, Janvier 2008

[GOK06] E.Göktürk, "Design of a higher level architecture for network simulators", dans *Proc. of the 20th European Conference on Modelling and Simulation (ECMS'O6)*, Mai 2006

[HUN06] C. Hunter, J. Camp, P. Murphy, A. Sabharwal, C. Chris, "A Flexible Framework for Wireless Medium Access Protocols", dans *Conference Record of the Fortieth Asilomar conference on Signals, Systems and Computers, (ACSSC '06)*, pp. 2046 – 2050, Pacific Grove, CA, Etats-Unis, Octobre 2006
WARP: Wireless Open-Access Research Platform,
Disponible : http://warp.rice.edu/trac/wiki

Conclusion

Dans ce travail de thèse, nous nous sommes intéressés à la prise en compte des besoins non fonctionnels de correction, de flexibilité et de réutilisabilité dans les simulateurs de réseaux sans fil. Nous avons opté dans ce cadre pour une approche modulaire et eu recours aux techniques d'ingénierie FODA et FORM pour modéliser les caractéristiques offertes par une interface réseau et les mapper dans une architecture logicielle conceptuelle. Pour illustrer le potentiel des principes, deux prototypes sont proposés, l'un pour la validation – ie le moteur ELYSE –, l'autre pour la simulation – ie l'interface CoW.

Cette conclusion générale présente une synthèse des principaux résultats et les perspectives de continuation du projet.

1. Synthèse

La conception d'un simulateur innovant est une entreprise difficile impliquant une exigence de crédibilité et un positionnement clair par rapport aux produits concurrents. Deux périodes se détachent dans notre travail :

- la première a vu le développement d'outils d'analyse permettant l'audit des simulateurs déjà existants
- la seconde a vu les concepts définis se matérialiser dans un prototype

De façon générale notre approche propose une refonte du domaine de la simulation des réseaux sans fil à travers des concepts dont chacun pourra s'inspirer. La crédibilité de la solution générale repose sur l'intégration sans concession de considérations a priori antinomiques dans le domaine à savoir, la prise en compte des détails du bas niveau et l'abstraction des concepts réseaux de haut niveau.

1.1. La validation des simulateurs

La validation des simulateurs est un domaine dans lequel il est difficile d'engager des projets car l'effort doit porter sur un cœur d'implémentation développé par des tiers et être accompagné de justifications solides. La stratégie élaborée dans ce cadre vise :

- à une évaluation stricte n'établissant pas de hiérarchie dans les défauts
- au développement d'outils aux conclusions irréfutables

Nous avons opté pour ce faire pour une méthode par force brute loggant l'ensemble de l'exécution de la couche MAC dans des fichiers de traces au format générique. Bien que la méthode s'avère lourde sous certains aspects (instrumentation des simulateurs), elle se distingue des autres tentatives dans le domaine, par sa précision et ses possibilités en terme de validation formelle et de comparaison. Pour le développeur

de protocoles, l'instrumentation en code neutre permet la génération de frises temporelles, la mesure de performance et la certification par rapport aux spécifications établies.

Nos principales contributions dans ce domaine, sont la démonstration de la viabilité du concept via la définition de la liste de caractéristiques la plus complète à ce jour sur les simulateurs NS-2.30 et Glomosim-2.03, la spécification d'un format de trace générique appelé langage neutre et le développement d'un framework dédié à la compréhension des simulateurs. La pièce maîtresse de ce framework est le moteur de validation ELYSE (cf chapitre 1). ELYSE se base sur une modélisation à deux niveaux des protocoles inspirée du simulateur YAVISTA. Les principales propriétés d'ELYSE sont le scriptage intégral des règles de validation au format StateChart, la validation quasi native du code, la possibilité de continuer la vérification du code neutre en cas d'erreur dans une séquence, la séparation des règles de validation côté récepteur et émetteur et la séparation des règles pour chaque séquence de messages.

1.2. La flexibilité et la réutilisabilité dans les simulateurs sans fil

La mise au point d'une interface flexible ne se limite pas à un jeu de réflexion inutile. Le problème est réel puisqu'une reprise en main des simulateurs généraux est déjà lancée avec YANS et NS-3. Il s'agit également de s'attaquer au mythe des machines totalement recomposables. Le domaine de la simulation offre en cela les méthodes et les moyens de programmation nécessaires pour y parvenir. L'entreprise suppose deux étapes. La compréhension du fonctionnement de la couche liaison des réseaux et la définition d'une architecture logicielle.

L'aspect saisissant de notre étude, dans ce domaine, est le recours exclusif à des principes d'ingénierie existants – style architectural, variabilité (cf chapitre 2), méthode FODA (Feature Oriented Domain Analysis) et FORM (Feature Oriented Reused Method) – et l'imbrication étroite des concepts.

1° Analyse du domaine avec la méthode FODA

Le domaine analysé couvre les standards IEEE 802.11, OSI et UMTS. Les fonctionnalités offertes par la couche liaison sont capturées à l'aide d'un modèle de caractéristiques FODA (cf chapitre 3). Chaque caractéristique est à ce titre précisément redéfinie pour les besoins d'intégration à l'ensemble. Les caractéristiques proposées s'étendent de la file d'attente à la gestion de l'adaptation aux conditions environnementales. L'identification des caractéristiques variantes est une étape nécessaire dans le développement de logiciels adaptables.

2° Synthétisation d'une architecture avec la méthode FORM

Les caractéristiques du modèle FODA sont mappées dans une architecture logicielle à l'aide de la méthode FORM. La méthode FORM propose trois vues architecturelles, chacune ajoutant un niveau de détail supplémentaire. La vue en modules du prototype CoW exhibe les liens de couplage entre les principaux groupes de caractéristiques définies. La vue en processus permet la représentation du fonctionnement des processus parallèles gérant l'émission et la réception. Les Figures 4.4 et 4.5 illustrent à ce titre le fonctionnement de ces processus et leur couplage avec les autres caractéristiques. La vue en sous modules permet enfin l'identification des caractéristiques variantes à l'intérieur des composants. La viabilité des principes est démontrée à travers le portage des spécifications du mode DCF de 802.11 et de l'implémentation TSMA de Glomosim. L'assemblage des cartes est réalisé à l'aide d'un contrôleur

détectant chaque nouvelle insertion de composants dans l'environnement du middleware.

2. Perspectives

Les perspectives du projet s'inscrivent selon deux lignes : l'aide à la recherche et la recherche.

2.1. L'aide à la recherche avec la suite Yavista

Compte tenu de l'ancrage du projet dans le logiciel, et le succès relatif de la mise en production de la suite Yavista (voir chapitre 1), un effort sera engagé pour créer une vitrine du projet et développer les services du portail yavista.sourceforge.net. Les objectifs seront de faire de Yavista a) une autorité dans le domaine de la certification des simulateurs MANET, b) une référence en mesure de performance, c) de promouvoir les travaux de recherche associés.

Les services du nouveau portail concerneront :

- la publication de recommandations sur les simulateurs MANET,
- la diffusion de YAVISTA sequencer, YAVISTA grapher et YAVISTA cerifier,
- l'amélioration des outils comme l'interface de YAVISTA grapher pour afficher les chemins IP dans le réseau et prendre en compte les fichiers de grande taille (à l'aide d'un mécanisme de segmentation des traces).
- l'instrumentation des simulateurs et des réseaux réels

La stratégie reposera sur une communication moderne, via du contenu vidéo, et l'utilisation de licences libres. Initialement conçu comme démonstrateur du simulateur Yavista, le logiciel pourrait opter pour un nouveau standard de trace dans le cas d'un rapprochement avec d'autres projets de simulation.

2.2. La recherche avec le simulateur Yavista et ELYSE

Plusieurs perspectives de recherche plus ou moins directes sont envisageables. Dans l'ordre d'effort croissant, nous citons :

- la standardisation du langage neutre : Du format du langage neutre dépend la complexité du code d'instrumentation et du certifieur. Plus le langage neutre s'affranchit des spécificités de modélisation des simulateurs et devient complet, plus le certifieur se simplifie. La contrepartie est la complexification du code d'instrumentation. Standardiser le code neutre consiste à proposer le langage le plus simple à valider en connaissance des spécificités de modélisation des simulateurs à instrumenter.
- la validation avec ELYSE : cet axe concerne l'extension du langage neutre à d'autres protocoles. Les études concerneront plus particulièrement les protocoles du plan de contrôle des réseaux (routage, etc).
- le développement du simulateur Yavista : l'effort doit être mesuré en fonction qu'il est destiné à l'intégration du modèle aux systèmes réels ou reste cantonné au domaine de la simulation. En tout état de cause, le contrôle du modèle par une logique adaptative agent doit être la priorité. Un prototype réflexif proposant une interface FODA sera proposé à ce titre.
- l'implémentation d'un sous arbre de caractéristiques plus complet : les recherches concerneront l'implémentation de 802.11e et UMTS.

Bibliographie personnelle

Conférences internationales

W. Berrayana, *R. Ben-El-Kezadri*, F. Kamoun, G. Pujolle, "DelayEDD-HCCA and RT-HCCA: two new IEEE 802.11e schemes based HCCA supporting Real-time applications", dans *Proc of the 2nd International Mobile Multimedia Communications Conference (MobiMedia'06)*, Alghero, Italie, Septembre 2006

R. Ben-El-Kezadri, F. Kamoun, "Graphic Visualization of the 802.11 DCF Protocol Under the NS-2 Simulator", dans *Proc of the Fourteen Annual Simulation Symposium (ANSS'07)*, Norfolk, VA, Etats-Unis, Mars 2007

R. Ben-El-Kezadri, F. Kamoun, "Towards MANET Simulators Massive Comparison and Validation", dans *Proc of the 18th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications (PIMRC'07),CAMAD Workshop,* Athènes, Grèce, Septembre 2007

R. Ben-El-Kezadri, F. Kamoun, "A Flexible Channel Access Model for Wireless Network Interface Cards", dans *Proc of the 3rd International Conference on Communication Systems Software and Middleware (COMSWARE'08),* Bangalore, Inde, Janvier 2008

Journaux

R. Ben-El-Kezadri, F. Kamoun, "YAVISTA: A Graphical Tool for Comparing 802.11 Simulators", Journal of Computers (JCP), 3(1), 2008

Annexe A : Identification des Caractéristiques de NS-2.30 et Glomosim 2.03

Cette annexe détaille précisément l'ensemble des caractéristiques de NS-2 et Glomosim présentées à la Figure 1.16. Chaque caractéristique est classée comme interprétation du standard, simplification de la norme ou erreur de modélisation. Les tables A.1, A.2 et A.3 recensent l'ensemble des caractéristiques de chaque type et les clauses ou sections du standard [IEE99] auxquelles elles contreviennent.

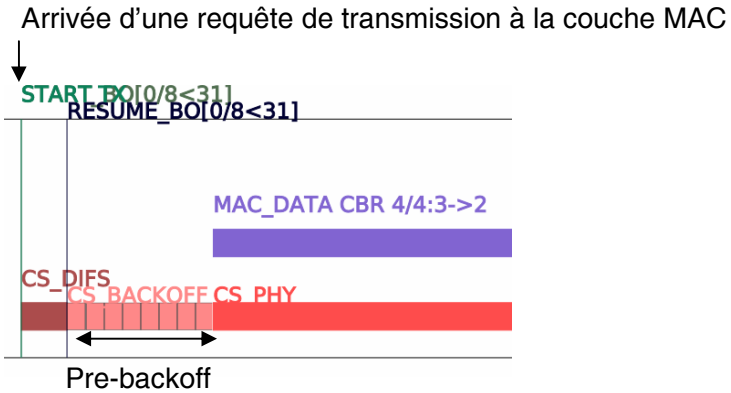
1. Interprétations du Standard

Table A.1 Interprétations du standard IEEE 802.11

	NS-2	Glomo	Clause du standard 802.11
1) Accès au canal lorsque le medium est libre	×		9.2.5.1
2) Utilisation de EIFS si erreur PLCP	×		9.2.3.4, pp391, pp460

1.1. Accès au canal lorsque le medium est libre

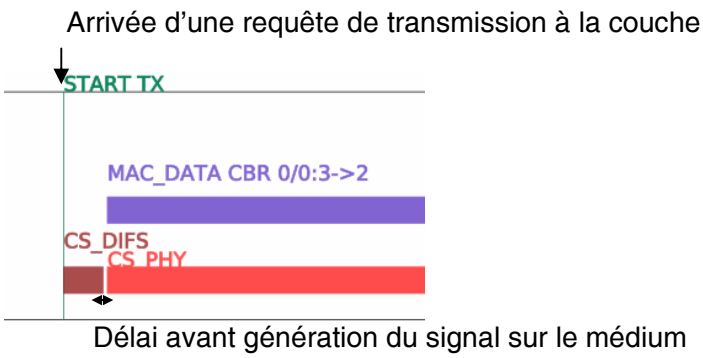
NS-2) Sous NS-2, une station attend une procédure de pre-backoff avant d'émettre même si le medium est libre. La frise A.1.a décrit le comportement d'une station NS lorsque le medium est libre. Comme le montre la frise, la période de pre-backoff comprend un IFS (*CS_DIFS*) et un nombre aléatoire de slots de backoff (*CS_BACKOFF*). La transmission est indiquée sur la frise par l'évènement *MAC_DATA*.



Frise A.1.a Accès au canal lorsque le medium est libre (NS-2)

Glomosim) Glomosim propose une interprétation plus proche du standard. Celui-ci stipule que l'accès au medium est immédiat si le canal est libre depuis une période supérieure ou égale à DIFS. La frise A.1.b décrit l'interprétation de Glomosim de l'accès au medium. La trame est transmise (*MAC_DATA*) un IFS (*CS_DIFS*) après la demande de transmission de la couche MAC (*START TX*). A noter que si le canal est

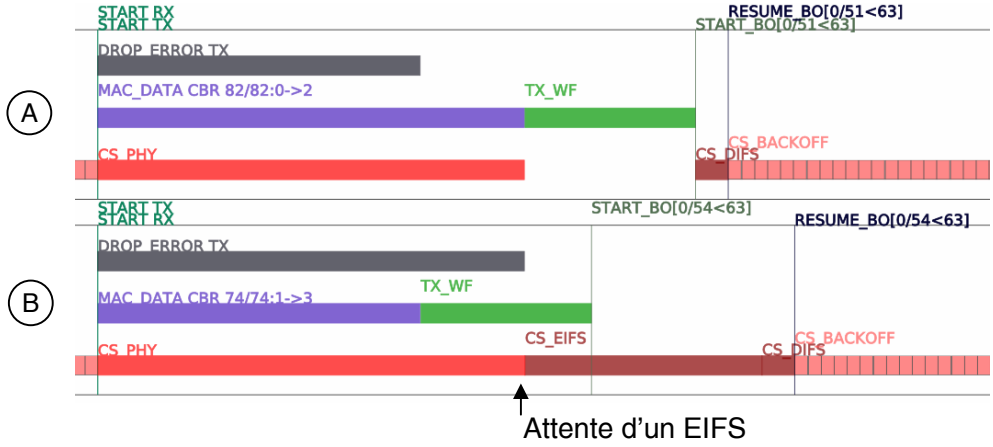
perçu occupé avant l'expiration de l'IFS, la transmission est réessayée un IFS après que le canal soit redevenu libre.



Frise A.1.b Accès au canal lorsque le medium est libre (Glomosim)

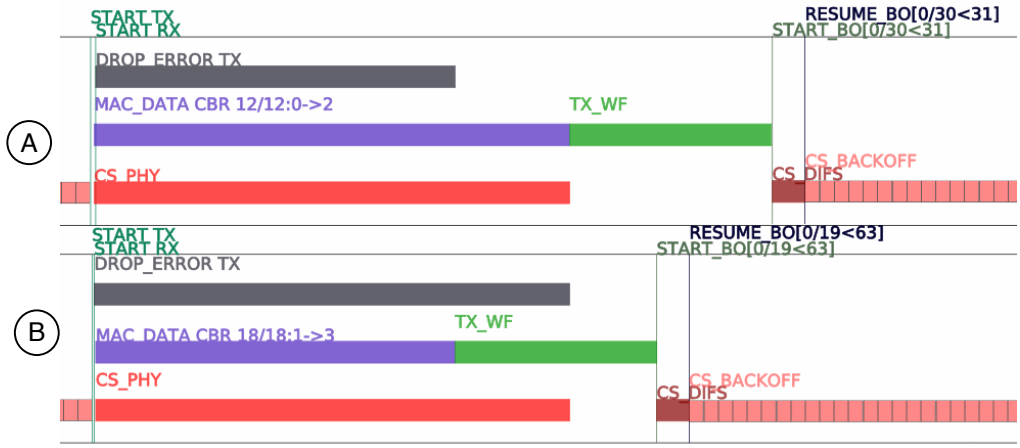
1.2. Utilisation de EIFS si erreur PLCP

NS-2) Quand une station reçoit une trame partielle sans entête PLCP, elle attend un intertrame EIFS après que le medium redevienne libre. Etant donné que l'EIFS ne peut être utilisé que quand l'entête PLCP est correctement acquise, ce comportement contrevient au standard. La frise A.2.a illustre ce phénomène avec un chevauchement de transmissions et de réceptions. Les stations A et B sont à portée. Puisque la station B est en train d'émettre au début de la réception du paquet de A, elle ne peut pas décoder correctement son entête PLCP. B attend pourtant un EIFS (*CS_EIFS*) après que le medium soit redevenu libre.



Frise A.2.a Utilisation de EIFS si erreur PLCP (NS-2)

Glomosim) Glomosim est d'avantage conforme à la norme que NS-2 dans ce cas, car il n'utilise pas d'EIFS. Dans le cas présenté sur la frise A.2.b, la station B relance la procédure de backoff à l'expiration du timer d'ACK (*TX_WF*) indiquant l'échec de sa transmission.



Frise A.2.b Utilisation de EIFS si erreur PLCP (Glomosim)

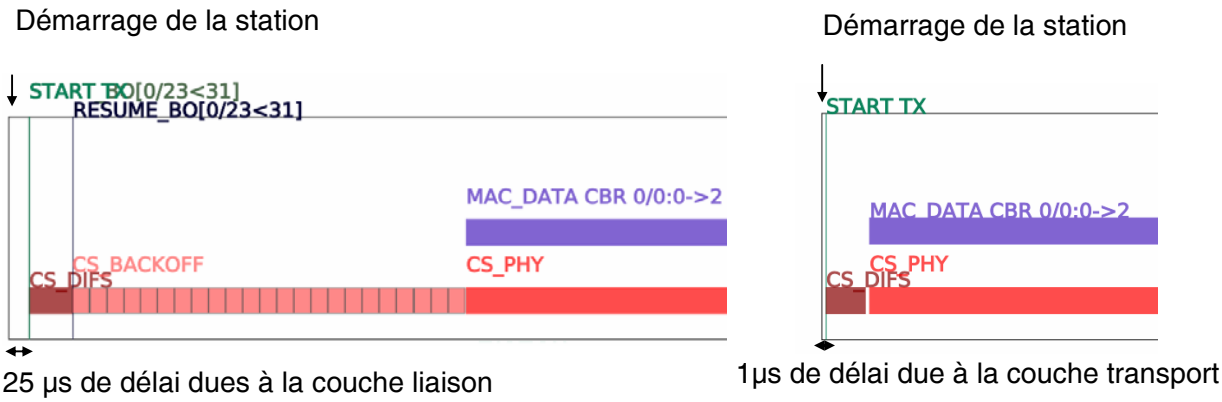
2. Simplifications du Système

Table A.2 Simplifications du système

	NS-2	Glomo	Clause du standard 802.11
1) Comportment des stations au démarrage	×	×	pp338, pp457
2) Capture RTS	×		N/A
3) Décrémentation du backoff		×	9.2.5.2
4) Modèle de capture	×	×	N/A

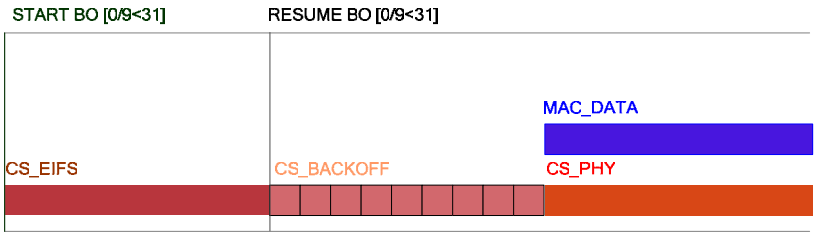
2.1. Comportement des stations au démarrage

NS-2, Glomosim) Lorsque les stations rejoignent le canal, celles-ci n'exécutent pas la procédure d'initialisation standard consistant à attendre un EIFS et un nombre aléatoire de slots de backoff avant la transmission. La frise A.3.a montre le comportement d'une station possédant du trafic pendant (c'est-à-dire en attente dans les files) au démarrage sous NS-2 et Glomosim. La procédure d'accès est globalement similaire à celle décrite aux frises A.1.a et A.1.b.



Frise A.3.a Comportement des stations au démarrage de NS (à gauche) et Glomosim (à droite)

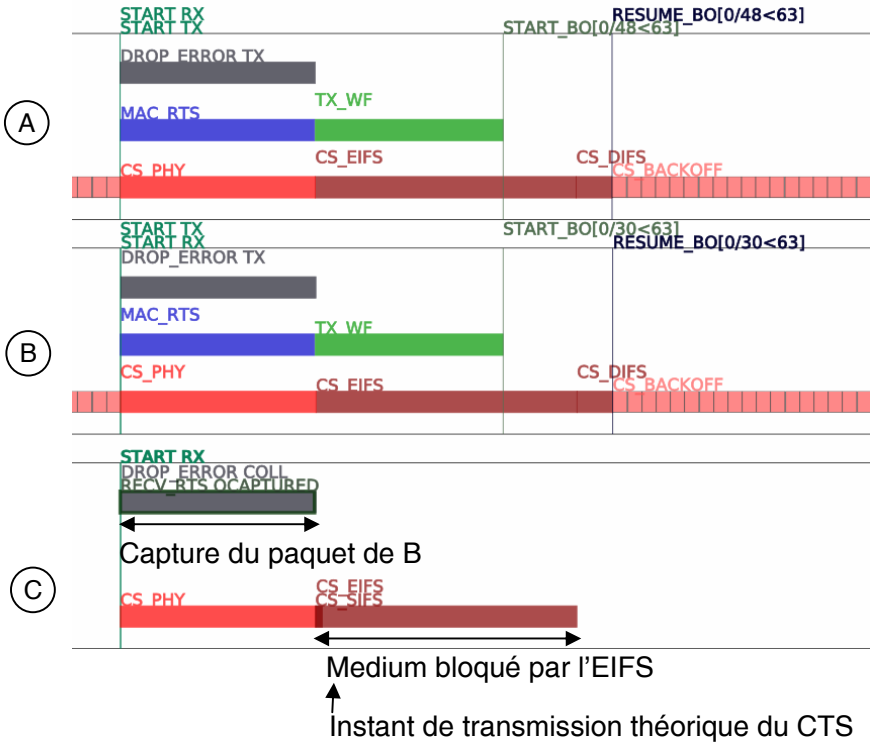
Standard 802.11) La frise A.3.b montre comment une station se conformant au standard se serait comportée. Comme illustré, la station aurait attendu un EIFS (*CS_EIFS*) plus une période de backoff (*CS_BACKOFF*).



Frise A.3.b Comportement des stations au démarrage (Standard 802.11)

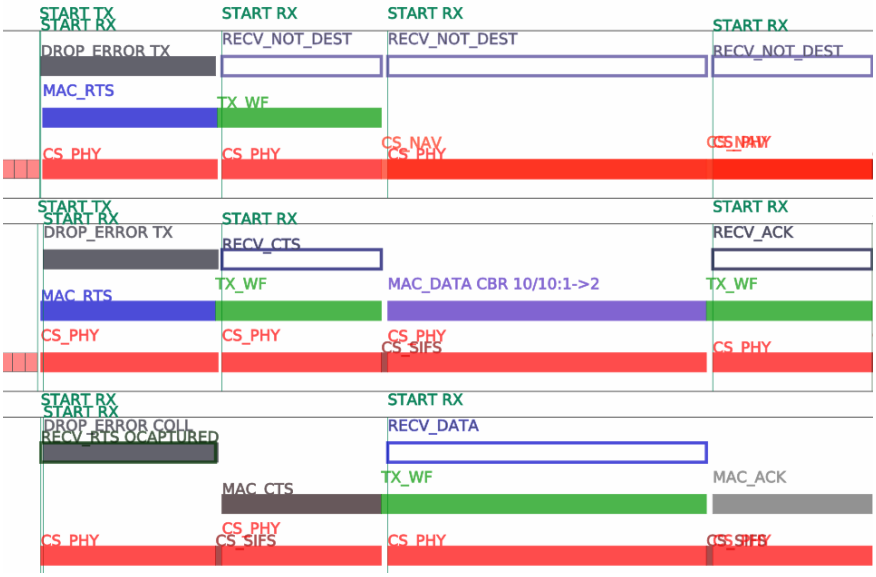
2.2. Capture RTS

NS-2) NS-2 avorte la procédure de RTS dans le cas d'une collision suivie d'une capture de RTS. Sur la frise A.4.a, les stations A et B communiquent avec la station C en utilisant le mécanisme RTS/CTS. Deux RTS sont transmis simultanément et entrent en collision à la station C. Compte tenu de la proximité entre les stations B et C, le paquet de la station B est correctement décodé et celui de A est ignoré. Mais malgré que C décode correctement le paquet RTS de B et que le médium est libre, C ne répond pas à la station B avec un paquet CTS. La raison est que la station C doit attendre la fin de l'EIFS – généré après la collision – avant de pouvoir transmettre. Comme l'EIFS n'est pas terminé au moment d'envoyer le CTS, le CTS ne peut donc jamais être envoyé dans ce cas. Les conséquences de ce phénomène sont analysées dans [KOC04].



Frise A.4.a Capture RTS (NS-2)

Glomosim) Contrairement à NS-2, sous Glomosim la procédure de RTS continue normalement après une collision entre deux paquets RTS. La frise A.4.b illustre le comportement de Glomosim en cas de collision de RTS.



Frise A.4.b Capture RTS (Glomosim)

2.3. Décrementation du backoff

La décrémentation du backoff met en jeu le temps de propagation sur le medium et le temps de passage de l'état de réception à l'état d'émission dans la couche physique. La façon dont un simulateur prend en compte ces éléments peut donner lieu à deux modèles de décrémentation différents : discret comme NS-2 ou continu comme Glomosim.

NS-2) Sous NS-2, lorsqu'une trame est transmise sur le medium, toutes les stations du réseau décrémentent leur backoff du même nombre de slots. La preuve est géométrique : soit B, la station ayant transmis la dernière trame, C la station ayant transmis l'avant dernière trame et A une station quelconque. Soit N_A et N_B, le nombre de slots de backoff restants de A et B avant la transmission de B ($N_B < N_A$). Les temps de propagation t_{AC}, t_{AB} et t_{BC} entre les différentes stations sont représentés à la Figure A.1. A, B et C sont chacune à portée respective. C peut être confondue avec A ou B.

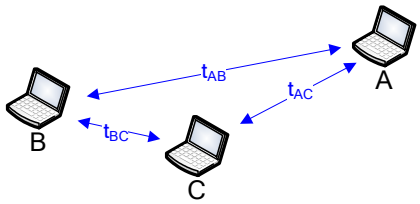


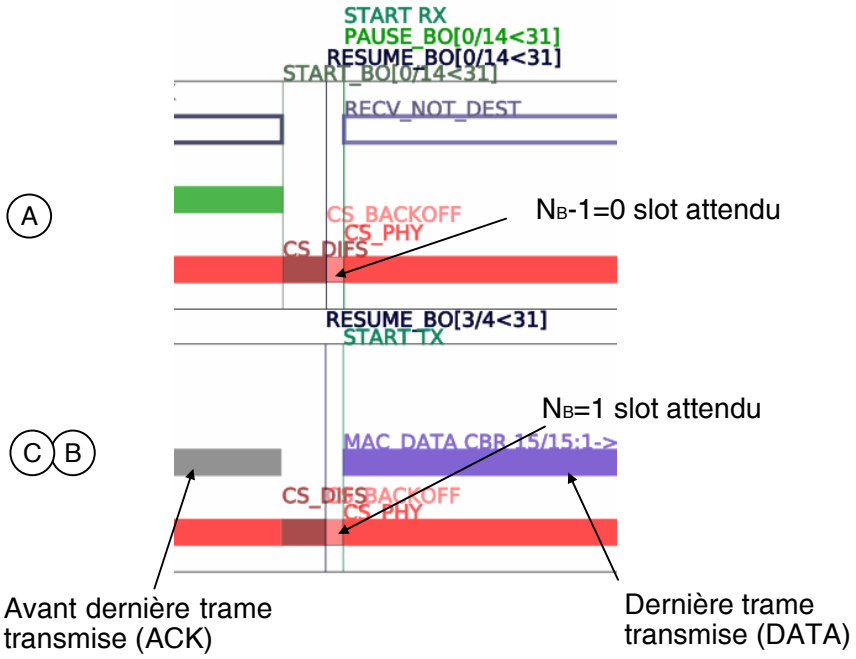
Figure A.1 Temps de propagation sur le réseau

Un simple raisonnement géométrique montre que si *B* attend $T_B = N_B * SlotTime$ avant de transmettre, la station *A* considérera le medium libre durant la période :

$$T_A = N_B * SlotTime + t_{CB} + t_{AB} - t_{AC}$$ [43]

[43] NS ne prend pas en compte le temps de passage de l'état de réception à l'état d'émission dans la couche physique.

Si $t_{CB} + t_{AB} - t_{AC} < 0$, la station A attend moins de N_B slots. En conséquence, NS-2 décrémente N_A de N_B-1 slots seulement. Sinon, N_A est décrémenté de N_B slots. La preuve du lemme repose sur le fait que $t = t_{CB} + t_{AB} - t_{AC}$ est toujours positif. Malheureusement, lorsque les trois nœuds sont alignés ou que B et C sont confondues, le manque de précision de l'unité flottante FPU dans le calcul de T_A à l'interruption du timer de backoff biaise le calcul de t. Dans ce cas, le nombre de slots à décrémenter (N_B-*1* ou N_B) est aléatoire (voir frise A.5.a) et le nombre de collision/capture sous estimé par rapport à la théorie (en théorie une collision/capture survient si A et B transmettent dans le même slot, c'est à dire si ($|N_A - N_B| = 0$).



Frise A.5.a Décrémentation du backoff dans un réseau à deux nœuds. B et C sont confondues (NS-2)

Glomosim) Sous Glomosim, deux éléments majeurs régissent la décrémentation du backoff :

- il existe une période de temps irréversible Δ de 5 µsec entre le moment où une station décide de transmettre un RTS ou un paquet de données[44] et le moment où le premier bit de la trame est effectivement transmis sur le medium
- quand une trame est détectée sur le medium, chaque station décrémente son backoff de la durée pendant laquelle le médium a été perçu libre. Un raisonnement géométrique montre que si la station B attend $T_B = S_B$ avant de transmettre[45], la station A perçoit le medium libre pendant:

$$T_A = S_B + t_{CB} + t_{AB} - t_{AC} + \Delta$$

Sachant que T_A et T_B sont réels, le backoff des stations n'est pas synchronisé et expire à des instants approximativement multiples de Δ. Une collision/capture survient entre les stations A et B si S_A et S_B expirent dans un intervalle de Δ. De fait, si A et B s'apprêtent à transmettre dans le même slot, la probabilité de collision/capture entre les stations A et B vaut approximativement:

[44] appartenant à une séquence DATA/ACK ou DATA
[45] S_B est la valeur de backoff en sec

$$P = \frac{2\Delta}{SlotTime} = 0{,}5$$

Le résultat est illustré à la Figure A.2 :

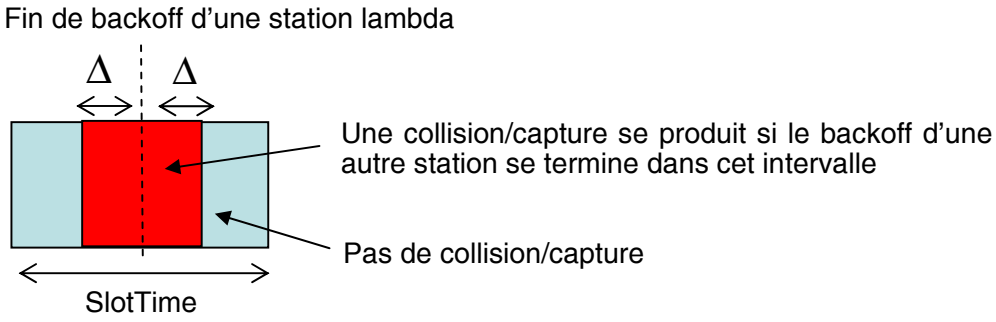
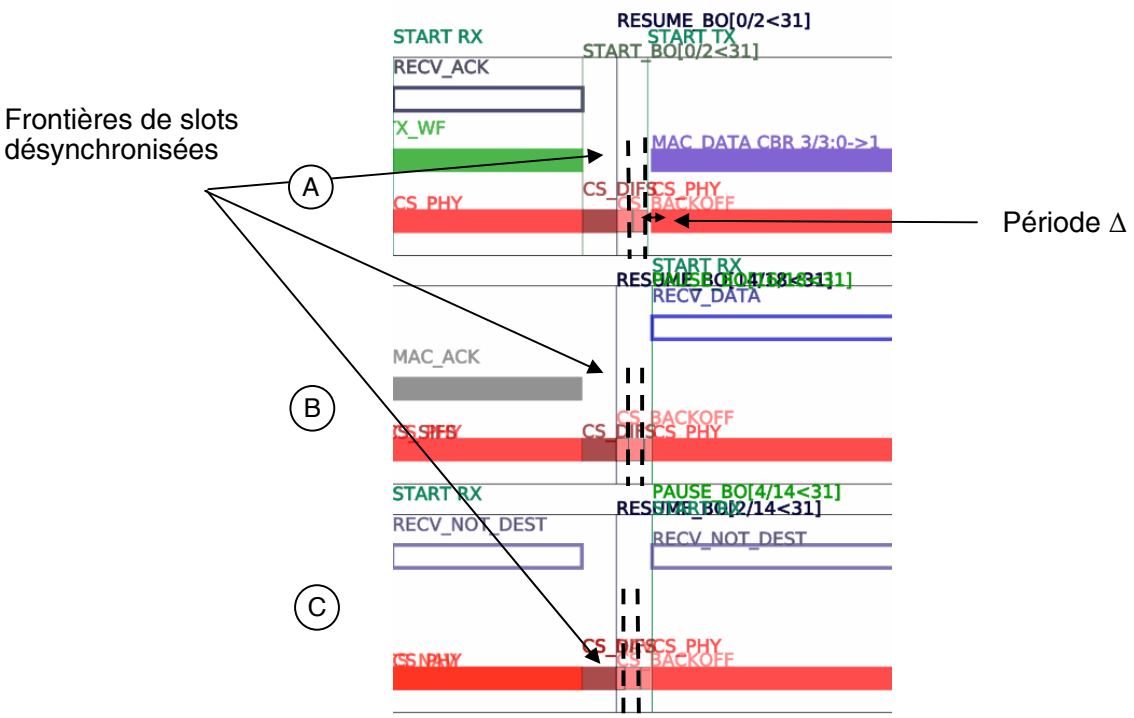


Figure A.2 Illustration du cas de capture/collision sous Glomosim

A cause de la désynchronisation des backoffs, ce modèle produit en moyenne deux fois moins de collisions/captures que NS-2 quand le medium atteint sa capacité limite. La frise A.5.b montre l'évolution des backoffs des stations *A*, *B* et *C* entre la transmission d'un ACK et la trame de donnée suivante. La désynchronisation se manifeste par le fait que les frontières de slots des différentes stations ne correspondent pas exactement.



Frise A.5.b Décrémentation du backoff sous Glomosim

La différence théorique de 100% entre NS-2.30 et Glomosim 2.03 est confirmée par les résultats expérimentaux des Figures A.3 et A.4. Les graphes montrent la probabilité de collision des paquets. Ils sont obtenus en moyennant les résultats de 9 simulations (3 topologies différentes[46] pour 3 configurations différentes des générateurs de nombres aléatoires). Une collision est comptée à chaque fois qu'un paquet transmis ne reçoit pas d'acquittement. Les intervalles de confiance sont calculés à 95%. Les Macs PDUs ont pour taille 76 octets. Les résultats théoriques de [TAY01], validés contre le simulateur de Bianchi, sont également reportés sur les figures.

Les expérience reproduisent les conditions décrites dans [TAY01]: les noeuds sont tous à portée, il n'y a pas de bruit, le medium atteint sa capacité limite, la charge UDP est équilibrée, les simulations durent 10 secondes. Seules 5 secondes de régime permanent sont conservées.

A la différence de l'expérience de la Figure A.4, l'expérience de la Figure A.3 neutralise complètement l'effet du modèle de capture en associant la même puissance à tous les paquets reçus. Un constat intéressant est la convergence relative du rapport des probabilités de collisions observées sous les deux simulateurs. La raison est que la valeur théorique P calculée sous Glomosim, n'est plus complètement valable lorsque le nombre de nœuds, devant théoriquement entrés en collision à la fin d'un backoff, dépasse 2.

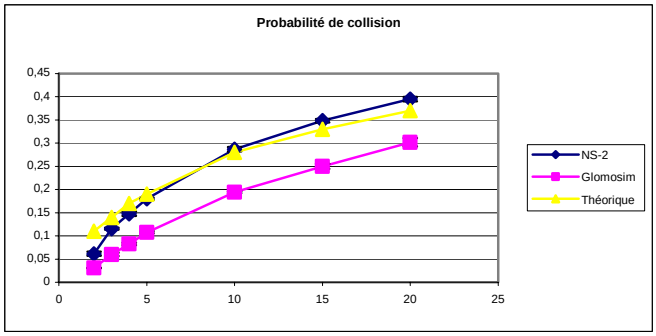


Figure A.3 Comparaison de la probabilité de collisions des paquets sous NS-2, Glomosim et la théorie (neutralisation du modèle de capture, dans une zone de $1m^2$)

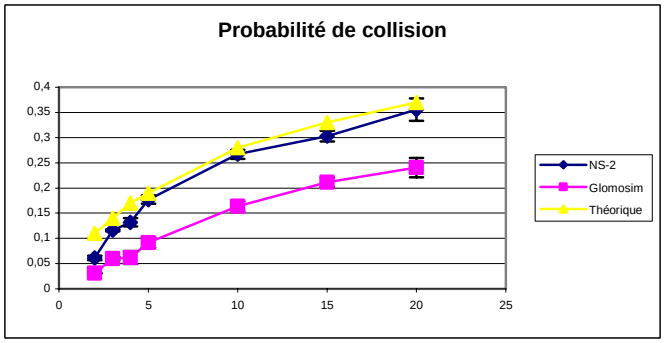


Figure A.4 Comparaison de la probabilité de collisions des paquets sous NS-2, Glomosim et la théorie (modèle de capture actif, dans une zone de $1m^2$)

[46] Une topologie correspond à un placement aléatoire des nœuds dans une zone à $1m^2$. Du fait de la proximité des stations dans cette scène, le rapport à puissance des paquets transmis simultanément est généralement proche de 1. De sorte, le modèle de capture des deux simulateurs agit de même et conclue à la collision et la perte des deux paquets [Bek07]. Du fait de la proximité des stations, le recours à l'EIFS est également limité.

2.4. Modèle de capture

NS-2 et Glomosim n'utilisent pas le même modèle de capture. La Figure A.5 montre que les deux simulateurs se comportent identiquement seulement si le ratio de puissance entre le nouveau paquet reçu et l'ancien appartient à l'intervalle [0.1, 10]. De plus dans certains cas, Glomosim et NS-2 présentent des simplifications importantes comme expliqué ci-dessous.

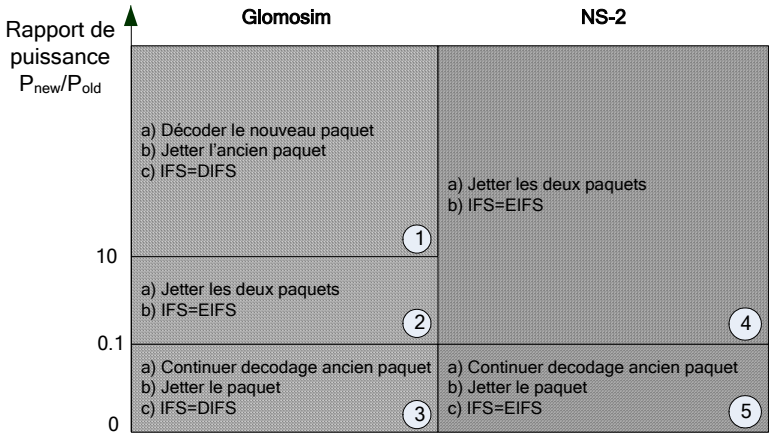
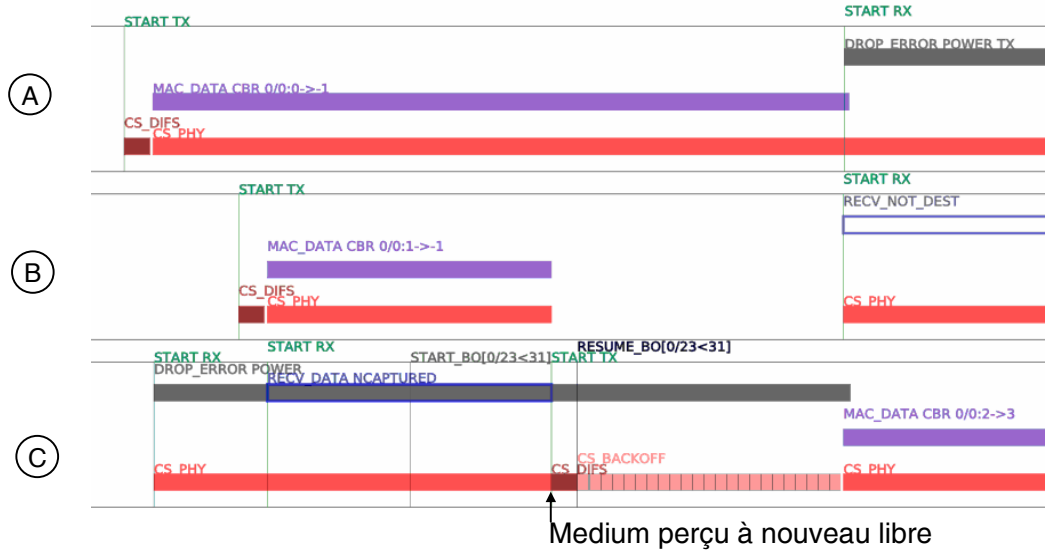
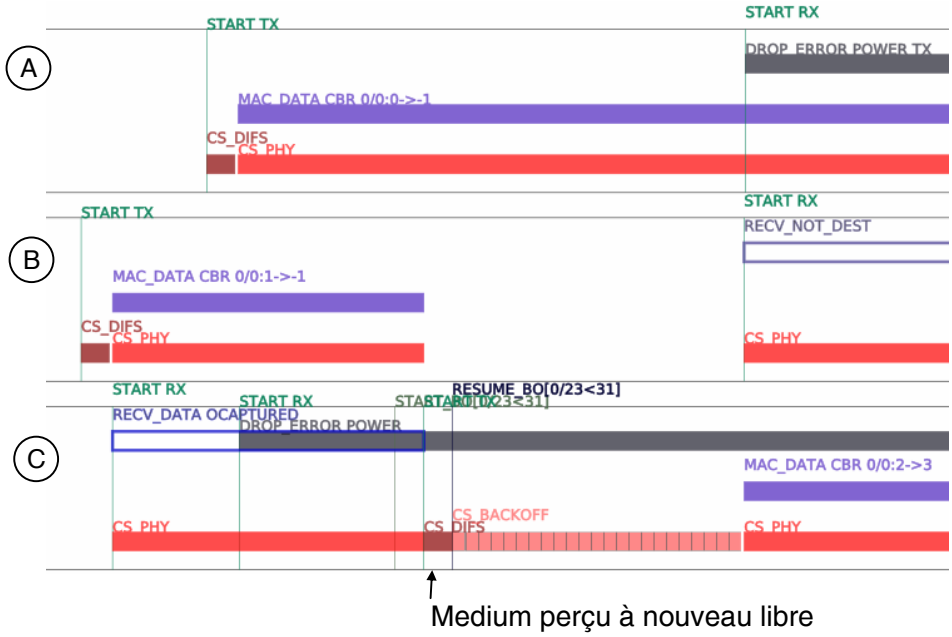


Figure A.5 Comportement de NS-2 et Glomosim en cas de capture en fonction du ratio de puissance des paquets reçus

Glomosim) Dans les cas 1 et 3, si c'est le paquet se terminant en dernier qui est ignoré, la vue du canal retourne à l'état libre à la fin de la réception du paquet se terminant en premier. Les frises A.6.a et A.6.b montrent deux scénarios de capture correspondants aux cas 1 et 3. Comme nous pouvons le voir, malgré que les stations A et C soient à portée d'entente, le paquet broadcasté par A est ignoré par la station C, car la trame est rejetée par le modèle de capture. Dans les deux cas, C perçoit de fait le canal libre pendant la transmission de A.



Frise A.6.a Modèle de capture (cas 1) (Glomosim)



Frise A.6.b Modèle de capture (Glomosim) (cas 3)

NS-2) Dans le cas 5, l'EIFS peut être mal utilisé si le second paquet se termine après le paquet capturé sous NS. En effet, dans ce cas, l'EIFS commence à t_0 (fin du paquet se terminant en premier) au lieu de t_1 (fin de la détection d'énergie théorique sur le canal). La frise A.6.c montre comment deux paquets broadcastés par les stations A et B sont capturés au niveau de la station C.

Frise A.6.c Modèle de capture (NS-2)

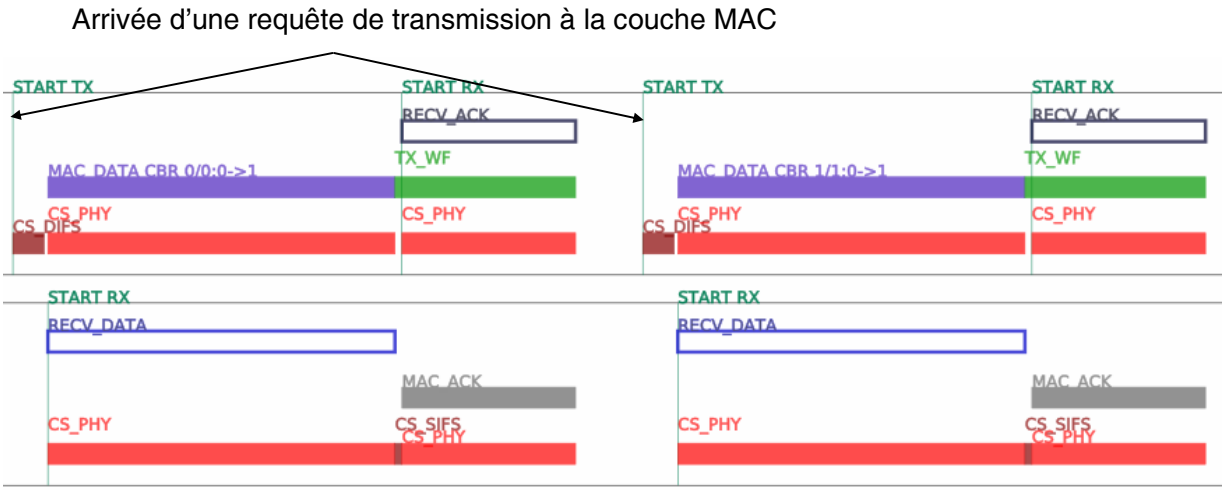
3. Erreurs de Modélisation

Table A.3 Erreurs de modélisation

	NS-2	Glomo	Clause du standard 802.11
1) Gestion du backoff quand les files sont vides		×	9.2.5.2
2) Durée IFS	×	×	9.2.10
3) Resynchronisation EIFS	×	×	9.2.3.4
4) Perturbation des échanges de trames		×	9.2.5.1
5) Remise à zéro EIFS		×	9.2.5.2

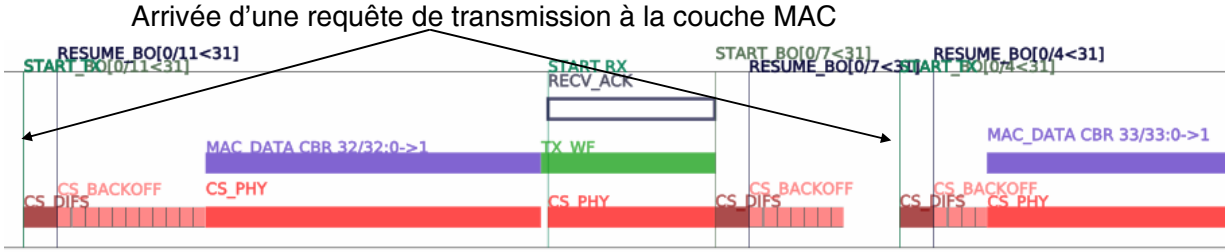
3.1. Gestion du backoff quand les files sont vides

Glomosim) Glomosim ne lance pas la procédure de backoff à la fin d'une transmission s'il ne reste plus de paquets à écouler dans les files d'attente de la couche MAC. La frise A.7.a montre comment Glomosim se comporte lorsqu'arrive une requête de transmission alors que la file est vide. Observons la fin de la première transmission. Celle-ci n'est pas suivie d'une période de backoff et le prochain paquet arrivant dans la file est transmis directement après un DIFS (*CS_DIFS*).



Frise A.7.a Gestion du backoff quand les files sont vides (Glomosim)

NS-2) La frise A.7.b reproduit la même expérience sous NS. Comme illustré, NS relance une nouvelle procédure de backoff après la fin de la transmission de la première trame.



Frise A.7.b Gestion du backoff quand les files sont vides (NS-2)

Ce dysfonctionnement affecte particulièrement le calcul de la capacité d'un réseau MANET simple constitué d'un émetteur CBR (noeud A) et d'un récepteur (noeud B). Dans l'expérience suivante, la taille de la file d'attente de A est fixée à l'infini et le débit physique à 2Mbps/sec. Chaque simulation est répétée trois fois. On mesure le nombre de trames transmises de A vers B durant une seconde en diminuant successivement la période de génération des paquets de A. La capacité maximale mesurée sous NS diffère de 29% avec celle de Glomosim lorsque la taille des MAC PDUs vaut 126 octets. Le scénario au pire des cas correspond à la taille minimale de PDU configurable sous les deux simulateurs à savoir 76 octets. Dans ce cas la différence atteint 36% (voir Figure A.6). La discontinuité observée sur Glomosim est due à la différence de comportement quand la file est vide ou non.

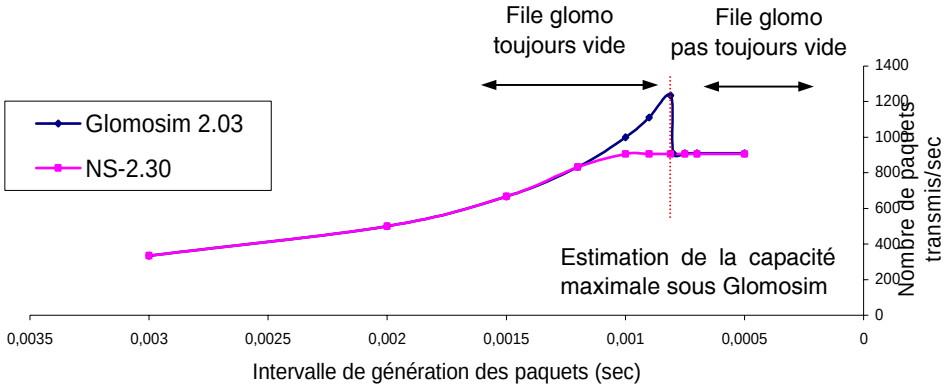
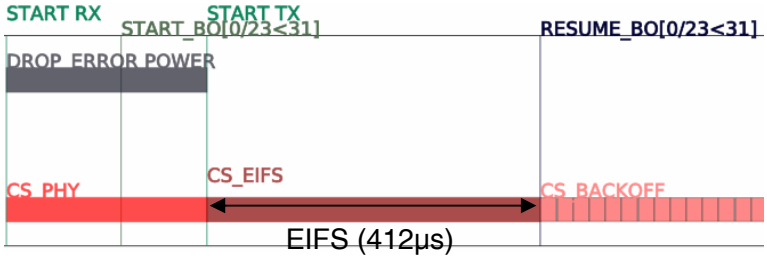


Figure A.6 Comparaison de performances entre NS-2 et Glomosim (MAC PDU =76 octets) dans un réseau de deux noeuds.

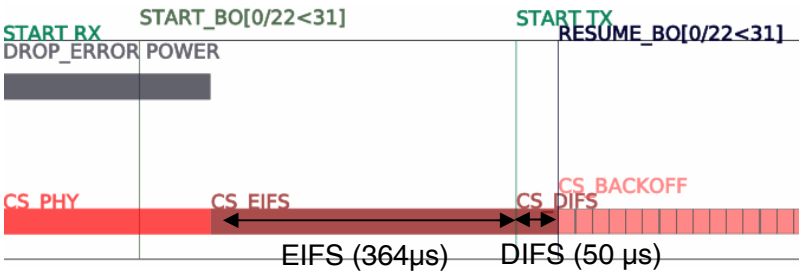
3.2. Durée IFS

Glomosim) Dans la norme 802.11, la durée de l'EIFS vaut 364 µs et celle du DIFS 50 µs. Cependant, sous Glomosim, la durée de l'EIFS (*CS_EIFS*) est égale à 412 µs (cf frise A.8.a) et le DIFS à 45 µs (50 – Δ).



Frise A.8.a Durée EIFS (Glomosim)

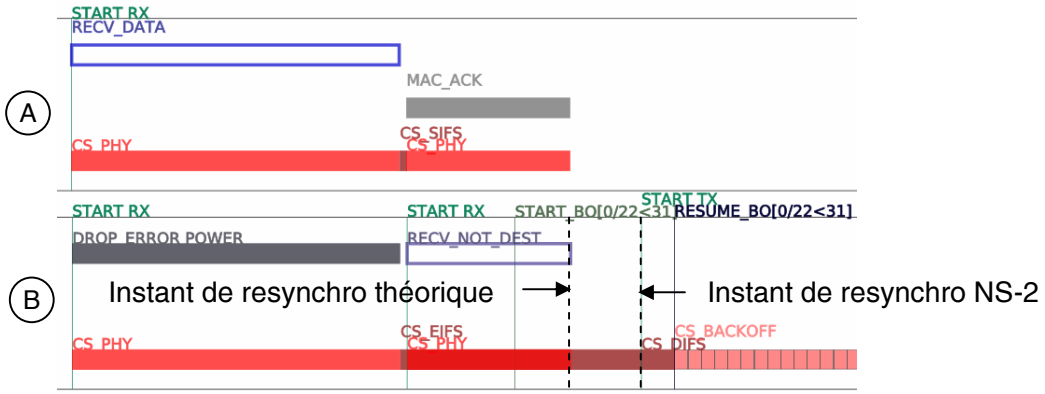
NS-2) Après une réception erronée, les stations sous NS-2 attendent un DIFS en plus de l'EIFS avant de décrémenter à nouveau les slots de backoff. Il résulte que l'EIFS est prolongé de 50 µs en trop sous NS. La frise A.8.b montre le décodage d'un signal erroné (*DROP_ERROR*). Comme nous pouvons le voir, la station attend un DIFS (*CS_DIFS*) en plus de l'EIFS (*CS_EIFS*) suivant la réception de la trame.



Frise A.8.b Durée EIFS (NS-2)

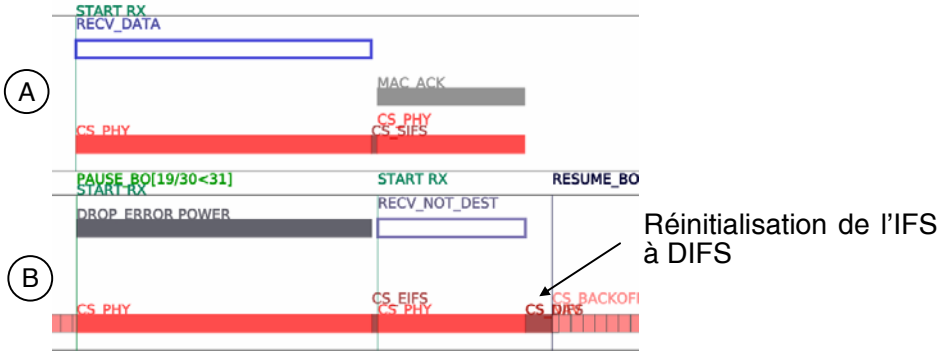
3.3. Resynchronisation EIFS

NS-2) Sous NS, une station qui décode correctement une trame pendant l'intervalle EIFS, ne se resynchronise pas. A titre d'exemple, la frise A.9.a montre une station (la station B) arrivant uniquement à décoder l'acquittement d'un échange DATA/ACK. La station relance normalement sa procédure de backoff à la suite de l'échange. Mais comme nous pouvons l'observer, le début de la procédure n'est pas synchronisé avec la fin de l'acquittement (*MAC_ACK*). En effet, NS-2 oblige la station B à attendre la fin de l'EIFS avant de relancer la procédure de backoff.



Frise A.9.a Resynchronisation EIFS (NS-2)

Glomosim) La frise A.9.b répète la même expérience sous Glomosim. Comme la station B n'arrive pas à décoder la trame de DATA de l'échange DATA/ACK, elle règle initialement son timer IFS avec la valeur EIFS (*CS_EIFS*) à la fin de la réception des DATA. Mais comme l'ACK de l'échange DATA/ACK est correctement reçu, le timer IFS est réinitialisé avec la valeur DIFS et la procédure générale de backoff reprend à la fin de la réception de l'ACK.



Frise A.9.b Resynchronisation EIFS (Glomosim)

3.4. Perturbation des échanges de trames

Glomosim) Sous Glomosim, lorsqu'une station reçoit un paquet d'une station qui ne lui est pas destiné pendant l'un de ses propres échanges, la procédure de collision qui s'en suit peut différer de celle du standard[47]. Deux exemples types mettant en jeu ce comportement sont présentés à la Figure A.7. Dans les deux cas la station A retransmet sa trame sans retirer de backoff.

Dans le cas 1, les stations A et B transmettent simultanément une trame à C. La station B étant beaucoup plus proche de la station que C, le paquet de A est capturé et celui de C est jeté. De sorte A reçoit seulement l'ACK destiné à B.

Dans le cas 2, B transmet à B' et A à A'. Comme A et A' ne sont pas a portée, A reçoit uniquement l'ACK destiné à B.

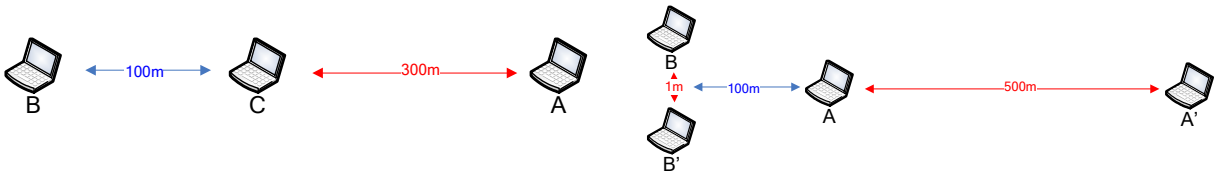
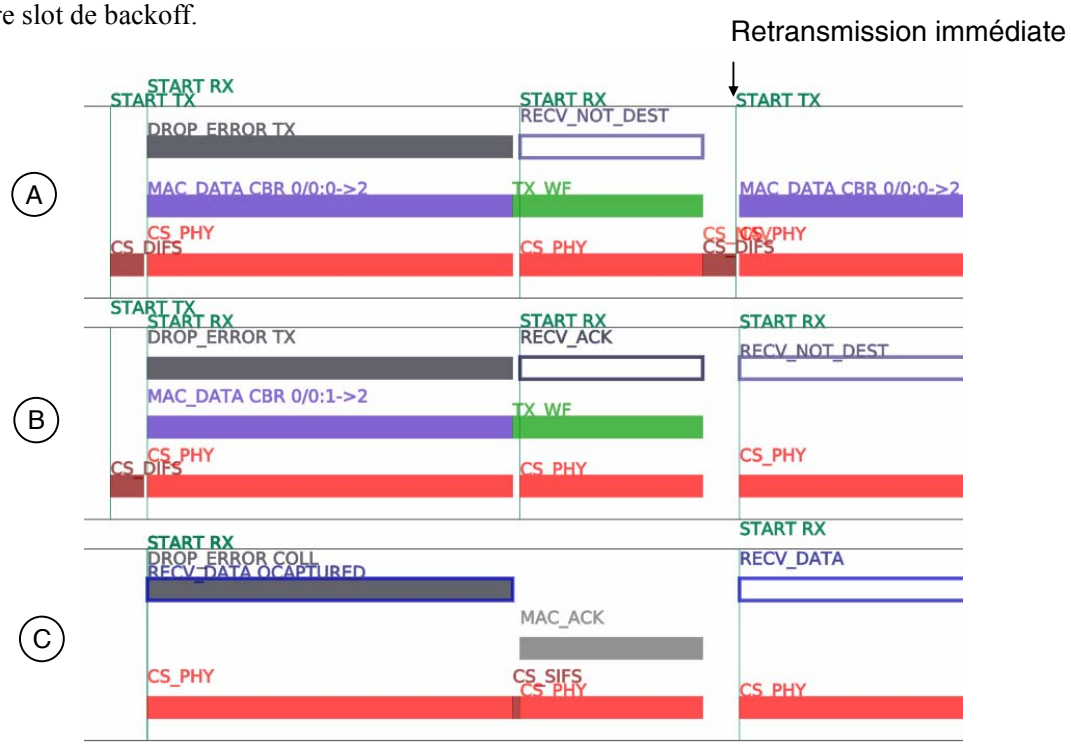


Figure A.7 Scenarios de test des transmissions simultanées (cas 1 à gauche, cas 2 à droite)

La frise A.10.a illustre le cas 1 avec un échange DATA/ACK. Comme nous pouvons le voir, à la réception de l'ACK destiné à la station B, A retransmet immédiatement après un DIFS sa trame sans attendre le moindre slot de backoff.



Frise A.10.a Perturbation des échanges de trames (Glomosim)

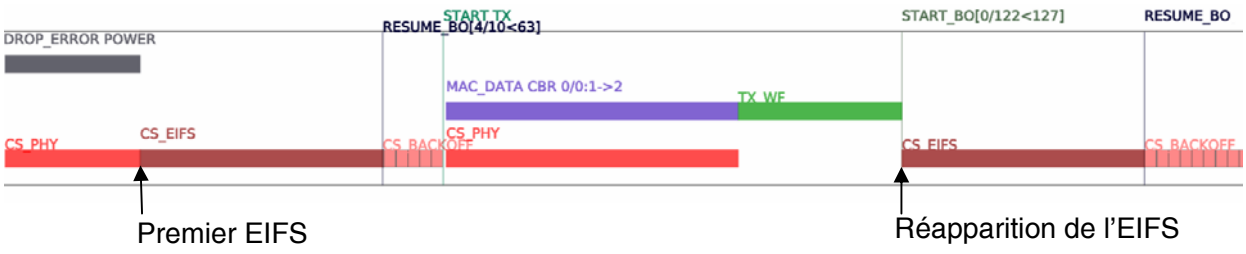
[47] Ce problème concerne autant les échanges RTS/CTS/DATA/ACK et DATA/ACK. Pour que celui-ci survienne à 2Mbps/sec, la longueur des paquets générés par les stations sources ne doit pas différer de plus de 3 octets. Dans le cas contraire, Glomosim se comporte normalement.

NS-2) Conformément à Glomosim, sous NS-2, la station C relance correctement la procédure de backoff comme le montre la frise A.10.b.

Frise A.10.b Perturbation des échanges de trames (Glomosim)

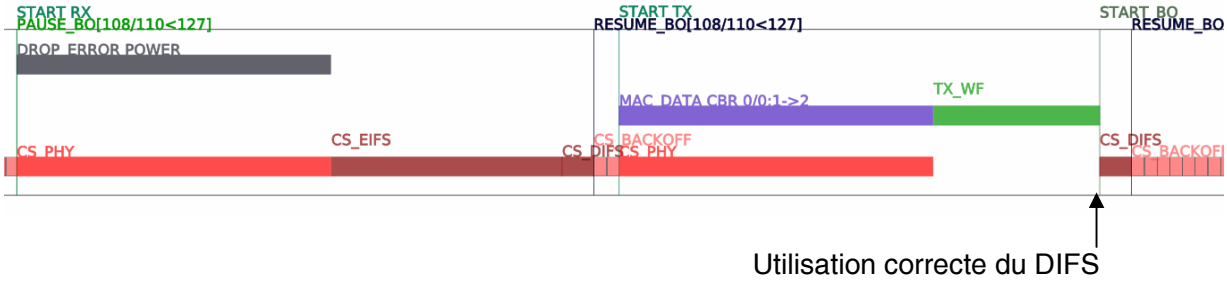
3.5. Remise à zéro EIFS

Glomosim) Selon la norme, l'EIFS est attendu uniquement lorsque l'indication d'occupation de canal immédiatement précédente est causée par la réception d'une trame erronée. Pourtant sous Glomosim, lorsqu'une station reçoit une trame en erreur, elle continue à utiliser l'EIFS à la place du DIFS jusqu'à ce que lui arrive une trame correcte. Ce phénomène est illustré sur la frise A.11.a. Le premier EIFS (*CS_EIFS*) est effectivement conforme au standard car il démarre après la réception d'une trame erronée. Mais cela n'est pas vrai, pour le deuxième EIFS qui est juste une résurgence du premier.



Frise A.11.a Remise à zéro EIFS (Glomosim)

NS-2) La frise A.11.b décrit la même expérience sous NS-2. Comme nous pouvons le voir, NS-2 utilise correctement un DIFS à la fin de la période d'attente de l'ACK (*TX_WF*).



Frise A.11.b Remise à zéro EIFS (NS-2)

Références

[IEE99] ANSI/IEEE Std, 802.11 1999 Edition (R2003), IEEE Standard for Information Technology — Telecommunications and Information Exchange between Systems — Local and Metropolitan Area Network — Specific Requirements — Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications, 1999

[KOC04] A. Kochut, A. Vasan, A.U. Shankar, A. Agrawala, "Sniffing out the correct physical layer capture model in 802.11", dans *Proc. of the 12th IEEE International Conference on Network Protocols (ICNP'04),* pp. 252 – 261, Berlin, Allemagne, Octobre 2004

[TAY01] Y. C. Tay, K. C. Chua, "A capacity analysis for the IEEE 802.11 MAC protocol", Wireless Networks, 7 (2), pp. 159 – 171, Kluwer Academic Publishers Hingham, MA, Etats-Unis, Mars/Avril 2001

Annexe B : Instrumentation de 802.11 dans NS-2.29 et Glomosim 2.03

Cette annexe détaille, dans le cadre du développement de Yavista 1.0:

- les processus, les états et les flots de contrôle identifiés lors de la phase de modélisation de NS-2 [Net-NS-2] et Glomosim [Net-Glomosim]
- l'instrumentation en code neutre (instruction en rouge sur les Figures B.1, B.3 et B.5)

1. NS-2

1.1. Processus de décodage

Comme illustré à la Figure B.1, les paquets circulant sur le médium sont traités par une machine composée de trois états :

- *MAC_IDLE* : indique que la carte n'est pas actuellement en train de décoder un paquet,
- *MAC_RECV* : indique que la carte essaye de décoder un paquet,
- *MAC_COLL* : indique que la carte est en train de décoder un paquet ayant produit une collision.

Sous NS-2.29, un paquet arrivant avec une puissance :

- inférieure au seuil de sensibilité de la carte n'est pas traité par la couche MAC,
- supérieure au seuil de sensibilité active la transition *MAC_IDLE→MAC _RECV*,
- supérieure au seuil de sensibilité mais inférieure au seuil de réception arrive à la couche MAC avec une erreur.

Dans l'état *MAC_RECV*, une collision survient lorsque arrive un paquet avec une puissance plusieurs fois supérieure à la puissance du paquet en cours de décodage. Le paquet à l'origine de la collision devient le nouveau paquet en cours de décodage s'il se termine en dernier.

Si la puissance du second paquet est négligeable par rapport au paquet en cours de décodage, le décodage de l'ancien paquet se poursuit, le nouveau paquet est jeté et la carte reste en *MAC_RECV*. Sous NS-2, ce processus, consistant à garder l'ancien paquet et jeter le nouveau est appelé 'capture'. Dans ce cas le NAV est mis à jour avec la durée du nouveau paquet plus un EIFS pour simuler l'utilisation de l'EIFS de la norme.

Lorsque le paquet en cours de décodage se termine, la machine à états retourne à l'état *MAC_IDLE*. Un paquet (RTS, CTS, ACK ou DATA) est correctement décodé s'il n'a pas été marqué en erreur par une autre réception ou une transmission.

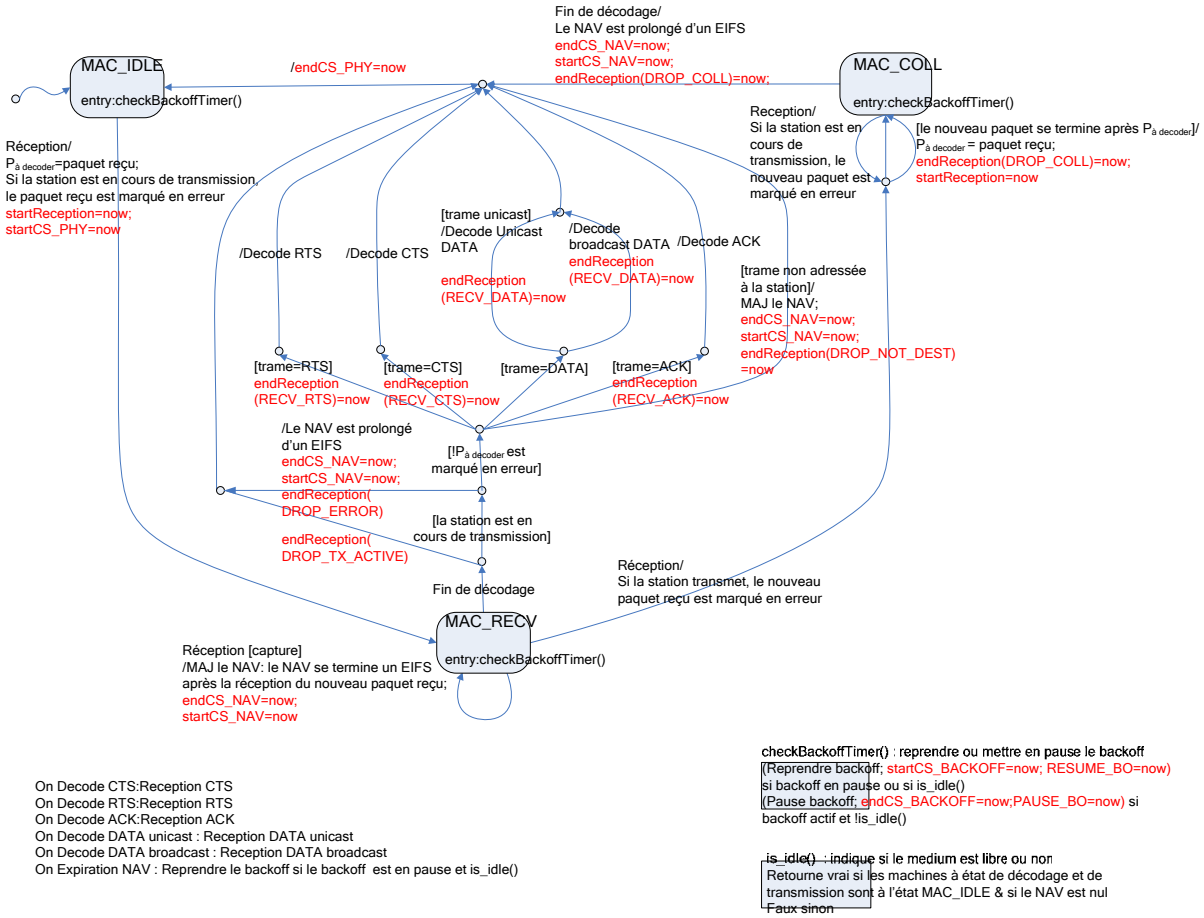


Figure B.1 Processus de décodage sous NS-2

1.2. Processus de transmission

Sous NS-2, la machine gérant les transmissions à la charge de transmettre (voir Figure B.2) :

- les trames de données (via l'état MAC_SEND),
- les acquittements (via l'état MAC_ACK)
- les trames RTS et CTS (via les états MAC_RTS et MAC_CTS).

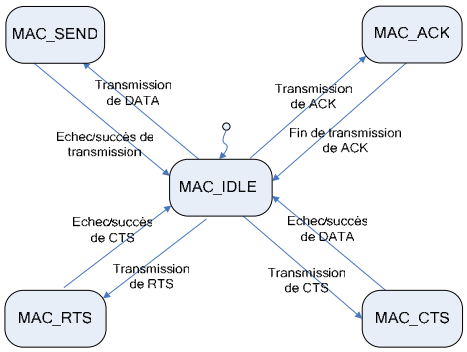


Figure B.2 Processus de transmission simplifié sous NS-2

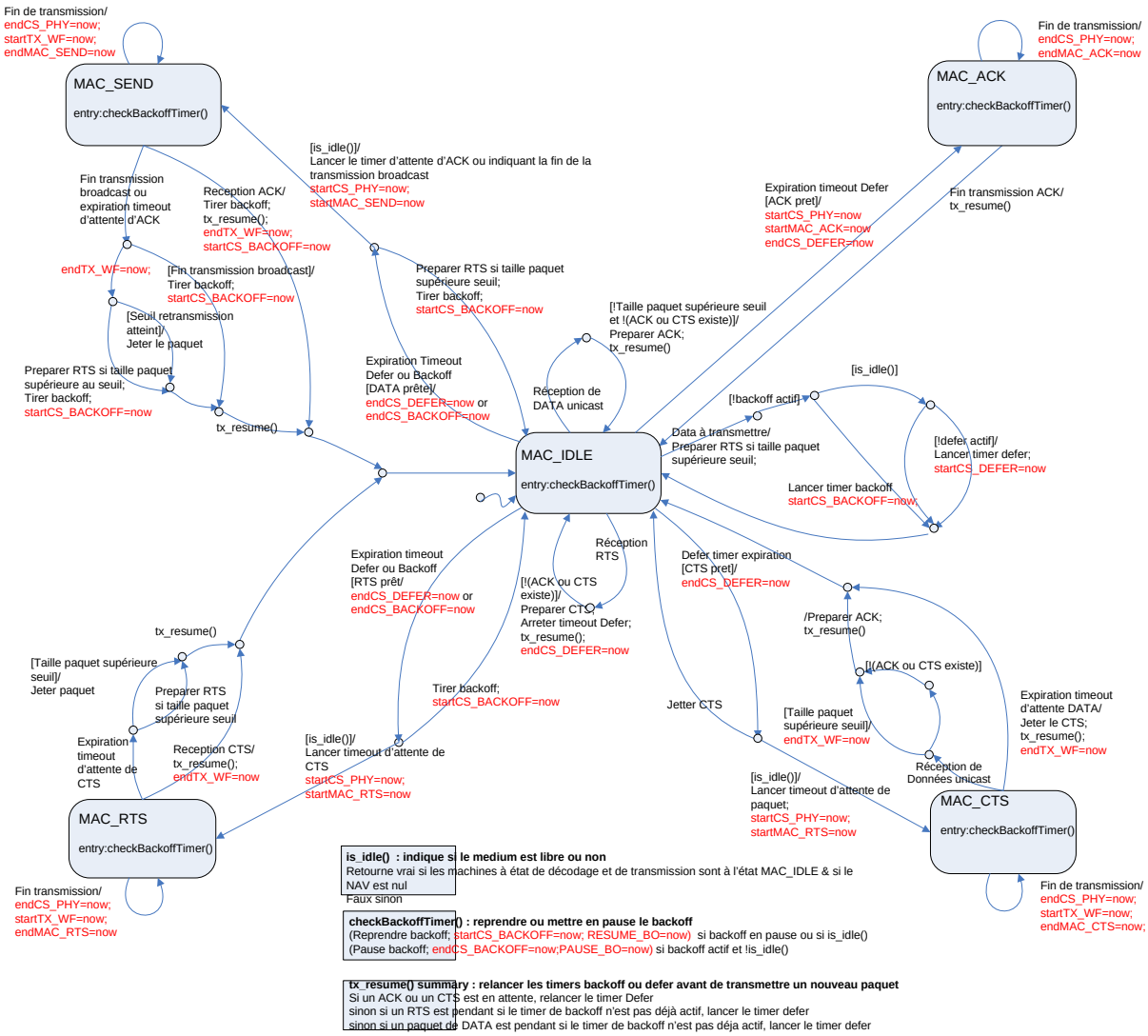


Figure B.3 Processus de transmission sous NS-2

L'état initial est *MAC_IDLE*. La transition *MAC_IDLE→MAC_SEND* est activée quand les timers de backoff ou de defer expirent indiquant la possibilité de transmettre une trame sur le medium. Ces timers sont utilisés pour modéliser la durée des intertrames et de la procédure de backoff. La machine à états retourne à l'état *MAC_IDLE* à la réception d'un ACK ou à l'expiration du timeout d'ACK qui indique l'échec d'une transmission. De façon similaire, les états *MAC_ACK*, *MAC_RTS* et *MAC_CTS* représentent respectivement la transmission d'un ACK, d'un CTS ou d'une trame RTS.

2. Glomosim

Sous NS, l'état courant de la machine en transmission dépend du paquet à transmettre. Sous Glomosim, il n'existe qu'une seule machine à états et celle ci reproduit le cycle de vie de la couche MAC lors d'une transmission : attente de transmission, transmission, attente de confirmation, retour à l'état libre. Les Figures B.4 et B.5 offrent des vues plus ou moins détaillées de ce processus.

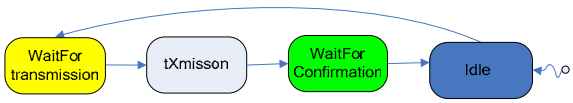


Figure B.4 Modèle simplifié du protocole DCF sous Glomosim

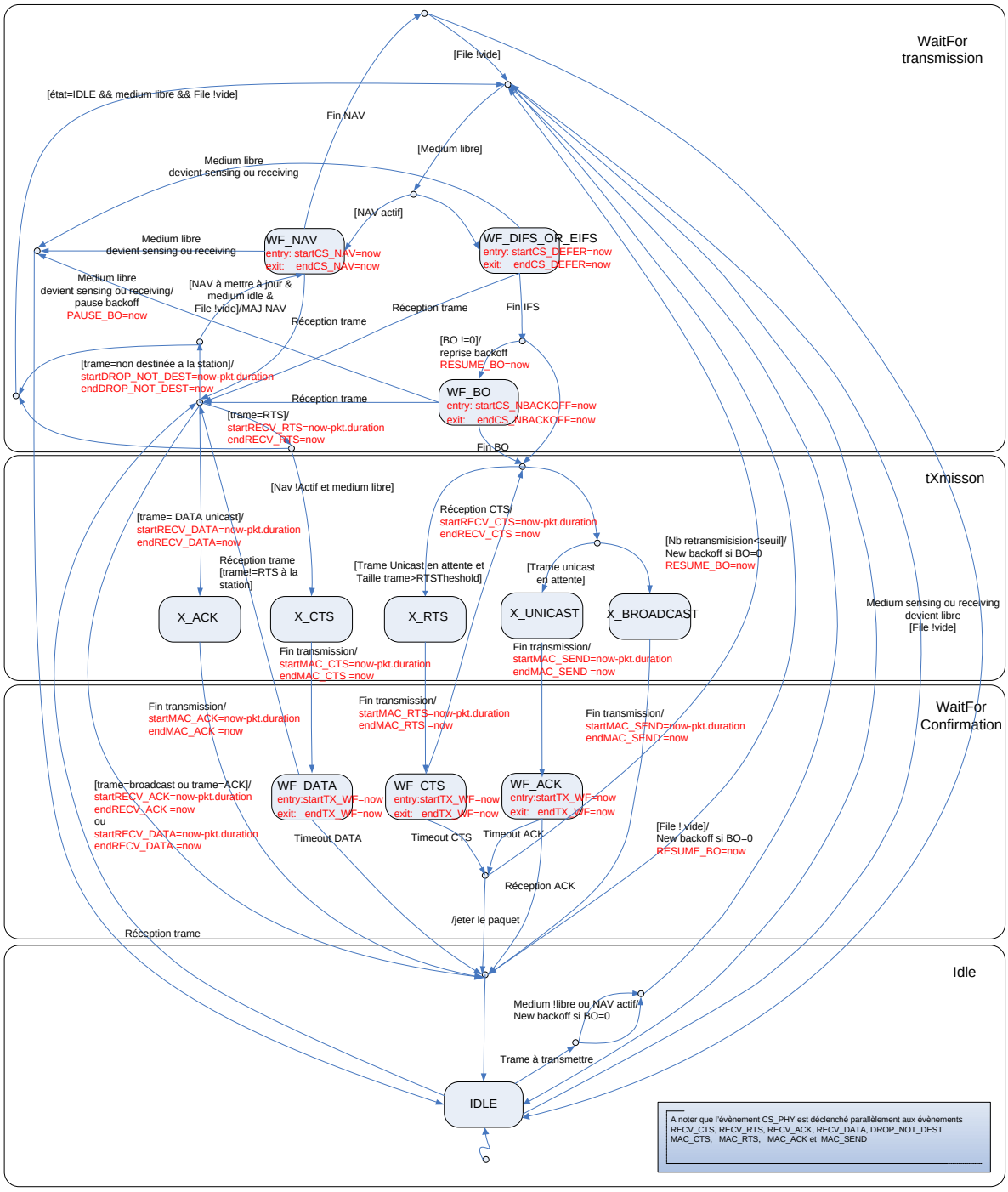


Figure B.5 Modèle du protocole DCF sous Glomosim

Netographie

[Net-NS-2] NS-2.29, Disponible: http://www.isi.edu/nsnam
Dernière visite le 31/08/06

[Net-Glomosim] Glomosim 2.03, Disponible: http://pcl.cs.ucla.edu/projects/glomosim/
Dernière visite le 31/08/06

Annexe C : Contrats de CoW

Les contrats des éléments cœurs de l'architecture de CoW sont présentés à la Figure C.1.

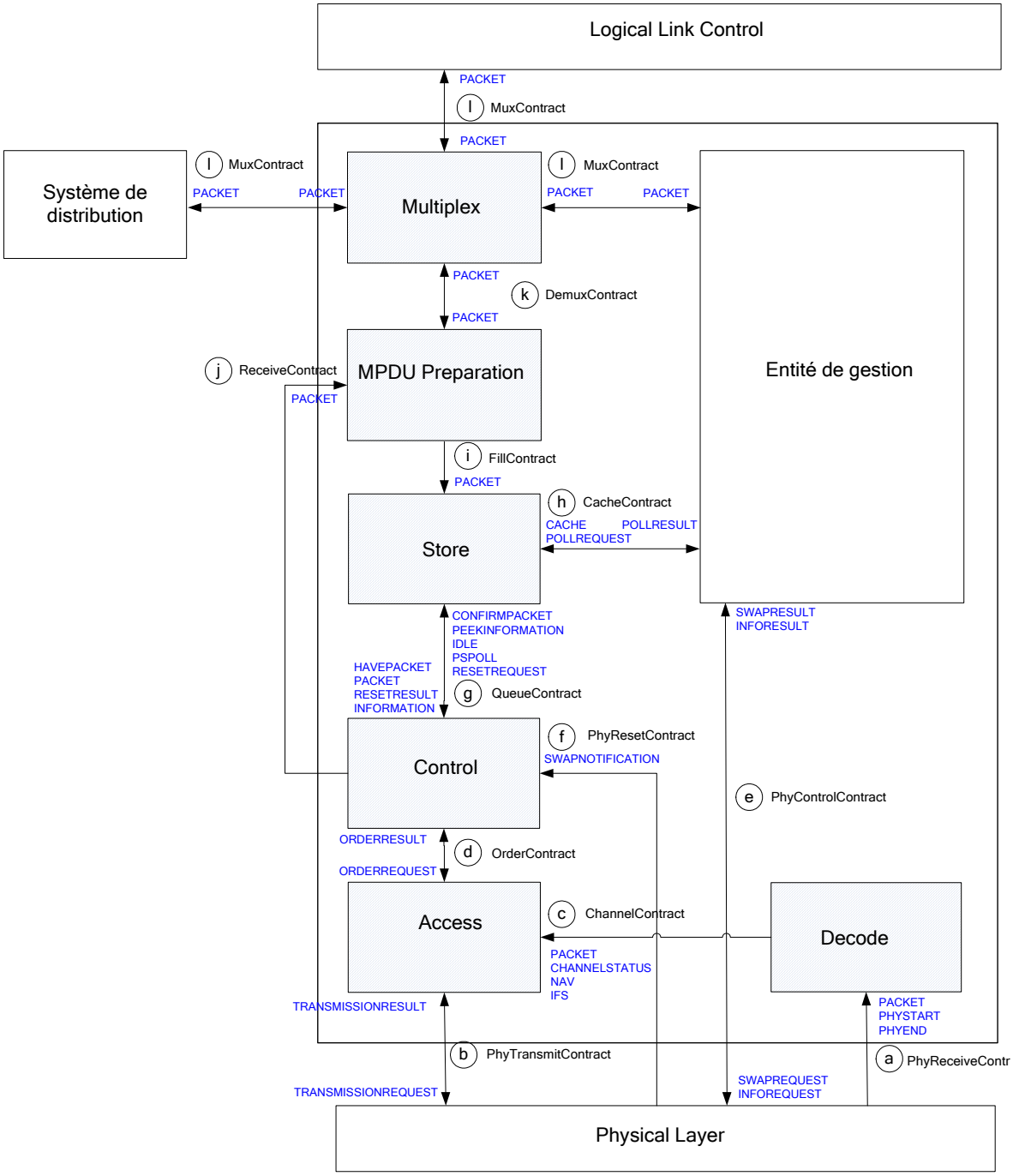


Figure C.1 Contrats définis dans CoW

a) Le contrat *PhyReceiveContract* indique à *Decode* quand un paquet commence à être assemblé (*PHYSTART*), quand l'assemblage se termine (*PHYEND*) et, à la fin de l'assemblage, le paquet reçu (*PACKET*)
b) Le contrat *PhyTransmitContract* permet l'envoi de paquets à la couche physique (*TRANSMISSIONREQUEST*). La fin de transmission est notifiée à *Access* par le message *TRANSMISSIONRESULT*.
c) Le contrat *ChannelContract* indique les périodes ou le medium est occupé physiquement (*CHANNELSTATUS*), les périodes ou le canal est occupé virtuellement (*NAV*), les paquets reçus (*PACKET*) et l'interframe (*IFS*) à utiliser selon que le paquet reçu est correctement décodé ou non.
d) Le contrat *OrderContract* permet, dans le sens *Control→Access*, de configurer le bloc *Access* pour réaliser les ordres de *Control*. Dans le sens inverse, il permet de notifier le résultat des ordres ou des évènements non attendus dans *Control* (exemple : réception d'un paquet de DATA d'une séquence DATA/ACK ou un RTS d'une séquence RTS/CTS/DATA/ACK). Dans les deux sens, les messages charrient des structures *ORDER*.
e) Le contrat *PhyControlContract* permet à un bloc optionnel de gestion de monitorer la couche PHY (*INFOREQUEST*) ou d'ordonner un changement de canal (*SWAPREQUEST*). Les valeurs monitorées et les résultats du swap remontent dans des messages *INFORESULT* et *SWAPRESULT*.
f) Le contrat *PhyResetContract* notifie au composant *Control* les demandes de changement de canal (cf contrat *PhyControlContract*).
g) Le contrat *QueueContract* transporte, dans le sens montant, les messages *IDLE* indiquant que *Control* est prêt à recevoir un nouveau paquet, *PEEKINFORMATION* indiquant une demande de profil de paquet et *RESETREQUEST* indiquant un ordre de mise à zéro des files. *PSPOLL* indique à *STORE* de mettre en cache tous les paquets des stations ayant décidées de quitter la cellule.
   Dans le sens descendant, le contrat *QueueContract* transporte les messages *HAVEPACKET* notifiant la présence d'un paquet à transmettre, *PACKET* contenant un paquet à transmettre, *RESETRESULT* indiquant le résultat d'une demande de mise à zéro des files et *INFORMATION* contenant le profil du prochain paquet à transmettre.
h) Le contrat *CacheContract* permet à un bloc optionnel de gestion d'ordonner le forwarding des paquets d'une station maintenant connectée à une autre cellule en les réinjectant dans le système de distribution optionnellement présent (*CACHE*). Il permet également de générer une requête d'envoi d'un paquet *PSPOLL* (*POLLREQUEST*) et de renvoyer à l'entité de gestion le résultat de cette requête (*POLLRESULT*).
i) Le contrat *FillContract* transporte les MAC PDU à transmettre sur le réseau (*PACKET*).
j) Le contrat *ReceiveContract* transporte les MPDU reçus (*PACKET*).
k) Le contrat *MuxContract* permet l'orientation des données entre l'entité optionnelle de gestion, le système de distribution éventuellement présent, les couches inférieures et les couches supérieures.
l) Le contrat *DemuxContract* agit comme *MuxContract*.

Enfin, le contrat Xport commun à tous les composants permet la mise à jour des variables partagées entre les différentes entités.

Liste des figures

Figure i.1 Impact des besoins fonctionnels et non fonctionnels sur l'architecture des simulateurs 14
Figure 1.1 Architecture des simulateurs 18
Figure 1.2 Architecture de standardisation des simulateurs 19
Figure 1.3 Méthode de comparaison des simulateurs traditionnelle (à gauche) et YAVISTA (à droite) 20
Figure 1.4 Approche de comparaison sous YAVISTA 20
Figure 1.5 Rôle des générateurs de code neutre 23
Figure 1.6 Implémentation de l'interface de sortie standard 24
Figure 1.7 Méthode d'implémentation d'un module de génération de code neutre 24
Figure 1.8 Architecture de YAVISTA 25
Figure 1.9 Frise temporelle générée avec YAVISTA sequenceur 26
Figure 1.10 Visualisation des éléments du réseau avec YAVISTA grapheur 27
Figure 1.11 Comparaison de courbes avec YAVISTA grapheur 27
Figure 1.12 Description de l'automate validant la séquence DATA/ACK côté émetteur (émission de Data) 28
Figure 1.13 Description de l'automate validant la séquence DATA/ACK côté récepteur (émission de ACK) 29
Figure 1.14 Implémentation de la séquence DATA/ACK en émission (haut niveau) 30
Figure 1.15 Implémentation de bas niveau de l'état d'attente de transmission (fichier TX_WF_X_implem.xml) 30
Figure 1.16 Caractéristiques de NS-2.30 et Glomosim 2.03 32
Figure 1.17 Temps de propagation sur le réseau 33
Figure 1.18 Décrémentation du backoff dans une chaîne de trois noeuds sous NS-2 34
Figure 1.19 Illustration du cas de capture/collision sous Glomosim 35
Figure 1.20 Décrémentation du backoff sous Glomosim 35
Figure 1.21 Comparaison de la probabilité de collisions des paquets sous NS-2, Glomosim et la théorie. Neutralisation du modèle de capture, dans une zone de $1m^2$ à gauche & modèle de capture actif, dans une zone de $1m^2$ à droite 36
Figure 2.1 Module, produit et ligne de produits 42
Figure 2.2 Principe de modularité du point de vue du module, du produit et de la ligne de produits 42
Figure 2.3 Ligne de produits à trois points de variété : couche LLC, file d'attente, sous couche MAC 43
Figure 2.4 Exemple d'architecture centralisée 44
Figure 2.5 Exemple d'architecture avec programme principal et sous routines 45
Figure 2.6 Exemple d'architecture en couches 46
Figure 2.7 Exemple d'architecture orientée objet 46
Figure 2.8 Exemple d'architecture par flot de donnée 47
Figure 2.9 Mesures de la morphologie. A gauche le graphe de dépendances du programme, à droite les métriques de morphologie. 51
Figure 2.10 Complexité cyclomatique d'un graphe de flot. A gauche le programme original, au centre le graphe de flot, à droite la mesure de complexité 52
Figure 2.11 Exemple montrant les caractéristiques pertinentes d'une automobile 54
Figure 2.12 Utilisation des méta-classes 58
Figure 2.13 Utilisation des méta-objets 59
Figure 2.14 Représentation du méta niveau d'une architecture logiciel à l'aide d'un graphe 59
Figure 2.15 Architecture logique d'un système adaptable 60

Figure 2.16 Exemple de calcul d'index d'adaptabilité de l'architecture AAI 61
Figure 3.1 Diagramme de structure représentant la couche liaison dans le modèle TCP/IP augmenté d'un mécanisme de communication inter couche 70
Figure 3.2 Diagramme de contexte représentant les principaux flux d'informations avec les autres entités du système 70
Figure 3.3 Caractéristiques définissant le fonctionnement d'une carte réseau 70
Figure 3.4 Caractéristiques définissant la méthode d'accès au canal 71
Figure 3.5 Caractéristiques définissant la gestion des files d'attente 71
Figure 3.6 Caractéristiques définissant les mécanismes de séquencement et de détection et de correction d'erreurs 72
Figure 3.7 Caractéristiques définissant la compression, la fragmentation et le chiffrement 73
Figure 3.8 Caractéristiques définissant les fonctions de contrôle des ressources 73
Figure 3.9 Caractéristiques définissant la gestion de l'adaptation 74
Figure 3.10 Caractéristiques définissant le mapping des flux réseaux sur les flux physiques 75
Figure 3.11 Caractéristiques définissant le séquencement et les fonctions de détection de porteuse 76
Figure 3.12 Diagramme en couches de NS 76
Figure 3.13 Caractéristiques disponibles dans NS 77
Figure 3.14 Diagramme en couches de Glomosim 80
Figure 3.15 Caractéristiques disponibles dans Glomosim 81
Figure 3.16 Diagramme en couches de J-Sim 84
Figure 3.17 Caractéristiques disponibles dans J-Sim 85
Figure 3.18 Architecture des stations ad-hoc des NQAPs et NQSTAs (stations non QoS) 88
Figure 3.19 Architecture à qualité de service 89
Figure 3.20 Caractéristiques disponibles sous YANS 90
Figure 3.21 Hiérarchie de la ligne de produit 90
Figure 3.22 a. et b. Comparaison des mesures de généralité de Glomosim, J-Sim, NS-2 et YANS 95
Figure 3.23 a. et b. Taille en arcs et nœuds et impureté moyennes de Glomosim, J-Sim, NS-2 et YANS 96
Figure 3.24 a. et b. Mesure du nombre de lignes de code, de complexité, de couplage entre objets et de flots d'information moyens de Glomosim, J-Sim, NS-2 et YANS 97
Figure 4.1 Modèle de caractéristiques FODA 105
Figure 4.2 Implantation des caractéristiques dans CoW 107
Figure 4.3 Vue en processus des modules *Control* et *Access* dans CoW 108
Figure 4.4 Implémentation de la séquence DATA/ACK en émission sous CoW 109
Figure 4.5 Implémentation de la séquence DATA/ACK en réception sous CoW 109
Figure 4.6 Implémentation du protocole 802.11 sous Yavista 110
Figure 4.7 Mapping entre les processus de COW et les machines à états de la norme 111
Figure 4.8 Implémentation de la machine virtuelle 802.11 de CoW 113
Figure 4.9 Mapping entre les processus de CoW et la conception TSMA de Glomosim 114
Figure 4.10 Adaptation des composants sous CoW 116
Figure 4.11 Caractéristiques disponibles dans CoW 118
Figure 4.12 Hiérarchie de la ligne de produits CoW 119
Figure 4.13 a. et b. Comparaison des mesures de généralité de CoW et YANS 122
Figure 4.14 a. et b. Taille en arcs et nœuds et impureté moyennes de CoW et YANS 123
Figure 4.15 a. et b. Mesures du nombre de lignes de code, de complexité, de couplage entre objets et de flots d'information moyens de CoW et YANS 123
Figure 5.1 Mapping entres les processus et le modèle de caractéristiques Yavista 129
Figure 5.2 Architecture du simulateur Yavista pour plusieurs types d'accès au canal 129
Figure 5.3 Parallèle entre stations Yavista et ELYSEE. Validation du processus RX de *Control* 131

Figure A.1 Temps de propagation sur le réseau 143
Figure A.2 Illustration du cas de capture/collision sous Glomosim 145
Figure A.3 Comparaison de la probabilité de collisions des paquets sous NS-2, Glomosim et la théorie (neutralisation du modèle de capture, dans une zone de $1m^2$) 146
Figure A.4 Comparaison de la probabilité de collisions des paquets sous NS-2, Glomosim et la théorie (modèle de capture actif, dans une zone de $1m^2$) 146
Figure A.5 Comportement de NS-2 et Glomosim en cas de capture en fonction du ratio de puissance des paquets reçus 147
Figure A.6 Comparaison de performances entre NS-2 et Glomosim (MAC PDU =76 octets) dans un réseau de deux noeuds. 150
Figure A.7 Scenarios de test des transmissions simultanées (cas 1 à gauche, cas 2 à droite) 152
Figure B.1 Processus de décodage sous NS-2 156
Figure B.2 Processus de transmission simplifié sous NS-2 156
Figure B.3 Processus de transmission sous NS-2 157
Figure B.4 Modèle simplifié du protocole DCF sous Glomosim 158
Figure B.5 Modèle du protocole DCF sous Glomosim 158
Figure C.1 Contrats définis dans CoW 161

Liste des tableaux

Table 1.1 Description des éléments XML 21
Table 1.2 Description des attributs XML 22
Table 1.3 Format des éléments XML 22
Table 2.1 Modèle d'assurance qualité logicielle de McCall 40
Table 2.2 Catégories de cohésion 48
Table 2.3 Catégories de couplage 48
Table 2.4 Techniques de fixation de la variabilité dans les lignes de produits 55
Table 3.1 Dépendances inter caractéristiques obligatoires et exclusions sous NS 78
Table 3.2 Présentation du mapping caractéristiques/composants et des caractéristiques adaptables des produits Mac simple, SMAC, TDMA et 802.11 79
Table 3.3 Calculs des EAI, AAI et SAI sous NS 79
Table 3.4 Mesures de morphologie et d'impureté de NS 79
Table 3.5 Mesures de flots d'information, de couplage et de complexité de NS 80
Table 3.6 Dépendances inter caractéristiques obligatoires et exclusions sous Glomosim 82
Table 3.7 Présentation du mapping caractéristiques/composants et des caractéristiques adaptables des produits MACA, CSMA, 802.11 et TSMA 82
Table 3.8 Calculs des EAI, AAI et SAI sous Glomosim 83
Table 3.9 Mesures de morphologie et d'impureté de Glomosim 83
Table 3.10 Mesures de complexité de Glomosim 83
Table 3.11 Dépendances inter caractéristiques obligatoires et exclusions sous J-Sim 85
Table 3.12 Présentation du mapping caractéristiques/composants et des caractéristiques adaptables du produit 802.11 sous J-Sim 86
Table 3.13 Calculs des EAI, AAI et SAI sous J-Sim 86
Table 3.14 Mesures de modularité de morphologie et d'impureté de J-Sim 86
Table 3.15 Mesures de flots d'information, de couplage et de complexité de J-Sim 87
Table 3.16 Corrélation entre les composants Mac High et les autres éléments de l'architecture 90
Table 3.17 Dépendances inter caractéristiques obligatoires et exclusions sous YANS 91
Table 3.18 Présentation du mapping caractéristiques/composants et des caractéristiques adaptables sous YANS 92
Table 3.19 Calculs des EAI, AAI et SAI sous YANS 92
Table 3.20 Mesures de morphologie et d'impureté de YANS 93
Table 3.21 Mesures de flots d'information, de couplage et de complexité des stations ad hoc 93
Table 3.22 Mesures de flots d'information, de couplage et de complexité des NQSTA 94
Table 3.23 Mesures de flots d'information, de couplage et de complexité des NQAP 94
Table 3.24 Mesures de flots d'information, de couplage et de complexité des QSTA 94
Table 3.25 Mesures de flots d'information, de couplage et de complexité des QAP 95

Table 4.1 Caractéristiques variables de CoW 111
Table 4.2 Mapping entres les sous caractéristiques de 802.11 et les machines à états de la norme 112
Table 4.3 Mapping entres les sous caractéristiques de TSMA et Glomosim 114
Table 4.4 Dépendances inter caractéristiques obligatoires et exclusions sous CoW 119
Table 4.5 Présentation du mapping caractéristiques/composants et des caractéristiques adaptables sous CoW 120
Table 4.6 Calculs des EAI, AAI et SAI sous CoW 120
Table 4.7 Mesures de morphologie et d'impureté de CoW 121
Table 4.8 Mesures de flots d'information, de couplage et de complexité des stations TSMA 121
Table 4.9 Mesures de flots d'information, de couplage et de complexité des stations 802.11 ad-hoc 121
Table 4.10 Mesures de flots d'information, de couplage et de complexité des stations 802.11 infrastructure 122
Table A.1 Interprétations du standard IEEE 802.11 139
Table A.2 Simplifications du système 141
Table A.3 Erreurs de modélisation 149