Vers une approche de spécification formelle des protocoles à l'aide de Troll*

Mourad Maouche — Mohamed Bettaz

Institut d'Informatique
Université de Constantine
25000 Algérie

RÉSUMÉ. Cet article présente une approche de spécification formelle de systèmes de communication basée sur Troll*, extension du langage formel objet Troll. Fondée sur une sémantique saine et rigoureuse, cette approche adopte un style de description orienté ressources, style motivé par un souci de maîtrise de la complexité et d'amélioration du degré de réutilisation ; ce qui la distingue des techniques de description formelle des protocoles telles que Lotos et Estelle. Troll* est en fait le résultat de l'intégration du formalisme des ECATNets au langage Troll, intégration motivée par un souci de prise en charge correcte du comportement parallèle inhérent aux systèmes de communication. Une étude de cas simple du monde des protocoles illustre notre intérêt pour le langage Troll.

ABSTRACT. This paper presents a Troll* based approach of communication systems. Troll* is an extension of the formal object oriented language Troll. It is based on a sound and rigorous semantics and uses an augmented oriented resources description. This distinguishes it from other formal protocol description techniques such as Lotos and Estelle. In fact, Troll* is a result of the ECATNet formalism integration with the Troll language. This integration is motivated by a correct handling of the parallel behaviour of the communication systems. A simple case study related to communication protocols illustrates our interest in Troll.

MOTS-CLÉS : protocoles de communication, spécification formelle, approche objet, vrai parallélisme, réseaux algébriques.

1. Introduction

Les techniques de description formelle sont largement adoptées dans le domaine des systèmes de communication. Leur succès est dû essentiellement au
fait qu’ils rendent possible la vérification de la correction des systèmes décrits.
L’aptitude à exprimer naturellement et correctement le parallélisme inhérent à
ces systèmes, à maîtriser leur complexité, sans cesse croissante, et la facilité
d’utilisation sont, à notre sens, des paramètres pertinents susceptibles de re-
hausser la qualité et l’efficacité d’une technique de description formelle.
Les langages de spécification formelle utilisés par la communauté des proto-
coles de communication sont, en général, conçus sur la base d’une combinaison
de formalismes complémentaires tels que les automates à états finis, l’algèbre des
processus, les réseaux de Petri, les types abstraits algébriques et la logique tem-
poirelle. L’objectif visé est, bien sûr, de bénéficier des avantages de chacun de ces
formalismes. Cependant ces combinaisons ad-hoc de formalismes hétérogènes,
conduisent le plus souvent à des langages démunis d’une sémantique "sauve" et
propre. Ce qui constitue un obstacle sérieux à une démarche de vérification de
propriétés rigoureuse et fondée. C’est le cas, par exemple, aussi bien de Lotos
qui est une une combinaison ad-hoc [GOG 93] de ces [MIL 80] et d’Acte One
[EHR 85] que de celui d’Estelle qui est une combinaison ad-hoc d’automates à
états finis et d’éléments du langage Pascal.

Le recours à des langages de description formelle dotés d’une sémantique
"sauve" [GOG 93] fondée sur des théories mathématiques solides semble être
une voie d’investigation nécessaire.

C’est dans ce sens que nous nous sommes proposés de nous investir dans le
langage de description formelle Troll [EHR 93a], [SAA 93], [JUN 91], et cela
pour trois raisons essentielles:

- Troll part de concepts faisant l’unanimité [GOG 91], [MES 92], [BOC
90]. Il réunit, autour du paradigme objet, des idées issues des types abstraits
algébriques, des systèmes réactifs et concurrents, des modèles sémantiques de
données et des bases de données [SAA 93]. Ceci ne distingue pas Troll d’autres
langages de la même catégorie tels que Foops [GOG 91], Maude [MES 92],
Mondel [BOC 90].

- Troll est pourvu d’une sémantique "sauve" fondée sur une approche logique
[EHR 93b]. La dynamique des objets d’un système est décrite par le biais d’une
logique appropriée. Un modèle sémantique, basé sur la théorie des catégories,
permet de donner aux concepts d’héritage, d’agrégation et d’interaction inter-
objets une interprétation en termes de morphismes [EHR 93a].

Ceci ne distingue pas Troll de l’approche Foops. Cependant notre choix
s’est porté sur Troll du fait qu’il suggère des indications sur l’implémentation.
En effet Troll définit les morphismes d’héritage et d’interaction à deux niveaux
d’abstraction différents, alors que Foops "confond" ces deux types de concepts.

- Troll est "ouvert" en ce sens qu’il autorise la description du comporte-
dement dynamique des objets du système par divers formalismes (processus,
réseau de Petri...) tout en garantissant la cohésion de la cohabitation entre
les différents formalismes qui le constituent. Ce n’est pas le cas de Foops qui décrit le comportement dynamique à l’aide d’équations récursives moyennant un “aménagement” au niveau de l’Institution à sortes cachées [GOG 92]. Cette institution est “fermée” en ce sens qu’elle ne se prête pas à l’intégration de formalismes réputés appropriés.

Nous proposons, dans cet article, d’utiliser les ECATNets [BET 92], [BET 93a], [BET 96] pour décrire le comportement des objets d’un système spécifié en Troll. L’intérêt majeur de ce formalisme réside dans sa capacité à exprimer le comportement parallèle d’un objet selon la sémantique du vrai parallélisme, et non selon la sémantique de l’entrelacement, comme c’est le cas pour ccs. De plus ce formalisme, basé sur une combinaison saine des réseaux de Petri de haut niveau et des types abstraits algébriques, est à notre sens plus accessible aux non initiés que ne l’est ccs.

L’objectif du présent article est double :
- situer, sans entrer dans les détails, l’intégration des ECATNets dans Troll.
- développer sur un exemple détaillé une approche de spécification des protocoles de communication selon le style orienté ressources [TUR 93], [GOG 92]. À notre connaissance, aucune expérience relative à la spécification des protocoles à l’aide de Troll n’a été menée.

La section 2 présente les idées essentielles relatives à l’intégration des ECATNets dans Troll. La section 3 développe une esquisse de l’approche de spécification de protocoles à l’aide de Troll que nous préconisons. Nous avons délibérément choisis, pour des raisons d’espace, d’illustrer cette approche par un exemple simple et commu de tous, le protocole du bit alterné. Une conclusion de l’article est présentée dans la section 4.

2. Troll, comportement dynamique et ECATNets

Avant d’aborder les idées essentielles relatives à l’intégration du formalisme ECATNet dans Troll, il est utile de faire une présentation générale du langage troll.

2.1. Présentation générale de Troll

Troll est un langage de spécification formelle orienté objet qui est fondé sur une approche logique. La spécification d’un système en Troll s’interprète en termes de formules logiques exprimées selon une logique appropriée [EHF 93b]. Un système est décrit en Troll en termes d’une communauté d’objets autonomes en interaction. L’interaction est un concept qui décrit un mécanisme abstrait de communication entre objets.

Un objet, en Troll, est en fait un processus doté d’un état interne et d’une identité. Son état interne est observable par le biais d’attributs. Cet état évolue suite à l’occurrence d’événements générés d’autres objets ou de manière
interne par l'objet lui même. L'événement n'est en fait qu'une abstraction de la "método" du paradigme objet.

Les valeurs des attributs et ceux des paramètres associés aux événements sont décrits par le biais de types abstraits algébriques. L'effet d'un événement sur les attributs d'un objet s'exprime de manière déclarative par le biais de termes algébriques. Cet effet, pour un événement donné, peut dépendre non seulement de l'état de l'objet mais aussi de son histoire. Troll permet de décrire ce type de situations.

Des contraintes d'intégrité sur les valeurs d'attributs peuvent être spécifiées à l'aide d'opérateurs de la logique temporelle et modale.

Troll offre, par ailleurs, des mécanismes d'abstraction et de structuration de haut niveau permettant de construire :

- des collections d'objets grâce au concept de classe. Ce concept est à distinguer de celui de "template" qui représente en fait le "moule" à partir duquel il est possible d'instancier des objets.
- des objets composés grâce au mécanisme d'agrégation et d'interaction. Plusieurs variantes d'agrégation sont possibles en Troll: agrégation statique ou dynamique; agrégation autorisant ou non le partage de composants entre objets composés.

L'interaction permet, dans le cas de l'agrégation, de décrire les communications potentielles entre l'objet composé et ses composants, ou bien entre les composants eux mêmes.
- des objets par spécialisation ou généralisation d'objets déjà existants grâce au mécanisme d'héritage.

Enfin, Troll permet aussi la possibilité d'inclure, dans le "template", la description du comportement dynamique d'un objet instance. Cette description peut comporter deux aspects:
- Le premier sert à exprimer les contraintes liées à l'occurrence des événements qui concernent l'objet en question, ceci grâce aux concepts de la logique modale (permission, obligation, "commitment") et ceux de la logique temporelle.
- Le second sert à décrire les cycles d'évolution possibles de l'objet en termes de processus. Un formalisme proche de ccs et de csp est utilisé pour décrire ces cycles d'évolution.

2.2. Intégration des ECATNets dans Troll

Le parallélisme intra-objet et inter-objet, au sens vrai parallélisme, est "ignore" dans Troll. Ces lacunes de Troll motivent notre présente démarche. L'idée de base qui a guidé notre démarche consiste :
- d'une part, à ne pas retenir le formalisme ccs (ou csp) pour décrire le comportement dynamique d'un objet mais plutôt opter pour un formalisme apte à exprimer le vrai parallélisme et muni bien sûr d'une sémantique "sauve". Cette perspective est rendue possible grâce à la propriété "d'ouverture" de Troll [EHR 93a] que nous avons évoqué en introduction.
et d’autre part, à uniformiser l’expression du parallélisme intra et inter-objets par le biais d’une même sémantique, à savoir la logique de réécriture. Cette logique de la concurrence, réputée saine et complète, permet de dénoter correctement des systèmes concurrents, et de mener un raisonnement sur ces systèmes. Les preuves de propriétés sont alors établies par déduction logique [ES 95].

2.1. Parallélisme intra-objet

Nous suggérons d’adopter le formalisme ECATNet, combinaison des réseaux de Petri de haut niveau et des types abstraits algébriques, pour décrire le comportement dynamique d’un objet. Les ECATNets sont, en fait, une catégorie de réseaux algébriques qui se distinguent des autres catégories par sa sémantique optimisée en termes de logique de réécriture [BET 92], [MES 95]. Leur attraction graphique et leur sémantique "sauve" motivent donc ce choix.

L’utilisateur procède alors en deux étapes: il décrit d’abord l’aspect structuré de l’objet identifié selon la syntaxe de Troll, ensuite il construit de manière naturelle le réseau ECATNet qui modélise le comportement parallèle de l’objet question. Les événements de l’objet seront représentés par des transitions, leurs attributs par des places. Des transitions et des places additives, liées aux problèmes de synchronisation entre les activités internes de l’objet, sont nécessaires. Il est évident que les événements identifiés doivent avoir une granularité (événements élémentaires) telle que le parallélisme intra-objet puisse être capté. Le réseau ECATNet ainsi obtenu peut être transformé en une théorie de logique de réécriture [BET 92].

2.2. Parallélisme inter-objets

Qu’au parallélisme inter-objet, nous suggérons de retenir l’approche adoptée dans le langage Maude [MES 92]. Le comportement parallèle d’une communauté d’objets décrite en Maude s’exprime de manière élégante et correcte par le biais de règles de la logique de réécriture. La logique de réécriture peut donc servir de cadre unificateur pour décrire le parallélisme inter et intra objets : une spécification basée sur les objets. Des travaux ayant pour objectif la transformation d’une spécification Troll en une spécification Maude ont déjà été menés [DEN 94]. L’intérêt de Maude, outre la prise en charge du vrai parallélisme inter-objets, est sa qualité de langage exécutable, ce qui n’est pas le cas de Troll.

L’intégration des ECATNets dans Troll et la transformation d’une spécification ECATNet en une spécification Maude sont en cours.
3. Exemple

L’approche de spécification formelle de protocoles à l’aide de Troll, que nous préconsions, adopte le style de description orienté ressource, style de description souvent utilisé par la communauté des protocoles de communication, entre autres avec Lotos [TUR 93]. Troll supporte le paradigme objet, paradigme qui s’adapte bien à ce style. De plus, il améliore de manière appréciable les possibilités de ce style grâce aux mécanismes abstraits d’héritage, d’agrégation et d’interaction. Les techniques de description formelle citées en introduction n’offrent pas ces mécanismes malgré des tentatives d’extension comme par exemple dans [RUD 91].

L’exemple traité dans cette section, le protocole du bit alterné, montre l’intérêt de ces mécanismes. Ils seront pleinement exploités dans cet exemple.

Une analyse rapide du protocole du bit alterné conduit à l’identification naturelle des composantes suivantes : un agent responsable de l’émission de paquets, un agent responsable de la réception et un agent responsable du transfert des paquets de données et d’acquittements (canal de communication).

Les sous-sections suivantes présentent successivement la spécification en Troll de l’agent de transfert, de l’agent émetteur, de l’agent récepteur et enfin celle du système en entier. Dans la suite de cette section le terme "classe" sera employé dans son sens classique c’est à dire le sens de "moule".

3.1. Spécification d’un canal de communication non fiable

Un canal de communication fiable est souvent décrit par le biais d’une file d’attente à capacité finie. L’envoi d’un paquet est perçu comme le dépot d’un élément en queue de la file d’attente et la réception d’un paquet comme le retrait d’un élément en tête de la file d’attente. La description d’un canal avec perte peut, quant à elle, être obtenue par spécialisation en raffinant la classe File-attente.

Il est donc naturel de commencer par décrire la classe File-attente, supposée à capacité infinie, et de procéder, ensuite à son raffinement jusqu’à aboutir à la description de la classe canal de communication non fiable. Un premier raffinement conduit à la classe Buffer, dont les objets sont des files d’attente à capacité finie. Cette classe sera utilisée aussi bien pour modéliser un canal de communication fiable (réputé de capacité finie), que pour modéliser les entités de stockage temporaire des trames à émettre et reçues. La réutilisation est ainsi mise à profit.

La classe File-attente, de capacité infinie, est décrite de manière récursive. De plus elle est paramétrée par le type abstrait algébrique 'élément'.

En plus de ses attributs naturels (tête, queue et taille de la file d’attente), l’attribut supplémentaire Reste permet la description récursive de la file. La valeur de l’attribut 'Reste' d’une instance de la classe File-attente fait référence à une autre instance de cette classe. Celle ci n’est autre que la file obtenue en retirant à f son élément de tête.
Template File-attente (element) (* File d'attente*)
data types element, No;

Attributes
Tête: element; (*premier élément de la file d'attente*)
Queue: element; (*dernier élément de la file d'attente*)
Reste: No; (No :Type dont les valeurs représentent l'espace des identités des objets de cette classe)
Taille: nat; (*nombre d'éléments dans la file d'attente*)

Constraints
{ Taille = 0 } ==> Tête = 'erreur'; Reste.Taille = 0; Queue = 'erreur';
{ Taille = 1 } ==> Tête = Queue; Reste.Taille = 0;

Events
birth créer ( in Reste: No); (*Événement de création d'une instance de la classe File-attente; avec l'identité Reste comme paramètre d'entrée*)
depôt (in e: element); (*Événement de dépôt de l'élément e dans la file d'attente*)
retrait; (*Événement de retrait d'un élément de la file d'attente*)
death détruire; (*Événement de destruction de l'instance*)

Valuation (Exprime l'effet des événements sur les attributs)
variables e: element, Rest0: No;
{ créer( Rest0) Taille = Rest0.Taille; Tête = Rest0.Tête; Queue = Rest0.Queue; Reste = Rest0; }
{ Taille > 0 } —> [retrait] Taille = Taille -1; Tête = Reste.Tête; Queue = Reste.Queue;
{ Taille = 0 } —> [dépôt(e)] Tête = e; Taille = Taille +1; Queue = e;
{ Taille >0 } —> [dépôt(e)] Queue = e; Taille = Taille +1

Behavior
Permission
{ Taille >0 } retrait; (*L'événement retrait n'est autorisé que si la file d'attente est non vide*)

Commitments
variables e: element;
{ after (retrait) and Reste.Taille > 0 } ==> Reste.retrait; (*L'occurrence de l'événement 'retrait' d'un élément doit être suivie immédiatement par un événement 'retrait' d'un élément de la file d'attente Reste; respect de la cohérence e la file d'attente*)
{ after (dépôt(e)) and Reste.Taille >0 } ==> Reste.depôt(e); (*L'occurrence de l'événement 'dépôt' d'un élément dans une file vide doit être suivie immédiatement par un événement 'dépôt' de ce même élément dans la file vide Reste; respect de la cohérence de la file d'attente*)

End Template File-attente

La classe Buffer hérite de la classe File-attente. Elle est obtenue grâce à une restriction (contrainte) imposée à l'attribut Taille. Cet héritage respecte la
propriété d’encapsulation de la classe File-attente.

Template Buffer (element)
Specialisation of File-attente (element);
Constraints
initially Taille < MaxB; (*Indique que la taille de la classe Buffer est bornée*)
Events
birth buffer;
death annuler
Behavior
Permissions
variables c: element;
{ Taille < MaxB } dépot(c); (*L’occurrence de l’événement dépot d’un élément n’est autorisée que si la taille maximale du buffer n’est pas atteinte*)
End Template Buffer

La classe Canal, décrivant le concept de canal de communication, est une spécialisation de la classe Buffer. Elle est caractérisée par un événement supplémentaire qui modélise la perte de paquet, et par un attribut supplémentaire Ls, de type liste d’entier, qui reflète l’historique des séquences de perte de paquets survenues sur le canal de communication. L’attribut Ls dont la valeur peut être dérivée de celle de Ls, représente l’entier maximal de la liste Ls.

L’occurrence de l’événement ‘perte d’un paquet’ est supposée survenir soit suite à l’occurrence d’un événement ‘dépot’ de paquet, soit suite à l’occurrence d’un événement ‘perte de paquet’ (modélisation de la situation de pertes successives de paquets).

Dans le premier cas le paquet déposé en queue du canal est retiré, ce qui viole la propriété d’encapsulation de la classe Buffer, et de plus l’entier 1 est inséré dans la liste Ls.

Dans le second cas de figure le dernier élément de la liste Ls est incrémenté. L’opérateur temporel after permet d’exprimer ce comportement du canal de communication.

Template Canal (element)
Specialisation of Buffer (element)
Template
data types natlist; (*Liste ordonnée de naturels*)
Attributes
Ls: natlist;
L: nat;
Derivation L = max (Ls); (*Retourne l’entier maximal de la liste Ls*)
Constraints
initially L = 0; Ls = []; (* [] représente une liste vide*)
Events
active perdre; (*Evénement interne modélisant la perte d’une trame *)

Valuation
variables e, e': element;
{ (Queue = e') and (after depôt(e)) } ==> [perdre] La=concatener (La,1);
Queue = e';
{after (perdre)} ==> [perdre] La = Incrementer_dernier_element_liste

Behavior
variables e: element;

Permissions
sometimes(after (depôt(e)) or after (perdre)) } perdre; [1]
{ L < MaxL } perdre; [2]
End Template Canal

La formule [1] permet de spécifier que l’événement 'perte de paquet', considéré comme phénomène aléatoire et interne au canal, peut survenir au cours de l’évolution du canal.
La formule [2] exprime, de manière indirecte, que le nombre de retransmissions successives d’un paquet sur un canal de communication est limitée à une constante de valeur MaxL.

3.2. Spécification globale du système

La communauté d’objets constituée de l’objet EMT et de l’objet RCV en interaction modélise le système étudié. L’interaction entre ces deux objets complexes est assurée grâce au mécanisme de partage d’événements qui supporte le langage Troll. En effet deux canaux de communication sont considérés, dans notre modélisation, comme des objets partagés entre EMT et RCV.

L’objet complexe EMT se compose en fait:
- d’un objet, instance de la classe Buffer, pour le stockage des trames de données à émettre ou à réémettre
- de deux objets, instances de la classe générique Canal qu’il partage avec l’objet complexe RCV. L’un représente le canal qui véhicule les paquets de données et l’autre celui qui véhicule les acquittements.
- et un objet, instance de la classe Cellule booléenne, qui sert au suivi et au contrôle des acquittements attendus. Cette classe peut être vue comme un spécialisation de la classe Buffer: la valeur de l’attribut Taille est "forcée" à un, le paramètre ‘élément’ est instanci au type booléen.

Object EMT
Template
data types [Canal (paquet)],[Canal (bool)],[Cell (bool)],[Buffer (element)];
Attributes
SDCh: [Canal(paquetc)] (*Identité de l’objet canal de communication dédié aux trames de données*)
SACH: [Canal (bool)] (*Identité de l’objet canal de communication dédié aux acquittements*)
SBC: [Bcell (bool)] (*Identité de l’objet cellule booléenne*)
SBuf: [Buffer (element)] (*Identité de l’objet buffer émission*)

**Events**
birth créer;
active émettre; (*Evénement d’émission ou de réémission de trames*)

**Behavior**

**Interactions** (Exprime l’interaction entre l’objet composé EMT et ses objets composants)

\[
\{( \text{SBuf.Taille} <> 0 \text{ et } (\text{SACH.Tête} <> \text{SBC.Contenu})\} \Rightarrow \text{emettre} \}
\]
SACH.Retrait; SDCh.depot (paire (SBuf.Tête, SBC.Contenu)); [3]

\[
\{( \text{SBuf.Taille} <> 0 \text{ et } (\text{SACH.Tête} = \text{SBC.Contenu})\} \Rightarrow \text{emettre} \Rightarrow
\]
SBC.inverser; SBuf.retrait; SACH.retrait; [4]

**Pattern** (description du cycle d’évolution de l’objet ou processus)
créer \(\rightarrow\) EMETTRE
process EMETTRE = émettre; EMETTRE (*processus récursif modélisant l’activité d’émission*)

**End Object EMT**

L’objet complexe RCV se compose:
- d’un objet, instance de la classe Buffer, pour le stockage des trames reçues.
- de deux objets, instances de la classe générique Canal qu’il partage avec l’objet complexe EMT. L’un représente le canal qui véhicule les paquets de données et l’autre celui qui véhicule les acquittements.
- et d’un objet, instance de la classe Cellule Booléenne, qui sert au suivi et au contrôle des trames attendues.

**Object RCV**

**Template data types** [Canal (paquet)], [Canal (bool)], [Bcell (bool)], [Buffer (element)];

**Attributes**
RDCh: [Canal (paquet)] (*Identité de l’objet canal de communication dédié aux trames de données*)
RACH: [Canal (bool)] (*Identité de l’objet canal de communication dédié aux acquittements*)
RBc: [Bcell (bool)] (*Identité de l’objet cellule booléenne*)
RBuf: [Buffer (element)] (*Identité de l’objet buffer de réception*)

**Events**
birth créer;
active recevoir; (*Evénement de réception de trames*)

**Behavior**
Interactions (Exprime l'interaction entre l'objet composé RCV et ses objets composants)

\[((RDCh.Tête).Second) \leftrightarrow RBc.Contenu]\} recevoir \>
RACH.depot((RDCh.Tête).Second); RDCh.retrait [5]

\text{Pattern}
\text{créer} \rightarrow \text{RECEVOIR}
\text{process RECEVOIR = recevoir; RECEVOIR; (*processus récursif modélisant l'activité de réception*)}

End Object RCV

Remarques :
La classe Buffer a été mise à contribution pour représenter diverses autres classes utiles pour le système en cours de description (en tant que classe à affiner par spécialisation et en tant que classe composante d'une classe composite).


La spécification d'un système en Troll est décrite en termes de relation et d'interaction entre la communauté d'objets le constituant. Dans notre cas, la seule interaction est celle qui existe entre l'objet EMT et l'objet RCV par le biais des objets canaux, communs aux deux. Cette contrainte de partage peut être décrite aisément en Troll à l'aide du concept de "relationship".

Relationship PARTAGE.CANAL Between EMT, RCV

Constraints
SDCh = RDCh (*Exprime que les objets EMT et RCV partagent le même objet canal de communication dédié aux trames de données*)
SACh = RACh (*Exprime que les objets EMT et RCV partagent le même objet canal communication dédié aux acquittements*)

End Relationship PARTAGE.CANAL.

 Conclusion
Dans cet article nous avons présenté une approche de spécification formelle de protocoles de communication à l'aide du langage Troll. L'originalité de cette approche peut être résumée dans les trois points importants suivants : elle s'appuie sur un langage formel, Troll, muni d'une sémantique "sauve" fondée sur des bases mathématiques solides, telles que la théorie des catégories et la logique...
mathématique, ce qui le distingue, à notre sens, des autres langages formels traditionnellement utilisés par la communauté des protocoles de communication (Estelle et Lotos). Elle supporte, tout en l'améliorant, le style de description orienté ressource, style souvent adopté par cette même communauté, ceci grâce à l'apport des concepts du paradigme objet. Enfin, cette approche est apte à capter la description du vrai parallélisme intra et inter-objets inhérent au système à spécifier, ce qui la distingue des autres approches formelles connues. Cette aptitude est la conséquence du choix des ECATNets, formalisme élégant et convivial apte à décrire le vrai parallélisme selon une semantique "sauve" [BET 92], [MES 95], comme moyen d'expression du comportement dynamique d'un objet dans Troll.

Pour des raisons d'espace, un protocole simple et connu de tous illustre notre approche dans cet article. Mais en fait, celle-ci fait l'objet d'une expérimentation sur des exemples plus réalistes, complexes. Ces exemples relèvent du monde des réseaux de communication ; systèmes de télécommunications, administration réseau.

5. Références

OG 91] J. Goguen, "Types as Theories", Proceedings of the Conference on

OG 92] J. A. Goguen, R. Diaconescu, "Towards an algebraic semantics for

OG 92] J. A. Goguen, "Algebraic Semantics For the Object Paradigm", In-

JUN 91] R. Jungclaus, G. Saake, T. Hartmann, "Language Features for Ob-

MES 92] J. Meseguer, "A Logical Theory of Concurrent objects and its Real-

MES 95] J. Meseguer, N. Marti-Oliet, "From Abstract Data Types to Logical


UD 91] S. Rudkin, "Inheritance in Lotos", 4th International Conference on

SAA 93] G. Saake, T. Hartmann, R. Jungclaus, H.-D. Ehrich, "Object Ori-