bibLOTOS: Une bibliothèque de constructions prédéfinies pour un projet de gestion de réseau

Cristiano Maciel Mirela Sechi Moretti Annoni Notare Elenirse Maria Furlanetto

Erreur! Signet non défini. **Erreur! Signet non défini.** **Erreur! Signet non défini.**

Bernardo Gonçalves Riso Carlos Becker Westphall

Erreur! Signet non défini. **Erreur! Signet non défini.**

Université Féderale de Santa Catarina - UFSC

Département d´Informatique et de Statistique - Laboratoire de Réseaux et de Gestion - LRG

B.P. 476 88040 970 Florianópolis SC Brésil +55(48) 231 9739 fax +55(48) 231 9770

Resumé: Dans ce travail sont présentés les aspects concernant la conception, l´implémentation et l´usage de la bibLOTOS, une bibliothèque de constructions prédéfinies réalisées avec l´usage d´une technique de description formelle LOTOS. Cette bibliothèque compile des constructions en LOTOS Basique (où quelques aspects de comportement sont définis), en ACT ONE (où sont représentés seulement quelques types de données abstraites) et en LOTOS Complet (réunissant comportement et données en chaque construction). L´emploi de la bibLOTOS ainsi comme, de la méthodologie à elle associée sont illustrées comme un projet d´un système de gestion de réseaux. Dans cet exemple illustré, les travaux d'analyse et d´implémentation automatique sont réalisés avec l´emploi des outils comme: Eucalyptus, Caesar et Aldébaran.

Mots-Clés: Spécification formelle, Constructions prédéfinies, LOTOS, Gestion pro-active, Eucalyptus, Caesar et Aldébaran.

1. Introduction

Plusieurs secteurs industriels utilisent des produits pré-fabriqués (de l'usage général) pour la manufacture de nouveaux produits (de l´usage spécifique). C'est le cas, par exemple, de l´industrie de la construction civile où briques, tubes, armatures, blocs, fer et beaucoup d´autres produits pré-fabriqués sont utilisés dans la construction des édifices. Les exemples dans l´industrie alimentaire (qui utilise des aliments pré-élaborés) et dans d'autres industries (comme dans le cas de l´industrie électrique et électronique) sont nombreux. Le succès obtenu par toutes ces industries en ce qui concerne le rapide procès de production et réduction des coûts est indiscutable. Les entreprises de production de logiciel et en particulier, les entreprises de logiciel de communications (ainsi inclus les protocoles de communications pour l'industrie) pouvent se servir de cette approche qui consiste à utiliser dans ces projets un bloc de constructions pré-définies pour qui peuvent être réutilisées [CoOl 96]. Cette idée n´est pas nouvelle. Pour cela, on établit aujourd´hui des conditions qui résultent de la disponibilité de techniques de description formelle (TDF) et associés à ces TDF, méthodologies de projet qui s´appuient dans l´usage de puissants outils actuels et automatiques [ISO 8807] [BoBr 87] [QuCu 94] [Gara 96]. Ces conditions, une fois réunies, constituent un arsenal de grand effet pour la conception et l´implémentation de systèmes de communications (par exemple dans le domaine de gestion de réseaux) avec rapidité, sécurité et bas prix de développement. De cet arsenal une bibliothèque de constructions prédéfinies est un des principaux éléments [PiSi 92]. Pour donner au procès de développement de logiciel de communication et de gestion de réseaux une forme industrielle, il faut pour cela monter telle bibliothèque de constructions pré-définies et d´établir une méthodologie de projet qui oriente l´usage systématique de cette bibliothèque en projets spécifiques [BoMo 91]. C´est bien l´objectif du présent travail. Les réseaux d´ordinateurs et de télécommunications forment des milieux hautement hétérogènes, exigeant des mécanismes de gestion de plus en plus sophistiqués et automatiques [LePe 94]. Dans ce contexte, il est important d´adopter l´approche pro-actif dans les plateformes de gestion, à cause de la capacité de controler les ressources et détecter les possibles problèmes qui peuvent affecter le fonctionnement du réseau. Le domaine de la gestion pro-active des réseaux a évolué rapidement. Comme résultat de cette évolution, des applications pour la comptabilisation des ressources de réseau ont eté développées; le contrôle de congestionnement des blocs et le controle de l´accès au réseau [Roch 96]. En divers cas, le projet de ces applications est réalisé d´une façon informelle. L´étude et la spécification de ces applications avec l´usage de constructions prédéfinies est de grand intérêt pour la recherche.

Dans la section 2 de ce travail sont préséntés des aspects de la conception et de l´implémentation de la bibliothèque ´bibLOTOS´. Dans la section 3, on présente la méthodologie de projet associée avec l´usage de ´bibLOTOS´. Cette méthodologie consiste en plusieurs étapes de développement et étapes d´analyse. Pour illustrer l´usage de ´bibLOTOS´ et de la méthodologie associée à cette bibliothèque, le projet d´un système de gestion pro-active de réseaux est développé dans la section 4. Les conclusions sont établies dans la section 5. Finalement, la bibliographie (section 6).

2. La Bibliothèque bibLOTOS

La bibLOTOS réunit constructions prédéfinis organisées en trois segments principaux. (1) Constructions en LOTOS Basique; (2) Constructions en Act One; (3) Constructions en LOTOS Complet.

Génération Systématique de Constructions Prédéfinies: Dans la conception de la bibLOTOS est utilisé un procédé pour la génération systématique de constructions pré-définies, qui part des cas très simples (processus basiques comme stop et exit) évolue vers des constructions plus complexes (par exemple dans le contexte de LOTOS Basique: comportements finis et infinis, ainsi comme les comportements déterministes et non-déterministes).

Conception de la bibLOTOS: Pour chaque construction pré-définie, spécifiée en LOTOS, et présente dans la bibliothèque bibLOTOS, il y a un bloc d´informations associées qui sont accumulées. L´idée est d´offrir aux membres des ou à l´utilisateur un élément de facile utilisation qui lui permet de simplifier la construction de spécifications LOTOS permettant d´introduire des réprésentations graphiques et textuelles. Pour cela, les constructions de la bibLOTOS sont accumulées de façon que chaque construction corresponde à plusieurs champs. La bibLOTOS est constituée par des champs suivants: code, nom, spéc_informelle, spéc_formelle, spéc_C, LTS, représ_graphique et util.

Code: Champs de donnée type texte, dans lequel est accumulé le code structuré de la construction prédefinie, formé par les quatre premières caractères et le nom du processus défini par le spécificateur, comme suite: 1º caractère: LOTOS Basique / Act One / LOTOS Complet; 2º caractère: Processus Fini / Processus avec possibilité infinie; 3º caractère: Processus déterministes / non-déterministes; 4º caractère: Processus avec seulement des éléments observables / Processus avec des éléments internes; et 5º caractère: Souligne (_) qui sépare le code structuré du nom du processus. D´autres caractères = nom du processus défini par le projectiste. **Nom**: champs de donnée de type texte qui identifie la construction prédefinie selon le code. **Spéc_informelle**: champs de donnée de type texte qui montre informellement la construction préféfinie. **Spéc_formelle:** champs de donnée de type texte qui emmagasine la construction prédefinie en LOTOS déjà approuvée avec l´utilisation des données. **Spéc_C**: champs de donnée de type texte, dans lequel est donné une implémentation de référence, en code C, de la spécification correspondente en LOTOS. Ce code C est obtenu automatiquement avec l´aide d'outil TOPO et transporté pour la bibLOTOS. **LTS:** champs de donnée type texte où un système de transitions etiquetées correspondant à la spécification est représenté. **Repres_graphique**: champs de type object Microsoft ´OLE´™ dans lequel est représentée graphiquement la construction prédéfinie. Pour cette représentation est utilisée le language G-LOTOS (Graphique LOTOS) qui permet de représenter graphiquement les spécifications LOTOS. La représentation graphique permet de choisir et sélectionner des constructions prédéfinies dans les phases de développement et d´analyse de systèmes. **Util:** champs de donnée de type texte qui contient un historique de l´usage de la construction prédéfinie dans un système et que l´utilisateur va fournir durant l'usage de la bibliothèque. Ce champs n´a pas été encore instalé. Voir la Figure 2.1.

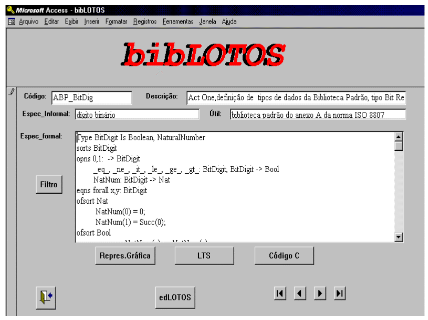

Figure 2.1. - Gérance de base de donées Microsoft Access.

Intégration de la bibLOTOS à l´environnement LOWE: Pour l´implémentation de la bibLOTOS, on avait utilisé le système administrateur de banque de données relationelle pour Windows, Microsoft Access™. La bibLOTOS est implémentée à l´environnement LOWE où est constaté aussi d´autres éléments d´aide au projet avec l´usage de LOTOS en phase de développement ou d´extension . Voir la Figure 2.2.

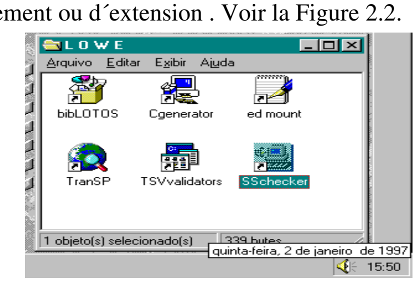

Figure 2.2 - Environnement d´outils LOWE, où est intégré à la bibLOTOS.

En tant que base de données relationelle, l'Access™ utilise des tableaux pour accumuler les données de l'usager. Ainsi, le premier pas pour l'implémentation de la bibLOTOS a été la création des tableaux de données qui feraient partie de la base. Ensuite, des relations entre les tableaux et les plataformes de recherches de données ont eté établies. Microsoft Access™ possède des «macros» qui permet d'automatiser quelques travaux (par exemple, dans le cas de recherche par différents champs des registres). On peut comparer les macros avec les outils de programmation qui peuvent être attribuées à des boutons. Microsoft Access™ possède un logiciel, dénommé Microsoft ADT™ responsable pour la formation du code d'installation pour une base de données d'Access™. Il dispense l'Access™ complet et permet d'utiliser une base de données avec l'extension MDB. Le choix de l'Access pour l'implémentation de la bibLOTOS se justifie par l'ensemble des ressources mentionnées au-dessus. En outre, Access est un logiciel d'usage généralisé.

3. Méthodologie associée à l'utilisation de la bibLOTOS

Les constructions prédéfinies sont élaborées, classifiées et après archivées dans un bloc. Elles sont classifiées en accord avec la ressemblance de ses comportements ou pour l'usage des structures des données. La classification des constructions prédéfinies permet le procès de recherche et sa réutilisation.

Aspects de développement: Pour développer la spécification d'un système, contenant constructions réutilisables, on peut suivre les indications suivantes: (1) Au début, il définit comment le système se comporte et représente ce comportement à travers de la composition de constructions prédéfinies. Il peut rechercher les constructions prédéfinies de son intérêt à travers du code structuré de l'accumulation de ces constructions. La recherche des constructions prédéfinies par le code structuré se donne dans une structure en arbre avec repertoires et sous-repertoires. (2) L'usager examine la construction prédéfinie et peut la sélectionner pour sa réutilisation. Les divers champs de la bibliothèque (bibLOTOS), tels que la spécification informelle, sa représentation graphique et son LTS (système de transitions etiquetées), par exemple, permettent la sélection de la construction qui satisfait au mieux l´utilisateur durant la phase de développement d'un système. (3) L´utilisateur sélectionne les constructions prédéfinies nécessaires à la spécification en développement. Le transfert de la construction prédéfinie de la bibLOTOS pour la spécification d'un système se fait à travers d'un bouton automatique (une macro répresentant les options copier et coller dans le menu Editer). (4) L'éditeur des spécifications du environnement LOWE permet que la construction prédéfinie récupérée soit modifiée jusqu'à ce que représente fidèlement l'intention du projectiste. En accord avec les indications schématisées au-dessus, les principales étapes de la méthodologie sont: accumulation, classification et récupération de constructions prédéfinies (pour l'alimentation de la bibliothèque), modifications et compositions de spécifications (utilisant de la bibLOTOS). La Figure 3.1 représente le processus de développement de la spécification d'un système avec l'usage des constructions prédéfinies accumulées dans la bibLOTOS.

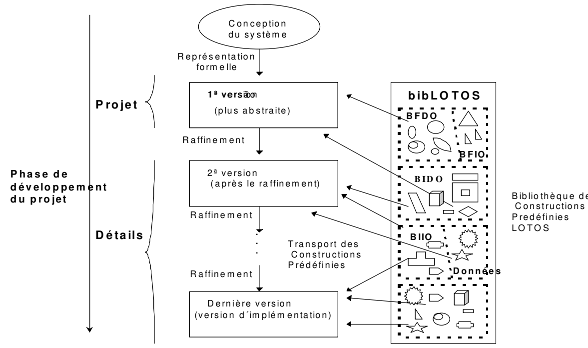

Figure 3.1- Vision schématique du processus de développement de système avec l'usage de constructions prédéfinies accumulées dans la bibLOTOS.

Aspects de l'analyse: La méthodologie proposée adopte une élaboration de projet descendante *top-down.* Partant d'une spécification en haut niveau d'abstration, des nouvelles spécifications, chaque fois plus raffinées, vont être obtenues à mesure que les décísions de projets sont introduites dans les spécifications. Chaque nouvelle version est comparée avec la précédente. La nouvelle version est considérée correcte si elle est équivalente à la précédente. La preuve d'équivalence est réalisée avec l'aide des outils automatiques. Voir la Figure 3.2.

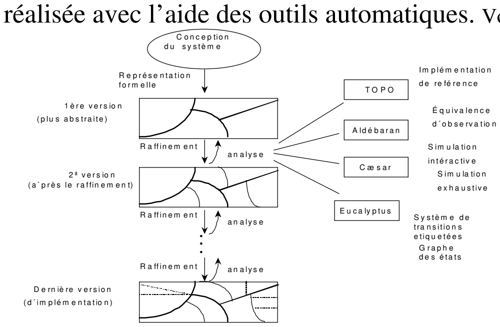

Figure 3.2 - Vision schématique du processus d´analyse des versions successives du système en développement avec l´usage des outils automatiques d´aide au projet (TOPO, Aldébaran, Caesar et Eucalyptus).

Implémentation automatique: Les implémentations de référence (qui peuvent être utilisées comme point de départ de chaque version de logiciels éficients) sont obtenues automatiquement à partir de chaque version de la spécification du système en développement. En particulier, la dernière version (dénommée version d'implémentation ou de réalisation du système) permet d'obtenir une implémentation de référence plus complète.

4. Projet d'un système de gestion proactive

La gestion proactive de réseaux a pour objectif de prévenir les problèmes qui peuvent être provoquées dans le réseau et éviter qu´ils ne perturbent les services offerts aux usagers. Pour cela, la gestion proactive doit être capable de détecter les indices de problèmes avant qu'ils ne se produisent. L'application de la gestion proactive se fait à travers d'un système complexe où plusieurs éléments spécifiques sont combinés pour réaliser ce type de gestionnement. La complexité de la gestion proactive de réseaux sugère l'usage de rigoureuses approches de projet. La structure de ces systèmes peut béneficier des advantages offerts par l'usage des Techniques de Description Formelle (TDFs). Dans les sous-sections suivantes, on développe une étude consacrée à une application pour la gestion de réseaux à travers de spécification formelle avec l'usage de constructions prédéfinies. Dans cette étude, l'application est partiellement spécifiée. La principale composante de la gestion proactive, qui est un agent proactif, a son comportement spécifié en accord avec la méthodologie proposée. L'application spécifiée utilise la gestion proactive pour controler le trafic de messages en bloc dans le réseau. La spécification formelle de cette application fait l'usage de constructions prédéfinies présentes dans la bibLOTOS.

Description Informelle: L'agent proactif PROAG est responsable pour contrôler et analyser le comportement du réseau. Il réalise des opérations sur les objets de la MIB (Management Information Base) et compare les valeurs obtenues avec celles accumulées dans la Baseline. (Les accès aux valeurs accumulées dans la Baseline sont représentés à travers d'actions à la porte **baseline_data**). **L'Architecture du Système:** En haut niveau d'abstraction, l'agent proactif peut être vu comme un système dont seules les portes d'entrée et de sortie sont observables. Pour ne pas considérer la structure interne du système on adopte une représentation en boîte noire. L'envoi de notifications au gérant est représenté par des actions dans la porte **notif**. De temps en temps, l'agent proactif **PROAG** peut recevoir des notifications provenant du **MIB**. C'est le cas, par exemple, quand la valeur d'un paramètre de fonctionnement du réseau dépasse la limite pré-établie. La spécification formelle correspondent en LOTOS peut être présentée comme suit (en spécifiant un haut avec niveau d'abstraction).

```
specification PROAG[notif,baseline_data,mo_notif,operat] : noexit
behaviour (*le comportement de l´agent pro-actif est défini par le processus PROA*)
       PROA[notif,baseline_data,mo_notif,operat]     where
process PROA[notif,baseline_data,mo_notif,operat]:noexit:= ... endproc endspec
```

Vu comme une boîte noire, le processus **PROA** possède quatre portes de communication avec le milieu externe (**notif**, **mo_notif**, **baseline-data**, **operat**). Le comportement du **PROA** est détaillé informellement comme suit. Dans le processus **PROA**, on a au début un choix non-déterministe (représenté par l'opérateur [ ]) dans lequel l'agent proactif demande une opération à la **MIB** à travers de la porte **opera***t* ou l'agent proactif reçoit de la **MIB** une notification, par exemple un **GET** (à travers de la porte **mo-notif**) quand cela est nécessaire. Après réalisation de l'une des alternatives du choix, l'agent obtient les informations sur la Baseline à travers d'une action dans la porte **baseline_data**. On introduit alors un nouveau choix où après une prise de décision, on envoie une notification au gestionaire avant que le processus **PROA** ne soit appelé recursivement.

```
process PROA [notif,baseline_data,mo_notif,operat]:noexit:=
   operat; (*réalise des opérations sur les objects *)
       baseline_data; (* et fait les comparaisons avec la baseline*)
           (i; (*décide de recommencer*)
          PROA [notif,baseline_data,mo_notif,operat]
   []      i; (*décide emettre des notifications*)
          notif;PROA [notif,baseline_data,mo_notif,operat])
   []   mo_notif; (*reçoit les informations de la MIB *)
        baseline_data; (*et fait les comparaisons avec la BASELINE*)
           (i; (*décide de recommencer*)
```

```
         PROA [notif,baseline_data,mo_notif,operat]
  []     i; (*décide emettre des notifications*)
       notif; PROA [notif,baseline_data,mo_notif,operat])  endproc
```

Observant le comportement réalisé par le processus PROA, on peut identifier deux parties semblables. Les dernières sont séparées par le premier niveau du choix. (La séparation est représentée par l'opérateur []). Considérant cette division, la spécification de l'agent proactif peut être divisée en deux parties semblables. Chacune de ces parties peut être spécifiée avec l'utilisation d'une instance d´une construction prédéfinie accumulée dans la bibLOTOS.

```
specification PROAG_1[notif,baseline_data,mo_notif,operat]:noexit
behaviour  PROA [notif,baseline_data,mo_notif,operat]  where
  process PROA [notif,baseline_data,mo_notif,operat]:noexit:=
    operat;BIII_PREDEFINIDO[notif,baseline_data,mo_notif,operat]
 [] mo_notif;BIII_PREDEFINIDO[notif,baseline_data,mo_notif,operat] where
process BIII_PREDEFINIDO[notif,baseline_data,mo_notif,operat]:noexit:=
  baseline_data; (i; PROA [notif,baseline_data,mo_notif,operat]
         []       i;notif; PROA [notif,baseline_data,mo_notif,operat])
       endproc   endproc   endspec
```

Dans ce cas, l'appréciation de constructions prédéfinies permet de donner à la spécification de l'agent proactif une forme. On peut prouver que les spécifications PROAG et PROAG_1 (deux versions différentes du même système) sont equivalentes.

Détail de L'Agent Proactif: En réduisant l'agent proactif, on peut considérer une organisation interne avec deux éléments: Le REMOTE_MONITOR et le VERIFY. La combinaison de ces deux éléments détermine la structure de l'agent proactif. Le REMOTE_MONITOR (moniteur de réseau) rassemble les informations des objets gestionnés des MIBs telles que le RMON MIB et le MIB II. Le VERIFY (verificateur) est responsable pour verifier les manières de dègradation du réseau. Cela est fait à travers des comparaisons entre les valeurs accumulées dans la BASELINE et celles obtenues par le moniteur de réseau. Les notifications sont envoyées pour un gestionnaire (administrateur) à travers d'éléments dans la porte notif. L'organisation interne de l'agent proactif peut être spécifiée comme suit.

```
specification PROAGENT [notif,baseline_data,operat,mo_notif]:noexit
behaviour
hide collected_data in
         VERIFY[notif,baseline_data,collected_data] | [collected_data] |
         REMOTE_MONITOR[operat,mo_notif,collected_data]   where
 process VERIFY[notif,baseline_data,collected_data]:noexit:= ... endproc
 process REMOTE_MONITOR[operat,mo_notif,collected_data]:noexit:= ... endproc
       endspec
```

Pour raffiner le composant VERIFY, on peut définir l´existence deux processus COMPARE et CALCULATE. Le processus VERIFY est composé de deux sous-processus: CALCULATE et COMPARE. Le processus CALCULATE reçoit des données de l'entrée collected_data, et réalise des calculs de variables pour les envoyer du processus COMPARE par l'entrée calculated_data. Ce processus est appelé recursivement. Pour sa part, le processus COMPARE reçoit des données de l'entrée calculated_data, en se comuniquant avec une base de donnée à travers de l'entrée `baseline_data`, afin de collecter les données sur le fonctionnement du réseau, et réalise un choix non-déterminé. Après une probabilité d'une action `i` une notification est envoyée au gérant (dans l' entrée `notif`) ou les données sont calculées et comparées à nouveau; le processus `COMPARE` est ensuite appelé recursivement. Les processus `COMPARE`, `CALCUTATE` et `REMOTE_MONITOR` sont des instances de constructions prédéfinies accumulées dans la bibLOTOS. Un projet plus détaillé de l'agent proactif est aprésenté en [Maci 97]. Dans ce projet, la spécification de l'implémentation est identifiée comme `PROAGENT`. La spécification du `PROAGENT` a l'aspect suivant:

```
specification PROAGENT[notif,baseline_data,operat,mo_notif]:noexit
behaviour hide collected_data in
  VERIFY[notif,baseline_data,collected_data]   |[collected_data]|
  REMOTE_MONITOR[operat,mo_notif,collected_data]      where
  process VERIFY[notif,baseline_data,collected_data]:noexit:=
    hide calculated_data in       CALCULATE[collected_data,calculated_data]
    |[calculated_data]|  COMPARE[notif,baseline_data,calculated_data]   where
     process CALCULATE[collected_data,calculated_data]:noexit:=
     collected_data;calculated_data;
     calculated_data;collected_data;
     CALCULATE[collected_data,calculated_data]        endproc
     process COMPARE[notif,baseline_data,calculated_data]:noexit:=
     calculated_data;baseline_data;
     (i;calculated_data;
     COMPARE[notif,baseline_data,calculated_data]
     []i;notif;calculated_data;
     COMPARE[notif,baseline_data,calculated_data])    endproc    endproc
 process REMOTE_MONITOR[operat,mo_notif,collected_data]:noexit:=
 operat;collected_data;collected_data;
 REMOTE_MONITOR[operat,mo_notif,collected_data]
 []mo_notif;collected_data;collected_data;
 REMOTE_MONITOR[operat,mo_notif,collected_data]     endproc    endspec
```

Validité des spécifications avec l´usage des éléments LOTOS: Deux éléments sont utilisés dans ce travail d'analyse, la validité et la traduction de spécifications. (1) CADP - Caesar Aldébaran Distribution Package) [Gara 96]; et (2) TOPO (topo@dit.upm.es). Les éléments qui composent le CADP sont contemplés dans l´environnement graphique Eucalyptus.

Analyse Syntatique et Sémantique: Au début, l'analyse syntatique et sémantique de l'agent est réalisée à l´aide de l´outil Caesar. Cette analyse peut être réalisée de deux façons; à partir de la ligne de commande Unix ou à partir de l´environnement Eucalyptus. Dans les deux façons, les pas exécutés dans cette analyse sont montrés dans la Figure 4.1.

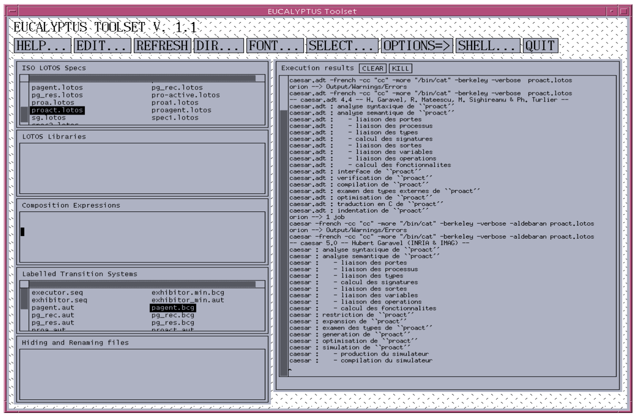

Figure 4.1 - Procédé d'analyse de l'agent proactif dans l´environnement Eucalyptus.

Simulation de l'agent proactif: La simulation contribue pour la validité de spécifications. Dans cette section sont présentées deux types de simulation: (a) La première est la simulation exhaustive où tous les états et les transitions sont représentés en un graphe des états. (b) l'autre est la simulation interactive dans laquelle l´utilisateur peut choisir le chemin qu´il veut parcourir.

Simulation Exhaustive: Durant la compilation, un réseau de Petri est formé. Ce réseau (archive **.net**) est présenté ci-dessous.

```
proagent.net

reseau pour la specification ``proagent''

nombre de variables: 0
nombre d'unites: 4
nombre de places: 12
nombre de transitions: 13
nombre de marques: 3
liste des variables
liste des registres
liste des constantes
liste des unites
unite 0
 place unique initiale: 0
 sous-unites: unite 1
 places: 1 ... 3
 place initiale: 1
unite 2
 places: 4 ...
 donnees
statistiques de l'optimisation
nombre d'unites detruites par l'optimisation E6 : 1 (20 %)
nombre de places detruites par l'optimisation E3 : 7 (37 %)
nombre de transitions detruites par l'optimisation E3 : 7 (35 %)
```

Une simulation exhaustive est obtenue avec le réseau de Petri. Le graphe des états (archive **.gph**) est aprésenté au-dessous.

```
graphe pour la specification ``agent''

nombre d'etats: 18
nombre d'arcs: 22
liste des arcs
0 --- <1> MO_NOTIF.0.0[1] ---> 1
0 --- <2> OPERAT.0.0[1] ---> 2
1 --- <11> COLLECTED_DATA.0.1[3] ---> 3
2 --- <9> COLLECTED_DATA.0.1[3] ---> 4
. . .
statistiques
nombre d'etats : 18
 taille d'un marquage : 2 octets
. . .
 taille de la table de hachage : 8329 entrees, soit 33316 octets
taille minimale des collisions : 0
 taille maximale des collisions : 1
```

Durant la compilation, l'ation -aldebaran permet de former un graphe compatible avec l'élémet de vérification Aldébaran. Le graphe obtenu (archive **.aut**) peut être utilisé pour vérifier les équivalences. Ce graphe est au-dessous représenté.

```
des (0,45,18)
(0,MO_NOTIF,13)
(0,OPERAT,13)
(1,BASELINE_DATA,
3)
(2,MO_NOTIF,5)
(2,OPERAT,5)
(2,NOTIF,13)
(3,REQ,11)
(4,MO_NOTIF,3)
(4,OPERAT,3)
(4,REQ,13)
(5,NOTIF,11)
(6,MO_NOTIF,1)
(6,OPERAT,1)
(6,BASELINE_DATA,
4)
(7,MO_NOTIF,2)
(7,OPERAT,2)
(7,NOTIF,0)
(8,MO_NOTIF,4)
(8,OPERAT,4)
(8,REQ,0)
(9,MO_NOTIF,6)
(9,OPERAT,6)
(9,BASELINE_DATA,
8)
(10,BASELINE_DATA
,11)
(11,i,6)
(11,i,12)
(11,i,15)
(12,MO_NOTIF,10)
(12,OPERAT,10)
(12,BASELINE_DATA
,13)
(13,i,9)
(13,i,14)
(13,i,16)
(13,MO_NOTIF,11)
(13,OPERAT,11)
(14,MO_NOTIF,12)
(14,OPERAT,12)
(14,BASELINE_DATA
,0)
(15,MO_NOTIF,17)
(15,OPERAT,17)
(15,BASELINE_DATA
,2)
(16,MO_NOTIF,15)
(16,OPERAT,15)
(16,BASELINE_DATA
,7)
(17,BASELINE_DATA
,5)
```

Simulation Interactive: La simulation interactive permet l'exécution d'une spécification à travers d'un seul chemin, de tel façon que chaque choix non-déterministe est décidé par l´utilisateur. Un exemple de telle simulation est présenté ci-dessous.

```
<initial state>
*** current state
at depth 1
successor #1:
MO_NOTIF
successor #2:
OPERAT
command 1 ? next
which successor
(between 1 and 2)
? 1
*** current state
at depth 2
successor #1: i
(COLLECTED_DATA
[3])
command 2 ? next
. . .
command 15 ? quit
```

Test de l'agent proactif: Le test peut être réalisé à travers de l'exécution de la spécification. Durant l'exécution, il est possible de choisir une parmi ces trois stratégies de test. 1. Le non-déterminisme n'est pas permis; 2. Le commencement du test est choisi aléatoirement; et 3. Le commencement de test est choisi par le spécificateur. Au dessous un exemple est présenté où la stratégie 3 a été sélectionné.

```
caesar.open:
fetching
. . .
available
execution
strategies:
   (1)
deterministic
executed
   (2) non-
deterministic
with random
seed
   (3) non-
deterministic
with chosen
seed
chosen strategy
(between 1 and
3) ? 3
chosen seed ? 3
OPERAT
BASELINE_DATA
NOTIF
MO_NOTIF
BASELINE_DATA
OPERAT
<17 states
visited, 6
visible
transitions
```

Vérification de l'agent proactif: Plusieurs types de vérification peuvent être réalisés pour valider la spécification de l'agent proactif. On présente ici une vérification que affirme l'absence d'impasse et l'équivalence d'observation. **Absence de blocage:** Utilisant l'outil Caesar.open, il est possible de prouver que la spécification de l'agent proactif ne contient pas d'impasse. `*** no deadlock found` Observation: Terminateur ne garantit pas l´absence de deadlock. Il va mieux utiliser plutôt `exhibitor` avec la sequence <any> <deadlock>. **Equivalence d´observation:** La vérification d'équivalence est réalisée en combinant l'outil Caesar avec l'outil Aldébaran. L'utilisation de Caesar permet de former un automate dans la forme convenable pour Aldébaran. L'utilisation d'Aldébaran permet de prouver que les deux automates représentatifs de l'agent proactif à différents niveaux d'abstraction sont équivalents pour un observateur externe. Au-dessous sont présentés les systèmes de transitions étiquetées correspondant à une représentation plus abstraite et à une représentation plus détaillée de l'agent proactif.

Système de transitions étiquetées de la spécification plus abstraite (proa.aut).

```
des (0, 10, 7)
(0, OPERAT, 1)
(0, MO_NOTIF, 2)
(1,
BASELINE_DATA, 3)
(2,
BASELINE_DATA, 4)
(3, i, 0)
(3, i, 5)
(4, i, 0)
(4, i, 6)
(5, NOTIF, 0)
(6,NOTIF,0)
```

Système de transitions étiquetées de la spécification raffinée (proagent.aut)

```
des (0, 22, 18)
(0, MO_NOTIF, 1)
(0, OPERAT, 2)
(1, i, 3)
(2, i, 4)
(3, i, 5)
(4, i, 6)
(5,
BASELINE_DATA, 7)
(6,
BASELINE_DATA, 8)
(7, i, 9)
(7, i, 10)
(8, i, 11)
(8, i, 12)
(9, i, 13)
(10, NOTIF, 14)
(11, i, 15)
(12, NOTIF, 16)
(13, i, 17)
(14, i, 13)
(15, i, 17)
(16, i, 15)
(17, MO_NOTIF, 1)
(17,   OPERAT,   2)
```

Résultat de la vérification de l'équivalence d'observation entre les deux spécifications.

```
aldebaran -oequ proa proagent                TRUE
```

Le résultat **`TRUE`** prouve que les deux spécifications sont équivalentes quant à l'observation. **Traduction de LOTOS pour le code C:** Utilisant l'élément TOPO, on obtient automatiquement le code C de référence correspondante à une spécification LOTOS. Le code C correspondant à une spécification LOTOS de l'agent proactif a 107 lignes. Quelques unes de ces lignes du code d'archive **`.c`** sont aprésentées au-dessous. Cette traduction est aussi disponible dans Eucalyptus.

```
# include "pagent.hh"

gate_list gts;
var_list  vrs;
int       i;

frame M;

PUBLIC void
c0 (b)
  board b;
{
  again:
  M = b->frm;

  switch (M->ent) {
case 0: {
  b = mkhd (b, 1);
  gts = b->fth->gte;
  gts[5] = TRUE;
  M = b->frm;

  mkpe (b, 2, 3);
  gts = b->gte;
  gts[5] = TRUE;
  c0 (b->sons);
  b = b->sons->bth;
  goto again;
 }
 case 2: {
. . .
```

5. Conclusions et futurs travaux

Ce travail présente des résultats d'une activité de recherche et de pour l´ingénierie des protocoles basée sur les méthodes formelles. Dans ce contexte, on a proposé une méthodologie de projet basée sur la réutilisation de constructions prédéfinies. Les auteurs ont implémenté une bibliothèque de constructions prédéfinies (bibliothèque bibLOTOS), ont établi une méthodologie de projet et le développement de protocoles qui utilise la technique de description formelle LOTOS. Cette méthodologie est d'utilité générale, mais dans ce travail, elle est illustrée à travers d'un projet de système de gestion proactive pour des réseaux de communications. Avec ce travail, les auteurs espèrent offrir une contribuition pour la diffusion de LOTOS, suposant que cette diffusion peut être stimulée par la disponibilité d´outils permettant une utilisation facile et une intégration à l´environnement de développement de spécifications LOTOS. Une fois développés, les méthodologies et éléments capable d'aider l´utilisateur, r endent possible le transfert de la technologie LOTOS, du milieu académique au milieu industriel. On pense avec la continuation des investigations déjà commencées, réaliser une étude pour enrichir automatiquement la bibliothèque bibLOTOS avec de nouvelles constructions prédéfinies. On pense aussi avec d'autres études, définir une méthode pour permettre la selection efficace des constructions prédéfinies durant l'élaboration de projets.

6. Remerciements: Nous remercions Hubert Garavel et équipe VASY (INRIA Rhône-Alpes) pour leur collaboration, utile pour le démarrage de nos recherches.

7. Références

[BoBr 87] **Bolognesi, T. ; Brinksma, E.**: *«Introduction to the ISO specification language LOTOS»*. Computer Networks and ISDN Systems, 14(1):25-59, janeiro 1987.

[BoMo 91] **Bochmann, G. v.; Mondain-Monval, P.; Leconte, L.**: *«Formal Description of Network Management Issues»*. Integrated Network Management, II. Elsevier Science Publ. B.V., North-Holland, 1991.

[CoOl 96] **Courtiat, J.-P.; Oliveira, R.C. De**: «*Formal Designs of Multimedia Documents»*. XIV SBRC. 1996.

[Gara 96] **Garavel, H.**: *«Cæsar reference manual»*. Grenoble - France, 1996.

[ISO 8807] **ISO**: *«International Standard - IS 8807. Information Processing Systems - Open Systems Interconnection - LOTOS - A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour»,* 1988.

[LePe 94] **Lehmann Jr., E. O.; Pedroza, A. de *C.* P.**: *«Spécification et Vérification de protocole CMIP pour le gestionnement de réseau»* (en portugais). XII SBRC, 16-20/05/94, Curitiba PR.

[Maci 97] **Maciel, C.**: *«Constructions Prédéfinies Appliquées au Projet de systèmes pour la gestion de réseaux: La Bibliothèque bibLOTOS»* (en portugais). UFSC 1997.

[PiSi 92] **Pires, L.F.; Sinderen, M. van; Vissers, C.**:«*On the use of pre-defined implementation constructs in distributed systems design*». U. of Twente, 1992.

[QuCu 94] **Queiroz, J.A.M. de ; Cunha, P.R.F**.: *»Systèmes distribués: De spécifications LOTOS à implémentations*» (en portugais). IX Escola de Computação, 24-31/07/94. Recife, 1994.

[Roch 96] **Rocha, M.A.**: «*Gestion pro-active de réseaux d´ordinateurs utilisant des agents et techniques d´intelligence artificielle* (en portugais). XIV Simpósio Brasileiro de Redes de Computadores, Fortaleza, CE, 1996.