MASTER EN SCIENCES INFORMATIQUES

Outil d'aide à la construction d'un cahier des charges orienté clients à partir d'une spécification Albert

Alaime, Sébastien

Award date:
1998

Facultés Universitaires Notre-Dame de la Paix

INSTITUT D'INFORMATIQUE

Rue Grandgagnage, 21 5000 NAMUR

Tél. 081/72.49.83 - Fax. 081/72.49.67

Outil d'aide à la construction d'un cahier des charges orienté clients à partir d'une spécification Albert

Par
Sébastien ALAIME

Promoteur : Eric DUBOIS

Année Académique 1997 – 1998

REMERCIEMENTS

Je tiens à exprimer toute ma reconnaissance à mon promoteur Monsieur Eric DUBOIS, pour ses nombreux conseils et l'attention qu'il m'a portée durant l'élaboration et la rédaction de ce mémoire.

Je remercie également Monsieur Patrick HEYMANS, Monsieur JUNGEN Bernard et toute l'équipe d'Albert II pour les nombreux conseils pratiques qu'ils m'ont fournis et pour leur grande disponibilité.

J'exprime ma profonde gratitude à tout le corps enseignant de l'Institut d'Informatique des Facultés Universitaires Notre de la Paix de Namur, pour l'aide et les conseils dispensés durant mes années d'études.

Je remercie toutes les personnes qui ont participé de près ou de loin à l'élaboration de ce mémoire ainsi que les nombreuses personnes qui m'ont soutenu durant mes études.

TABLE DES MATIERES

CHAPITRE I

CHAPITRE II

CHAPITRE III

CHAPITRE IV

CHAPITRE I

INTRODUCTION

CHAPITRE I

INTRODUCTION

1.1 CYCLE DE VIE D'UN LOGICIEL

Dans le cycle de vie d'un logiciel, il y a essentiellement la conception globale et la conception informatique.

- La conception globale du système résout le problème informatique du client ce qui amène à la rédaction du cahier des charges, elle est appelée « **Requirements Engineering** » ou ingénierie des besoins. Elle définit le système, le rôle du software, du hardware, elle identifie et décrit ce que l'on attend du logiciel. Cette phase est primordiale pour assurer le succès du développement et la maintenance d'un système d'information coopératif. Un système d'informations coopératif est une composition de différentes activités pouvant être exécutées en parallèle avec des possibilités de communications et d'interactions **[DU BOIS, 95]**. Le rôle de l'ingénierie des besoins est de saisir et d'analyser les attentes du client tout en respectant les propriétés fonctionnelles et non fonctionnelles du système d'information que l'on veut implémanter. Une des caractéristiques principales de l'ingénierie des besoins est la difficulté de communication entre l'analyste et les clients. L'ingénierie des besoins a les caractéristiques suivantes **[DU BOIS, 94b]** :
  - Une activité de découverte, de compréhension. L'analyste collecte des informations sur le problème du client. Ces informations sont obtenues par : interviews, discussions, observations, étude de la documentation disponible.
  - Une activité de modélisation. L'analyste traite les informations collectées durant l'activité précédente et à partir des descriptions informelles fournies par le client, les transforme en des concepts formels facilement intégrables dans un langage décrivant les besoins.

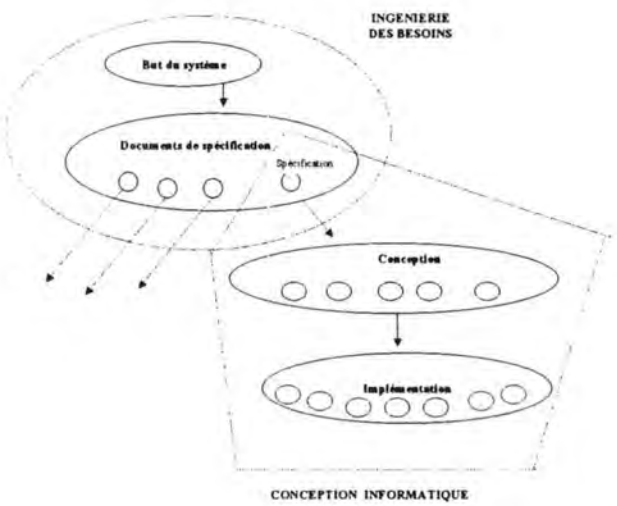
- Une activité d'analyse. L'analyste détecte les problèmes : contradictions, inconsistances, ambiguïtés, redondances dans un document détaillant les besoins.
- Une activité de validation. L'analyste contrôle l'adéquation de ses descriptions en les reformulant au client d'une manière appropriée.

- La conception informatique structure le travail et le décompose en unités en se basant sur le cahier des charges. Elle est appelée **« Software Engineering »**.

Dans un projet informatique, on dégage plusieurs grandes tâches à effectuer. Ces tâches sont souvent exécutées suivant un cycle de vie appelé «Waterfall Model».

- La spécification. Première étape du développement d'un logiciel. Son rôle est de déterminer la fonctionnalité du logiciel.
- La conception (Design). Seconde étape qui permet de dégager une solution abstraite indépendante d'éléments matériels.
- L'implémentation. Troisième étape détermine une solution physique en fonction de l'environnement.
- La phase de tests. Dans cette dernière étape, on élabore une stratégie de tests dont les points clés sont le choix des valeurs, et les endroits décisifs pour effectuer les tests.

Si une mauvaise compréhension du problème apparaît suite à un manque de précision du client par exemple, il est toujours possible de recommencer une étape.



1.2 ALBERT II : UN LANGAGE FORMEL

Parmi les différentes étapes du cycle de vie d'un logiciel, le langage Albert II se place en tout début de cycle c'est à dire en ingénierie des besoins.

Il est destiné à modéliser des systèmes d'informations coopératifs en termes d'agents hétérogènes caractérisés par leurs objectifs individuels, leurs responsabilités vis-à-vis d'actions et leurs moyens de coopération et de communication. Le langage supporte différents styles de spécification pour des situations de coopération prédéfinies et repose sur des bases formelles reposant sur la logique temporelle temps réel **[DU BOIS, 97b]**.

Ces bases formelles permettent de vérifier automatiquement la validité d'une modélisation des besoins.

L'inconvénient majeur est le manque de clarté pour les personnes non familiarisées avec la logique et les mathématiques. Actuellement, il existe des outils qui atténuent ce désavantage :

- Le paraphraseur. A partir d'une spécification formelle en Albert II, il traduit une partie de cette spécification en langage naturel.
- L'animateur. Cet outil permet à l'utilisateur de construire et d'explorer les comportements possibles du système modélisé à partir d'un sénario donné. Il autorise une relecture de l'historique du comportement ainsi que la possibilité de rejouer différemment une partie du sénario **[HEYMANS, 97]**.
- Outil d'aide à la construction d'un cahier des charges. Moins ambitieux que les deux précédents, cet outil élabore un document à partir des descriptions d'une spécification. Le document élaboré est structuré de façon à rassembler tous les éléments liés à un même concept. Le sujet de ce mémoire est le développement de cet outil.

1.3 DESCRIPTION DU PROBLEME

Il est primordial d'avoir de nombreux contacts avec le client durant la phase de développement d'un logiciel afin de bien cerner ses désirs et d'éviter de nombreux problèmes. Durant cette phase, les différents travaux réalisés sont basés sur des concepts pratiquement inconnus du client. Par exemple, dans le langage Albert II une formule mathématique ne fournit pas beaucoup d'informations au client. Le langage Albert II permet d'accompagner une formule d'une description écrite dans un langage

naturel. Actuellement, un client intéressé par la modélisation de ses exigences doit consulter l'éditeur Albert II. Malheureusement, dans cet éditeur, le client doit effectuer de nombreuses manipulations avant d'obtenir toutes les informations escomptées.

Exemple :

Pour obtenir toutes les descriptions relatives à un agent, le client doit cliquer une première fois pour obtenir la description générale de l'agent, plusieurs fois pour avoir accès aux descriptions des différentes actions et états de l'agent, ensuite cliquer quelques fois pour obtenir les descriptions des contraintes relatives à cet agent.

Dans le langage Albert II, les contraintes sont classées par type de patrons. Cette structure en patron est très utile pour l'analyste, elle lui permet d'écrire des contraintes complexes. Cette structure est appropriée à un langage formel mais ne convient pas à la lecture. En effet, il serait préférable de regrouper toutes les contraintes ratachées à un concept.

1.4 Objectif Du Travail

Pour remédier à ces multiples manipulations, il nous a semblé utile de créer un outil rassemblant les différentes descriptions d'une spécification dans un document structuré. Le regroupement de toutes les contraintes autour d'un concept fournit le cahier des charges. Celui-ci facilite une meilleure perception du travail de l'analyste et améliore sa collaboration avec le client. Grâce à cet outil, il est inutile pour le client d'apprendre à se servir de l'éditeur Albert II.

1.5 Description De L'Outil

Pour obtenir les descriptions d'une spécification Albert II, plusieurs approches sont envisageables

- Une première solution est de générer directement à partir de l'éditeur Albert II le fichier de descriptions. A première vue, cette solution est très attrayante étant donné sa facilité de mise en œuvre. Il suffit d'aller rechercher dans les variables les données correspondantes aux descriptions. Le premier désavantage est le faite que l'on est dépendant de l'architecture logicielle de l'éditeur Albert II. Un autre désavantage est l'obligation d'utiliser l'éditeur Albert II pour effectuer des

spécifications. Les spécifications textuelles ne pouvant pas être assimilées par l'éditeur Albert II, ne sont pas prises en compte.

- Prendre le fichier texte qui peut être généré à partir de l'éditeur Albert II et ensuite le parser. En effet le fichier texte représente la spécification en langage Albert II. Cette solution n'étant pas très appropriée est abandonnée. Il faudrait construire un nouveau parseur pour obtenir uniquement des descriptions d'objets, ce qui n'est pas du tout le rôle initial d'un parseur.
- Utiliser le repository d'Albert II et le questionner pour prendre les informations nécessaires à la construction du document. Nous avons choisi cette solution car des programmes sont déjà développés pour peupler le repository Albert II.

Comme nous le verrons plus loin, le repository Albert II est peuplé en miniTelos. Celui-ci est basé sur les concepts de Telos.

Le prochain chapitre décrit le langage Albert II ainsi que le rôle de l'outil développé.
Le troisième chapitre est consacré à la description des principaux concepts de Telos et de miniTelos illustrés par des exemples. MiniTelos est utilisé pour construire et peupler le repository Albert II.
Le quatrième chapitre présente une partie du repository d'Albert II.
Les chapitres suivants sont consacrés aux différentes phases du développement de l'application (phase de spécification et de conception).
Le dernier chapitre conclue ce travail.

CHAPITRE II

REDACTION D'UN CAHIER DES CHARGES

CHAPITRE II

REDACTION D'UN CAHIER DES CHARGES

Après une courte description des concepts principaux du langage Albert II, un exemple de spécification sera donné. A partir de cette spécification, un cahier des charges sera élaboré à partir de notre outil. Les concepts les plus importants du langage sont numérotés de **(1)** à **(8)** pour faciliter la création de liens entre les concepts, la spécification textuelle et le cahier des charges élaboré.

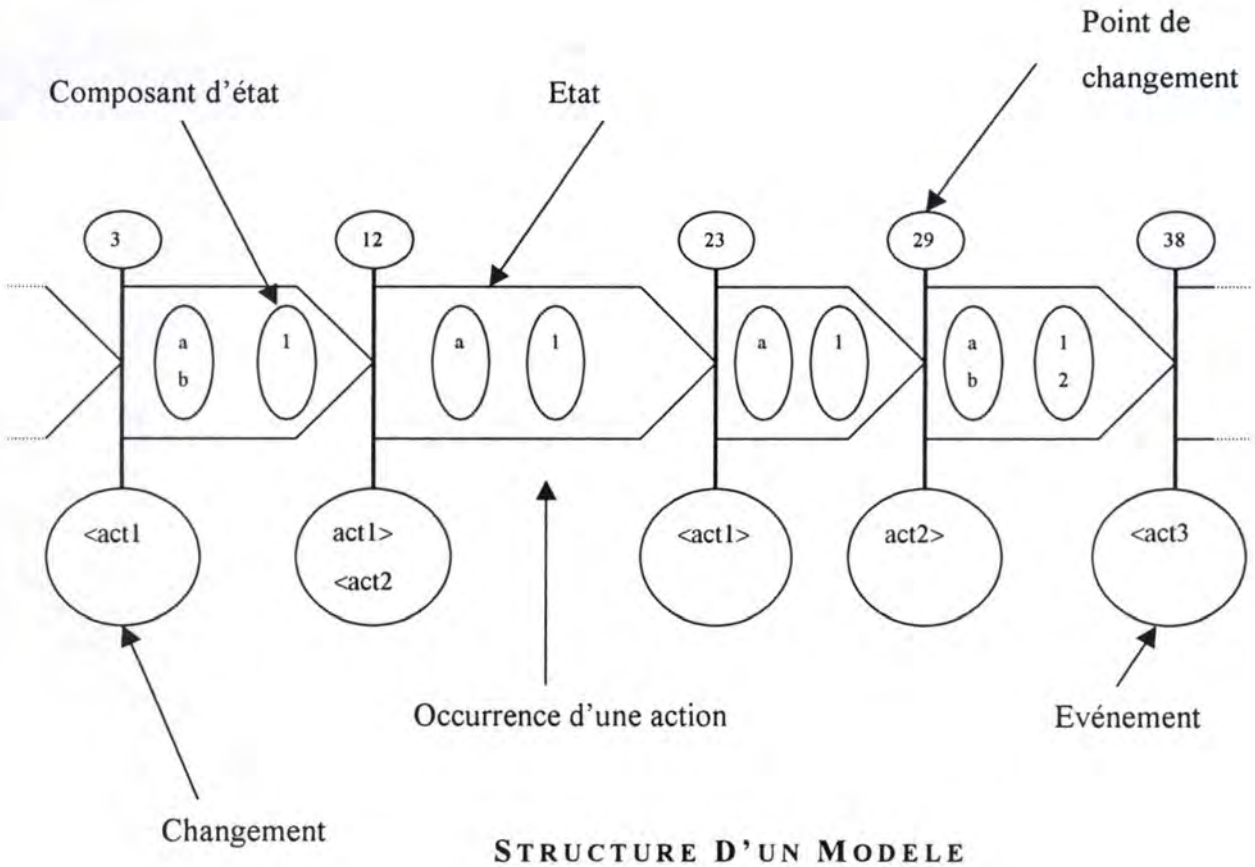
2.1 CONCEPTS DU LANGAGE ALBERT II

Depuis une vingtaine d'années, plusieurs modèles et langages ont été proposés pour modéliser les besoins fonctionnels. D'un côté, des langages ont évolué progressivement de notations graphiques (boîtes et flèches) vers des notations plus formelles basées sur la logique et les mathématiques. D'un autre côté, la plupart des langages existants sont basés sur la modélisation de l'information. Des langages plus récents ont incorporé le paradigme orienté objet déjà bien connu dans les langages de programmation et de conception. Il a prouvé son utilité au niveau de l'ingénierie des besoins. Le langage Albert II est dans cette ligné. Le concept d'agent employé par Albert II peut être vue comme une spécialisation du concept d'objets. Le mot agent a été préféré car plus approprié pour définir les composants appartenant à un système composé dont les responsabilités et les perceptions peuvent changer dans le temps **[DUBOIS, 94a]**.

Le but du langage Albert II est de décrire les comportements admissibles d'un système composé. Une **spécification (1)** Albert II définit des **types de données (2)** et des **opérations (3)** sur ces types de données. Elle est structurée en terme de **sociétés (4)** et d'**agents (5)** (utilisant les types de données définies dans une spécification Albert II). Une société est composée de plusieurs agents «fils » et sociétés «filles ». Un agent représente un sous système. Le modèle de spécification d'un agent consiste en une séquence de changements d'état.

Chaque changement est étiqueté avec une valeur temporelle, ces valeurs forment une séquence strictement croissante. Les intervalles entre les changements ne sont pas

nécessairement équidistants. Les points de changements représentent un instant où il se passe quelque chose dans le système. Un état est constitué de plusieurs **composants d'état (6)**. Ces valeurs restent inchangées entre deux changements adjacents. Un changement est composé d'événements simultanés.



STRUCTURE D'UN MODELE

Un événement correspond soit :

- A l'occurrence d'une action instantanée
- Au commencement d'une **action (7)**
- A la fin d'une action

Seul l'occurrence d'un événement peut induire un changement d'état. La valeur d'un état peut à un moment donné, dans une certaine vie toujours être dérivée des changements subis depuis son état initial. La spécification décrit un agent en définissant un ensemble de comportements admissibles **[DU BOIS, 95]**.

Le rôle de l'analyste est double

- Il définit le vocabulaire de son application (Qu'est-ce qu'un agent ? Quelle est la structure des états ? Quelles actions les agents peuvent-ils effectuer ?). Cette activité produit des déclarations qui peuvent être exprimées textuellement ou graphiquement grâce à l'éditeur Albert II.
- Il réduit l'ensemble des comportements admissibles en ajoutant des **contraintes (8)** sur le système. Ceci est effectué en utilisant des patrons textuels.

Six concepts fondamentaux ont été relevés dans cette description intuitive du langage. Dans le chapitre consacré au repository Albert II, nous constaterons que certains de ces concepts fondamentaux peuvent être raffinés et que de nouveaux concepts vont venir enrichir notre description intuitive du langage.

2.2 EXEMPLE D'UNE SPECIFICATION

L'exemple de spécification développé[1] est basé sur la **modélisation d'un contrôle aérien d'un aéroport (1)**. Un **radar (5)** a en charge la surveillance d'une piste. Chaque piste possède un radar. Le radar **avertit le moniteur (7)** de la présence d'un avion se rapprochant de la piste. Le moniteur est un outil informatique qui **avertit le contrôleur aérien (7)** sur la **situation de l'état des pistes (6)**. Le **contrôleur aérien (5)** prend les décisions nécessaires lorsque des situations risquent d'être dangereuses ou conflictuelles. Le **service de plan de vol (5)** à en charge la gestion de la base de données dans laquelle sont stockées toutes les informations relatives aux **routes suivies par les vols (6)**.

Spécification textuelle de la modélisation du contrôle aérien

```
(1) %SPEC Mgt_Controle_Aerien
    %BASIC TYPES
         | La position d'un avion (longitude,latitude)
        (2) POSITION
         | Il s'agit de l'altitude de vol
        (2) NIVEAU
        (2) DATE
        (2) FLIGHT
```

[1] N.B. Cet exemple a été exploité durant tout ce mémoire

```
| END_BASIC_TYPES
%CONSTRUCTED TYPES
     | La localisation d'un avion est caractrisée par sa
     | position ainsi que par son niveau
   (2)FLIGHT_LOCATION=CP[POSITION,NIVEAU]
     | Il s'agit de la localisation supposée d'un vol à une
     | certaine date
   (2)ESTIMATED_FLIGHT_LOCATION=CP[DATE,FLIGHT_LOCATION]
     | La trajectoire est composée d'une succession de
     | localisations
   (2)TRAJECTOIRE=SEQ[FLIGHT_LOCATION]
     | L'état de la situation autour d'une piste peut être normal
     | ou critique
   (2)ETAT=ENUM[Normal,Critique]
| END_CONSTRUCTED_TYPES
%OPERATIONS
     | Il y a danger lorsque deux trajectoires d'avions sont
     |convergentes
    (3)Danger:TRAJECTOIRE \x TRAJECTOIRE -> BOOLEAN
     | Chaque moniteur est en charge de la surveillance d'un
     | et un seul radar.
    (3)Surveille:Moniteur -> Radar; Surveille(m) = r\
          \with\
            (m1 <> m2 => Surveille(m1) <> Surveille(m2))\
| END_OPERATIONS
(4)%SOCIETY Mgt_Controle_Aerien
 | Le systme ERATO de contrôle aérien
    (Service_plan_de_vol)
    (Erato)
    (Radar)))
| END_SOCIETY
    (5)%AGENT Mgt_Controle_Aerien.Service_plan_de_vol
     | Il s'agit du service qui, à tout moment, s'occupe de la
     | gestion de l'ensemble des plans de vol
        %STATE COMPONENTS
         | Il s'agit de la base de données concernant l'ensemble
         | des informations relatives à la route suivie par un vol
        (6)Flight_Route \table-of ESTIMATED_FLIGHT_LOCATION
    \indexed-by FLIGHT  -> Mgt_Controle_Aerien.Erato.Moniteur
```

```
    | END_STATE_COMPONENTS
    | BASIC_CONSTRAINTS
    %DERIVED COMPONENTS
    %INITIAL VALUATION
    | DECLARATIVE CONSTRAINTS
    %STATE BEHAVIOUR
    %ACTION COMPOSITION
    %ACTION DURATION
    | OPERATIONAL CONSTRAINTS
    %PRECONDITIONS
    %EFFECTS OF ACTIONS
    %TRIGGERINGS
    | COOPERATION CONSTRAINTS
    %ACTION PERCEPTION
    %STATE PERCEPTION
    %ACTION INFORMATION
    %STATE INFORMATION
    | END_CONSTRAINTS
| END_AGENT
(4) %SOCIETY Mgt_Controle_Aerien.Erato
 | Le systme Erato est un système distribué où chaque
 | sous-système est associé à la surveillance d'un radar
 | (donc d'une piste)
   (Moniteur)
   (Controleur)
| END_SOCIETY
   (5) %AGENT Mgt_Controle_Aerien.Erato.Moniteur
    | Le moniteur est un outil informatique en charge de
    | fournir de l'information au contrôleur
      %STATE COMPONENTS
       | Il s'agit de l'état courant affiché au contrôleur.
       | Cet état reprend la trajectoire des vols suivis
       | (càd la dernière position enregistrée) ainsi que
       | les 9  (?) dernières positions repérées
       (6) Situation \table-of TRAJECTOIRE \indexed-by FLIGHT
           -> Mgt_Controle_Aerien.Erato.Controleur
       | Il s'agit d'une information donnée (normale ou
       | critique) donnée au contrôleur sur la situation en
       | cours
```

```
(6)Etat_situation \instance-of ETAT  ->
            Mgt_Controle_Aerien.Erato.Controleur
 | Situation considérée comme 'safe'
(6)Situation_safe \instance-of BOOLEAN
            \derived-from Situation
| END_STATE_COMPONENTS
%ACTIONS
 | Mise-à-jour de la situation sur base de la
 | réception d'une nouvelle information concernant un
 | vol
(7)*Maj_situation(FLIGHT_LOCATION)
 | Détection d'une anomalie concernant la route suivie
 | par un flight
(7)*Anomalie_route(FLIGHT,FLIGHT_LOCATION) ->
             Mgt_Controle_Aerien.Erato.Controleur
 | Processus de traitement associé à la réception
 | d'une information concernant un vol
(7)Traitement_message
 | Un signal d'alarme est envoyé dés lors que la
 | situation est critique depuis trop longtemps
(7)*Alarme -> Mgt_Controle_Aerien.Erato.Controleur
 | Déclenchement de  l'état critique
(7)*Enclen
 | Remise à l'état normal de l'état
(7)*Reset
| END_ACTIONS
| BASIC_CONSTRAINTS
%DERIVED COMPONENTS
 | Un conflit est détecté lorsque, dans la situation
 | courante, deux avions ont des trajectoires
 | dangereuses
(8)Situation_safe ==  In_dom(f1,Situation)
\and In_dom(f2,Situation)\\and f1<>f2 \and
\not Danger(Situation[f1],Situation[f2])
%INITIAL VALUATION
 | A l'état initial, la situation est considérée comme
 | normale
(8)Situation = Normal
| DECLARATIVE CONSTRAINTS
```

```
%STATE BEHAVIOUR
 | L'état de la situation est critique dés lors que,
 | pendant 5 secondes, la situation n'est pas 'safe'
(8)[]  \Lasted 5 sec: \not Situation_safe
      <=>  Etat_situation = Critique
%ACTION COMPOSITION
 | Le traitement d'un message consiste à mettre-à-jour
 | la situation à la suite de la réception d'une
 | information de vol en provenance du radar
(8)Traitement_message <-> rad.Message(fl)
      <> Maj_situation(fl)
%ACTION DURATION
 | Le traitement d'un message doit prendre moins de 2
 | secondes
(8)Traitement_message <= 2 sec
| OPERATIONAL CONSTRAINTS
%PRECONDITIONS
 | L'action Reset ne peut avoir lieu que si la
 | situation est à l'état Critique
(8)Reset: Etat_situation = Critique
 | L'action Enclenc ne peut avoir lieu que si la
 | situation est à l'état Normal
(8)Enclen: Etat_situation = Normal
%EFFECTS OF ACTIONS
 | Mise de l'état à la valeur normale
(8)Reset:\
          [ ]\
          Etat_situation := Normal
 | Mise de l'état à la valeur critique
(8)Enclen:\
      [ ]\
     Etat_situation := Critique
%TRIGGERINGS
 | Si la situation de l'état est critique depuis 30'
 | alors un signal d'alarme est envoyé
(8)Etat_situation = Critique // 30 sec -> Alarme
| COOPERATION CONSTRAINTS
%ACTION PERCEPTION
 | Le moniteur ne perçoit que les messages en
```

```
  | provenance du radar dont il assure la surveillance
       (8)\XK (r.Message(_) // Surveille(Self) = r)
       %STATE PERCEPTION
       %ACTION INFORMATION
       %STATE INFORMATION
        | A tout moment, le moniteur doit informer le
        | contrôleur de la situation
       (8)\XK(Situation.Controleur // TRUE)
       | END_CONSTRAINTS
  | END_AGENT
  (5)%AGENT Mgt_Controle_Aerien.Erato.Controleur
   | Le contrôleur est chargé des décisions à prendre en
   | cas de conflits et d'informations données par son
   | moniteur
      | BASIC_CONSTRAINTS
      %DERIVED COMPONENTS
      %INITIAL VALUATION
      | DECLARATIVE CONSTRAINTS
      %STATE BEHAVIOUR
      %ACTION COMPOSITION
      %ACTION DURATION
      | OPERATIONAL CONSTRAINTS
      %PRECONDITIONS
      %EFFECTS OF ACTIONS
      %TRIGGERINGS
      | COOPERATION CONSTRAINTS
      %ACTION PERCEPTION
      %STATE PERCEPTION
      %ACTION INFORMATION
      %STATE INFORMATION
      | END_CONSTRAINTS
  | END_AGENT
(5)%AGENT Mgt_Controle_Aerien.Radar
 | Un radar est associé à la surveillance d'une piste
   %ACTIONS
    | Le radar envoie aux moniteurs d'Erato un message
    | concernant la localisation dtecte d'un avion
   (7)*Message(FLIGHT_LOCATION)
            -> Mgt_Controle_Aerien.Erato.Moniteur
```

```
            | END_ACTIONS
            | BASIC_CONSTRAINTS
            %DERIVED COMPONENTS
            %INITIAL VALUATION
            | DECLARATIVE CONSTRAINTS
            %STATE BEHAVIOUR
            %ACTION COMPOSITION
            %ACTION DURATION
            | OPERATIONAL CONSTRAINTS
            %PRECONDITIONS
            %EFFECTS OF ACTIONS
            %TRIGGERINGS
            | COOPERATION CONSTRAINTS
            %ACTION PERCEPTION
            %STATE PERCEPTION
            %ACTION INFORMATION
            %STATE INFORMATION
            | END_CONSTRAINTS
        | END_AGENT
| END_SPEC
```

Remarque :

A la page douze, nous avons un exemple concret de la classification des contraintes en patron.

Copie d'une partie de la page douze

```
%EFFECTS OF ACTIONS
                | Mise de l'état à la valeur normale
               Reset:\
                         [ ]\
                         Etat_situation := Normal
                | Mise de l'état à la valeur critique
               Enclen:\
                     [ ]\
                    Etat_situation := Critique
```

Nous constatons que les contraintes sur des actions différentes : « *Reset* » et « *Enclen* » sont répertoriées dans le même patron de contraintes.

2.3 CAHIER DES CHARGES

A partir de la spécification textuelle du contrôle aérien, voici le document générer par notre application :

```
**************************************
* Spécification : Mgt_Controle_Aerien *(1)
**************************************

Liste des types définis par l'utilisateur(userType) :
=====================================================

        (2)POSITION
        La position d'un avion (longitude,latitude)

        (2)NIVEAU
        Il s'agit de l'altitude de vol

        (2)DATE


        (2)FLIGHT


        (2)FLIGHT_LOCATION
        La localisation d'un avion est caractrisée par sa position
        ainsi que par son niveau

        (2)ESTIMATED_FLIGHT_LOCATION
        Il s'agit de la localisation supposée d'un vol à une certaine
        date

        (2)TRAJECTOIRE
        La trajectoire est composée d'une succession de localisations

        (2)ETAT
        L'état de la situation autour d'une piste peut être normal ou
        critique
```

Liste des opérations définies par l'utilisateur(userOperation) :
==========

(3) Danger
Il y a danger lorsque deux trajectoires d'avions sont convergentes

(3) Surveille
Chaque moniteur est en charge de la surveillance d'un et un seul radar.

Société racine :
==========

(4) Mgt_Controle_Aerien

Société(Society): Erato
==========

(5) Agent : Moniteur

Le moniteur est un outil informatique en charge de fournir de l'information au contrôleur

Liste des Actions du Moniteur :
.....

(7) Action Maj_situation
Mise-à-jour de la situation sur base de la réception d'une nouvelle information concernant un vol

(7) Action Anomalie_route
Détection d'une anomalie concernant la route suivie par un flight

(7) Action Traitement_message
Processus de traitement associé à la réception d'une information concernant un vol

(8) contrainte de composition sur Traitement_message :
Le traitement d'un message consiste à mettre-à-jour la situation à la suite de la réception d'une information de vol en provenance du radar

(8) contrainte de durée(duration) sur Traitement_message :
Le traitement d'un message doit prendre moins de 2 secondes

(7) Action Alarme
Un signal d'alarme est envoyé dés lors que la situation est critique depuis trop longtemps

(8) contrainte de triggering sur Alarme :
Si la situation de l'état est critique depuis 30' alors un signal d'alarme est envoyé

(7) Action Enclen
Déclenchement de l'état critique

(8) contrainte de précondition(precondition) sur Enclen :
L'action Enclenc ne peut avoir lieu que si la situation est à l'état Normal

(8) contrainte d'effets(effect) sur Enclen :
Mise de l'état à la valeur critique

(7) Action Reset
Remise à l'état normal de l'état

(8) contrainte de précondition(precondition) sur Reset :
L'action Reset ne peut avoir lieu que si la situation est à l'état Critique

(8) contrainte d'effets(effect) sur Reset :
Mise de l'tat à la valeur normale

Liste des Composants d'états du Moniteur :
.....

(6) Composants d'état Situation
Il s'agit de l'état courant affiché au contrôleur. Cet état reprend la trajectoire des vols suivis (càd la dernière position enregistrée) ainsi que les 9 (?) dernières positions repérées

(8) contrainte de valeur initiale(initialValuation) sur Situation :
A l'état initial, la situation est considérée comme normale

(8) contrainte d'information sur Situation :
A tout moment, le moniteur doit informer le contrôleur de la situation

(6) Composants d'état Etat_situation
Il s'agit d'une information donnée (normale ou critique) donnée au contrôleur sur la situation en cours

(8) contrainte de comportement d'état(statebehaviour) :
L'état de la situation est critique dés lors que, pendant 5 secondes, la situation n'est pas 'safe'

(6) Composants d'état Situation_safe
Situation considérée comme 'safe'

(8) contrainte de dérivation(dérivation) sur Situation_safe :
Un conflit est détecté lorsque, dans la situation courante, deux avions ont des trajectoires dangereuses

(8) contrainte de comportement d'état(statebehaviour) :
L'état de la situation est critique dés lors que, pendant 5 secondes, la situation n'est pas 'safe'

(5) Agent : Controleur

Le contrôleur est chargé des décisions à prendre en cas de conflits et d'informations données par son moniteur

fin de la description de la sociétéErato_

(5) Agent : Service_plan_de_vol

Il s'agit du service qui, à tout moment, s'occupe de la gestion de l'ensemble des plans de vol

Liste des Composants d'états du Service_plan_de_vol :
.....

(6) Composants d'état Flight_Route
Il s'agit de la base de données concernant l'ensemble des informations relatives à la route suivie par un vol

(5)Agent : Radar

Un radar est associé à la surveillance d'une piste

Liste des Actions du Radar :

.....

(7)Action Message

Le radar envoie aux moniteurs d'Erato un message concernant la localisation détectée d'un avion

(8)contrainte de perception sur Message :
Le moniteur ne percoit que les messages en provenance du radar dont il assure la surveillance

Remarque :

Aux pages dix-sept et dix-huit, nous avons un exemple concret du regroupement d'informations autour d'un concept.

Copie d'une partie des pages dix-sept et dix-huit

Action Enclen
Déclenchement de l'état critique

contrainte de précondition(precondition) sur Enclen :
L'action Enclenc ne peut avoir lieu que si la situation est à l'état Normal

contrainte d'effets(effect) sur Enclen :
Mise de l'état à la valeur critique

Nous constatons que l'action « *Enclen* » est liée à deux contraintes :

- L'une de précondition
- L'autre d'effet d'actions.

Ceci permet une lecture plus aisée du cahier des charges[1].

[1] Voir remarque page quatorze

CHAPITRE III

TELOS & MINITELOS

CHAPITRE III

TELOS & MINITELOS

Durant tout ce chapitre, nous expliquerons les différents concepts fondamentaux de Telos. Ces concepts seront accompagnés d'un exemple basé sur la modélisation du concept de spécification d'Albert II.

Exemple :

Tous objets Albert II possèdent un nom et une description. Une spécification Albert II est composée d'une société racine, des types de données, des opérations sur ces types de données et des commentaires définis par un analyste (utilisateur).

Une spécification Albert II est un objet utilisateur (Albert II).

C'est sur le principe de Telos que le repository d'Albert II a été construit. Après une description générale de Telos et de ces principaux concepts, on analysera plus en détail mini-Telos qui lui est utilisé réellement pour construireet peupler le repository.

3.1 INTRODUCTION

Telos est un langage qui tente de supporter le développement des systèmes d'informations. Le langage est approprié pour représenter le savoir au niveau d'une variété de mondes en rapport avec un système d'informations **[MYLOPOULOS]** :

- Le monde du sujet (domaine d'application). Il correspond au domaine dans lequel les informations sont stockées (il a le même rôle que celui des tables de bases de données relationnelles).
- Le monde du système (conception). Il inclut les spécifications à différents niveaux de détails d'implémentation, la nature et le nombre de niveaux de spécification dépendent de la méthodologie adoptée. Par exemple : les niveaux peuvent inclure les besoins fonctionnels, la conception et l'implémentation. Pour chaque niveau, des concepts appropriés doivent être définis et faire partie du méta-modèle du monde du système
- Le monde du traitement (les modèles d'utilisateur, l'environnement). Il décrit l'environnement dans lequel le système est inclus. De telles descriptions prennent la

forme de relations d'entrée – sortie qui peuvent être incluses dans des classes différentes d'utilisateur ou des interfaces supportées par le système. Il peut également inclure une description de l'environnement dans lequel le système doit fonctionner.

- Le monde du développement (équipe et méthodologie). Il est focalisé sur les entités et activités qui prennent part au processus de conception lui-même. Cela peut inclure la formation d'une équipe de concepteurs, des décisions au niveau de la conception, le développement d'outils.

Remarque : actuellement seul les deux premiers mondes sont représentés dans le repository Albert II

3.2 DEFINITION DE BASE

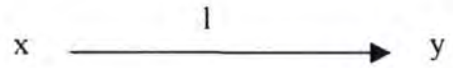
Une base de connaissance Telos est constituée d'une structure d'objets construits sur base de deux idées simples : les individus et les attributs **[JARKE, 96]**.

Les individus représentent des entités (qui peuvent être concrètes ou abstraites) tandis que les attributs représentent une relation binaire entre des entités ou d'autres relations.

Dans Telos, le mécanisme de structuration d'une base de connaissance traite les individus et les attributs de la même manière. Ils sont désignés par le terme «proposition». Ceci est une des principales caractéristiques de Telos.

Une base de connaissance Telos est un ensemble fini de propositions interreliées

$$KB = \{proposition(oid, x, l, y, tt) \mid x, y, tt \in ID, \& \in LABEL\}$$



ou

oid est la clef d'accès dans la base de connaissance

ID est un ensemble non vide d'identifiants.

LABEL est un sous-ensemble non vide de noms

Le composant «oid» est appelé identifiant d'une proposition

Le composant «x» est appelé source d'une proposition

Le composant «l» est appelé le label d'une proposition

Le composant «y» est appelé destination d'une proposition

Le composant «tt» est appelé temps de croyance

Nous pouvons interpréter **proposition(oid, x, l, y, tt)** :

L'objet «x» a une relation appelé «l» avec l'objet «y». Cette relation est acceptée par le système au temps «tt».

Plusieurs mécanismes de structuration existent dans le langage Telos **[MYLOPOULOS]**:

- L'agrégation. Elle consiste en un ensemble d'attributs qui ont en commun une proposition comme source.
- La généralisation/la spécialisation. Une classe peut être spécialisée suivant une certaine hiérarchie.
- La classification. Elle peut être considérée comme une instance d'une ou plusieurs propositions génériques ou classe. Les classes sont elles-mêmes des propositions.

Le principe de ce langage est basé sur la représentation du savoir. Telos supporte trois types de représentations différentes.

- Une représentation logique
- Une représentation en frame
- Une représentation graphique

La représentation graphique et la représentation en frame découlent de la représentation logique.

On distingue quatre patrons de propositions dont voici les noms

- **Individuals**

  proposition (oid, oid, l, oid, tt)

  (« oid est un objet avec le nom l accepté en tt »)

- **InstanceOf**

  proposition (oid, x, *instanceof, y, tt)

  (« x est une instance de classe y acceptée en tt »)

- **IsA**

  proposition (oid, x, *isa, y, tt) and

  (« x est une spécialisation d'y acceptée en tt »)

- **Attributes**

  Toutes les autres propositions

3.3 AXIOME

Telos impose quelques axiomes sur la base de connaissance.

Pour garantir une correspondance unique, on doit suivre les axiomes suivants **[JARKE, 96]** :

Le label d'un objet individu doit être unique dans la base de connaissance.

Le Label d'un attribut doit être unique dans toutes les propositions de type Attribute qui ont un objet source en commun.

Exemple de plusieurs frames Telos

Tous objets Albert II possèdent un nom et une description.

```
AlbertUserObject in AlbertClass with
       attribute
                      name :  String ;
                      description :  String
end
```

Une spécification Albert II est composée d'une société racine, des types de données, des opérations sur ces types de données et des commentaires définis par un analyste (utilisateur).

Une spécification Albert II est un objet utilisateur (Albert II).

```
AlbertSpecification in AlbertClass isa AlbertUserObject with
attribute
              rootAgentSociety : AlberAgentOrSociety ;
              userComment : AlbertPostIt ;
              userType : AlbertUserDefinedType ;
              userOperation : AlbertOperation
end
```

Le label de la source commune dans la première frame est AlbertUserObject. Il est déclaré comme une instance de la classe AlbertClass et possède deux attributs. Les «oid» précédés par # sont générés par le système. Cette frame correspond à un ensemble de propositions :

```
proposition (#AlbertUserObject,#AlbertUserObject,
             AlbertUserObject,#AlbertUserObject)

proposition (#1,#AlbertUserObject,
             *instanceof,#AlbertClass)

proposition (#2,#AlbertUserObject,
             name,#String)

proposition (#3,#AlbertUserObject,
             description,#String)

proposition (#AlbertSpecification,#AlbertSpecification,
             #AlbertSpecification,#AlbertSpecification)

proposition (#4,#AlbertSpecification,
             *instanceof,#AlbertClass)

proposition (#5,#AlbertSpecification,
             *isa,#AlbertUserObject)

proposition (#6,#AlbertSpecification,
             rootAgentSociety,#AlbertAgentOrSociety)

proposition (#7,#AlbertSpecification,
             userComment,#AlbertPostIt)

proposition (#8,#AlbertSpecification,
             userType,#AlbertUserDefinedType)

proposition (#9,#AlbertSpecification,
             userOperation,#AlbertOperation)
```

L'instantiation de la classe individu est donnée implicitement par la structure des deux propositions de type individu : `#AlbertUserObject` et `#AlbertSpecification`. De manière analogue, les attributs `#2`, `#3`, `#6`, `#7`, `#8`, `#9` sont automatiquement considérés comme des instances de la classe **Attribut**.
Les propositions `#1` et `#4` sont des instances de la classe **InstanceOf** et `#5` est une instance de la classe **IsA**

Voici un exemple d'instantiation d'AlbertSpecification

```
Albert_Spec_Mgt_Controle_Aerien in AlbertSpecification with
   name
        _AUTO_424 : "Mgt_Controle_Aerien"

userType
        _AUTO_433 : Albert_Type_Mgt_Controle_Aerien/POSITION ;
        _AUTO_450 : Albert_Type_Mgt_Controle_Aerien/NIVEAU ;
        _AUTO_467 : Albert_Type_Mgt_Controle_Aerien/DATE ;
        _AUTO_484 : Albert_Type_Mgt_Controle_Aerien/FLIGHT ;
        _AUTO_501 : Albert_Type_Mgt_Controle_Aerien/FLIGHT_LOCATION ;
        _AUTO_504 : Albert_Type_Mgt_Controle_Aerien
                              /ESTIMATED_FLIGHT_LOCATION ;
        _AUTO_507 : Albert_Type_Mgt_Controle_Aerien/TRAJECTOIRE ;
        _AUTO_510 : Albert_Type_Mgt_Controle_Aerien/ETAT

rootAgentSociety

        _AUTO_512 : Albert_Society_Mgt_Controle_Aerien
                              /Mgt_Controle_Aerien[1]
userOperation

        _AUTO_676 : Albert_UserOp_Mgt_Controle_Aerien/Danger ;
        _AUTO_680 : Albert_UserOp_Mgt_Controle_Aerien/Surveille
end
```

L'attribut **name** doit être défini dans une des classes d' Albert_Spec_Mgt_Contrôle

_Aerien. Dans notre cas, Albert_Spec_Mgt_Contrôle_Aerien est une instance d'AlbertSpecification et également une instance d'AlbertUserObject. Ceci est dû à l'axiome qui définit l'héritage des membres d'une classe en Telos :

La destination d'une spécialisation hérite de toutes les instances de sa source.

Un exemple est la spécialisation #5 : toutes les instances d'AlbertSpecification sont aussi des instances d'AlbertUserOjbect.
Telos force le typage des valeurs des attributs par l'axiome suivant :

Si « p » est une proposition qui est une instance de la proposition « P » alors la source de « p » doit être une instance de la source de « P » et la destination de « p » doit être une instance de la destination de « P ».

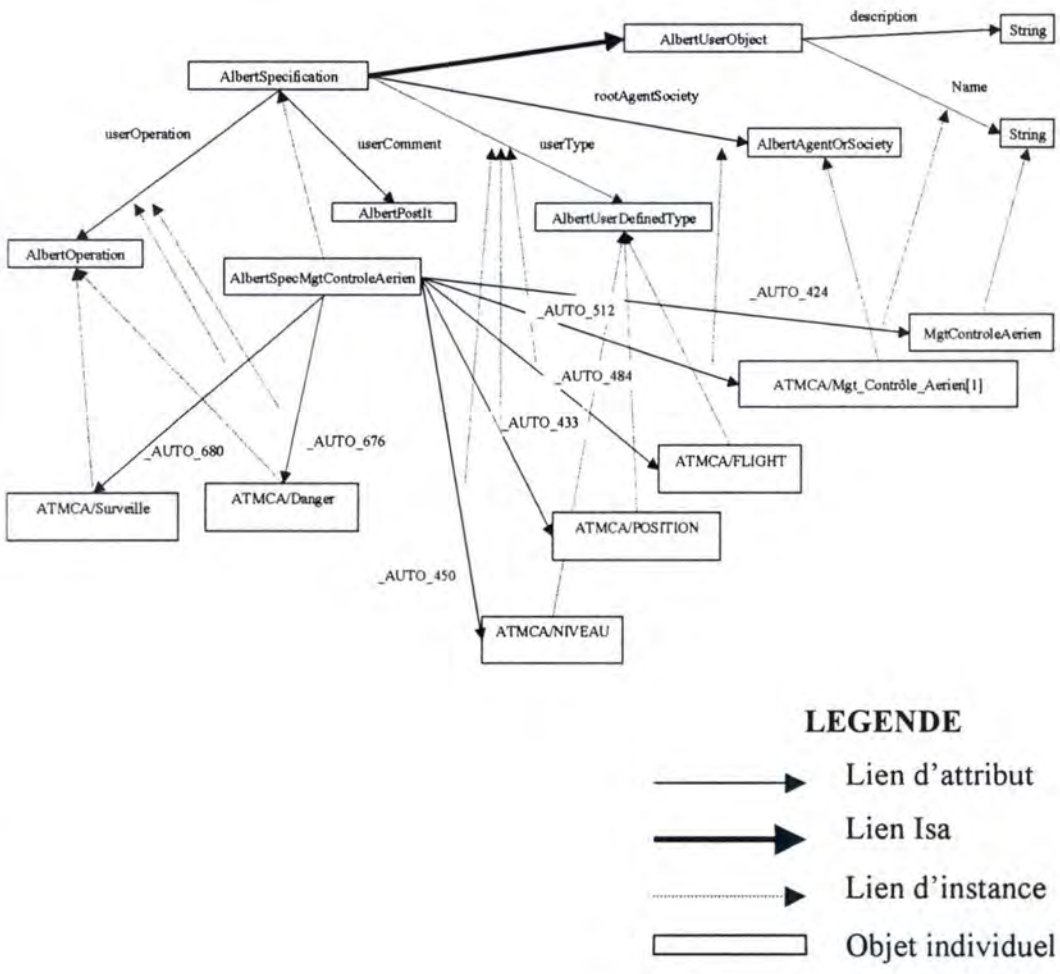
Par exemple «Mgt_Contrôle_Aerien» doit être une instance d'un **String.** L'individu Albert_Spec_Mgt_Contrôle_Aerien montre aussi d'autres caractéristiques : les attributs spécifiés au niveau d'une classe ne doivent pas nécessairement être instantiés. C'est le cas pour l'attribut description d'AlbertUserObject et l'attribut AlbertPostIt d'AlbertSpecification. D'un autre côté, il est possible d'obtenir plusieurs instances d'un même attribut. Ceci peut être observé dans notre exemple pour les attributs userType et useroperation. Dans certaines situations, les occurrences d'attributs d'une frame qui sont instantiés ne savent pas être déterminées. Ce problème de multiples généralisations et d'instantiations est résolu en suivant la condition suivante (respectée par la base de connaissance Telos).

Si p_1 et p_2 sont des attributs de deux classes c_1 et c_2 qui ont le même label l et i est une instance commune de c_1 et c_2 qui a un attribut de la catégorie l alors il doit exister une spécialisation commune c_3 de c_1 et c_2 avec un attribut l qui spécialise p_1 et p_2 et i est une instance de c_3.

Telos traite chacune des trois sortes de relation comme des objets. Donc chaque lien d'attribut d'instantiation ou de généralisation d'AlbertSpecification peut avoir ses propres attributs et instances. La syntaxe d'un objet attribut est la suivante : on concatène le nom d'un individu avec un point d'exclamation suivi du label d'un attribut. La relation entre name et _Auto_424 peut être exprimée comme :

```
Attribute Albert_Spec_Mgt_Controle_Aerien!_Auto_433_ in
                                    AlbertSpecification!userType
end
```

Voici sous forme graphique une Albertspecification et son instance.



3.4 Le Sous Langage Predicatif CBL

CBL est utilisé pour exprimer des contraintes d'intégrité des règles de déduction et des requêtes. Les variables à l'intérieur des formules doivent être quantifiées et assimilées à un type qui limite les instantiations possibles d'un ensemble de classes. Un ensemble de littérales a été défini pour le langage **[JARKE, 95]**.

- (x in c) ⇔ proposition (o , x , *in , c , ...)
  L'objet x est une instance de la classe c

- (c isA d) ⇔ proposition (o , c , *isa , d , ...)
  L'objet c est une spécialisation (sous-classe) de d

- (x l y) ⇔ proposition (o , x , l, y , ...)
  L'objet x a un lien avec l'objet y de nom l

- x < y , x > y , x <= y , x > = y , x <> y
  On remarque que x et y doivent être des instances de **Integer** ou **Real**

- x = = y
  Les objets x et y doivent être les mêmes

Restriction Sur Les Formules

Chaque constante(un argument qui n'est pas une variable) dans une formule « f » doit être le nom d'un objet existant dans la base de connaissance Telos ou alors c'est une constante d'une classe construite « integer, real ou string ».

Voici un exemple d'une frame Telos comprenant une règle de déduction et une contrainte d'intégrité.

```
Individual AlbertStateComponent in AlbertClass
                                    isA AlbertUserObject with
        attribute
              elementType : AlbertType ;
              isFixed : Boolean ;
```

```
            isDerivedFrom : AlbertStateComponent ;
            exportsTo : AlbertAgent ;
            derivation : AlbertDerivation ;
            initialValuation : AlbertInitialValuation ;
        rule // règle de déduction
           derivationsTCRule :
       $forall s1, s2/AlbertStateComponent ((s1 isDerivedFrom s2)
        or exists s3/AlbertStateComponent (s1 isDerivedFrom s3)
        and (s3 derivationsTC s2)) ==> (s1 derivationsTC s2) $

        constraint // contrainte d'intégrité
         noDerivationCircularity :
               $forall s/AlbertStateComponent not(s derivationsTC s)$;
         derivationInAgent :
               $forall a1,a2/AlbertAgent s1,s2/AlbertStateComponent
              (a1 state s1) and (a2 state s2) and(s1 derivationsTC s2)
                                        ==> (a1 == a2) $
end
```

3.5 LANGAGE DE REQUETES CBQL

Les requêtes sont définies comme une classe spéciale où les instances sont les résultats des requêtes **[JARKE, 96]**. Les requêtes sont des instances de la classe système QueryCLass

```
Individual QueryClass in Class isA Class with
       attribute
              retrieved_attribute : Proposition ;
              computed_attribute : Proposition
       attribute,single
              constraint : MSFOLquery
end QueryClass
```

Le retrived_attribute est comparable à une projection en algèbre relationnelle.
Le computed_attribute est un attribut qui est calculé à partir des relations existantes avec d'autres objets.

Les requêtes peuvent être réutilisées :

- Par héritage

```
GenericQueryClass getMgtControlAerien isa getAlbSpec
retrieved_attribute
              name : String
              description : String
       constraint
              mgtcontrolAerienconst :$AlbertSpecMgtControleAerien in
                                                          AlbertSociety
end getMgtControlAerien
```

- En tant que classe d'attribut

```
GenericQueryClass getaction isa AlbertAction
       retrieved_attribute
              name : String
              description : String
       computed_attribut,parameter
              agent : getAgOrSoc
       constraint
              actionconst :$ agent action this $
end getaction
```

- Dans la partie contrainte d'une requête

```
GenericQueryClass getinitialValuation isa AlbertInitialValuation
       retrieved_attribute
              description : String
       computed_attribut,parameter
              agent : AlbertAgent
       constraint
              initialValuationconst : $ exists st/getstateComponent
                                    (( agent stateComponent st )and
                                    ( st initialValuation this )) $
end getinitialValuation
```

- Récursivement

```
GenericQueryClass getAgOrSoc isa AlbertAgent , AlbertSociety
        retrieved_attribute
                name : String
                description : String
        computed_attribut,parameter
                rootsoc : AlbertSociety
        constraint
                AgOrSocconst :$ ( rootsoc containedAgentOrSociety this )
                                or exists soc/AlbertSociety
                                ( soc in getAgOrSoc ) and
                                ( soc containedAgentOrSociety this ) $
end getAgOrSoc
```

3.6 MINITELOS

MiniTelos[1] est un langage basé sur les principaux concepts de Telos, c'est à dire les concepts de propositions. Il n'inclut pas les langages CBL et CBQL. Le repository Albert II est créé et peuplé à l'aide des classes miniTelos.
MiniTelos est implémanté en Java et voici sa hiérarchie :

class java.lang.Objet

- classBE.ac.fundp.info.Telo.Multiplicity
- class java.util.Observable
  - class BE.ac.fundp.info.Telos.TelosBase
- class BE.ac.fundp.info.TelosObject
  - class BE.ac.fundp.info Telos.TelosBinaryRelation
    - class BE.ac.fundp.info.Telos.TelosAttribute
    - class BE.ac.fundp.info.Telos.TelosIn
    - class BE.ac.fundp.info.Telos.TelosIsa
  - class BE.ac.fundp.info.Telos.TelosIndividual
    - classBE.ac.fundp.info.Telos.TelosToken
    - classBE.ac.fundp.info.Telos.TelosTokenClass
- class java.lang.Throwable
  - class java.lang.Exception
    - class BE.ac.fundp.info.Telos.TelosException
- class java.util.Vector
  - class BE.ac.fundp.info.Telos.TelosVector

MUTIPLICITY

La classe *Mutiplicity* est utilisée pour contraindre le nombre d'instances d'un attribut d'un objet. Il n'est pas nécessaire d'entrer davantage dans les détails étant donné que cette classe n'est pas utilisée dans notre application.

---

[1] Pour plus d'information sur les classes de miniTelos : voir annexes « MiniTelos ».

TELOSBASE

La classe *TelosBase* permet de créer une base de connaissance Telos. Elle permet de créer et d'accéder aux différents objets enregistrés dans la base de connaissances. Seule la méthode *getIndividual* est utilisée par notre application. Cette méthode permet de retrouver un objet individu à partir d'un label donné. Elle renvoie le *TelosObject* correspondant

Exemple :

```
base.getIndividual (« AlbertSpecification ») base étant une base de connaissances
donne
proposition («#AlbertSpecification, #AlbertSpecification,
        « AlbertSpecification »,# AlbertSpecification)
```

D'autres méthodes intéressantes existent :

- *saveFrames* : cette méthode permet de sauver des frames à la Telos, ceci permet de garder une certaine compatibilité entre miniTelos et les autres outils basés sur Telos (ConceptBase)
- *doValidation* : valide la base de connaissance

Ces méthodes ne sont pas utilisées par notre application.

TELOSOBJECT

Un objet Telos est rattaché à une base de connaissance grâce à un champ *m_TelosBase* (qui est de type de *TelosBase*). Un objet *TelosObject* représente une proposition générique. Les méthodes utilisées par l'application sont :

- **TelosObject** *getAttibutes*(**String strAttr**) Cette méthode permet d'obtenir la proposition correspondante au nom spécifié en paramètre. Elle renvoie le *TelosObject* correspondant

Exemple :

```
to.getAttributes («userOperation »)
to étant proposition (#AlbertSpecification,#AlbertSpecification,
        « AlbertSpecification »,#AlbertSpecification)
donne
proposition (# ? ? ?,#AlbertSpecification, «userOperation »,#AlbertOperation )
```

- **TelosVector** *getInstances*() Cette méthode permet d'obtenir une liste de toutes les instances directes d'un objet Telos. Elle renvoie un *TelosVector* qui correspond à la liste.

Exemple :

```
to.getInstances()
 to étant proposition (#AlbertSpecification,#AlbertSpecification,
          « AlbertSpecification »,#AlbertSpecification)
donne
proposition (#ASMgtControleAerien, # ASMgtControleAerien,
    « ASMgtControleAerien »,# ASMgtControleAerien
```

- **String** *getName*() Cette méthode donne le nom de l'objet Telos.

Exemple :

```
to.getName ()
to étant la proposition (#AlbertSpecification,#AlbertSpecification,
« AlbertSpecification »,#AlbertSpecification)
donne
« AlbertSpecification »
```

- **TelosObject** *getDestination*() Cette méthode donne la destination d'un objet Telos.

Exemple :

```
to.getDestination()
to étant la proposition (# ? ? ?,#AlbertSpecification,
     «userOperation »,#AlbertOperation )
donne
proposition (#AlbertOperation,#AlbertOperation,
    « AlbertOperation »,#AlbertOperation)
```

- **TelosObject** *getSource*() Cette méthode donne la source d'un objet Telos.

Exemple :

```
to.getSource()
to étant la proposition (# ? ? ?,#AlbertSpecification,
    «userOperation »,#AlbertOperation )
donne
proposition (#AlbertSpecification, #AlbertSpecification,
        « AlbertSpecification »,#AlbertSpecification)
```

TELOSBINNARYRELATION

La classe *TelosBinnaryRelation* est une spécialisation de *TelosObject*. Elle représente toutes les propositions qui ne sont pas de type individu. Elles sont utilisées pour mettre en relation deux objets *TelosObject* : ces deux objets peuvent être deux individus, un individu avec un objet *TelosBinnaryRelation*.

Les méthodes utilisées sont :

- **TelosVector** *getInstancesWithSource*(**TelosObject theSource**) Cette méthode permet de déterminer toutes les instances d'un objet Telos ayant comme source l'autre objet Telos. Elle renvoie une liste d'instances sous la forme d'un *TelosVector*.

Exemple :

```
to1.getInstancesWithSource(to2)
to1 étant proposition (# ? ? ?,#AlbertSpecification,
    «userOperation »,#AlbertOperation )
to2 étant proposition(#ASMgtControleAerien,#ASMgtControleAerien,
    « ASMgtControleAerien »,#ASMgtControleAerien)
donne
proposition (# ? ? ?,#ASMgtControleAerien,
    « AUTO_676 »,#ATMCA/Danger)
proposition (# ? ? ?,#ASMgtControleAerien,
    « AUTO_680 »,#ATMCA/Surveille)
```

- **TelosVector** *getInstancesWithDestination*(**TelosObject**) Cette méthode permet de déterminer toutes les instances d'un objet Telos ayant pour destination l'autre objet Telos. Elle renvoie une liste d'instances sous la forme d'un *TelosVector*.

Les trois spécialisations de cette classe sont :

- TelosAttribute
- TelosIsa
- TelosIn

TELOSINDIVIDUAL

TelosIndividual est une autre spécification de la classe TelosObject. Elle correspond aux propositions qui sont de type individu. Ce qui inclut les propositions de type TelosTokenClass.

TELOSEXCEPTION

Elle correspond à tous les types d'exceptions spécifiques aux classes Telos. Elle signale une condition anormale qui doit être traitée de façon particulière pour éviter la fin prématurée d'un programme. Les exceptions peuvent être capturées ou gérées.

TELOSVECTOR

Elle représente un tableau d'objets Telos. Cette classe est souvent utilisée par d'autres méthodes pour retourner une liste de valeurs

Exemple :

TelosVector getInstances()

Pour connaître la valeur d'un élément d'un TelosVector, on utilise la méthode getAt.

3.7 CONCLUSION

MiniTelos reprend les concepts de base Telos : propositions, liens entre les propositions (TelosObject), principes de regroupement (TelosIn), d'attributs (TelosAttribute) et le mécanisme d'héritage (TelosIsa).

Par contre des concepts plus élaborés tel que : requêtes, règles et contraintes ne sont pas implémantés dans miniTelos. Ceci va influencer l'architecture de l'application ce qui a pour conséquence une obligation de travailler à un niveau très bas (niveau propositions).

CHAPITRE IV

DESCRIPTION DU REPOSITORY ALBERT II

CHAPITRE IV

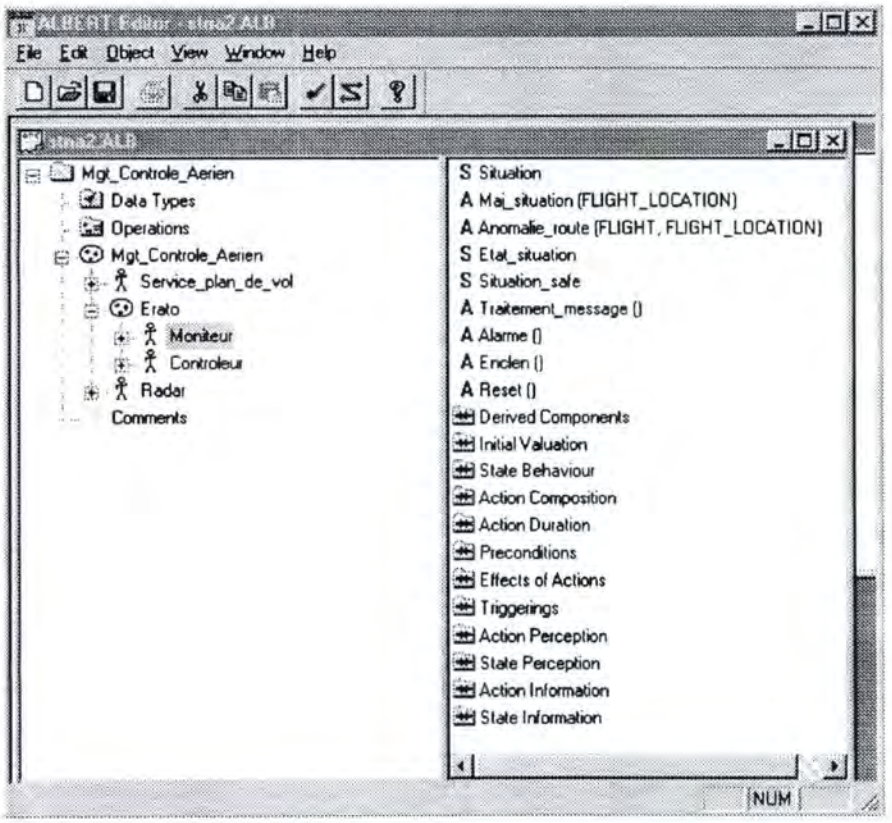
DESCRIPTION DU REPOSITORY D'ALBERT II

Pour rappel, l'objectif du mémoire est de construire un cahier des charges à partir d'une spécification Albert II. Seuls les concepts pouvant posséder une description nous serons utiles pour rédiger un cahier des charges. Ce chapitre explique les concepts d'Albert II modélisés dans le repository.

4.1 REPOSITORY

Lorsque l'on analyse la syntaxe du langage Albert II et son éditeur on constate qu'il est possible de donner une description à plusieurs éléments de la spécification. Cette description est énoncée dans un langage naturel.

COPIE D'ECRAN DE L'EDITEUR ALBERT

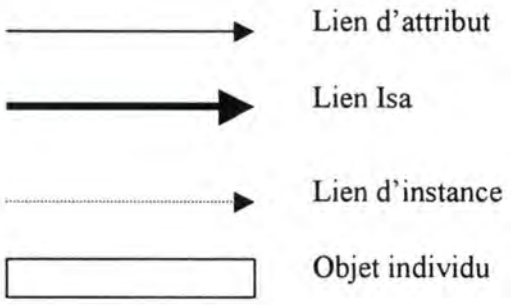


Voici la liste de ses concepts :

- Spécification
  - ✓ Opérations
  - ✓ Types
  - ✓ Sociétés
    - ❖ Agents
      - ❑ Actions
      - ❑ Composants d'état
      - ❑ Contraintes
        - ♦ Dérivations
        - ♦ Valeur initiale
        - ♦ Comportement de l'état
        - ♦ Composition
        - ♦ Durée
        - ♦ Précondition
        - ♦ Effets
        - ♦ Triggering
        - ♦ Perception d'action
        - ♦ Perception d'un état
        - ♦ Information sur une action
        - ♦ Information sur un état

Pour chaque concept modélisé dans le repository, on donne sa version en « frame » et sa version graphique, dans laquelle on trouve des exemples d'instantiation extraits de la spécification du contrôleur aérien.

Légende des graphiques



OBJET GENERIQUE ALBERT

Tous les concepts modélisés héritent des caractéristiques de l'objet *AlbertUserObject*.
Cet objet possède un nom **(a)**[1] et une description **(b)**. Le contenu du nom et de la description nous sera utile pour rédiger notre cahier des charges.

```
AlbertUserObject in AlbertClass UserObject with
        attribute
                        (a) name : String ;
                        (b) description : String
end
```

SPECIFICATION

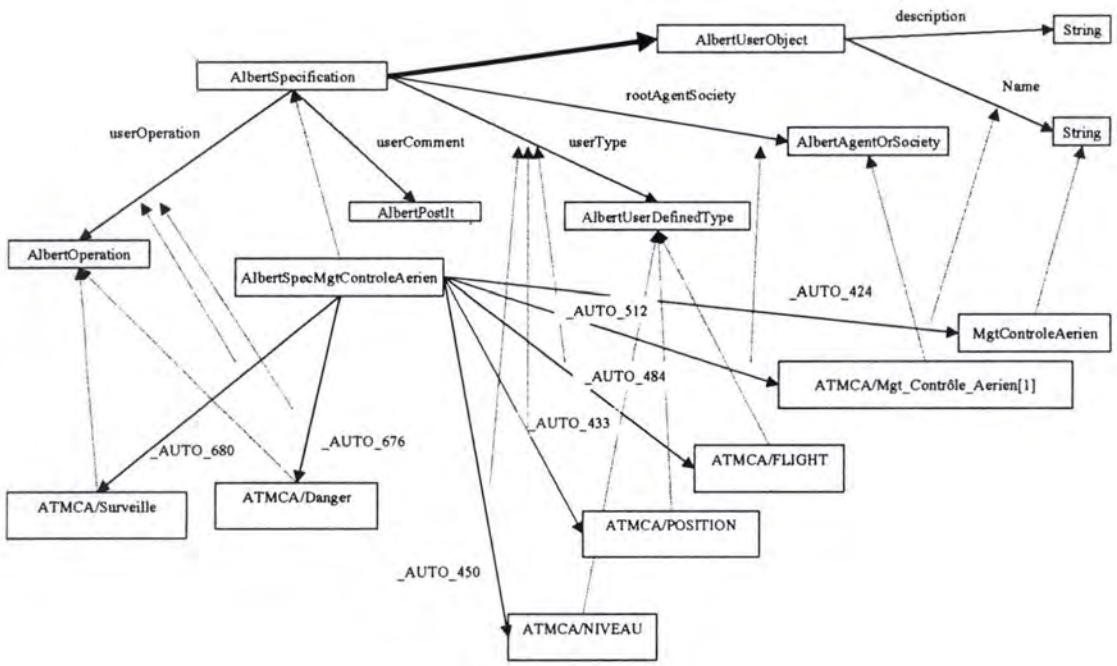
Une spécification **(a)** est composée de plusieurs sections.

- La définition d'une société **(b)**. La manière dont les agents se groupent pour former une hiérarchie.
- Dans l'éditeur Albert II, l'analyste a la possibilité de coller des post-it **(c)** électroniques sur un agent, une contrainte, une action....
- La définition de type de données **(d)**.
- Les opérations pouvant être effectuées sur ce type de données **(e)**.

Une spécification est composée d'au moins une société. Il doit y avoir au moins un agent dans chaque société si la société « mère » ne possède pas de société « fille » **[DU BOIS, 97b]**.

[1] Des lettres sont placées dans des (), en vue d'établir le lien entre le texte et la partie modélisée.

```
(a) AlbertSpecification in AlbertClass isa AlbertUserObject with
        attribute
                (b) rootAgentSociety : AlbertAgentOrSociety ;
                (c) userComment : AlbertPostIt ;
                (d) userType : AlbertUserDefinedType ;
                (e) userOperation : AlbertOperation
end
```



TYPES DE DONNEES

Il existe deux types de données **(a)** : les types de base et les types composés.

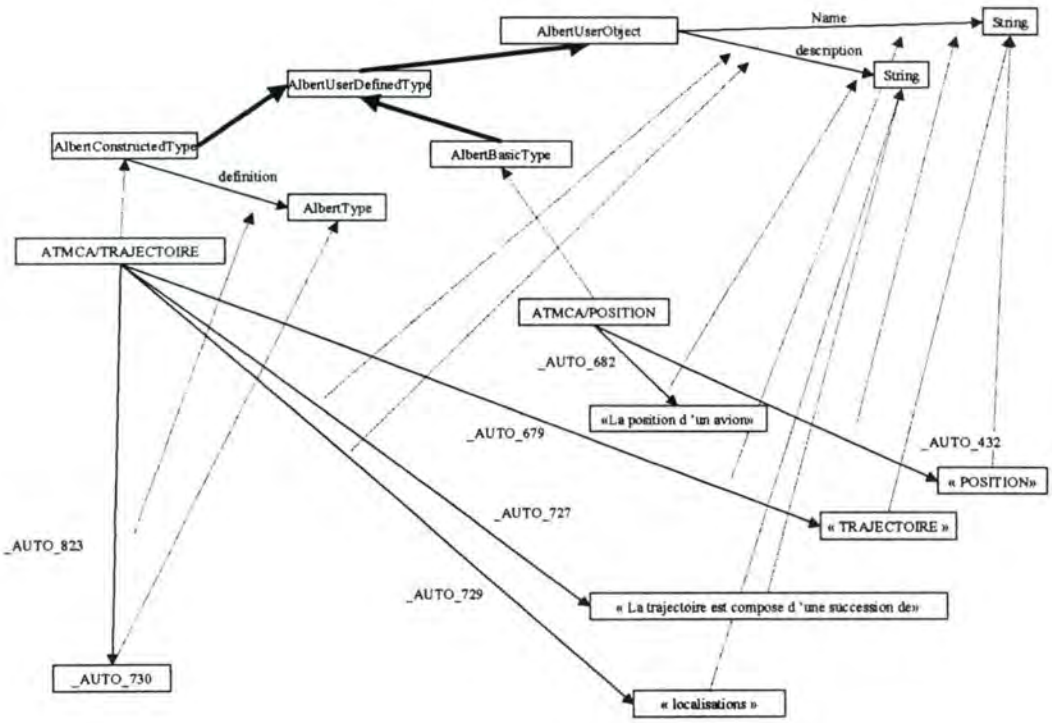
La déclaration des types de bases **(b)** est définie par l'analyste. Quelques types de base sont prédéfinis dans le langage. Les types de bases prédéfinis sont : BOOLEAN, CHAR, STRING, INTEGER, RATIONAL, DURATON.

L'analyste peut définir une structure des types composés**(c)**. Les constructeurs utilisés pour construire des expressions de types composés sont : CP, SET, BAG, UNION, ENUM

```
(a) AlbertUserDefinedType in AlbertClass isa AlbertUserType with
end

(b) AlbertBasicType in AlbertClass isa AlbertUserObject with
end

(c) AlbertConstructedType in AlbertClass isa AlbertUserDefinedType with
              attribute
                              definition : AlbertType
end
```



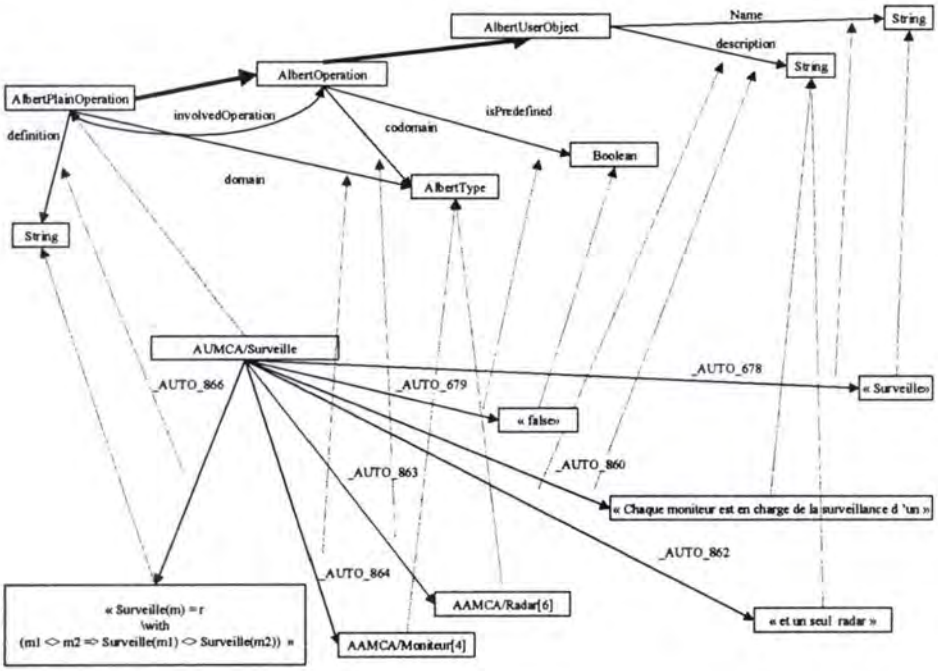
OPERATIONS SUR LES TYPES DE DONNEES

Il existe des opérations prédéfinies**(a)** sur les types de données prédéfinies :

tel que =, <, >, ...

Les opérations **(b)** sur les données ne doivent pas être confondues avec les actions des agents. Les opérations sont seulement des fonctions mathématiques. Elles peuvent être utilisées pour simplifier les expressions dans des contraintes mais ne servent pas à modéliser des comportements dynamiques du système **[DU BOIS, 95]**. Le codomain **(c)** correspond au résultat retourner par l'opération.

```
(b) AlbertOperation in AlbertClass isa AlbertUserObject with
        attribute
                (c) codomain : AlbertType ;
                (a) isPredefined : Boolean
end
```



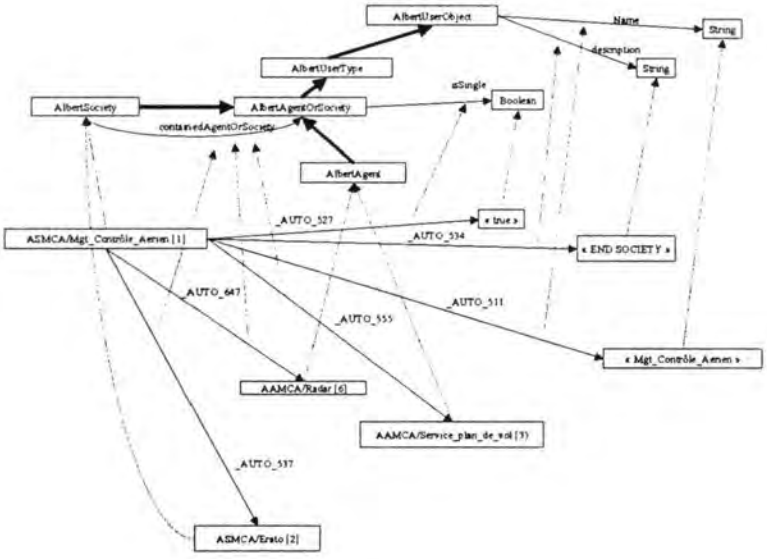
DEFINITION DES SOCIETES

Dans une spécification Albert II les agents sont groupés dans des sociétés **(a)** qui sont utilisées pour structurer une spécification. Leur comportement est déduit de la spécification des agents qui font partis de celle-ci. Ainsi la spécification associée à une société consiste

seulement en une liste de sous systèmes **(b)** qui la compose**(c)**. Chacun de ces sous systèmes peut être une autre société, un agent ou une classe **(d)** d'agents ou de société (un ensemble d'agent avec la même spécification) **[DU BOIS, 95]**. Les soci étés et agents sont considérés comme des types de données **(e)** dans le repository.

```
(a) AlbertSociety in AlbertClass isa AlbertAgentOrSociety with
        attribute
                (c) containedAgentOrSociety : AlbertAgentOrSociety
end

(b) AlbertAgentOrSociety in AlbertClass isa (e) AlbertUserType with
        attribute
                (d) isSingle : boolean ;
end
```



DEFINITION D'AGENT

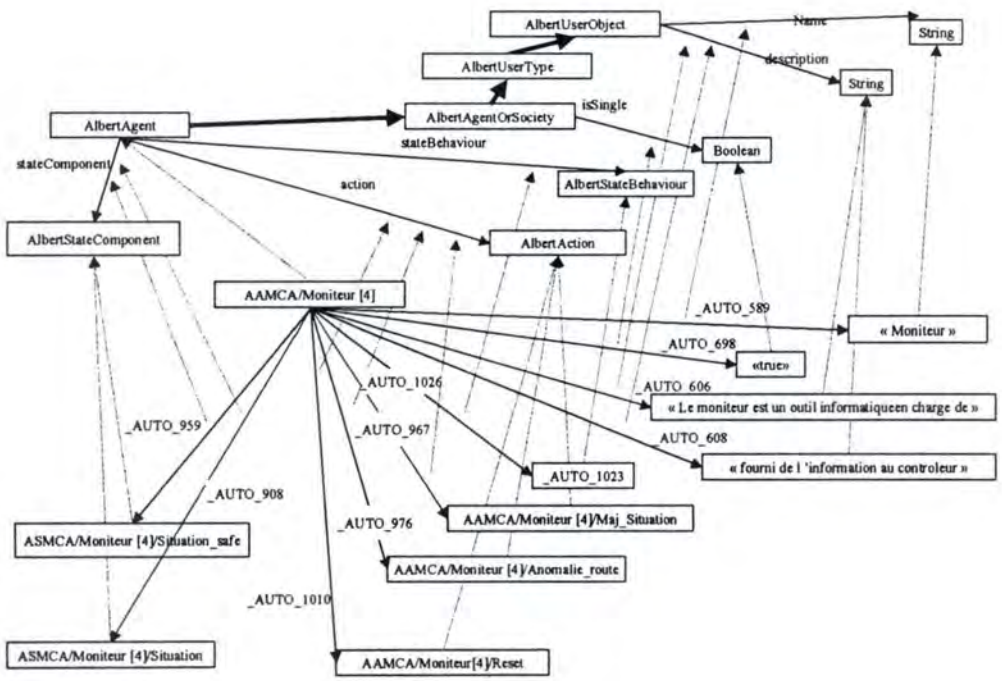
L'agent **(a)** est le concept le plus important dans Albert II. La déclaration d'un agent consiste à décrire les composants d'état **(b)** et les actions **(c)** dont il est responsable ainsi que ses contraintes. Celles-ci sont introduites pour réduire l'ensemble des comportements

admissibles des agents **[DU BOIS, 95]**. Normalement, les contraintes sont ratachées soit au composant d'état concerné ou à l'action concernée.

Une seule exception à cette règle : la contrainte de comportement d'état **(d)**.

En effet cette contrainte peut se rapporter à plusieurs états. Pour éviter de la placer d'une façon arbitraire sur un des composants d'où difficulté de la retrouver et pour empêcher de la dupliquer sur chaque composant, il est préférable de la rattacher au niveau supérieur qui est le niveau agent.

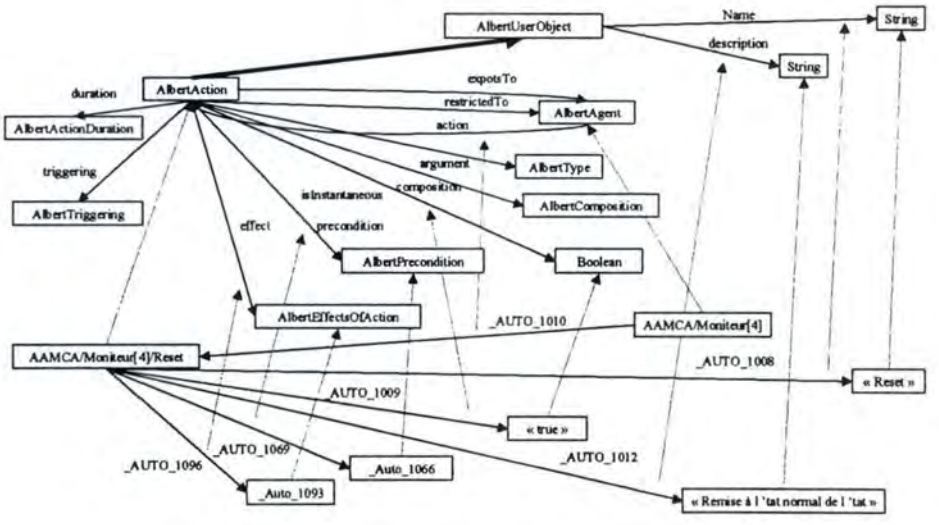
```
(a) AlbertAgent in AlbertClass isa AlbertAgentOrSociety with
        attribute
                (c) action : AlbertAction ;
                (b) stateComponent : AlbertStateComponent ;
                (d) stateBehaviour : AlbertStateBehaviour
end
```



ACTIONS

Une action **(a)** peut être instantanée **(b)** ou avoir une durée, elle peut être exportée **(c)** vers d'autres agents ou sociétés. Des contraintes sur ces exportations[1] peuvent être formulées. D'autres types de contrainte peuvent être exprimés : de durée **(d)**, de composition **(e)**, de précondition **(f)**, de triggering **(g)** et d'effet d'action **(h)**. Une action peut être paramétrisée **(i)**.

```
(a)AlbertAction in AlbertClass isa AlbertUserObject with
       attribute
              (i)argument : AlbertType ;
              (b)isInstantaneous : Boolean ;
              restrictedTo : AlbertAgent ;
              (c)exportsTo : AlbertAgent ;
              (d)duration : AlbertActionDuration ;
              (e)composition : AlbertActionComposition ;
              (f)precondition : AlbertPrecondition ;
              (g)triggering : AlbertTriggering ;
              (h)effect : AlbertEffectOfAction ;
end
```

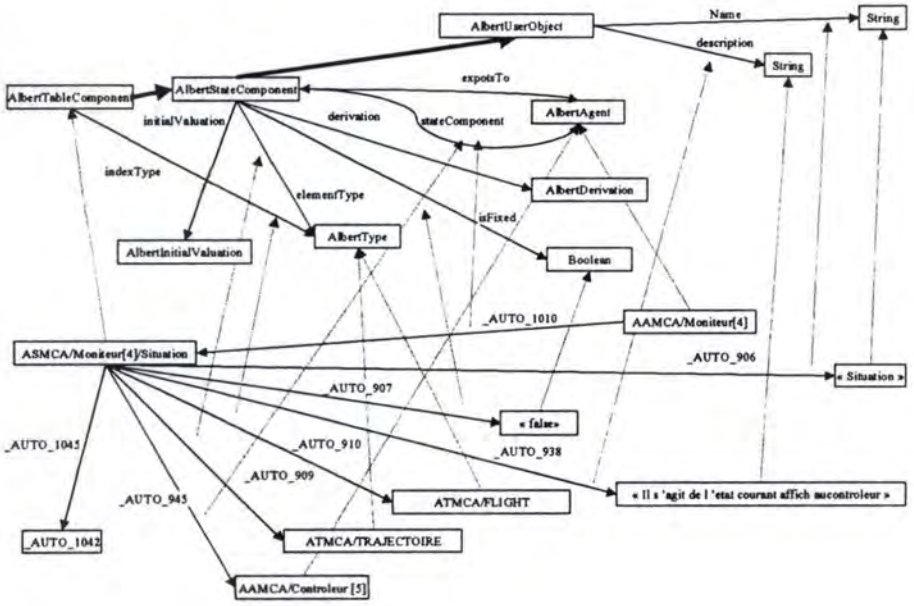


[1] Voir paragraphe concernant les contraintes de coopération p.5 2

COMPOSANTS D'ETAT

Un composant d'état **(a)** peut être fixe **(b)** ou varié dans le temps. Il peut dériver **(c)** d'autres composants, il peut également être exporté **(d)** vers d'autres agents ou sociétés. Des contraintes sur les exportations[1] peuvent être formulées. D'autres types de contraintes peuvent être exprimer : de dérivation **(f)** et de valeurs initiales Un composant d'état peut être un ensemble de valeurs d'un type donné **(e)** **[DU BOIS, 95]**. **(g)**.

```
(a)AlbertStateComponent in AlbertClass isA AlbertUserObject with
        attribute
                (e)elementType : AlbertType ;
                (b)isFixed : Boolean ;
                (c)isDerivedFrom : AlbertStateComponent ;
                (d)exportsTo : AlbertAgent ;
                (f)derivation : AlbertDerivation ;
                (g)initialValuation : AlbertInitialValuation ;
end
```



[1] Voir paragraphe concernant les contraintes de coopération p. 52

CONTRAINTES

Les contraintes **(a)** sont écrites en accord avec des patrons prédéfinis qui permettent d'aider l'analyste dans l'écriture de formules complexes et consistantes. Les patrons ont été introduits pour des raisons méthodologiques **[DU BOIS, 95]**.

```
(a)AlbertConstraint in AlbertClass isa AlbertUserObject with
end
```

Les composants dérivés

Les contraintes de composants dérivés **(a)** sont utilisées pour décrire la relation mathématique entre plusieurs composants d'état d'un agent. Un composant dérivé a toujours une valeur qui peut être calculée à partir d'autres composants**(b)**. Ce type de contraites est utilisé pour simplifier la lisibilité des expressions. Une expression complexe peut être remplacée par un nom grâce aux composants dérivés.

```
(a)AlbertDerivation in AlbertClass isa AlbertConstraint with
        attribute
                (b)constrainedBy :AlbertStateComponent
end
```

Les valeurs initiales

Les contraintes de valeurs initiales **(a)** permettent de fixer une valeur initiale à un composant d'état.

```
(a)AlbertInitialValuation in AlbertClass isa AlbertConstraint with
end
```

Comportement D'Etat

Les contraintes de comportement **(a)** sont utilisées pour donner les propriétés des composants d'état **(b)** d'un agent. Chaque fois que les actions appropriées**(c)** sont exécutées, la contrainte doit être vérifiée.

```
(a)AlbertStateBehaviour in AlbertClass isa AlbertConstraint with
        attribute
                (b)involvedStateComponent : AlbertStateComponent ;
                (c)restrictedTo : AlbertAction
end
```

Composition D'Actions

Les actions peuvent être considérées à différents niveaux de granularité. Les contraintes de composition d'actions **(a)** expriment la manière dont les actions peuvent être affinées **(b)** ou inversement la manière dont les actions peuvent être agrégées. Les actions peuvent être combinées de manière séquentielle en parallèle, travaillées simultanément...

```
(a)AlbertActionComposition in AlbertClass isa AlbertConstraint with
        attribute
                (b)componentAction : AlbertAction ;
                withClauseStateComponent[1] : AlbertStateComponent
end
```

La Durée Des Actions

Ces contraintes sont utilisées pour mettre une valeur sur la durée **(a)** des occurrences d'une action. Cette valeur peut correspondre à une durée exacte à une limite supérieure ou inférieure

```
(a)AlbertActionDuration in AlbertClass isa AlbertConstraint with
        attribute
                withClauseStateComponent[2]: AlbertStateComponent
end
```

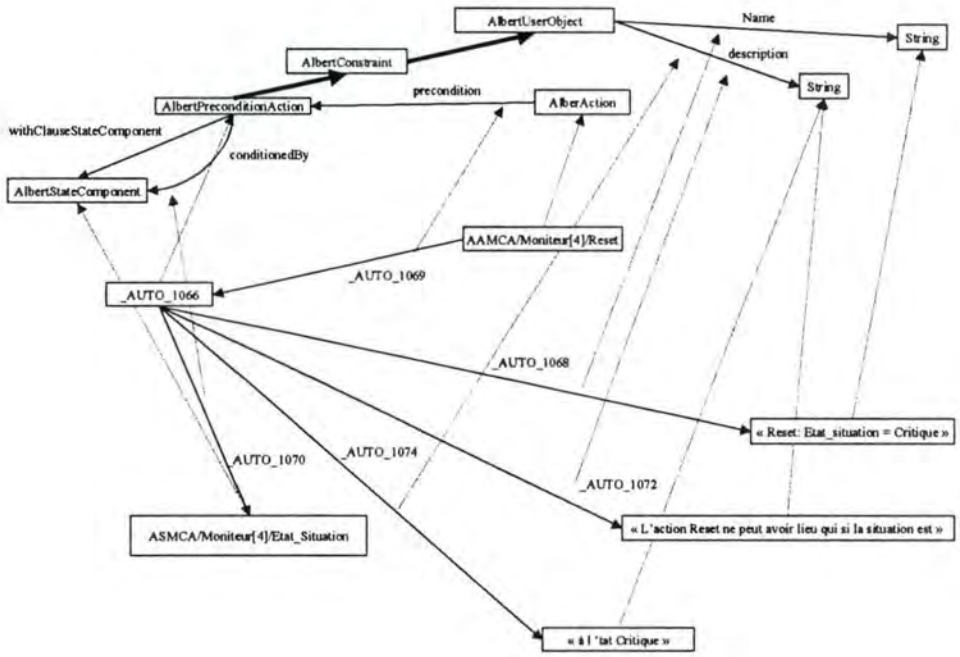
[1] Donne les composants d'état qui réduisent la portée de cette contrainte

[2] Idem

PRECONDITIONS

Les préconditions **(a)** définissent les conditions qui doivent être vérifiées sur un composant d'état **(b)** pour qu'une action donnée puisse avoir lieu dans un futur.

```
(a)AlbertPrecondition in AlbertClass isa AlbertConstraint with
       attribute
              (b)conditionedBy : AlbertStateComponent ;
              withClauseStateComponent[1] : AlbertStateComponent
end
```



Effets Des Actions

Ces contraintes sont utilisées pour exprimer la manière dont l'occurrence d'une action effectue **(a)** des changements dans des composants d'état d'un agent. Les changements

[1] Donne les composants d'état qui réduisent la portée de cette contrainte

d'état d'un agent peuvent être causés par des actions sous sa responsabilité mais aussi par des actions provenant d'autres agents.
Le pré-effet **(b)** d'une action peut être formulé, ce qui correspond aux changements de valeur des composants d'état d'un agent au début de l'occurence d'une action.
Le post-effet **(c)** d'une action peut être formulé, ce qui correspond aux changements de valeur des composants d'état d'un agent à la fin de l'occurance d'une action.

```
(a)AlbertEffectsOfAction in AlbertClass isa AlbertConstraint with
        attribute
                (b)preEffect : AlbertStateAssignment ;
                (c)postEffect : AlbertStateAssignment ;
                postEffectConditionedBy : AlbertStateComponent ;
                withClauseStateComponent[1] : AlbertStateComponent
end
```

Triggering

Les contraintes de triggering **(a)** sont utilisées pour exprimer une condition sur un composant d'état **(b)** sur lequel une action particulière à lieu.

```
(a)AlbertTriggering in AlbertClass isa AlbertConstraint with
        attribute
                (b)conditionedBy : AlbertStateComponent ;
                withClauseStateComponent[2] : AlbertStateComponent
end
```

Contraintes coopératives

Les contraintes coopératives sont utilisées pour spécifier la manière dont les agents inter agissent entre eux.
Elles agissent sur une action **(a)** ou sur un composant d'état **(b)**. Elles sont soit de type perception **(c)** ou informations **(d)**.

---

[1] Donne les composants d'état qui réduisent la portée de cette contrainte

[2] Idem

```
(a)AlbertAction!exportsTo in attribute with
        attribute
                (c)perception : AlbertActionPerception ;
                (d)information : AlbertActionInformation
end

(b)AlbertStateComponent!exportsTo in attribute with
        attribute
                (c)perception : AlbertStatePerception ;
                (d)information : AlbertStateInformation
end
```

Perception Des Actions Et Des Etats

Ces contraintes permettent de déterminer dans quelles conditions **(a)**(suivant la valeur des composants d'état) un agent perçoit les informations provenant d'autres agents (une partie d'un état ou l'occurrence d'une action).

```
AlbertStatePerception in AlbertClass isa AlbertConstraint with
        attribute
                (a)conditionedBy : AlbertStateComponent ;
                withClauseStateComponent[1] : AlbertStateComponent
end

AlbertActionPerception in AlbertClass isa AlbertConstraint with
        attribute
                (a)conditionedBy : AlbertStateComponent ;
                withClauseStateComponent[2] : AlbertStateComponent
end
```

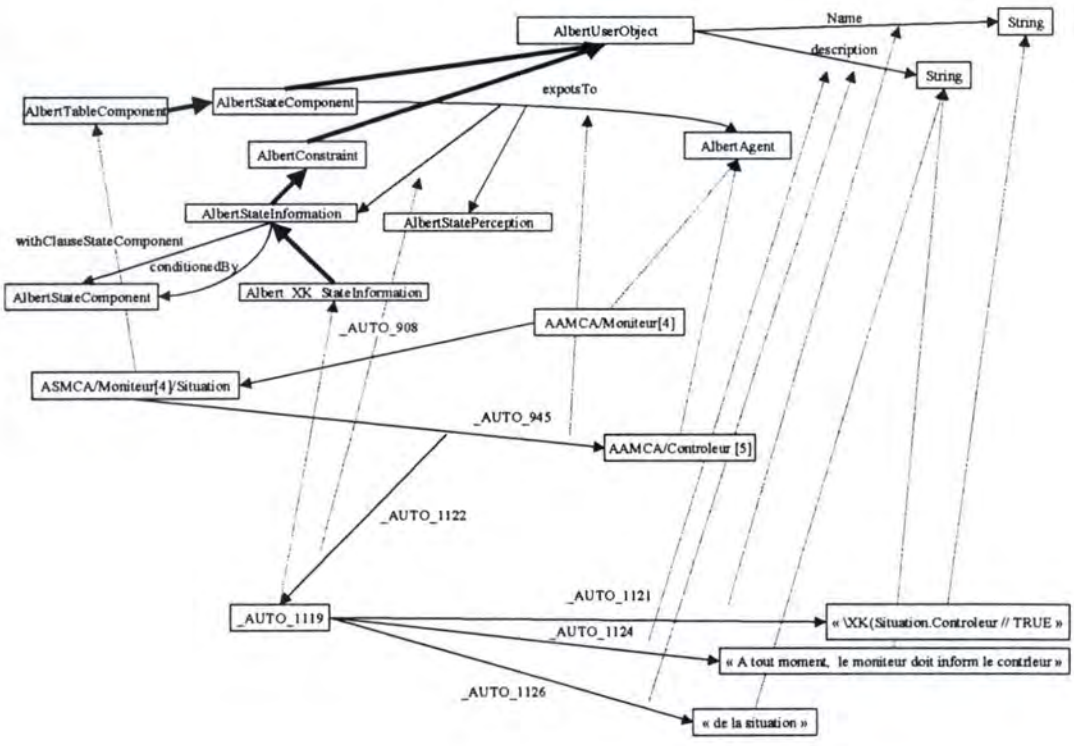
[1] Donne les composants d'état qui réduisent la portée de cette contrainte

[2] Idem

Information Des Actions Et Des Etats

Ces contraintes permettent de déterminer dans quelles conditions **(a)** (suivant la valeur des composants d'état) un agent montre les informations (une partie d'un état ou l'occurrence d'une action qu'il réalise) à d'autres agents.

```
AlbertActionInformation in AlbertClass isa AlbertConstraint with
        attribute
                (a) conditionedBy : AlbertStateComponent ;
                withClauseStateComponent[1] : AlbertStateComponent
end
AlbertActionInformation in AlbertClass isa AlbertConstraint with
        attribute
                (a) conditionedBy : AlbertStateComponent ;
                withClauseStateComponent[2] : AlbertStateComponent
end
```



[1] Donne les composants d'état qui réduisent la portée de cette contrainte

[2] Idem

CHAPITRE V

SPECIFICATION

CHAPITRE V

SPECIFICATION

La fonctionnalité principale de notre application est la recherche de toutes les descriptions liées aux différents concepts d'Albert II instantiés. On peut la décomposer en sous fonctionnalités spécifiques à chaque concept propre à Albert II.

Les différentes spécifications définies dans ce chapitre permettent de se familiariser davantage avec l'architecture du repository d'Albert II.

Pour exprimer les différentes fonctionnalités de l'application, nous avons utilisé le langage CBQL. Celui-ci a été décrit dans le chapitre consacré au langage Telos[1].

Les différentes sous fonctionnalités

- getAlbSpec
- getuserType
- getuserOperation
- getRootAgSoc
- getAgOrSoc
- getaction
- getstateComponent
- getderivation
- getinitialValuation
- getstateBehaviour
- getcomposition
- getduration
- getprecondition
- geteffect
- gettriggering

[1] Voir chapitre III

- getactionperception
- getstatecompperception
- getactioninformation
- getstatecompinformation

5.1 GETALBSPEC

Cette fonctionnalité permet de retrouver toutes les spécifications **(a)** existantes dans le repository et de donner pour chaque spécification son nom et sa description.

```
QueryClass getAlbSpec isa (a)AlbertSpecification
retrieved_attribute
        name : String
        description : String
end AlbSpec
```

5.2 GETUSERTYPE

Cette fonctionnalité retrouve les types de données utilisateurs **(a)** liés à une spécification donnée. Elle prend en paramètre une spécification (AlbertSpecification) et renvoie le nom et la description de chaque userType lié à cette spécification.

```
GenericQueryClass getuserType isa AlbertUserDefinedType
retrieved_attribute
        name : String
        description : String
        computed_attribut,parameter
                spec : AlbertSpecification
        constraint
                userTypeconst :$ spec (a)userType this $
end getuserType
```

5.3 GETUSEROPERATION

Cette fonctionnalité retrouve les opérations sur les types de données **(a)** (définits par l'analyste) liés à une spécification donnée. Elle prend en paramètre une spécification **(b)** et renvoie le nom et la description de chaque userOperation lié à cette spécification.

```
GenericQueryClass getuserOperation isa AlbertUserOperation
retrieved_attribute
        name : String
        description : String
        computed_attribut,parameter
                spec : (b)AlbertSpecification
        constraint
                userOperationconst :$ spec (a)userOperation this $
end getuserOperation
```

5.4 GETROOTAGSOC

A partir d'une instance d' une spécification **(a)** cette fonctionnalité donne la racine de l'arbre **(b)** qui contient les agents et les sociétés. Elle renvoie le nom et la description de la société racine liée à une instance de la spécification.

```
GenericQueryClass getRootAgSoc isa AlbertAgent , AlbertSociety
retrieved_attribute
        name : String
        description : String
        computed_attribut,parameter
                spec : (a)AlbertSpecification
        constraint
                rootconst :$ spec (b)rootAgentSociety this $
end getRootAgSoc
```

5.5 GETAGORSOC

A partir d'une société **(a)**, cette fonctionnalité donne tous les agents et les sociétés qui la composent **(b)** et cela quelle que soit la profondeur de l'arbre.

```
GenericQueryClass getAgOrSoc isa AlbertAgent , AlbertSociety
retrieved_attribute
       name : String
       description : String
       computed_attribut,parameter
              rootsoc : (a)AlbertSociety
       constraint
              AgOrSocconst :$ ( rootsoc containedAgentOrSociety this )
                            or exists soc/AlbertSociety
                            ( soc in getAgOrSoc ) and
                            ( soc (b)containedAgentOrSociety this ) $
end getAgOrSoc
```

5.6 GETACTION

Cette fonctionnalité donne à partir d'une instance d'un agent **(a)** le nom et la description de toutes les actions **(b)** qui sont liées à celui-ci.

```
GenericQueryClass getaction isa AlbertAction
retrieved_attribute
       name : String
       description : String
       computed_attribut,parameter
              agent : (a)AlbertAgent
       constraint
              actionconst :$ agent (b)action this $
end getaction
```

5.7 GETSTATECOMPONENT

Cette fonctionnalité donne à partir d'une instance d'un agent **(a)** le nom et la description de tous les composants d'état **(b)** qui sont liés à celui-ci.

```
GenericQueryClass getstateComponent isa AlbertstateComponent
retrieved_attribute
        name : String
        description : String
        computed_attribut,parameter
                agent : (a)AlbertAgent
        constraint
                stateComponentconst :$ agent (b)stateComponent this $
end getstateComponent
```

5.8 GETDERIVATION

Cette fonctionnalité donne à partir d'un composant d'état **(a)** la description de toutes les contraintes de type dérivation **(b)** liées à celui-ci.

```
GenericQueryClass getderivation isa AlbertDerivation
retrieved_attribute
        description : String
        computed_attribut,parameter
                st : (a)AlbertstateComponent
        constraint
                derivationconst : $ ( st (b)derivation this )) $
end getderivation
```

5.9 GETINITIALVALUATION

Cette fonctionnalité donne à partir d'un composant d'état **(a)** la description de toutes les contraintes de type valeur initiales **(b)** qui sont liées à celui-ci.

```
GenericQueryClass getinitialValuation isa AlbertInitialValuation
retrieved_attribute
        description : String
        computed_attribut,parameter
                st : (a)AlbertstateComponent
        constraint
                initialValuationconst : $ st (b)initialValuation this  $
end getinitialValuation
```

5.10 GETSTATEBEHAVIOUR

Cette fonctionnalité donne à partir d'un composant d'état **(a)** la description de toutes les contraintes de type comportement d'état **(b)** qui sont liées à celui-ci.

```
GenericQueryClass getstateBehaviour isa (b)AlbertstateBehaviour
retrieved_attribute
       description : String
       computed_attribut,parameter
              st : (a)AlbertstateComponent
       constraint
              stateBehaviourconst :$ this involvedStateComponent st $
end getstateBehaviour
```

5.11 GETCOMPOSITION

Cette fonctionnalité donne à partir d'une action **(a)** la description de toutes les contraintes de type composition **(b)** qui sont liées à celle-ci.

```
GenericQueryClass getcomposition isa AlbertActionComposition
retrieved_attribute
       description : String
       computed_attribut,parameter
              act : (a)AlbertAction
       constraint
              compositionconst : $  act (b)composition this  $
end getcomposition
```

5.12 GETDURATION

Cette fonctionnalité donne à partir d'une action **(a)** la description de toutes les contraintes de type durée **(b)** qui sont liées à celle-ci.

```
GenericQueryClass getduration isa AlbertActionDuration
retrieved_attribute
        description : String
        computed_attribut,parameter
                act : (a)AlbertAction
        constraint
                durationconst : $ act (b)duration this  $
end getduration
```

5.13 GETPRECONDITION

Cette fonctionnalité donne à partir d'une action **(a)** la description de toutes les contraintes de type précondition **(b)** qui sont liées à celle-ci.

```
GenericQueryClass getprecondition isa AlbertPrecondition
retrieved_attribute
        description : String
        computed_attribut,parameter
                act : (a)AlbertAction
        constraint
                preconditionconst : $ act (b)precondition this  $
end getprecondition
```

5.14 GETEFFECT

Cette fonctionnalité donne à partir d'une action **(a)** la description de toutes les contraintes de type effet **(b)** d'une action qui sont liées à celle-ci.

```
GenericQueryClass geteffect isa AlbertEffectsOfAction
retrieved_attribute
       description : String
       computed_attribut,parameter
              act : (a)AlbertAction
       constraint
              effectconst : $ act (b)effect this $
end geteffect
```

5.15 GETTRIGGERING

Cette fonctionnalité donne à partir d'une action **(a)** la description de toutes les contraintes de type triggering **(b)** qui sont liées à celle-ci.

```
GenericQueryClass gettriggering isa AlbertTriggering
retrieved_attribute
       description : String
       computed_attribut,parameter
              act : (a)AlbertAction
       constraint
              triggeringconst : $ act (b)triggering this  $
end gettriggering
```

5.16 GETACTIONPERCEPTION

Cette fonctionnalité donne à partir d'une action **(a)** la description de toutes les contraintes de type perception **(b)** d'action qui sont liées à celle-ci.

```
GenericQueryClass getactionperception isa Albert_I_ActionPerception
                                    , Albert_K_ActionPerception
                                    , Albert_XK_ActionPerception
retrieved_attribute
       description : String
       computed_attribut,parameter
              act : (a)AlbertAction
       constraint
              actionperceptionconst :
 $ exists exp/AlbertAction!exportsTo (exp in act!exportsTo)
       and ( exp (b)perception this )$
end getactionperception
```

5.17 GETSTATECOMPPERCEPTION

Cette fonctionnalité donne à partir d'un composant d'état **(a)** la description de toutes les contraintes de type perception **(b)** d'un composant d'état qui sont liées à celui-ci.

```
GenericQueryClass getstatecompperception isa Albert_I_StatePerception
                                    , Albert_K_StatePerception
                                    , Albert_XK_StatePerception
retrieved_attribute
       description : String
       computed_attribut,parameter
              st : (a)AlbertstateComponent
       constraint
              statecompperceptionconst :
 $ exists exp/AlbertStateComponent!exportsTo
 ((exp in st!exportsTo)and ( exp (b)perception this ))$
end getstatecompperception
```

5.18 GETACTIONINFORMATION

Cette fonctionnalité donne à partir d'une action **(a)** la description de toutes les contraintes de type information **(b)** sur une action qui sont liées à celle-ci.

```
GenericQueryClass getactioninformation isa Albert_I_ActionInformation
                                    , Albert_K_ActionInformation
                                    , Albert_XK_ActionInformation
retrieved_attribute
       description : String
       computed_attribut,parameter
              act : (a)AlbertAction
       constraint
              actioninformationconst :
 $ exists exp/AlbertAcion!exportsTo
((exp in act!exportsTo)and ( exp (b)information this ))$
end getactioninformation
```

5.19 GETSTATECOMPINFORMATION

Cette fonctionnalité donne à partir d'un composant d'état **(a)** la description de toutes les contraintes de type information **(b)** sur un composant d'état qui sont liées à celui-ci.

```
GenericQueryClass getstatecompinformation isa
                                    Albert_I_StateInformation
                                    , Albert_K_StateInformation
                                    , Albert_XK_StateInformation
retrieved_attribute
       description : String
       computed_attribut,parameter
              st : (a)AlbertstateComponent
       constraint
              statecompinformationconst :
 $ exists exp/AlbertStateComponent!exportsTo
 ((exp in st!exportsTo)and ( exp (b)information this ))$
end getstatecompinformation
```

CHAPITRE VI

CONCEPTION

CHAPITRE VI

CONCEPTION

Notre application génère un cahier des charges. On peut l'enregistrer soit en format «*txt* » ou « *html* ». Le document sous format « *html* » est composé de deux parties. La partie de gauche comprend une liste des sociétés et agents qui composent la spécification. Ces agents et sociétés sont des index vers les informations du contenu de la partie de droite. Ceci permet de retrouver plus facilement des informations concernant un agent particulier ou une société particulière.

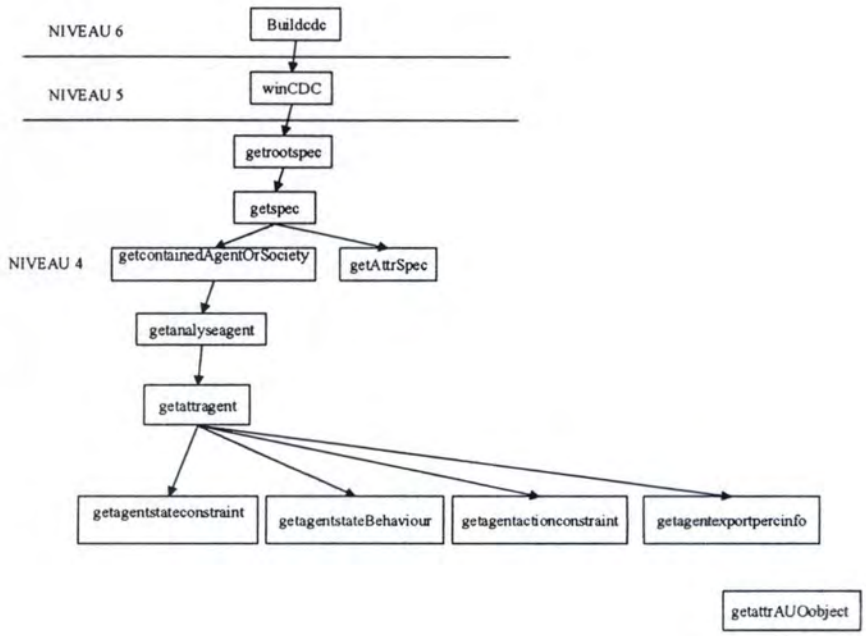
Le repository est créé et peuplé à partir de miniTélos qui est implémanté en java. Nous n'avons donc pas eu le choix et le langage d'implémentation de notre application est également en java.

Comme nous avons pu le constater lors de la description du repository, les données recherchées sont liées entre elles. En effet une contrainte appartient à une action qui elle-même appartient à un agent qui lui appartient à une société, celle-ci appartenant à une spécification. Il nous a semblé normal de garder cette structure arborescente pour l'architecture de notre application. Néanmoins, nous avons regroupé certaines fonctions présentant des caractéristiques semblables au point de vue du traitement des données.

Exemple :

Les fonctionnalités getprecondition et geteffect... sont devenues une fonction getagentstateconstraint avec comme paramètre soit « precondition » ou « effect ».

ARCHITECTURE LOGICIELLE DE L'APPLICATION



Remarque :

Etant donné la multitude des liens existant entre un grand nombre de fonctions de notre application et la fonction getattrAUOobject, ces liens ne sont pas représentés.

6.1 NIVEAU 6 : MODULE COORDINATEUR

BUILDCDC

_ → _

C'est le point d'entrée de notre application. Elle appelle la fonction winCDC si les paramètres en lignes ne sont pas définis ou mal formulés. Dans le cas contraire, elle appelle directement la fonction getrootspec.

utilise

Niveau 5 : winCDC

Niveau 4 : getrootspec

6.2 NIVEAU 5 : MODULE IHM

WINCDC

_ → _

Ceci est la fenêtre principale de notre application. Elle est composée essentiellement de trois boutons.

- Un bouton ouvrir qui ouvre une fenêtre de dialogue classique dans laquelle on donne le nom de la spécification (fichier) pour laquelle on veut rédiger un chaier des charges.
- Un bouton enregistrer qui ouvre une fenêtre dans laquelle on spécifie le nom et l'endroit où l'on veut enregistrer le cahier des charges.
- Un bouton quitter qui ferme l'application.

utilise

Niveau 4 : getrootspec

6.3 NIVEAU 4 : MODULE DE TRAITEMENT

GETROOTSPEC

Rep : String X CDC : String → _

A partir de Rep on ouvre la base de connaissance Telos. Ceci permet de la rendre accessible à toutes les bases de données. Elle fait appel à la fonction getspec qui va rédiger le cahier des charges.

utilise

Niveau 4 : getspec

GETSPEC

_ → _

Dans cette fonction, la première opération est de déterminer les différentes instances d'une spécification Albert II. Pour chaque spécification, on écrit son nom et sa description dans un fichier. On recherche également les types définis par l'utilisateur, les opérations définies par l'utilisateur et la société ou l'agent racine d'une spécification. Cette fonctionnalité correspond à la fonction getAlbSpec[1]

utilise

Niveau4 : getattrAUOobject
getAttrSpec
getcontainedAgentOrSociety

Niveau3 : getIndividual(TelosBase)
getInstances(TelosObject)
getDestination(TelosAttribute)
size(TelosVector)
getAt(TelosVector)

GETATSPEC

TelosIndividual X attr : String→SEQ[item:TelosIndividual]

Comme les propriétés des opérations sur les types sont semblables à celles des types définis (c'est à dire qu'ils sont tous les deux des attributs de l'individu

[1] Voir page 56

AlbertSpecification), cette fonction est utilisée pour déterminer le nom et descriptions des opérations existantes dans le repository.
Pour les mêmes raisons, cette fonction est utilisée pour déterminer la société racine.
Pour une spécification particulière, on va rechercher les liens existants entre cette spécification et un attribut.
Ensuite, on se positionne sur la destination du lien, ce qui nous permet d'obtenir le nom et la description de cette instance d'attribut.
Cette fontion suivant le paramètre attr correspond aux fonctionnalités : getuserType[1], getuserOperation[2], getRootAgSoc[3]

utilise

Niveau4 : getattrAUOobject

Niveau3 : getIndividual(TelosBase)
getAttribute(TelosObject)
getInstancesWithSource(TelosAttribute)
getDestination(TelosAttribute)
size(TelosVector)
getAt(TelosVector)

GETCONTAINEDAGENTORSOCIETY

TelosIndividual → _

A partir d'un individu (qui est une instance d'un AlbertSociety), on recherche d'abord les liens entre cet individu et d'autres sociétés ou agents.
Ensuite, on se positionne sur la destination du lien, si la destination est une société alors on recherche de nouveau les agents «fils » ou sociétés «filles »contenus dans cette société. On appelle de nouveau la fonction getcontainedAgentOrSociety avec comme

[1] Voir page 56
[2] Voir pages 56-57
[3] Voir page 57

paramètre cette société. Autrement, on analyse l'agent en appellant la fonction getanalyseagent.

Cette fonction correspond à la fonctionnalité getAgOrSoc[1].

utilise

Niveau4 : getcontainedAgentOrSociety
getanalyseagent
getattrAUOobject

Niveau3 : getIndividual(TelosBase)
getAttribute(TelosObject)
getInstancesWithSource(TelosAttribute)
getDestination(TelosAttribute)
size(TelosVector)
getAt(TelosVector)

GETANALYSEAGENT

TelosIndividual → _

Cette fonction consiste à donner le nom et la description d'un agent (passé comme paramètre de la fonction) ainsi que les actions et les composants d'état qui composent cet agent.

Cette fonction correspond à la fonctionnalité getAgOrSoc[2].

Niveau4 : getattragent
getattrAUOobject

GETATTRAGENT

ag : TelosIndividual X attr : String→ _

---

[1] Voir pages 57-58

[2] Voir pages 57-58

Suivant le paramètre attr passé à la fonction, on recherche les actions ou les composants d'état associés à l'ag. Si attr = action alors la fonction effectue ce qui suit :
A partir d'une instance d'AlbertAgent, on recherche les liens entre cet agent et une instance d'action. Ensuite, pour chaque lien on se positionne sur la destination de celui-ci. Ce qui nous donne une instance de l'individu AlbertAction. Ceci nous permet de connaître le nom et la description de toutes les actions d'un agent.
Il est aussi nécessaire de connaître les différentes contraintes associées aux actions et aux composants d'état de l'agent.
La fonction correspond aux fonctionnalités : getaction[1] et getstateComponent[2].

utilise

Niveau4 : getattrAUOobject
getagentstateBehaviour
getagentstateconstraint
getagentactionconstraint
getagentexportpercinfo

Niveau3 : getIndividual(TelosBase)
getAttribute(TelosObject)
getInstancesWithSource(TelosAttribute)
getDestination(TelosAttribute)
size(TelosVector)
getAt(TelosVector)

GETAGENTSTATEBEHAVIOUR

TelosIndividual → _

A partir d'une instance d'un AlbertstateComponent, on recherche les liens entre les contraintes de comportement et les composants d'état. Ensuite, pour chaque lien on se

[1] Voir page 58

[2] Voir pages 58-59

positionne sur la source de celui-ci. Ce qui nous donne une instance de l'individu AlbertStateBehaviour. Ceci nous permet de connaître le nom et la description de toutes les actions d'un agent.

Cette fonction correspond à la fonctionnalité getstateBehaviour[1].

utilise

Niveau4 : getattrAUOobject

Niveau3 : getIndividual(TelosBase)
getAttribute(TelosObject)
getInstancesWithDestination(TelosAttribute)
getSource(TelosAttribute)
size(TelosVector)
getAt(TelosVector)

GETAGENTSTATECONSTRAINT

TelosIndividual X attr : String → _

Pour un composant d'état particulier, on va rechercher tous les liens existant entre ce composant d'état et la contrainte spécifiée par l'attr. On se positionne ensuite sur la destination du lien ce qui permet d'obtenir une instance d'une contrainte. Il suffit maintenant d'aller rechercher la description de cette contrainte.

Cette fonction correspond aux fonctionnalités : getderivation[2] et getinitialValuation[3].

utilise

Niveau4 : getattrAUOobject

Niveau3 : getIndividual(TelosBase)
getAttribute(TelosObject)
getInstancesWithSource(TelosAttribute)

---

[1] Voir page 60

[2] Voir page 59

[3] Voir page 59

getDestination(TelosAttribute)
size(TelosVector)
getAt(TelosVector)

GETAGENTACTIONCONSTRAINT

TelosIndividual X attr : String → _

Pour une action particulière, on va rechercher tous les liens existant entre cette action et la contrainte spécifiée par l'attr. On se positionne ensuite sur la destination du lien ce qui permet d'obtenir une instance d'une contrainte. Il suffit maintenant d'aller rechercher la description de cette contrainte.
Cette fonction correspond aux fonctionnalités : getcomposition[1], getduration[2], getprecondition[3], geteffect[4] et gettriggering[5].
utilise

Niveau4 : getattrAUOobject

Niveau3 : getIndividual(TelosBase)
getAttribute(TelosObject)
getInstancesWithSource(TelosAttribute)
getDestination(TelosAttribute)
size(TelosVector)
getAt(TelosVector)

---

[1] Voir page 60
[2] Voir page 61
[3] Voir page 61
[4] Voir page 62
[5] Voir page 62

GETAGENTEXPORTPERCINFO

Pour une action particulière, on recherche toutes les exportations liées à un agent. Elles sont soit de type action ou composant d'état suivant la valeur du paramètre baseattr.
Ensuite parmi ces contraintes, on ne prend que celles qui possèdent un attribut perception ou information (le choix de l'attribut dépend du paramètre percinf).
Finalement, on recherche les descriptions associées à ces contraintes.
Cette fonction correspond aux fonctionnalités : getationperception[1], getstatecompperception[2], getactioninformation[3] et getstatecompinformation[4].
utilise

Niveau4 : getattrAUOobject

Niveau3 : getIndividual(TelosBase)
getAttribute(TelosObject)
getInstancesWithSource(TelosAttribute)
getDestination(TelosAttribute)
size(TelosVector)
getAt(TelosVector)

---

[1] Voir page 63

[2] Voir page 63

[3] Voir page 64

[4] Voir page 64

GETATTRAUOOBJECT

On recherche l'attribut name ou description (suivant la valeur de attr) d'un individu.

utilise

Niveau3 : getIndividual(TelosBase)
getAttribute(TelosObject)
getInstancesWithSource(TelosAttribute)
getDestination(TelosAttribute)
size(TelosVector)
getAt(TelosVector)

6.3 NIVEAU 3 : MODULE DES DONNEES PERSISTANTES

La description du niveau a été faite dans le chapitre consacré à miniTelos[1].

[1] Voir pages 34-38

CHAPITRE VII

CONCLUSIONS

CHAPITRE VII

CONCLUSIONS

Le but de cette application est de faciliter et d'améliorer la collaboration entre le client et l'analyste.

Dans cette application, le rôle de l'analyste est primordial, il doit toujours avoir conscience que la qualité du document généré est directement liée à ses explications. Sans aucune explication sur la spécification qu'il réalise le client sera incapable de savoir si ses exigences sont réalisées. Il est donc très important pour l'analyste de donner un maximum de descriptions.

Dans le but de d'améliorer la perception de la modélisation des exigences du client, quelques perfectionnements peuvent être apportés à l'application :

- Les informations mises par l'analyste dans les post-it peuvent être utilisées pour compléter le document généré. Les post-it sont associés à une contrainte, à une action, à un agent.... Ils sont utilisés pour apporter un éclaircissement sur des points sensibles d'une contrainte, d'une action, d'un agent....
- Pour certains concepts d'Albert II tel que les agents, les sociétés, les actions et les composants d'état, il existe une équivalence graphique. Celle-ci peut faciliter la compréhension d'une spécification. Il eet donc souhaitable de les intégrer au document généré.
- Dans le cadre du langage Albert II, d'autres outils ont été développés. Il existe un outil (paraphraseur) qui génère à partir de certains points d'une spécification l'équivalent en langage naturel. L'idéal serait de créer un outil qui permettrait de stoker ces informations dans le repository Albert II. Dans ce cas, ces informations pourraient être de nouvelles descriptions liées aux différents concepts du langage Albert II. Grâce aux performances de cet outil, notre application serait capable de générer un document moins dépendant des explications données par l'analyste.

BIBLIOGRAPHIE

BIBLIOGRAPHIE

[CAMPIONE] : Mary CAMPIONE, Kathy WALRATH, 'The Java™Tutorial. Object-Orient Programming for the Internet. Java Soft.

Description des classes MiniTelos. Disponible sous http://www.info.fundp.ac.be/~lcl/MiniTelos, 1998.

[DU BOIS , 95] : Philippe DU BOIS, 'The Albert II Language' : On the Design and the Use of a Formal Specification Language for Requirements Analysis ». PhD thesis, Computer Science Department, University of Namur, Namur (Belgique), September 1995.

[DU BOIS, 97a] : Philippe DUBOIS, 'The Albert II Reference manual' : Language Constructs and Informal Semantics Version 2.0., March 1997.

[DU BOIS, 97b] : Philippe DU BOIS, Eric DUBOIS, Jean-Marc ZEIPPEN, 'On the use of a Formal Requirements Engineering Language' Revised version of the paper published in the Proceedings of the Third International Symposium on Requirements Engineering (RE'97), Annapolis, Maryland (USA), January 5 – 8, 1997.

[DUBOIS, 94a] : Eric. DUBOIS, 'Albert at the Age of Two' Position Paper presented at the Dagstuhl Seminar 'System Requirements : Analysis, Management, and Exploitation ', Dagstuhl Castle, Germany, October 4 – 7, 1994.
Click here to see the abstract (and have the possibility to fetch the paper).

[DUBOIS, 94b] : Eric DUBOIS, Philippe DU BOIS, F.DUBRU, 'Animating Formal Requirements Specifications of Cooperative Information Systems, Toronto (Canada),May 1994.
Click here to see the abstract (and have he possibility to fetch the paper).

[DUBOIS, 94c] : Eric DUBOIS, Philippe DU BOIS, F. DUBRU, M. PETIT, 'Agent – oriented Requirements Engineering : a Case Study using the Albert Language' Proc. of the Fourth Intl. Work. Conf. on Dynamic Modelling and Information System (DYNMOD-IV), Noordwijkerhoud (The Netherlands), September 28-30 ? 1994.
Click here to see the abstract (and have the possibility to fetch the paper).

[DUBOIS, 95] : Eric DUBOIS, Philippe DU BOIS , Jean-Marc ZEIPPEN, 'A Formal Requirements Engineering Method for Real-Time, Concurrent, and Distributed System', Proc. of the ICSE Workshop on Formal Methods Application in Software Engineering Practice, Seattle, WA (USA), April 24-25, 1995.
Click here to see the abstract (and have the possibility to fetch the paper).

[HEYMANS, 97] : P. HEYMAN, 'Some throughts about the animationof formal specifications written in the Albert II langage', in (the informal) Proc. of the Doctorat Consortium of the third IEEE International Symposium on Requirements Engineering (RE'97), Annapolis, MD, (USA), January 6-10, 1997.
Click here to download the full paper (gzipped PostScript).

[JARKE, 95] : Matthias JARKE, R. GALLERSDÖRFER, M.-A. JEUSFELD, M. STAUDT, S. EHERER, 'ConceptBase – a deductive object base for meta data management'. Journal of Intelligent Information Systems, Special Issue on Advances in Deductice Object-Oriented Databases, 4 (2) : 167 – 192. InformatikV, RWTH Aachen (Germany).

[JARKE, 96] : Matthias JARKE, M.-A. JEUSFELD, M. STAUDT, 'ConceptBase V4.0 User Manual'. April 12, 1996.

Java in a Nutshell. Sebastopol, O'Reilly & Associates Inc, 1996.

Java™Object Serialization Specification, *Object serialization in the Java™system is the process of creating a serialized.* Garcia (California USA), Sun Microsystems, Inc., 1997.

[MYLOPOULOS] : John MYLOPOULOS, Alex BORGIDA, Matthias JARKE, Manolis KOUBARAKIS, 'Telos : Representing Knowledge About Information Systems'. Toronto, University, Department of Computer Science.

Projet CAT (Rapport jumelé d'activité. Période juillet - décembre 1997), 1998.

```
package BE.ac.fundp.info.Albert.jap;

import BE.ac.fundp.info.Telos.*;
import BE.ac.fundp.info.Albert.TelosBase.*;

import java.io.*;
import java.io.InputStream;
import java.io.FileInputStream;
import java.io.File;
import java.io.PrintWriter;
import java.io.OutputStream;

public class getdescr {

      AlbertTelosBase base;
      ObjectInput s1 ;
      File inputFile;
      File outputFile ;
      FileOutputStream foutput ;
      DataOutputStream dataout ;
      String nomfichhtml;
      String ext;


  public  DataOutputStream rootspec(String fichin,String fichout)
       throws Exception {


      int n= 0;


      repcreate(fichin);

      outputFile = new File (fichout);
      foutput = new FileOutputStream(outputFile);
      dataout = new DataOutputStream(foutput);

      String path,extension,fichier;

      Filename fichier1;
      fichier1 = new Filename(fichout,'\\');
      extension = fichier1.extension();

      if (extension.equals("html")  )
            {
            String ff,mfile,dfile;
            Filename fichhtml;
            fichhtml = new Filename(fichout,'\\');
            extension = fichhtml.extension();
            path = fichhtml.path();
            mfile = path+"\\m"+fichhtml.filename()+'.'+extension;
            dfile = path+"\\d"+fichhtml.filename()+'.'+extension;
            htmlindexwrite(fichout);
```

```
            File filemenu ;
            FileOutputStream fosmenu ;

            filemenu = new File (mfile);
            fosmenu = new FileOutputStream(filemenu);
            dataout = new DataOutputStream(fosmenu);

            htmlmenuwrite();
            nomfichhtml = "d"+fichhtml.filename()+'.'+extension;
            ext = "mhtml";
            rAShtmlwrite();

            File filedata ;
            FileOutputStream fosdata ;
            filedata = new File (dfile);
            fosdata = new FileOutputStream(filedata);
            dataout = new DataOutputStream(fosdata);

            ext = "dhtml";
            htmldatawrite();
            ext = "dhtml";
            spec();

            }
      else {
            ext = extension;
            spec();
            }
      return dataout;
      }

/************************
* infogeneralwritehtml *
************************/

public void infogeneralwritehtml(TelosIndividual ti)
                  throws Exception
      {

      String out;

      out = "<h1>Spécification : "
            +getattrAUOobject(ti,"name")+"</h1><BR>";
      dataout.writeBytes(out);

      out = "<BR>"+getattrAUOobject(ti,"description")+"<BR>";
      dataout.writeBytes(out);

      }
```

```
/**********************
* infogeneralwritedoc *
**********************/

public void infogeneralwritedoc(TelosIndividual ti)
                  throws Exception
      {

      String out,auxout;

      out = "* Spécification : "
            +getattrAUOobject(ti,"name")+" *\n";
      auxout = sousligne(out.length()-1,"*")+"\n";
      dataout.writeBytes(auxout);
      dataout.writeBytes(out);

      out = sousligne(out.length()-1,"*")+"\n";
      dataout.writeBytes(out);

      out = "\n"+getattrAUOobject(ti,"description")+"\n";
      dataout.writeBytes(out);

      }

/*********
*  spec  *
**********/

public void  spec()
                  throws Exception

      {

      TelosIndividual tiSp;
      TelosObject toSp;
      TelosVector tvinSp;

      TelosIndividual tiinSp;
      TelosVector tvRAg;
      TelosIndividual tiras;
      TelosAttribute taras;


      int nbrspec;
      int nbrRAg;

      tiSp = base.getIndividual("AlbertSpecification");
      toSp =(TelosObject)tiSp;
      tvinSp = toSp.getInstances();

      nbrspec = 0;
```

```
      while (nbrspec < tvinSp.size())
            {
            tiinSp = (TelosIndividual)tvinSp.getAt(nbrspec);
            if(ext.equals("dhtml"))
                  {
                  infogeneralwritehtml(tiinSp);
                  }
            else
                  {
                  infogeneralwritedoc(tiinSp);
                  }

            /*** userType ***/
            AtSpec(tiinSp,"userType");

            /*** userOperation ***/
            AtSpec(tiinSp,"userOperation");

            /*** rootAgentSocietywrite ***/
            tvRAg = AtSpec(tiinSp,"rootAgentSociety");
            nbrRAg =0;

            while (nbrRAg < tvRAg.size())

                        {
                        taras = (TelosAttribute)tvRAg.getAt(nbrRAg);
                        tiras = (TelosIndividual)taras.getDestination();


                        containedAgentOrSocietyget(tiras);
                        nbrRAg++;
                        }
            nbrspec++;
            }
      }

/************
* repcreate *
*************/

public  void repcreate(String fichin)

      {
       FileInputStream f ;
       MessagePool messages= new MessagePool(null);

      try {

       f= new FileInputStream(new File(fichin));
       AlbertRootNode spec= new AlbertRootNode(f,messages);
       base= new AlbertTelosBase();
       spec.fillTelosBase(base);
      }catch (Exception e)
            {System.out.println("probléme du parser"+e);}

      }
```

```
/***************
* rAShtmlwrite *
****************/

public  void rAShtmlwrite() throws Exception

      {

      TelosIndividual tiASpec;
      TelosObject toASpec;
      TelosObject totmp;
      TelosAttribute taASpec;
      TelosVector tvASpec;

      TelosAttribute tatmp;
      TelosIndividual titmp;

      int nbrASpec;
      String out;

      tiASpec = base.getIndividual("AlbertSpecification");
      toASpec = tiASpec.getSource();
      taASpec = toASpec.getAttribute("rootAgentSociety");
      tvASpec = taASpec.getInstances();
      nbrASpec =0;

      while (nbrASpec < tvASpec.size())
            {
            tatmp = (TelosAttribute)tvASpec.getAt(nbrASpec);
            titmp = (TelosIndividual)tatmp.getDestination();

            out ="<BR><U><FONT COLOR=\"#FFCC99\"><A HREF=\""
                        +nomfichhtml+"#"+titmp+"\" TARGET=\"data\">"
                        +"<h2>"+getattrAUOobject(titmp,"name")
                        +"/<h2>"+"</A></FONT></U>"+'\n';


                  dataout.writeBytes(out);

                  cAgSocget(titmp);
            nbrASpec++;
            }

      }
```

```
/************
* cAgSocget *
*************/

public  void cAgSocget
            ( TelosIndividual tisoc) throws Exception

      {
      TelosIndividual tiAS;
      TelosObject toAS;
      TelosAttribute tacAOS;
      TelosVector tvin_acAOS;

      TelosAttribute tain_cAOS;
      TelosIndividual tiin_cAOS;
      int nbrAOSdescr;
      boolean issoc;

      tiAS = base.getIndividual("AlbertSociety");
      tacAOS = tiAS.getAttribute("containedAgentOrSociety");
      tvin_acAOS = tacAOS.getInstancesWithSource(tisoc.getSource());

      nbrAOSdescr =0;

      while (nbrAOSdescr < tvin_acAOS.size())
            {
            tain_cAOS = (TelosAttribute)tvin_acAOS.getAt(nbrAOSdescr);
            tiin_cAOS = (TelosIndividual)tain_cAOS.getDestination();
            issoc = IsSociety(tiin_cAOS.getName());
            if (issoc) {
                        socmenuhtmlwrite(tiin_cAOS);
                        }

                  else  {
                        aghtmlwrite(tiin_cAOS);
                        }

            nbrAOSdescr++;
            }
      }

/*******************
* socmenuhtmlwrite *
********************/

public  void socmenuhtmlwrite
      ( TelosIndividual tiinsociety ) throws Exception

      {
      String out;

      out ="<BR><U><FONT COLOR=\"#FFCC99\"><A HREF=\""
                  +nomfichhtml+"#"+tiinsociety+"\" TARGET=\"data\">"
                  +"<h3>"+getattrAUOobject(tiinsociety,"name")
                  +" (society)"+"</h3>"
```

```
                +"</A></FONT></U>"+'\n';
        dataout.writeBytes(out);

        out = "<P><FONT COLOR=\"#000000\">**********</FONT>";
        dataout.writeBytes(out);

        cAgSocget(tiinsociety);

        out = "<P><FONT COLOR=\"#000000\">**********</FONT>";
        dataout.writeBytes(out);

        }

/**************
* aghtmlwrite *
***************/

public  void aghtmlwrite
        (TelosIndividual tiagent ) throws Exception

        {
        String out;

        if (ext.equals("mhtml"))
                {

                out ="<BR><U><FONT COLOR=\"#FFCC99\"><A HREF=\""
                      +nomfichhtml+"#"+tiagent+"\" TARGET=\"data\">"
                      +"<h5>"+getattrAUOobject(tiagent,"name")
                      +" (agent)"+"</h5>"
                      +"</A></FONT></U>"+'\n';
                dataout.writeBytes(out);
                }
        }

/****************
* htmlmenuwrite *
*****************/

public  void htmlmenuwrite()
        throws Exception

        {
        String out,menufile,datafile;
        String path,extension,fichier;

        out= "<HTML>"+"\n"
        +"<HEAD>\n"
        +"<META HTTP-EQUIV=\"Content-Type\" CONTENT=\"text/html;
charset=iso-8859-1\">\n"
        +" <META NAME=\"GENERATOR\" CONTENT=\"Mozilla/4.03 [en] (WinNT; I)
[Netscape]\">\n"
        +"<TITLE>Society et agent</TITLE>\n"
        +"</HEAD>\n\n";
        dataout.writeBytes(out);

        }
```

```
/****************
* htmldatawrite *
*****************/

public  void htmldatawrite()
      throws Exception

      {
      String out,menufile,datafile;
      String path,extension,fichier;


      out= "<HTML>"+"\n"
      +"<HEAD>\n"
      +"<META HTTP-EQUIV=\"Content-Type\" CONTENT=\"text/html;
charset=iso-8859-1\">\n"
      +" <META NAME=\"GENERATOR\" CONTENT=\"Mozilla/4.03 [en] (WinNT; I)
[Netscape]\">\n"
      +"<TITLE>description</TITLE>\n"
      +"</HEAD>\n\n";
      dataout.writeBytes(out);

      }

/*****************
* htmlindexwrite *
******************/

public  void htmlindexwrite(String namefile)
      throws Exception

      {
      String out,menufile,datafile;
      String path,extension,fichier;

      Filename fichhtml;
      fichhtml = new Filename(namefile,'\\');
      extension = fichhtml.extension();
      path = fichhtml.path();
      menufile = "\\m"+fichhtml.filename()+'.'+extension;
      datafile =  "\\d"+fichhtml.filename()+'.'+extension;


      out= "<HTML>"+"\n"
      +"<HEAD>\n"
      +"<META HTTP-EQUIV=\"Content-Type\" CONTENT=\"text/html;
                                  charset=iso-8859-1\">\n"
      +" <META NAME=\"GENERATOR\" CONTENT=\"Mozilla/4.03 [en]
                                 (WinNT; I) [Netscape]\">\n"
      +"<TITLE>Description de la spec</TITLE>\n"
      +"<FRAMESET COLS=\"200,*\" frameborder=1
                                 framespacing=1 border=1>\n"
      +"<FRAME NAME=\"menu\" SRC=\""+menufile+"\" NORESIZE>\n"
      +"<FRAME NAME=\"data\" SRC=\""+datafile+"\" NORESIZE>\n"
      +"</FRAMESET>\n"
      +"</HEAD>\n\n"
```

```
      +"</BODY>\n"
      +"</HTML>\n";
      dataout.writeBytes(out);

      }

/*********
* AtSpec *
**********/

public  TelosVector AtSpec
            ( TelosObject tospec,
              String tattribute) throws Exception

      {

      TelosIndividual tiASpec;
      TelosObject toASpec;
      TelosAttribute taASpec;
      TelosVector tvinASpec;

      TelosAttribute tatmp;
      TelosIndividual tiattrd;

      int nbrASpec;
      String out  ;
      String auxout;
      String otherattribute;

      tiASpec = base.getIndividual("AlbertSpecification");
      toASpec = tiASpec.getSource();
      taASpec = toASpec.getAttribute(tattribute);
      tvinASpec = taASpec.getInstancesWithSource(tospec);

      otherattribute = "";


      if (tattribute.equals("userType"))
             {otherattribute ="type défini par l'utilisateur";}
      if (tattribute.equals("userOperation"))
             {otherattribute= "opération défini par l'utilisateur";}
      if (tattribute.equals("rootAgentSociety"))
             {otherattribute= "rootSociety";}
      if (ext.equals("dhtml"))
            {
            if(tattribute.equals("rootAgentSociety"))
                  {
                  out = "<BR><h2>Société racine : </h2> ";
                  auxout = "";
                  }
            else
                  {
                  out = "<BR><h2>Liste des "+otherattribute
                              +"("+tattribute+") :"+" </h2> "+"<BR>";
                  auxout = "";
                  }
            }
```

```
      else
            {
            if(tattribute.equals("rootAgentSociety"))
                  {
                  out = "\nSociété racine : \n";
                  auxout =     sousligne(out.length(),"=")+'\n'+'\n';
                  }
            else
                  {
                  out = "\nListe des "
                        +otherattribute+"("+tattribute+") :"+'\n';
                  auxout =     sousligne(out.length(),"=")+'\n'+'\n';
                  }
            }
      dataout.writeBytes(out);

      dataout.writeBytes(auxout);

      nbrASpec =0;
      while (nbrASpec < tvinASpec.size())
            {
            tatmp = (TelosAttribute)tvinASpec.getAt(nbrASpec);
            tiattrd =(TelosIndividual)tatmp.getDestination();
            auxout = getattrAUOobject(tiattrd,"name");

            if (auxout != "")
                  {
                  if (ext.equals("dhtml") )
                        {
                        out = "     "+auxout+"<BR>"
                              +"     "
                              +getattrAUOobject(tiattrd,"description")
                              +"<BR>"+"<BR>";
                        }
                  else
                        {
                        out = "     "+auxout+'\n'
                              +"     "
                              +getattrAUOobject(tiattrd,"description")
                              +'\n'+'\n';
                        }
                  dataout.writeBytes(out);
                  }
            nbrASpec++;

            }

      return tvinASpec;
      }

/***************
* getattragent *
***************/

public  void getattragent
            (TelosIndividual tiag,
                  String attribut)
```

```
            throws Exception

{
TelosIndividual tiAgent;
TelosObject toAgent;
TelosObject toattragent;
TelosAttribute taattragent;
TelosIndividual tiAUO;

TelosVector tvinattragent;

TelosAttribute tainattragent;
TelosIndividual tiinattragent;

int nbrattragent;
String nameattr;
String out  ;

tiAgent = base.getIndividual("AlbertAgent");
toAgent =(TelosObject)tiAgent;
taattragent = toAgent.getAttribute(attribut);
tvinattragent = taattragent.getInstancesWithSource(tiag);

nbrattragent = 0;
nameattr ="";

if (attribut.equals("action"))
            {     nameattr ="Action";}
if (attribut.equals("stateComponent") )
            {     nameattr ="Composants d'état";}
if (tvinattragent.size() > 0)
      {
      if (ext.equals("dhtml"))
            {
            out = "<BR><BR><h5>Liste des "+nameattr+"s du "
            +getattrAUOobject(tiag,"name")
            +" :  <BR></h5>";//,,,,,,
            dataout.writeBytes(out);
            }
      else
            {
            out = "\n\nListe des "+nameattr+"s du "
                  +getattrAUOobject(tiag,"name")
                  +     " : \n";
            dataout.writeBytes(out);
            out = sousligne(out.length(),".")+"\n";
            dataout.writeBytes(out);

            }

      }

while (nbrattragent < tvinattragent.size())
      {
      tainattragent =
            (TelosAttribute)tvinattragent.getAt(nbrattragent);
      tiinattragent =
```

```
            (TelosIndividual)tainattragent.getDestination();

      if (ext.equals("dhtml"))
            {
            out = "<BR> "+nameattr+" "
            +getattrAUOobject(tiinattragent,"name")+"<BR>";
            dataout.writeBytes(out);

            out = getattrAUOobject(tiinattragent,"description")
                  +"<BR><BR>";
            dataout.writeBytes(out);

            }
      else
            {
            out = "\n "+nameattr+" "
                  +getattrAUOobject(tiinattragent,"name")+'\n';
            dataout.writeBytes(out);

            out = getattrAUOobject(tiinattragent,"description")
                  +'\n'+'\n';
            dataout.writeBytes(out);

            }

if (attribut.equals("stateComponent") )
            {

agentstateconstraintwrite(tiinattragent,"derivation");
agentstateconstraintwrite(tiinattragent,"initialValuation");

wagentstateBehaviour(tiinattragent);

agentexportpercwrite
      ((TelosIndividual)tiinattragent.getDestination()
                                    ,"AlbertStateComponent");
agentexportinfowrite
      ((TelosIndividual)tainattragent.getDestination()
                                    ,"AlbertStateComponent");
            }
if (attribut.equals("action"))
            {

agentactionconstraintwrite(tiinattragent,"composition");
agentactionconstraintwrite(tiinattragent,"duration");
agentactionconstraintwrite(tiinattragent,"precondition");
agentactionconstraintwrite(tiinattragent,"effect");
agentactionconstraintwrite(tiinattragent,"triggering");

agentexportpercwrite
      ((TelosIndividual)tiinattragent.getDestination()
                                          ,"AlbertAction");
agentexportinfowrite
      ((TelosIndividual)tiinattragent.getDestination()
                                          ,"AlbertAction");
            }
      nbrattragent++;
```

```
            if (ext.equals("dhtml"))
                  {
                  dataout.writeBytes("<BR><BR>");
                  }
            else
                  {
                  dataout.writeBytes("\n\n");
                  }

      }

      }


/************************
* wagentstateBehaviour *
************************/

public void wagentstateBehaviour
            (TelosIndividual tiinstate) throws Exception
      {
      TelosIndividual tiAgent;
      TelosObject toAgent;

      TelosAttribute tasb;
      TelosVector tvinsb;
      int nbrsb;

      TelosAttribute taST;
      TelosIndividual tiST;
      TelosAttribute tainvolvedSC;
      TelosIndividual tiStateBehaviour;
      TelosObject toStateBehaviour;
      TelosVector tvST;

      int nbrst;

      String out  ;
      String auxout     ;

      tiStateBehaviour = base.getIndividual("AlbertStateBehaviour");
      toStateBehaviour = (TelosObject)tiStateBehaviour;
      tainvolvedSC =
            toStateBehaviour.getAttribute("involvedStateComponent");
      tvST = tainvolvedSC.getInstancesWithDestination(tiinstate);


      tiAgent = base.getIndividual("AlbertAgent");
      toAgent =(TelosObject)tiAgent;
      tasb = toAgent.getAttribute("stateBehaviour");

      nbrsb = 0;
      out ="";
```

```
      while (nbrsb < tvST.size())
            {
            taST = (TelosAttribute)tvST.getAt(nbrsb);
            tiST = (TelosIndividual)taST.getSource();


            if (ext.equals("dhtml"))
                  {
                  out = out
                        +getattrAUOobject(tiST,"description")
                        +"<BR>";
                  }
            else
                  {
                  out = out
                        +getattrAUOobject(tiST,"description")
                        +'\n';
                  }

            nbrsb++;
            }

      if (out !="")
            {
            if (ext.equals("dhtml"))
                  {
                  auxout ="   contrainte de comportement"
                        + "d'état(statebehaviour)"
                        +" :<BR>";
                  }
            else
                  {
                  auxout ="   contrainte de comportement "
                        +"d'état(statebehaviour)"+" :\n";
                  }
            dataout.writeBytes(auxout);
            dataout.writeBytes("    "+out);
            }
      }

/*******************
* getattrAUOobject *
*******************/

 public  String getattrAUOobject
      (TelosIndividual ti,
      String attribut) throws Exception

      {
      TelosObject toAUO;
      TelosIndividual tiAUO;
      TelosVector tvAUO;
      TelosAttribute taname;
      int nbrname;
      String strout;
```

```
      tiAUO = base.getIndividual("AlbertUserObject");
      toAUO = (TelosObject)tiAUO;
      taname = toAUO.getAttribute(attribut);
      tvAUO = taname.getInstancesWithSource(ti);
      nbrname = 0;
      strout = "";
      while (nbrname < tvAUO.size())
            {
            strout = strout +enleverd((String)((TelosToken)
                  tvAUO.getAt(nbrname).getDestination()).getValue());
            nbrname++;
            }
      return strout;
      }

/*****************************
* containedAgentOrSocietyget *
*****************************/

  public  void containedAgentOrSocietyget
            ( TelosIndividual tisoc) throws Exception

      {
      TelosIndividual tiAS;
      TelosObject toAS;
      TelosAttribute tacAOS;
      TelosVector tvin_acAOS;

      TelosAttribute tain_cAOS;
      TelosIndividual tiin_cAOS;
      int nbrAOSdescr;
      boolean issoc;

      tiAS = base.getIndividual("AlbertSociety");
      tacAOS = tiAS.getAttribute("containedAgentOrSociety");
      tvin_acAOS = tacAOS.getInstancesWithSource(tisoc.getSource());

      nbrAOSdescr =0;

      while (nbrAOSdescr < tvin_acAOS.size())
            {
            tain_cAOS = (TelosAttribute)tvin_acAOS.getAt(nbrAOSdescr);
            tiin_cAOS = (TelosIndividual)tain_cAOS.getDestination();
            issoc = IsSociety(tiin_cAOS.getName());
            if (issoc) {
                        analysesociety(tiin_cAOS);
                        }

                  else  {
                        analyseagent(tiin_cAOS);
                        }

            nbrAOSdescr++;
            }
      }
```

```
/***************
* analyseagent *
****************/

public  void analyseagent
      (TelosIndividual tiagent ) throws Exception

      {
      TelosVector tvinAct;
      TelosVector tvinScom;
      String out,outaux;

      if (ext.equals("dhtml"))
            {
            out ="<A NAME= \""+tiagent+"\"></A>"+ "<h4><BR>Agent : "
                  +getattrAUOobject(tiagent,"name")+"</h4><BR>";
            dataout.writeBytes(out);

            out = " <BR> <BR> "
                  +getattrAUOobject(tiagent,"description")
                  +"<BR>";
            dataout.writeBytes(out);


            }
      else
            {

            out = "\nAgent : "
                  +getattrAUOobject(tiagent,"name")+"\n";
            dataout.writeBytes(out);

            out=sousligne(out.length(),"-");
            dataout.writeBytes(out);

            out = "\n "+'\n'
                  +getattrAUOobject(tiagent,"description");
            dataout.writeBytes(out);


            }

       getattragent(tiagent,"action");
       getattragent(tiagent,"stateComponent");


      }
```

```
/*******************
* socdatahtmlwrite *
*******************/

public  void socdatahtmlwrite
      ( TelosIndividual tiinsociety ) throws Exception

      {
      String out;

      out = "<A NAME= \""
                  +tiinsociety+"\"></A>"+"<BR><h3>Société(Society) : "
                  +getattrAUOobject(tiinsociety,"name")+"</h3>";
      dataout.writeBytes(out);

      out = "<BR>"
            +getattrAUOobject(tiinsociety,"description");
      dataout.writeBytes(out);

      out = "<P><FONT COLOR=\"#000000\">**********************</FONT>";
      dataout.writeBytes(out);

      containedAgentOrSocietyget(tiinsociety);

      out = "<P><FONT COLOR=\"#000000\">fin"
                  +"  de la description de la société"
                  +getattrAUOobject(tiinsociety,"name")+"</FONT>";
      dataout.writeBytes(out);

      }


/**************
* socdocwrite *
***************/

public  void socdocwrite
      ( TelosIndividual tiinsociety ) throws Exception

      {
      String out;

      out = "Société(Society): "
            +getattrAUOobject(tiinsociety,"name");
      out =out + "\n"+sousligne(out.length(),"=");
      dataout.writeBytes(out);
      out = "\n\n"
            +getattrAUOobject(tiinsociety,"description");
      dataout.writeBytes(out);

      containedAgentOrSocietyget(tiinsociety);
      out = "\n\nfin de la description de la"+
             "société"+getattrAUOobject(tiinsociety,"name")+"\f\n\n";
      dataout.writeBytes(out);

      }
```

```
/*****************
* analysesociety *
******************/

public  void analysesociety
      (TelosIndividual tiinsociety ) throws Exception

      {
      String out;

      if (ext.equals("mhtml"))
            {
            socmenuhtmlwrite(tiinsociety);
            }

      if (ext.equals("dhtml"))
            {
            socdatahtmlwrite(tiinsociety);
            }

      if (  !(ext.equals("dhtml") || ext.equals("mhtml")) )
            {
            socdocwrite(tiinsociety);
            }

      }

public  String enleverd(String descr)
            {
            if (descr.equals(" END_SOCIETY")
            ||descr.equals("END_SOCIETY")
            || descr.equals(" END_BASIC_TYPES")
            || descr.equals("END_BASIC_TYPES")
            || descr.equals(" END_CONSTRUCTED_TYPES")
            || descr.equals("END_CONSTRUCTED_TYPES")
            || descr.equals("END_OPERATIONS")
            || descr.equals(" END_OPERATIONS")
            || descr.equals(" END_CONSTRAINTS")
            || descr.equals("END_CONSTRAINTS")
            || descr.equals(" END_STATE_COMPONENTS")
            || descr.equals("END_STATE_COMPONENTS")
            || descr.equals(" END_ACTIONS")
            || descr.equals("END_ACTIONS")
            || descr.equals(" BASIC_CONSTRAINTS")
            || descr.equals("BASIC_CONSTRAINTS")
            || descr.equals(" DECLARATIVE CONSTRAINTS")
            || descr.equals("DECLARATIVE CONSTRAINTS")
            || descr.equals(" OPERATIONAL CONSTRAINTS")
            || descr.equals("OPERATIONAL CONSTRAINTS")
            || descr.equals(" COOPERATION CONSTRAINTS")
            || descr.equals("COOPERATION CONSTRAINTS")
            || descr.equals(" END_CONSTRAINTS")
            || descr.equals("END_CONSTRAINTS")
            || descr.equals(" END_AGENT")
            || descr.equals("END_AGENT")
```

```
            || descr.equals("END_SPEC")
            || descr.equals(" END_SPEC")

            )
                  {
                  return "";
                  }
            else  {
                  return descr;
                  }
            }
public  boolean IsSociety
      (String soc)

            {
            String subname;
            boolean issoc;
            int underscor1 =soc.indexOf('_');

            if ( soc.regionMatches(underscor1+1,"Society",0,7) )
                  {
                  issoc = true;
                  }
            else  {
                  issoc = false;
                  }
            return issoc;
            }


/*****************************
* agentactionconstraintwrite *
*****************************/

public  void agentactionconstraintwrite
            (TelosIndividual tiaction,
            String attribute) throws Exception

      {
      String out,auxout;
      String otherattribute;
      otherattribute = "";

      if (attribute.equals("duration"))
            {
            otherattribute = "de durée(duration)";
            }
      if (attribute.equals("composition"))
            {
            otherattribute = "de composition";
            }

      if (attribute.equals("triggering"))
            {
            otherattribute = "de triggering";
            }
```

```
      if (attribute.equals("effect"))
            {
            otherattribute = "d'effets(effect)";
            }

      if (attribute.equals("precondition"))
            {
            otherattribute = "de précondition(precondition)";
            }

      auxout = agentactionconstraintget(tiaction,attribute);

      if (auxout != "")
            {
            if (ext.equals("dhtml"))
                  {
                  out = "     contrainte "+otherattribute
                        +" sur "+getattrAUOobject(tiaction,"name")
                        +" : <BR>";
                  dataout.writeBytes(out);

                  }
            else
                  {
                  out = "     contrainte "+otherattribute
                        +" sur "+getattrAUOobject(tiaction,"name")
                        +" : \n";
                  dataout.writeBytes(out);
                  }

            dataout.writeBytes("    "+auxout);
            }
      }

/***************************
* agentactionconstraintget *
***************************/

public  String agentactionconstraintget
      (TelosIndividual tiinact,
      String attribute) throws Exception

      {
      TelosIndividual tiAAction;
      TelosObject toAAction;
      TelosAttribute taAAction;

      TelosVector tvattrconst;

      TelosAttribute tainattr;
      TelosIndividual tiinattr;

      String actout;
      int nbrconstr;
      int nbract;

      actout ="";
```

```
      tiAAction = base.getIndividual("AlbertAction");
      toAAction = (TelosObject)tiAAction;
      taAAction = toAAction.getAttribute(attribute);


            tvattrconst =
                  taAAction.getInstancesWithSource(tiinact.getSource());

            nbrconstr = 0;

            while (nbrconstr < tvattrconst.size())
                  {
                  tainattr =
                        (TelosAttribute)tvattrconst.getAt(nbrconstr);
                  tiinattr = (TelosIndividual)tainattr.getDestination();
                  if(ext.equals("dhtml"))
                        {
                        actout = actout
                              +getattrAUOobject(tiinattr,"description")
                              +"<BR><BR>";
                        }
                  else
                        {
                        actout = actout
                              +getattrAUOobject(tiinattr,"description")
                              +"\n\n";
                        }
                  nbrconstr++;
                  }

      return actout;
      }

/*****************************
* agentstateconstraintwrite *
*****************************/

public  void agentstateconstraintwrite
            (TelosIndividual tiScomp,
            String attribute) throws Exception

      {
      String out,outaux;
      String otherattribute;
      otherattribute = "";
      if (attribute.equals("derivation"))
            {
            otherattribute = "de dérivation(derivation)";
            }
      if (attribute.equals("initialValuation"))
            {
            otherattribute = "de valeur initial(initialValuation)";
            }


      outaux = agentstateconstraintget(tiScomp,attribute);
```

```
      if (outaux != "")
            {
            if (ext.equals("dhtml"))
                  {
                  out = "     contrainte "+otherattribute
                        +" sur "+getattrAUOobject(tiScomp,"name")
                        +" : <BR>";
                  }
            else
                  {
                  out = "     contrainte "+otherattribute
                        +" sur "+getattrAUOobject(tiScomp,"name")
                        +" :\n";

                  }

            dataout.writeBytes(out);
            dataout.writeBytes("    "+outaux);
            }
      }

/*************************
* agentstateconstraintget *
*************************/

public  String agentstateconstraintget
            (TelosIndividual tiinScomp,
            String attribute) throws Exception

      {
      TelosIndividual tiScomp;
      TelosObject toScomp;

      TelosAttribute taScomp;
      TelosVector tvinconScomp;

      TelosAttribute taattrconst;
      TelosIndividual tiattrconst;

      String stout;
      int nbrconstr;

      tiScomp = base.getIndividual("AlbertStateComponent");
      toScomp = (TelosObject)tiScomp;
      taScomp = toScomp.getAttribute(attribute);

      stout = "";

            tvinconScomp =
                  taScomp.getInstancesWithSource(tiinScomp.getSource());

            nbrconstr = 0;
```

```
            while (nbrconstr < tvinconScomp.size())
                  {
                  taattrconst =
                        (TelosAttribute)tvinconScomp.getAt(nbrconstr);
                  tiattrconst =
                        (TelosIndividual)taattrconst.getDestination();
                  if(ext.equals("dhtml"))
                        {
                        stout = stout
                              +getattrAUOobject(tiattrconst,"description")
                              +"<BR><BR>";
                        }
                  else  {
                        stout = stout
                              +getattrAUOobject(tiattrconst,"description")
                              +"\n\n";
                        }
                  nbrconstr++;
                  }
      return stout;
      }

/************************
* agentexportpercwrite *
************************/

public  void agentexportpercwrite
            (TelosIndividual tiexpoerc,
            String baseattr) throws Exception

      {
      String out,auxout;
      String otherattribute;

      if (baseattr != "")
            {
            otherattribute = baseattr;
            }
      else {otherattribute = baseattr;
            }

      auxout = agentexportpercinfget(tiexpoerc,baseattr,"perception");

      if (auxout != "")
            {
            if (ext.equals("dhtml"))
                  {
                  out = "     contrainte de perception sur "
                        +getattrAUOobject(tiexpoerc,"name")+" :<BR>";
                  }
            else
                  {
                  out = "     contrainte de perception sur "
                        +getattrAUOobject(tiexpoerc,"name")+" :\n";
                  }
            dataout.writeBytes(out);
```

```
            dataout.writeBytes("    "+auxout);
            }
      }

/************************
* agentexportpercinfget *
************************/

public  String agentexportpercinfget
      (TelosIndividual tiperc,
      String baseattr,String percinf) throws Exception

      {
      TelosIndividual tiAOexp;
      TelosAttribute taAOexp;

      TelosAttribute taPercInf;

      TelosVector tvinAOexp;

      TelosAttribute tainAOexp;

      TelosObject toAOexp;

      TelosVector tvPercInf;

      TelosAttribute tainPercInf;
      TelosIndividual tiinPercInf;

      int nbrexport,nbrperc;

      String expout;

      tiAOexp = base.getIndividual(baseattr);
      taAOexp = tiAOexp.getAttribute("exportsTo");

      taPercInf = taAOexp.getAttribute(percinf);

      tvinAOexp = taAOexp.getInstancesWithSource(tiperc);

      nbrexport = 0;
      expout = "";

      while (nbrexport <      tvinAOexp.size())
            {
            tainAOexp = (TelosAttribute)tvinAOexp.getAt(nbrexport);
            tvPercInf = taPercInf.getInstancesWithSource(tainAOexp);

            nbrperc =0;
            while (nbrperc < tvPercInf.size() )
                  {
                  tainPercInf = (TelosAttribute)tvPercInf.getAt(nbrperc);
                  tiinPercInf =
                        (TelosIndividual)tainPercInf.getDestination();

                  if (ext.equals("dhtml"))
                        {
```

```
                        expout = expout
                              +getattrAUOobject(tiinPercInf,"description")
                              +"<BR>";
                        }
                        else
                        {
                        expout = expout
                              +getattrAUOobject(tiinPercInf,"description")
                              +"\n";
                        }

                  nbrperc++;
                  }
            nbrexport++;
            }
      return expout;
      }


/************************
* agentexportinfowrite *
************************/

public  void agentexportinfowrite
            (TelosIndividual tiActSta,
            String baseattr) throws Exception

      {
      String out,auxout;
      String otherattribute;

      if (baseattr != "")
            {
            otherattribute = baseattr;
            }
      else {otherattribute = baseattr;
            }

      auxout = agentexportpercinfget(tiActSta,baseattr,"information");

      if (auxout != "")
            {
            if (ext.equals("dhtml"))
                  {
                  out = "     contrainte d'information sur "
                        +getattrAUOobject(tiActSta,"name")+" :<BR>";
                  }
            else
                  {
                  out = "     contrainte d'information sur "
                        +getattrAUOobject(tiActSta,"name")+" :\n";
                  }

            dataout.writeBytes(out);
            dataout.writeBytes("    "+auxout);
            }
      }
```

```
public  String sousligne(int longeur ,String type)
      {
      String sout ;
      sout = "";
      while(0 <longeur)
            {
            sout = sout +type;
            longeur --;
            }
      return sout;
      }

}
```