Introduction aux langages ESTELLE, LOTOS et SDL et essai d'application à la gestion de réseaux

Mohammed Ouzzif, André Schaff

CENTRE de
RECHERCHE en
INFORMATIQUE de
NANCY

Campus Scientifique
Boîte Postale 239
54506 Vandoeuvre-les-Nancy Cedex
téléphone : 83.91.20.00

Introduction aux langages ESTELLE, LOTOS et SDL et essai d'application à la gestion de réseaux

Mohammed OUZZIF
et André SCHAFF

Résumé

La spécification des systèmes informatiques est l'une des étapes les plus importantes du cycle de vie du développement d'un système. Vu les exigences de plus en plus pointues en matière de concurrence, sécurité et conformité, les méthodes de description conventionnelles ne sont plus adaptées.
Les méthodes de spécification formelles s'appuient sur des techniques formelles permettant des modélisations précises, rigoureuses et testables.
Dans ce rapport, nous présentons les trois langages de spécification formelle normalisés : ESTELLE, LOTOS et SDL. Nous présentons aussi une spécification formelle du modèle gestionnaire-agent en utilisant ces langages.

Mots clés : spécification formelle, FDT, LOTOS, ESTELLE, SDL, gestion des réseaux.

Sommaire

I- Introduction

Les systèmes et applications informatiques sont devenus de plus en plus complexes. Il en est de même pour leurs spécifications. L'utilisation des langages conventionnelles (informelles) pour leurs descriptions n'est plus adaptée. En effet, ces langages ne permettent ni des descriptions claires à interprétation unique ni des tests de conformité des implémentations. Ceci est d'une très grande importance du moment qu'une simple omission ou erreur dans leurs descriptions peut être très coûteuse à rectifier et peut induire de graves conséquences.

Des méthodologies de spécification rigoureuses et puissantes ont été développées. Elles adoptent des techniques de description formelle (*FDT : Formal Description Techniques*) qui visent à assurer des spécifications claires, complètes, consistantes, exécutables et testables.

Ces techniques sont justifiées surtout pour les systèmes [1] :

- concurrents tels que les systèmes distribués,

- à sécurité critique tels que les systèmes bancaires,

- à précision pointue tels que les systèmes utilisés dans les domaines militaire, médecine, signalisation, etc.

- à qualité critique tels que les systèmes d'exploitation et de communication.

Elles servent aussi pour la description des standards internationaux de protocoles et services de communication. Trois langages ont été standardisés : (ESTELLE, LOTOS et SDL) et feront l'objet des paragraphes suivants. Tout au long de leur présentation, un exemple simple de spécification du modèle gestionnaire-agent [2] sera traité.

II ESTELLE

ESTELLE est un langage de spécification formelle développé pour la description des services et protocoles OSI [3] en particulier et, des systèmes distribués en général. Il est basé sur trois concepts principaux à savoir les modules, les canaux et la structuration d'un module en sous modules et la notion d'automates d'états fini étendu.

II-1 Modules

Un module ESTELLE peut être vu comme une boîte noire possédant des points d'interactions (ip) à partir desquels, il reçoit des entrées et envoie des sorties. Sa structure interne est représentée par une machine d'état fini. Chaque transition consomme une entrée, change d'état et produit une sortie (figure 1.).
ESTELLE permet aussi des transitions sans entrées, appelées transitions spontanées, qui dépendent de contraintes temporelles.
Un module possède soit une file pour chaque point d'interaction recevant des entrées, soit une file commune pour tous ces points d'interaction. Ces files vont servir pour garder les entrées avant leur traitement.

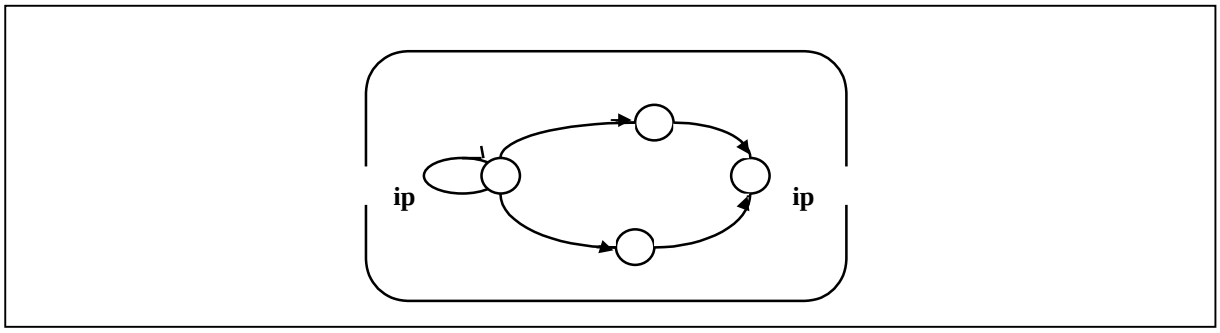

figure 1. concept module

Pour mieux voir ces concepts, on va traiter le modèle gestionnaire-agent [2] de la gestion des réseaux OSI (figure 2).

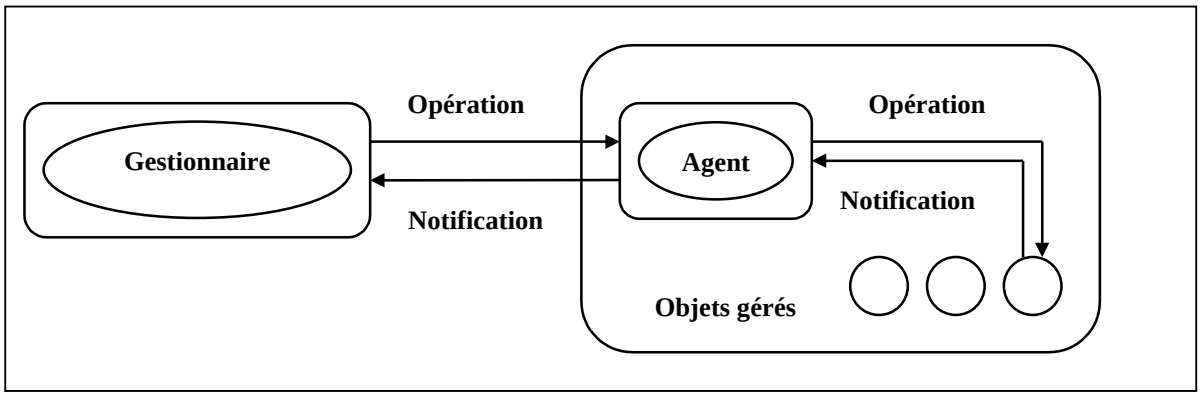

Figure 2. Modele gestionnaire-agent

Dans ce modèle, on a deux processus principaux : le processus gestionnaire et le processus agent. Le processus gestionnaire envoie des opérations à l'agent qui seront répercutées par ce dernier sur l'objet géré. L'objet géré, quant à lui, envoie des notifications à l'agent qui les transmet au gestionnaire.

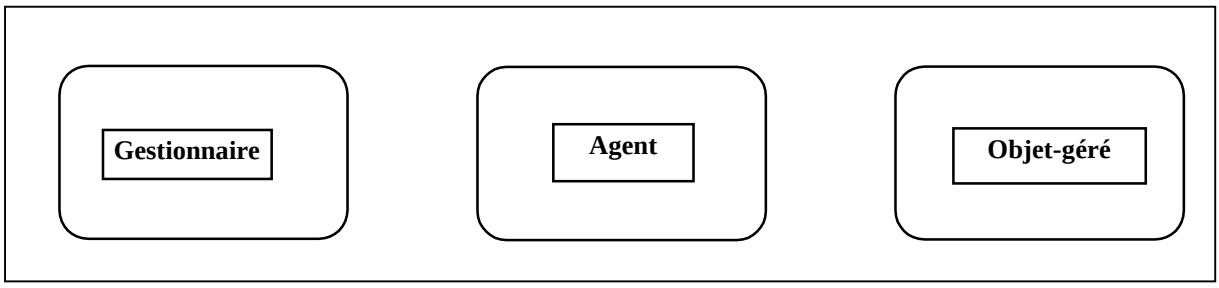

figure 3. Modules du modèle gestionnaire-agent

D'après cette spécification informelle, on peut faire une modélisation ESTELLE avec les trois modules suivants : le module « gestionnaire », le module « agent » et le module «objet-géré » (figure 3.).

Le module « gestionnaire » peut être modélisé comme un automate à deux états : « passif » et « actif » et à trois transitions. La notation d'une transition est « ip.entree/ip.sortie ». Un « **.** » indiquera une entrée ou sortie vide (figure 4.).

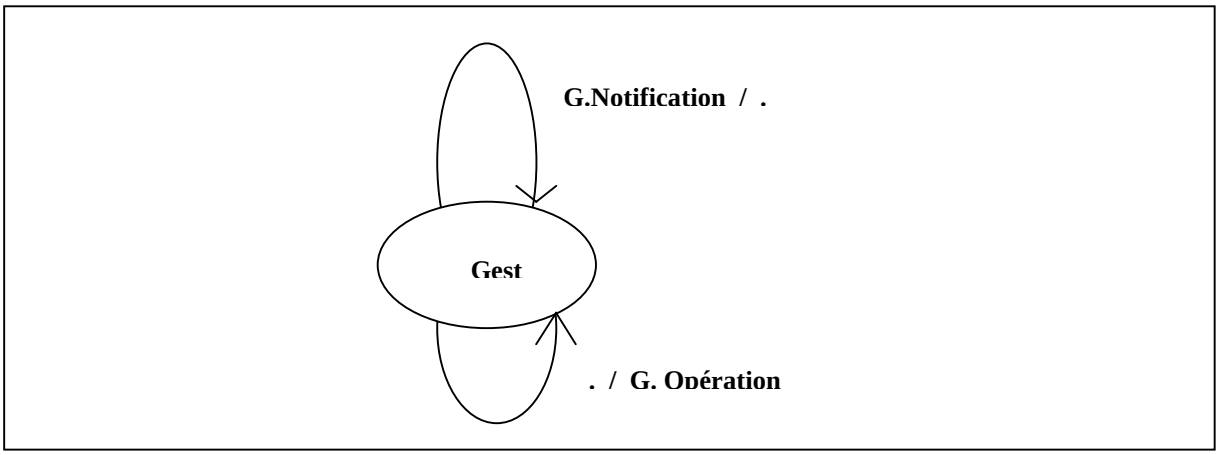

Figure 4. Automate du module gestionnaire

Ce module peut recevoir comme entrée, l'interaction «notification» par le point d'interaction G tout en restant dans le même état (Gest). Lorsqu'il détermine l'opération à effectuer, il émet en sortie l'opération via le point d'interaction G.
Pour les modules «agent » et « objet-géré», leurs automates seront constitués d'un seul état (figure 5).

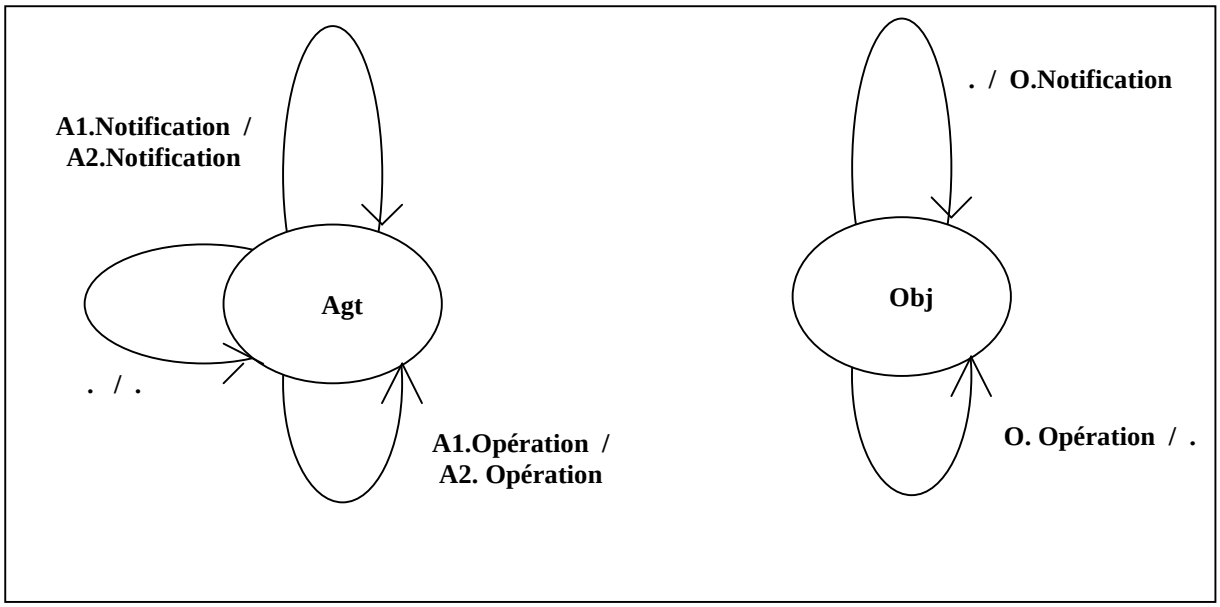

Figure 5. Module agent et Module objet-géré

II-2 Canaux

En ESTELLE, les modules communiquent à travers des canaux. Ces derniers possèdent deux extrémités liés aux points d'interactions des deux modules en question, qui chacun envoie ou reçoit des interactions (entrées/sorties) selon le rôle qui lui est assigné.

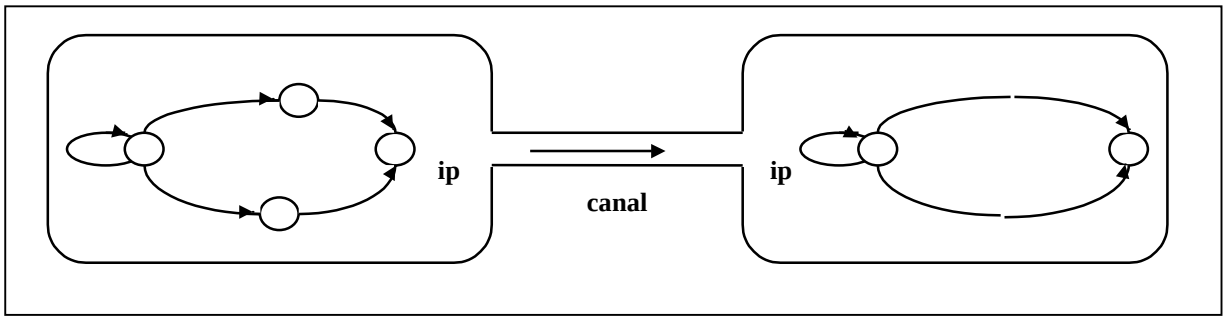

Figure 6. Canal ESTELLE

Dans notre exemple, on aura deux canaux : le canal «Gest-Agt » et le canal «Agt-Obj » (figure 7.).

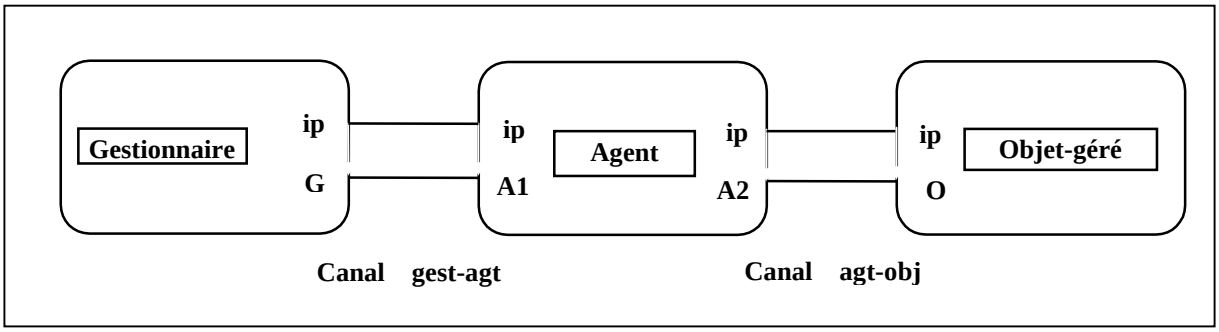

figure 7. Canaux du modèle gestionnaire-agent

Le gestionnaire va envoyer l'interaction « opération » et va recevoir l'interaction « notification » à travers le canal « gest-agt ». L'objet géré va envoyer l'interaction « opération » à travers le canal « agt-obj ». L'agent, quant à lui, va pouvoir envoyer l'interaction « notification » sur le canal « gest-agt » et va recevoir l'interaction « opération » sur le canal « agt-obj ».

II-3 Structuration

En ESTELLE, un module peut être structuré en sous modules. Cette structuration n'est pas vue de l'extérieur, i.e. le comportement d'un module est non déduit de sa structure interne.
La structuration introduit de nouveaux concepts, à savoir : module père, modules frères, ancêtre, descendant.
Un module peut communiquer avec ses modules fils. Pour cela, il est nécessaire d'avoir des mécanismes d'établissement de communication entre un père et les fils.
On distingue deux types de communication :

- attachement : un module père attache un de ses points d'interaction à un point d'interaction de l'un de ses fils (figure 8.).
- connexion : un module père connecte deux points d'interactions de deux de ses fils (figure 9.).

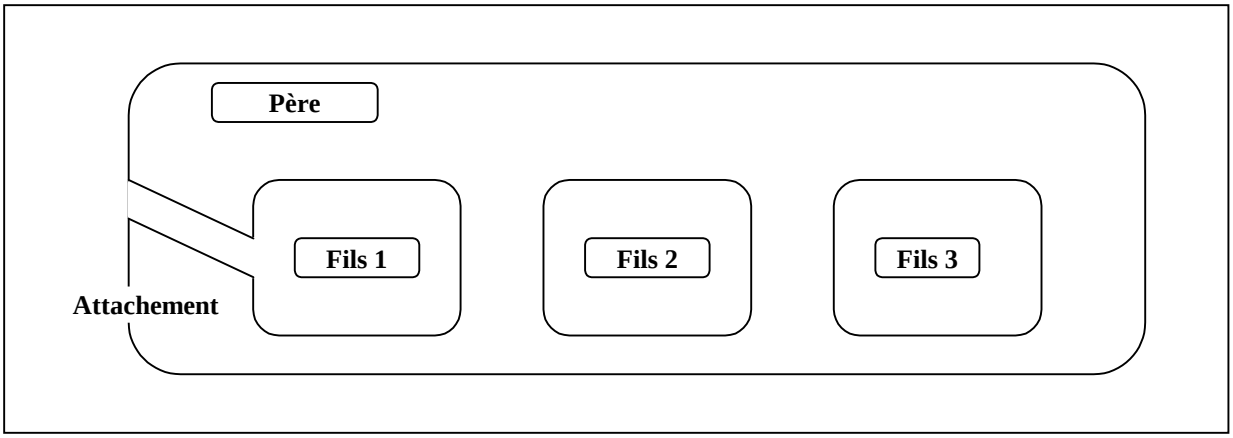

Figure 8. Attachement

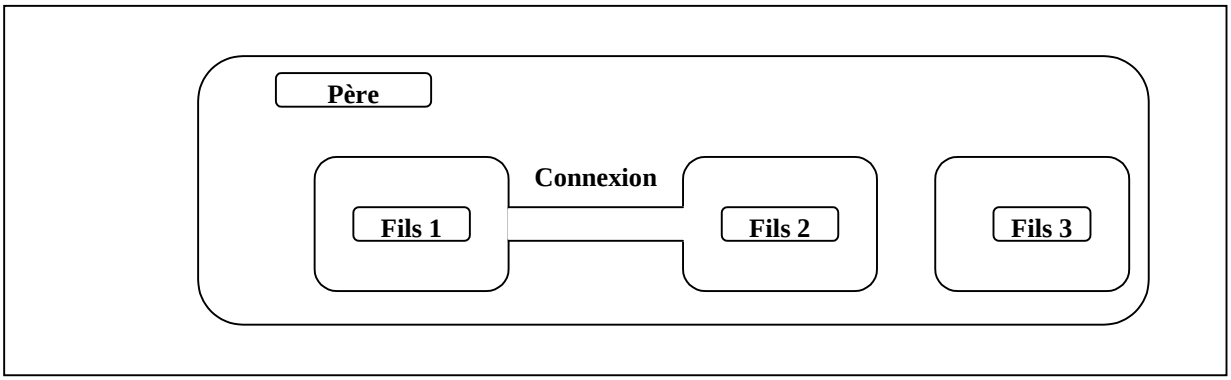

Figure 9. connexion

Il y a deux possibilités de synchronisation entre les modules fils d'un module père :

- les deux modules s'exécutent en parallèle si le module père est déclaré de classe « process » (voir paragraphe II-5-2).
- l'un des deux modules fils est exécuté si le module père est déclaré de « classe activity » (voir paragraphe II-5-2).

II-4 Systèmes de modules

En ESTELLE, un système peut être structuré en sous systèmes qui s'exécutent de manière asynchrone i.e. que les actions de l'un sont indépendantes de l'autre. Cependant, les actions dans un même sous système sont coordonnées.
Dans un sous système, la priorité est donnée aux ancêtres. En commençant par le module racine du sous système, si ce dernier a une transition, il la déclenche, sinon, s'il est déclaré de classe « process », il donne la possibilité à tous ses fils de déclencher leurs transitions ; s'il est déclaré de classe « activity », il choisit un fils à qui il donne la possibilité de déclencher une transition.

II-5 Langage de spécification ESTELLE

Ce paragraphe survole la syntaxe de exprimant les principaux concepts du langage ESTELLE.

II-5-1 Description d'un canal

La description d'un canal nomme les deux rôles associés aux modules se trouvant à ses extrémités. Il détermine les interactions que chacun de ces modules peut initier sur ce canal et les paramètres qui les accompagnent.

Exemple : pour notre modèle gestionnaire-agent (figure 7.), la description de ses canaux est :

```
Channel gest-agt (gest, agt)
    by  gest :
          operation;
    by  agt :
          notification;
```

Sur ce canal, deux interactions peuvent avoir lieu : « opération » envoyée par le module qui assure le rôle « gest » et « notification » envoyée par le module qui assure le rôle « agt ».

```
Channel agt-obj (agent, objet)
    by  agt :
          operation;
    by  obj :
          notification;
```

Sur ce canal, deux interactions peuvent être envoyées : « opération » envoyée par le module qui assure le rôle « agt » et « notification » envoyée par le module qui assure le rôle « obj ».

II-5-2 Description d'un module

La description d'un module est constituée de la description de son en-tête ainsi que la description de son corps.
L'en-tête d'un module identifie son nom et sa classe (activity ou process). Il décrit aussi les points d'interaction et le type de file adopté. En effet, l'en-tête établit le type d'un module.

Exemple :

Pour notre modèle gestionnaire-agent (figure 7.), la description de ses modules est :

```
Module  Gestionnaire  Activity;
    ip
        G : gest-agt (gestionnaie) individual queue;
end; {Gestionnaire}
```

Ce module possède un seul point d'interaction sur lequel le module jouant le rôle « gest » peut envoyer son interaction.
La description des deux autres modules (Agent et Objet-gere) est:

```
Module  Agent  Activity;
  ip
    A1 : gest-agt (gestionnaire) common queue;
    A2 : agt-obj (agent) common queue;  { Les deux points d'interaction ont
                                          la même queue }
end; {Agent}
```

```
Module  Objet-gere  Activity;
        ip
            O : agt-obj (objet) individual queue;
end; {Objet-gere}
```

Le corps d'un module identifie son en-tête et décrit les actions qu'il peut entreprendre sous forme de transitions. Ces transitions sont décrites à l'aide de clauses de garde.

Exemple de clauses de garde :

- ***When*** : spécifie un point d'interaction et l'interaction qui doit être à la tête de la queue associée au point d'interaction.
- ***From*** : spécifie l'état de contrôle dans lequel l'automate doit être avant la transition.
- ***To*** : spécifie l'état de contrôle dans lequel l'automate doit être après la transition.
- ***Provided*** : spécifie des conditions pour le déclenchement d'une transition (en dehors des entrées)

Il y a d'autres clauses (***priority***, ***any***, ***delay***) que nous ne décrivons pas dans ce rapport.

Exemple

```
Body Objet-gerebody for objet-gere;
        State Obj
        initialise      {*Partie initialisation du module*}
            to Obj
                begin
                end;
        trans           {*Partie des transitions  du module*}
            When O.operation
                from Obj to Obj
                    begin
                    end;
            Provided evenement
                from Obj to Obj
                    begin
                    output  O.Notification
                    end;
end;
```

Le corps du module « objet-gere » est constitué d'une partie initialisation à l'état « obj » et de la partie des transitions. La première transition reçoit sur le point d'interaction « O » une opération, elle reste dans le même état et ne donne rien en sortie. La deuxième transition est une transition spontanée qui reste dans le même état et donne en sortie une notification.
Les descriptions suivantes concernent celles des modules « agent » et « gestionnaire ».

```
Body Agentbody for agent;
        State Agt
        initialise      {Partie initialisation du module}
            to Agt
                begin
                end;
        trans           {Partie des transitions  du module}
            When A2.Notification
                from Agt to Agt
                    begin
                    output A1.Notification
                    end;
            When A1.operation
                from agt to Agt
                    begin
                    output A1.Operation
                    end;
end;
Body Gestionnairebody for gestionnaire;
        State Gest
        initialise      {Partie initialisation du module}
            to Gest
                begin
                end;
        trans           {Partie des transitions  du module}
            When G.Notification
                from Gest to Gest
                    begin
                    end;
            Provided  operat
            from Gest to Gest
                    begin
                    output G.Operation
                    end;
end;
```

III LOTOS

LOTOS a été développé pour spécifier les standards formels des protocoles et services OSI [4]. Il est structuré en deux parties nettement séparées. La première fournit le modèle comportemental dérivé de l'algèbre des processus. La deuxième partie permet la spécification des types de données abstraits et est basée sur le langage ACT ONE. On va surtout s'intéresser au premier aspect.

III-1 Vue générale

En LOTOS, les systèmes et leurs composants sont représentés par des ***processus.*** Ces derniers traduisent un comportement observable. Ils peuvent être vus comme des boîtes noires qui communiquent avec leur environnement externe via des portes (figure 10).

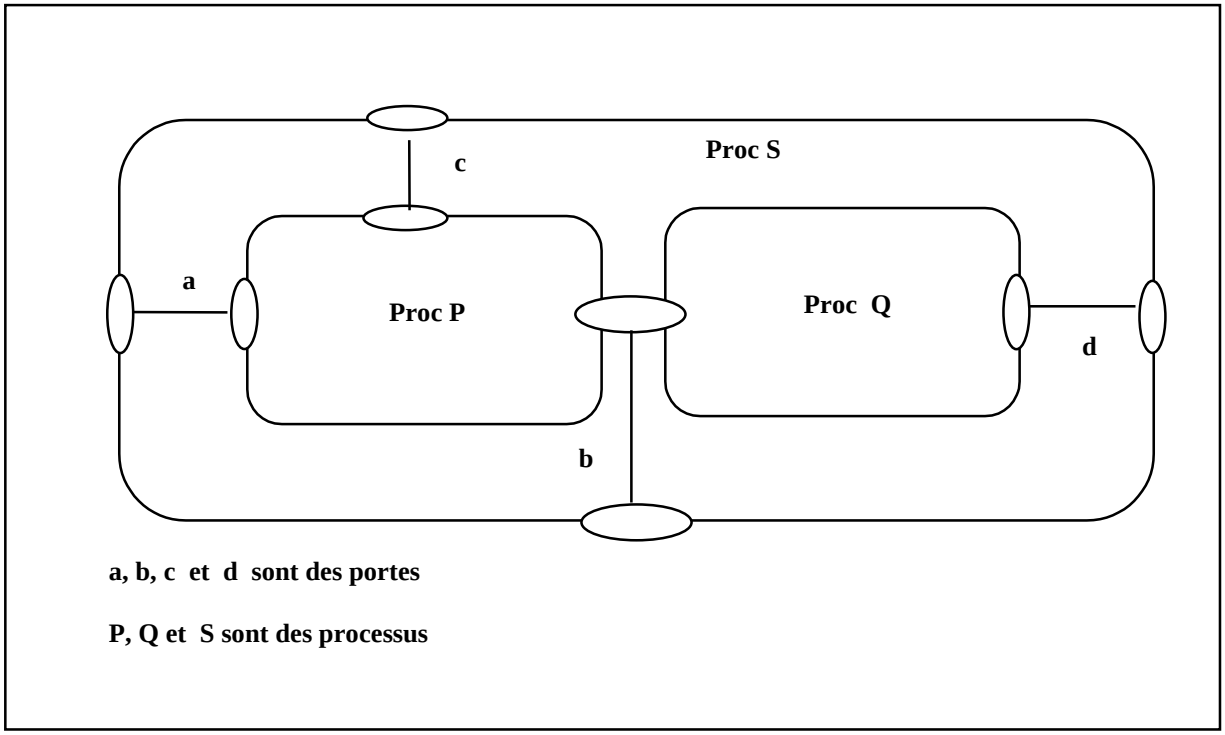

Figure 10. Exemple de structure de processus

Un comportement observable d'un système est décrit en LOTOS par des ***expressions de comportement.*** Une expression comportementale décrit l'ensemble des séquences d'interactions possibles dans lesquelles la participation du système en question est permise. Ces interactions sont représentées par des ***événements.*** Chaque événement est associé à une ***porte***.

La figure 10 pourrait correspondre aux définitions des processus suivants :

```
process S[a,b,c,d] : noexit :=

      P[a,b,c]|[b]|Q[b,d]                {*expression comportementale*}

where
process P[t,u,v] : noexit :=
      t; (u; stop   []   v;stop)     {*expression comportementale*}
endproc

process Q[x,y] : noexit :=
      x; y; stop                     {*expression comportementale*}
endproc
```

P[a,b,c] et Q[b,d] représentent une instanciation des processus P et Q. L'opérateur |[b]| entre P et Q signifie que ces deux processus interagissent via leur porte b.
Dans la suite, on explore les différents constructeurs de BASIC LOTOS qui permettent de construire les expressions comportementales. Et, pour illustrer ces constructeurs, on introduira en parallèle un exemple de système de contrôle d'accès. Ensuite, on fera une spécification du modèle gestionnaire-agent.

III-2 Basic LOTOS

LOTOS sans ACT ONE est appelé Basic LOTOS. Il fournit le modèle de comportement de LOTOS en se basant sur l'algèbre de processus.

III-2-1 Opérateurs de base

Les opérateurs de base de LOTOS permettent de construire des expressions de comportement complexe, sont :

- l'inaction : « stop »
- le rendez-vous : « ; »
- le choix : « [] »

L'opérateur « stop » dénote un comportement inactif. Il peut être utilisé pour décrire une impasse i.e. la situation où aucune action n'est possible.
L'opérateur « ; » exprime qu'un événement « a » par exemple doit survenir avant un comportement « B » : a ; B . Ceci est représenté par le graphe ci dessous.

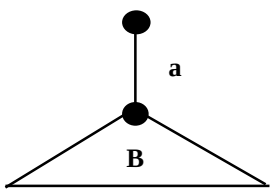

Figure 11. Graphe de l'opérateur « ; »

L'opérateur « choix » permet le choix entre deux comportements. Etant donné deux expressions de comportement B1 et B2. B1 [] B2 est représenté par le graphe suivant (figure 12).

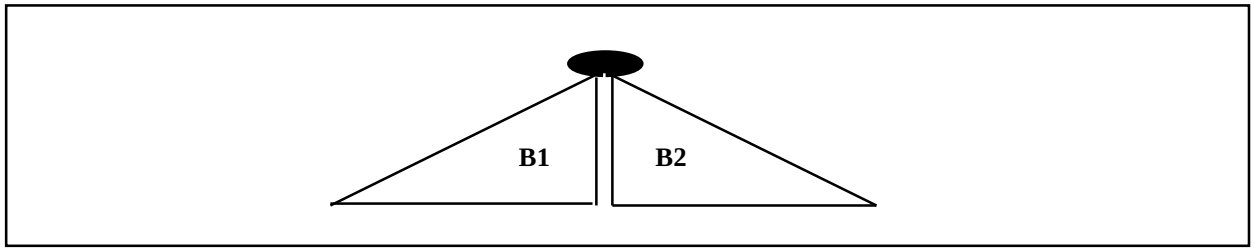

Figure 12. Graphe de l'opérateur « [] »

Exemple :

Considérons un système de contrôle d'accès qui requiert l'introduction de deux clés avant de permettre l'accès. Les clés ne peuvent être extraites avant l'accès.
La première étape est la définition de l'interface observable du système. Il y a cinq événements abstraits pour sa modélisation : In1, In2, Access, Out1 et Out2. Les deux premiers représentent l'insertion des clés, le troisième représente l'accès et les deux derniers représentent l'extraction.
Cet exemple peut être modélisé de plusieurs façons différentes dont voici deux illustrations au hasard.
Comme première modélisation, avec l'opérateur « ; », on peut spécifier ce système via quatre comportements (Figure 13).

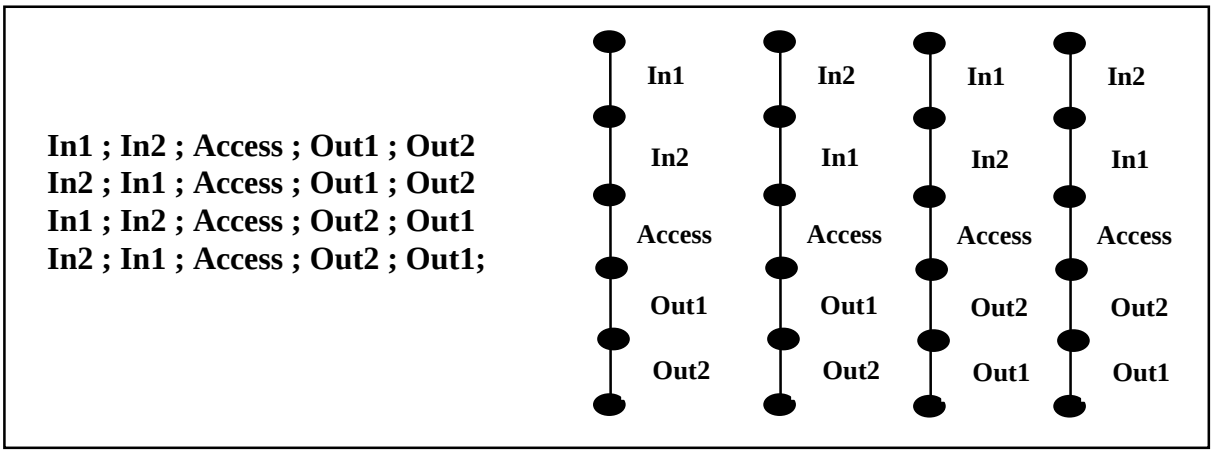

Figure 13. Modélisation du système de contrôle d'accès par l'opérateur « ; »

Une autre modélisation est possible en combinant les deux opérateurs « ; » et « [] » (figure 14). Cette combinaison va permettre une spécification complète de ce même système, ce qui n'est pas possible en utilisant uniquement l'opérateur « ; ».

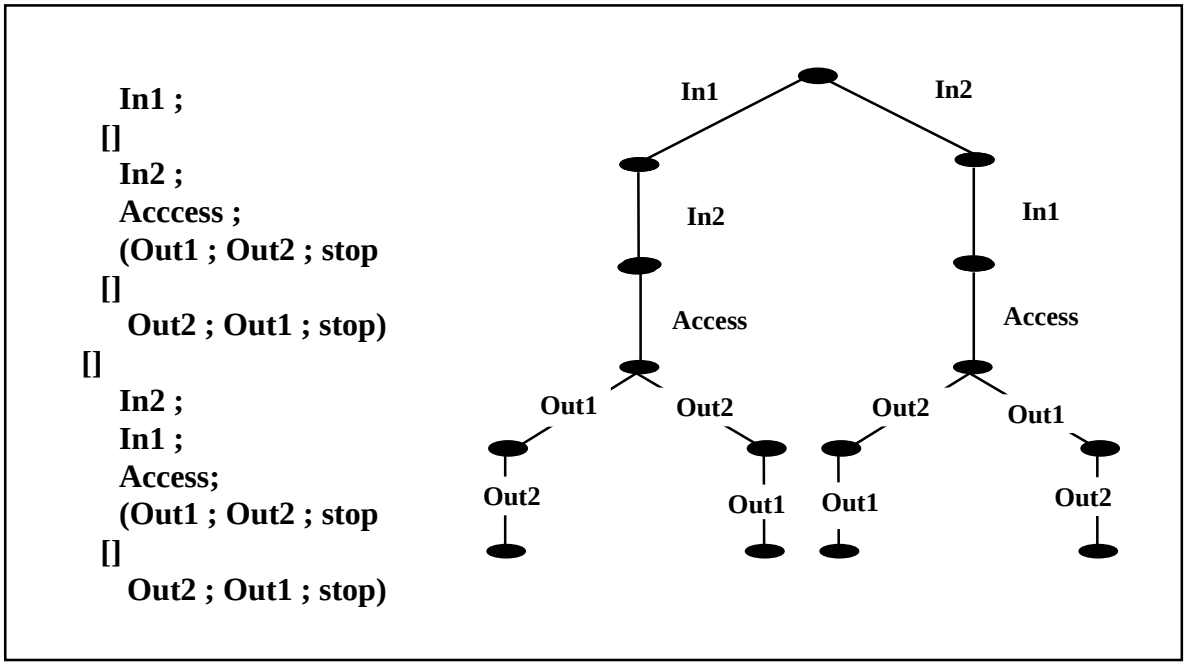

Figure 14. Modélisation du système de contrôle d'accès en combinant les operateurs « ; » et « [] »

III-2-2 Evénement interne et non déterminisme

En dehors des événements qui représentent l'interaction d'un système avec son environnement externe, LOTOS introduit la notion ***d'événement interne*** « i » qui exprime qu'un système peut faire une décision non observable, laquelle peut affecter son comportement futur.
LOTOS permet aussi l'expression du non déterminisme. Un comportement non déterministe se produit lorsque plusieurs comportements sont possibles et ne sont pas contrôlés par l'environnement externe. Il est modélisé en combinant le choix (« [] ») et la notion d'événement interne.
Soit « P » et « Q » deux expressions de comportement et « a » un événement. Trois façons possibles d'exprimer le non déterminisme sont présentées ci dessous :

(a ; P) [] (i ; Q)
(a ; P) [] (a ; Q)
(i ; P) [] (i ; Q)

III-2-3 Récursivité

Un comportement peut être répétitif. Ceci est modélisé en LOTOS avec des instanciations de processus de manière récursive.

Reprenons l'exemple du système de contrôle d'accès à deux clés. Ce système peut être modélisé de façon à ce qu'il permette des accès de manière infini. Cette modélisation se fait par l'instanciation d'un processus à l'intérieur d'un autre comme c'est le cas des deux processus « double-cle » et « Acc » (figure 15).

Double-cle [In1, In2, Acces, Out1, Out2]

Where

process Double-cle [I1, I2, Ac, O1, O2] : noexit :

I1; I2; Acc [I1, I2, Ac, O1, O2]
[]
I2; I1; Acc[I1, I2, Ac, O1, O2]

endproc (*Double-cle*)

process Acc [I1,I2,Ac,O1,O2] : noexit:

Access; O1; O2; Double-cle [I1, I2, Ac, O1, O2]
[]
Access; O2; O1; Double-cle [I1, I2, Ac, O1, O2]

endproc (*Acc*)

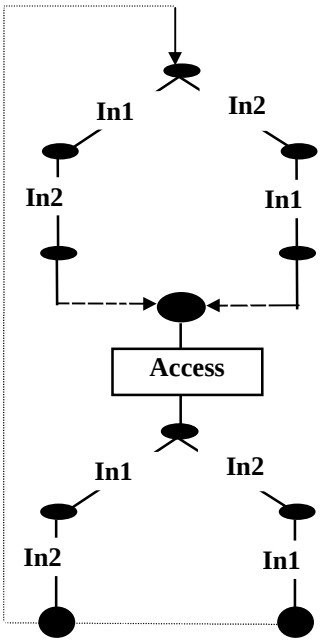

Figure 15. Modélisation du système de contrôle d'accès en utilisant la récursivité

Dans cette figure 15, le processus « Double-cle » attend l'événement d'introduction des clés puis fait appel au processus « Acc ». Ce dernier attend les événements accès et retrait des clés et fait appel au processus « Double-cle ».

III-2-4 Exit

Pour ce qui est de la terminaison, l'opérateur « stop » ne permet pas une séquence de processus. Pour remédier à cela, LOTOS modélise une terminaison avec succès à l'aide de l'opérateur « exit » représenté par **$**. Cet opérateur modélise une terminaison avec succès d'un processus.

III-2-5 Composition parallèle

Soient g1, g2, ..., gn des portes et B1 et B2 deux expressions de comportement. Dans l'expression générale **B1 | [g1, g2, ..., gn] | B2**, les événements qui appartiennent à la liste des portes ne peuvent se produire qu'avec la participation de Bl et B2 en même temps. Les événements qui n'y figurent pas peuvent se produire avec la participation d'un des deux.

Cas particulier :

Dans le cas où la liste des ports est vide, tous les événements se produisent avec l'une ou l'autre des deux expressions. Ceci est représente par l'opérateur ||| : **B1|||B2.**

L'opérateur || **: B1||B2,** représente le cas où les deux comportements se synchronisent sur toutes les portes.

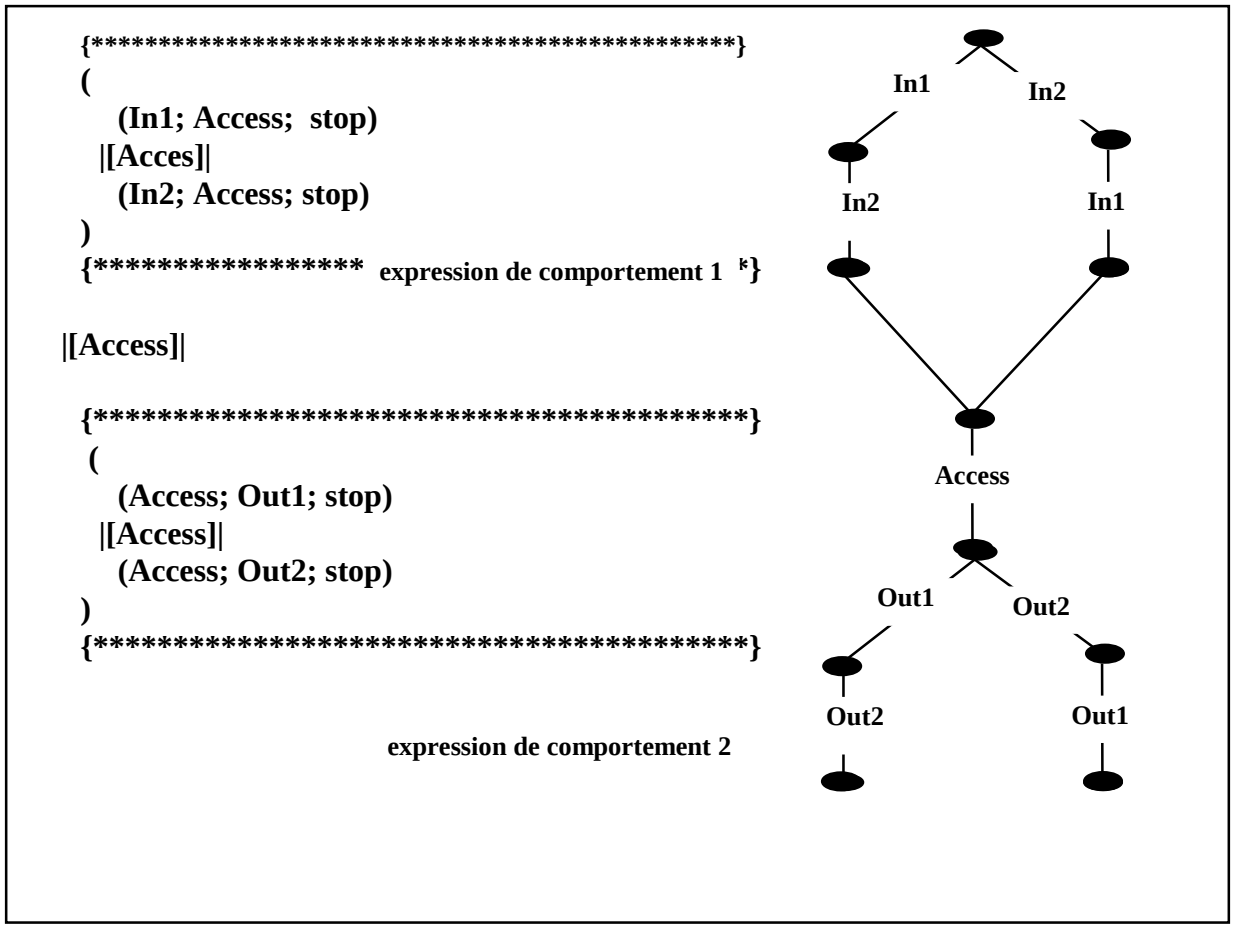

Figure 16. Modélisation du système de contrôle d'accès en utilisant la composition parallèle

La figure 16 montre une autre alternative de modélisation du système de contrôle d'accès à deux clés. Celle-ci utilise la composition de deux expressions comportementales « expression de comportement 1 » et « expression de comportement 2 » qui se synchronisent sur l'événement « Access ».

III-2-6 Composition séquentielle

L'opérateur « ; » permet un ordonnancement des événements dans une expression de comportement. Pour représenter l'ordonnancement des expressions de comportement, LOTOS introduit la composition séquentielle '>>'.
L'expression B1 >>B2 signifie que l'expression de comportement B2 n'est permise que si l'événement **$** (exit) se produit dans B1.

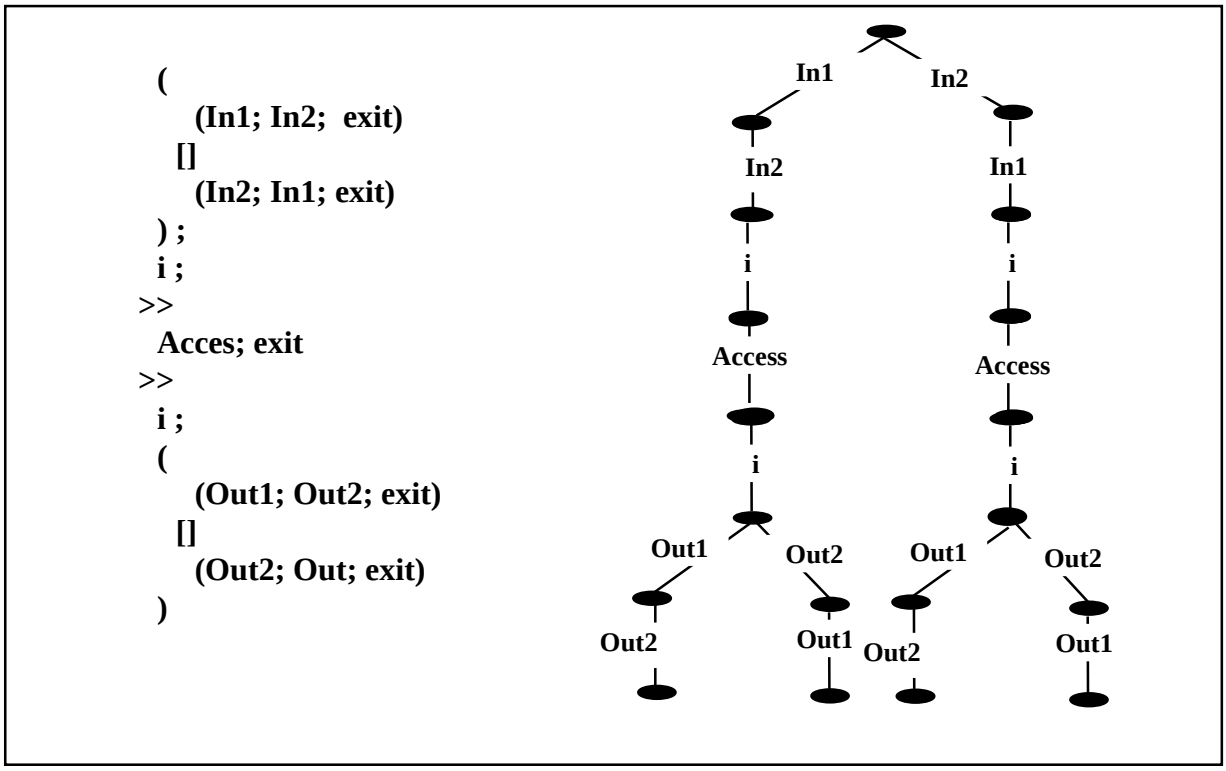

Figure 17. Modélisation du système de contrôle d'accès en utilisant la composition séquentielle

La figure 17 présente une modélisation du système d'accès à deux clés en utilisant la composition séquentielle des trois expressions comportementales : « *(In1; In2; exit) [](In2; In1; exit) ; i »* **,** **«** *Access; exit »* et « *i ;(Out1; Out2; exit) [] (Out2; Out; exit) ».*

III-3 Spécification de l'exemple gestionnaire-agent

Dans cet exemple on aura trois processus: un processus gestionnaire, un processus agent et un processus objet géré; et deux portes a et b.

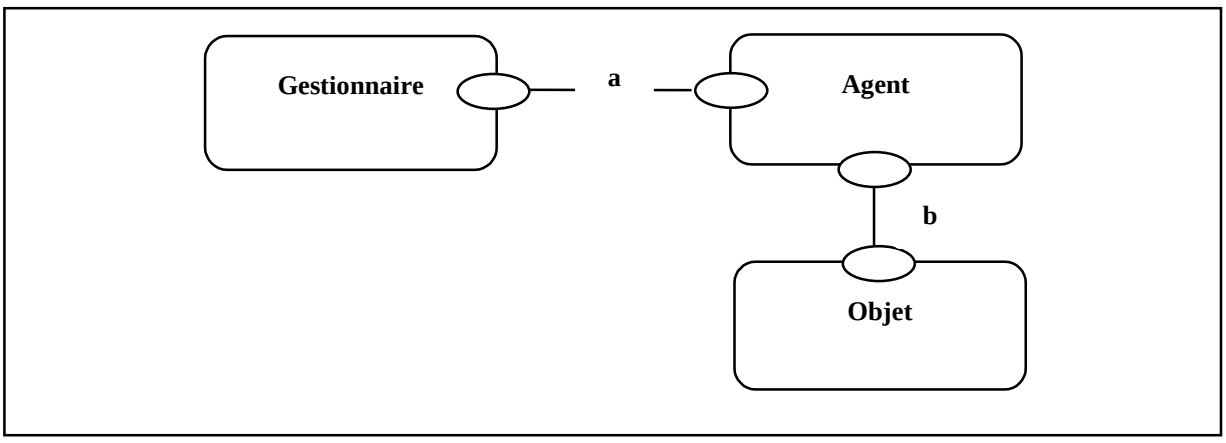

Figure 18. Processus du modèle Gestionnaire-agent.

On donnera ci dessous la description LOTOS de chacun de ces processus.

```
Process gestion [a,b] : noexit :=
   Gestionnaire [a] | [a] | Agent[a,b] |  [b] | Objet[b]

 where

    Process Gestionnaire [u] : noexit :=
        u ? notification      {* Le symbole '?' designe que le processus gestionnaire
                                 lit une notification par la porte u*}
      []

        u ! operation         {* Le symbole '!' designe que le processus gestionnaire
                                 envoie une operation sur la porte u*}
    End proc

    Process Agent[v,t] : noexit :=
        t ? notification ;
        v  ! notification
      []
        v ? operation ;
        t ! operation
    End proc

    Process Objet[w] : noexit :=
        w  ! notification
      []
        w ? operation
    End proc

End proc
```

IV SDL *(Spécification and Description Langage)*

SDL est un langage de spécification formelle normalisé à l'UIT-T [5], et maintenu durant toute son évolution par cet organisme. Ce langage adopte des techniques de description formelles basées sur les machines à états fini et fait partie des langages de spécification standards.
SDL a l'avantage d'avoir des moyens de description graphique. Ses concepts peuvent être regroupés en trois ensembles : concepts de structuration, concepts d'expression de comportement et concepts de représentation de données.
Dans la suite, on s'intéressera surtout aux deux premiers types de concepts, puis on donnera une description de l'exemple gestionnaire-agent en utilisant SDL.

IV-1 Concepts de structuration

Les concepts de structuration de SDL permettent la décomposition modulaire d'un système en vue de réduire sa complexité. Cette structuration se fait en terme de « système », « blocs » et « processus ».

IV-1-1 Concept « système »

Le concept « système » permet de représenter le système distribué tout entier. Il consiste en des « blocs » connectés soit à d'autres blocs ou à l'environnement externe du système via des canaux (figure 19). Ces canaux sont caractérisés par les signaux (messages) qu'ils transportent et peuvent être uni- ou bidirectionnels.

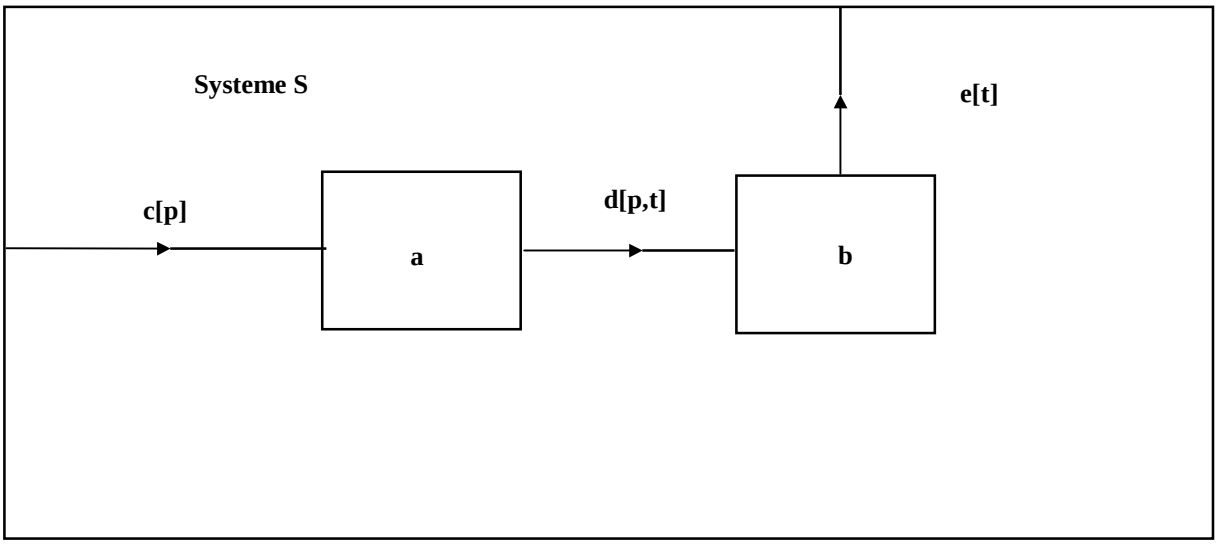

Figure 19. Concept système

Cette figure exprime que le système S est formé de deux blocs : **a** et **b** et de trois canaux : **c, d** et **e**. Le canal **e**, par exemple, permet de transporter le signal **t** du bloc **b** à l'environnement externe.

IV-1-1 Partitionnement

Un bloc peut être décrit soit en un diagramme « sous_structure » du bloc, soit par un diagramme du bloc.

- Diagramme « sous_structure » de blocs
Ce premier type de partitionnement implique une décomposition du bloc en des « sous_blocs » qui seront à leur tour considérés comme des blocs. Par conséquent, de nouveaux canaux apparaissent à travers cette décomposition (figure 20.).

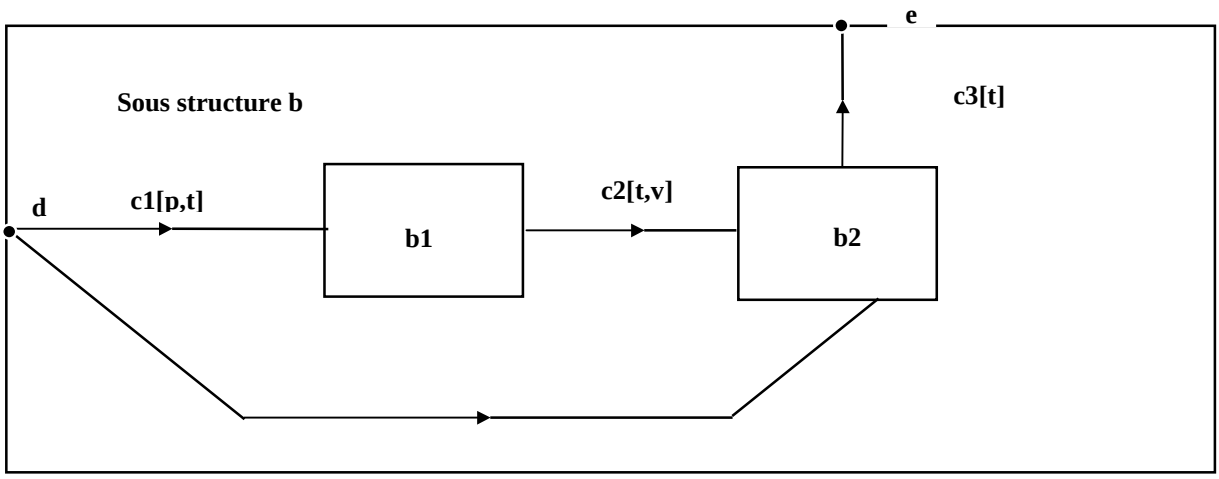

Figure 20. Partitionnement

Cette décomposition introduit le concept de point de connexion qui indique la liaison entre un canal externe et un canal interne à la sous structure.

Exemple :

Le point de connexion **d** lie le canal **d**[[**p,t**] au canal **c1**[**p,t**] et le point de connexion **e** lie le canal **c3**[[**t**] au canal **e**[**t**].

- Diagramme du bloc:

Un bloc peut aussi être partitionné en des processus connectés par des routes (figure 21.).

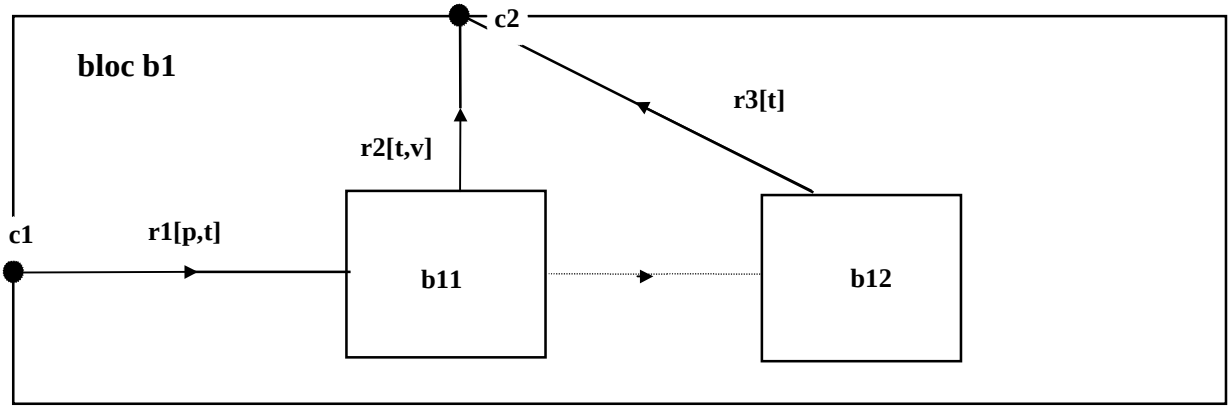

Figure 21. Concept diagramme

La différence entre le concept de canal et celui de route réside dans le fait que le premier est un chemin de communication sans délai et sert à connecter des entités localisées dans différents noeuds alors que le deuxième permet la connexion d'entités d'un même noeud.
Dans la figure 21, **b11** et **b12** représentent des processus : **r1, r2** et **r3** sont des routes , **c1** et **c2** sont des points de connexion permettant la liaison des canaux externes aux routes. Le trait pointillé entre **b11** et **b12** exprime que l'instance du processus **b11** crée une instance du processus **b12**.

IV-2 Concepts d'expression de comportement

La spécification du comportement d'un système se fait par la description de ses processus. Celle-ci s'effectue en SDL en utilisant la notion de machines d'états finis. Un processus consiste en des états et des transitions. Le passage d'un état à un autre se fait par la réception d'une entrée et par le déclenchement d'une transition. Les principaux concepts d'expression de comportement dont on aura besoin pour l'exemple gestionnaire-agent, sont décrits dans ce qui suit sous forme de symboles.

- Initialisation et terminaison d'un processus :

L'initialisation (respectivement terminaison) d'un processus est représentée par le symbole « start » (respectivement « stop ») (figure 22.).

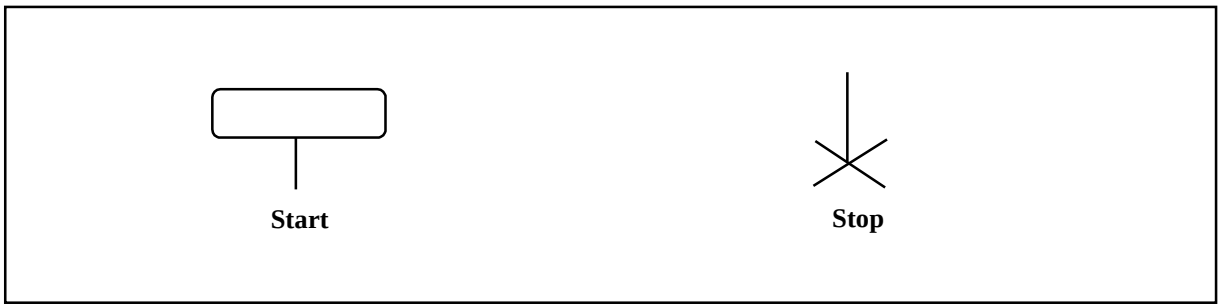

figure 22. Initialisation et terminaison de processus

- Etat

Ce concept permet de représenter à chaque instant le statut d'un processus. En SDL, un état ou une liste d'états est représenté par le symbole de la figure 23.

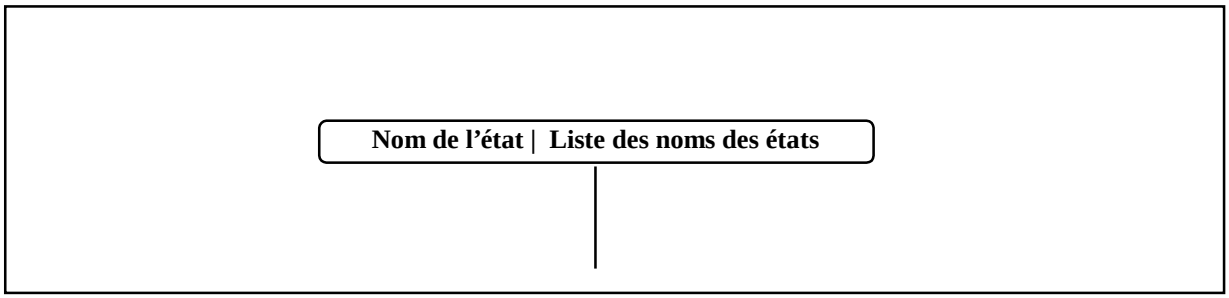

Figure 23. symbole de l'état d'un processus

- Entrées :

Les entrées représentent des stimulus (signal, expiration d'un temporisateur) consommés par un processus via son port d'entrée et donne lieu à des déclenchements de transitions. En SDL, elles sont représentées par le symbole de la figure 24.

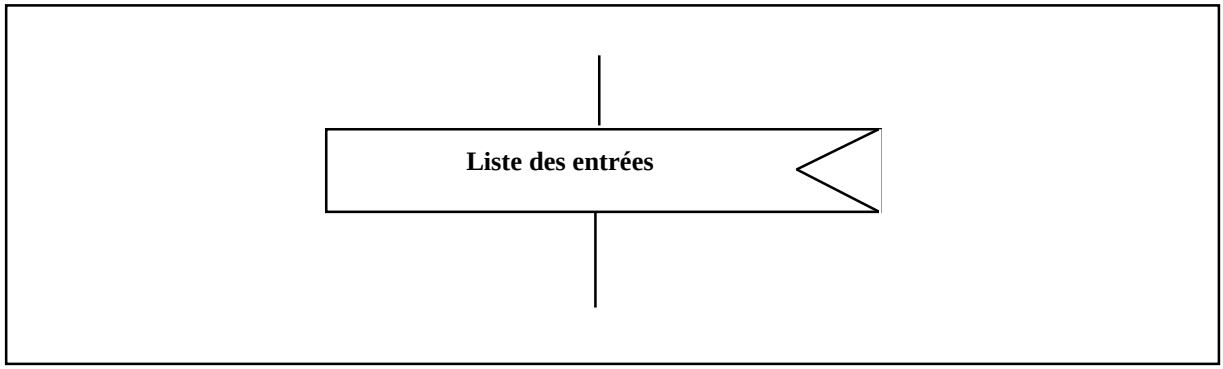

Figure 24. Symbole d'entrée

Les transitions peuvent aussi se déclencher lorsqu'une condition est satisfaite (figure 25.)

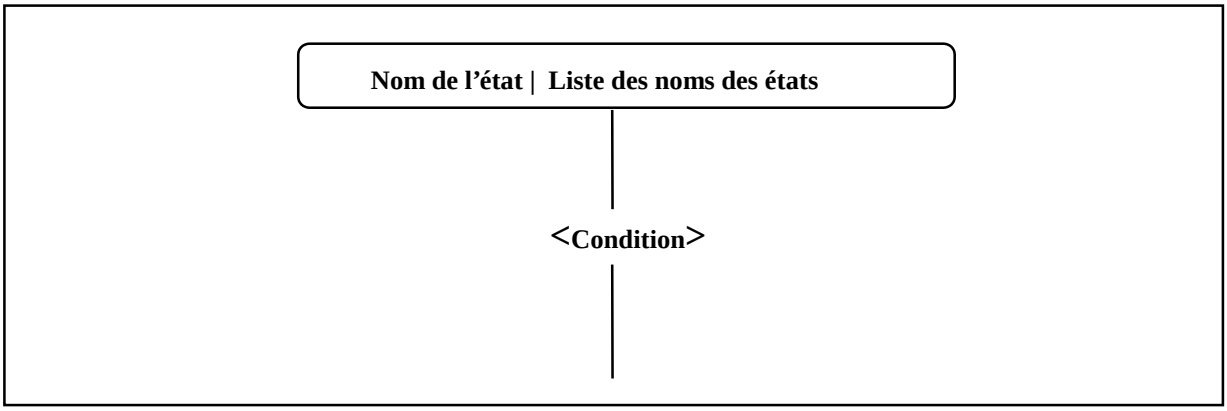

Figure 25. Représentation d'une condition

- Transitions

Une transition permet à un processus de passer d'un état à un autre lors de la réception d'un stimulus. Elle peut être constituée de plusieurs actions. Ces dernières peuvent être des « outputs » (sorties), tâches, création de processus, branchement ou appel de procédure.

- Sorties :

Cette action consiste à envoyer un stimulus à un autre processus. Elle est représentée par un symbole où on précise le signal, ses paramètres s'il y en a, et l'adresse du processus récepteur (figure 26).

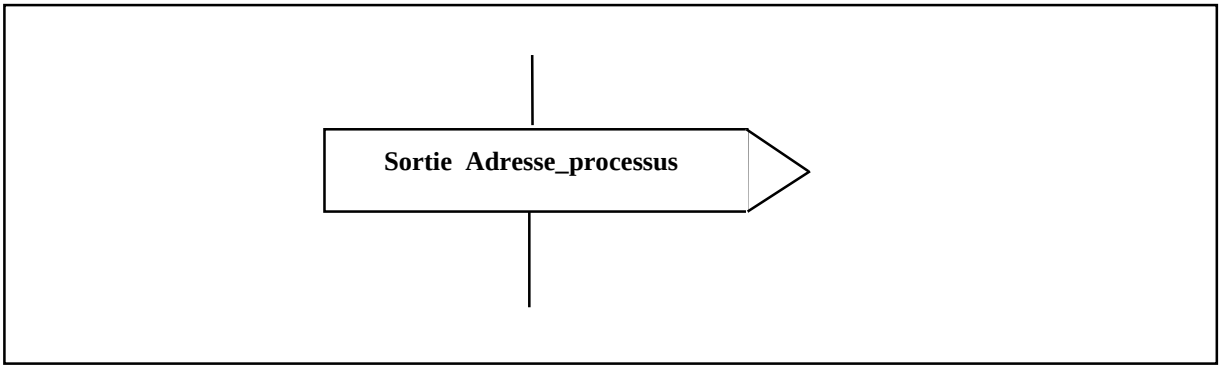

Figure 26. Sortie

- tâche

Une tâche permet de manipuler les variables (changement de valeur) de manière formelle ou informelle comme indiqué dans la figure 27 et dont un exemple est donné dans la figure 28.

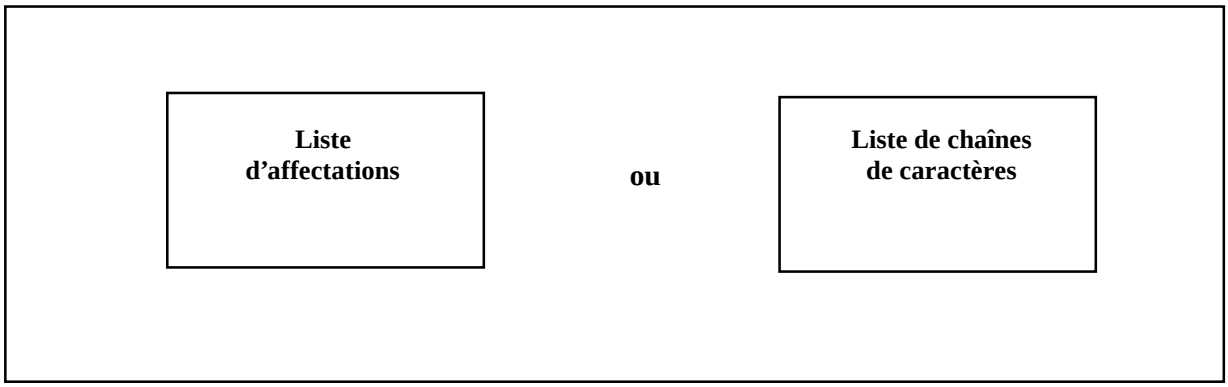

Figure 27. Description formelle et informelle d'une tâche

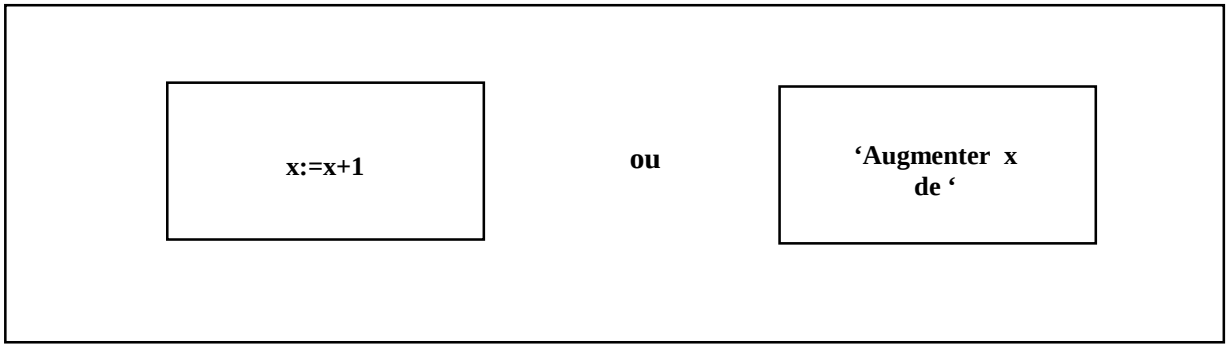

Figure 28. Exemple de tâche

- Création d'un processus

La création dynamique d'une instance de processus peut être faite par une instance existante d'un autre processus. Elle est représentée par le symbole « créer » de la figure 29.

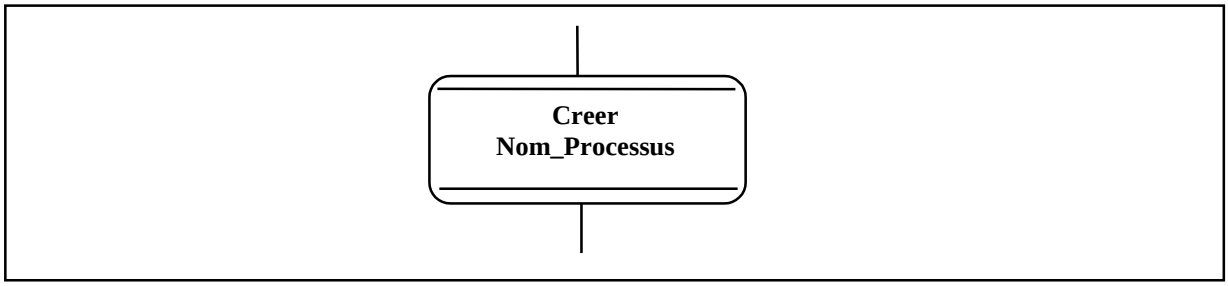

Figure 29. Création d'un processus

- Branchement

Le branchement dans une transition est exprimé par le symbole « décision » (figure 30.). Elle consiste en une condition suivie de deux branches et des actions pour chacune d'elle.

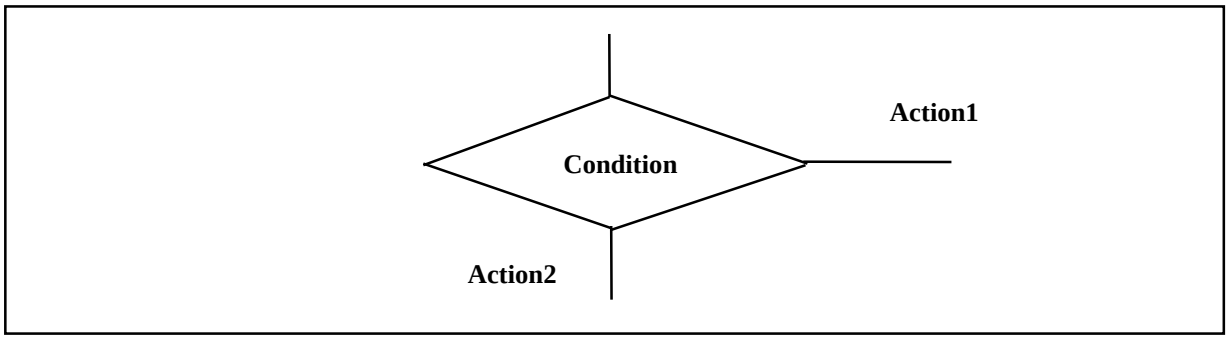

Figure 30. Branchement

- Appel de procédure

L'appel d'une procédure est représenté dans SDL par le symbole de la figure 31.

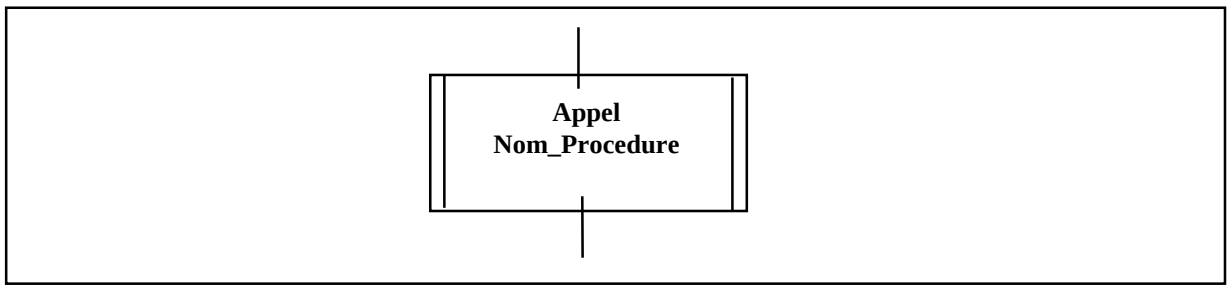

Figure 31. Appel d'une procedure

- Terminaison d'une transition

La terminaison d'une transition est représentée par l'état « next state ». Dans le cas où cet état est le même que l'état d'origine, il sera noté '_'.

A chacun des symboles graphiques de SDL, correspond une syntaxe permettant ainsi d'écrire des spécifications de manière textuelle. Il existe aussi des outils permettant le passage du mode graphique au mode textuel.

IV-3 Spécification de l'exemple du gestionnaire-agent

Nous allons reprendre la spécification de l'exemple Gestionnaire-agent cf figure 2 paragraphe II-1 .

- Structuration

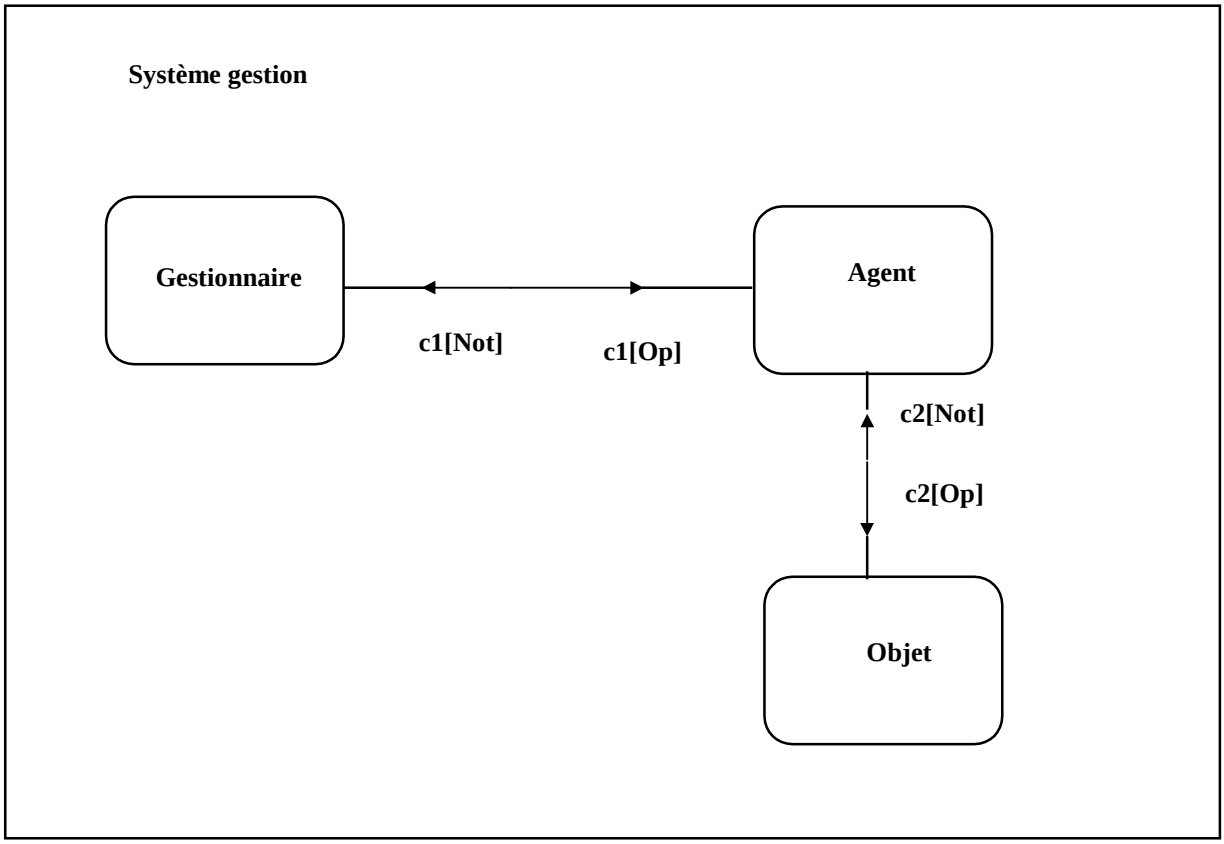

Figure 32. Système gestion

La figure 32 modélise une structuration de l'exemple gestionnaire-agent en SDL. Celle-ci est composée de trois blocs : gestionnaire, agent et objet géré. Ces blocs sont connectés par les canaux bidirectionnels c1 (entre gestionnaire et agent) et c2 (entre agent et objet). Ils seront considérés comme des processus car on ne peut pas les décomposer en blocs ou processus.

- Description du comportement

La figure 33 décrit le processus gestionnaire.

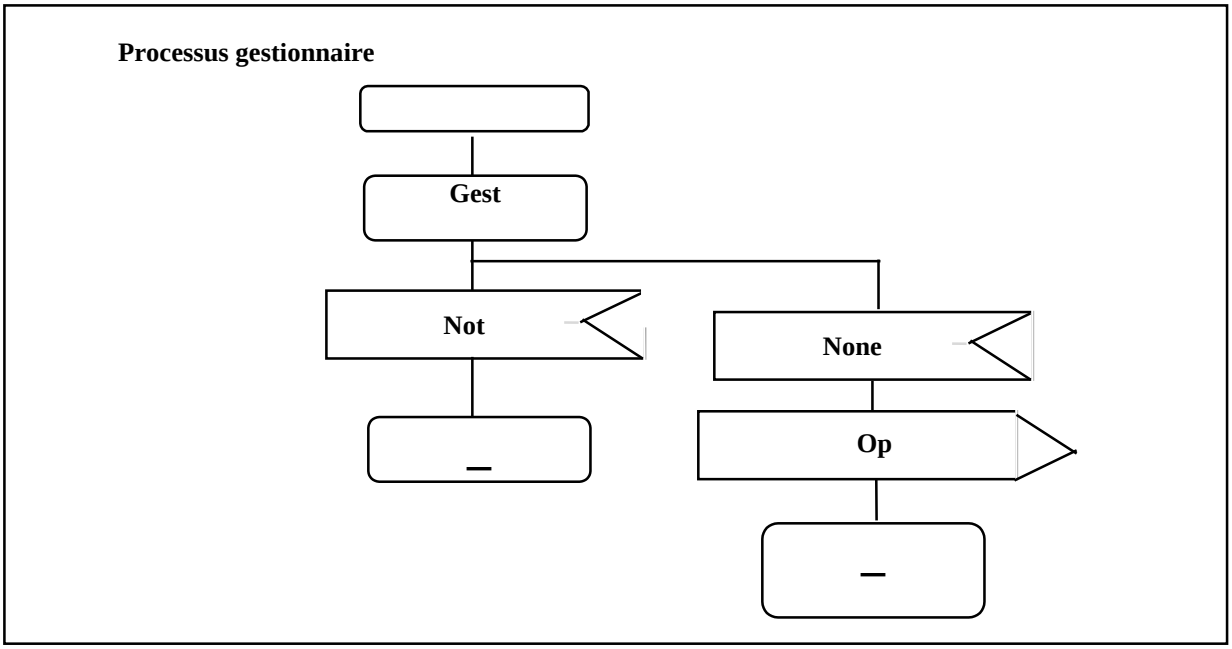

Figure 33. Processus gestionnaire

Le processus gestionnaire peut être soit à l'état actif soit à l'état passif. Dans le premier cas, s'il reçoit une notification (Not), il passe à l'état actif et s'il ne reçoit pas de stimuli (None), il reste dans le même état. Dans le deuxième cas, il passe à l'état passif en émettant une opération (Op).

La figure 34 décrit le processus agent.

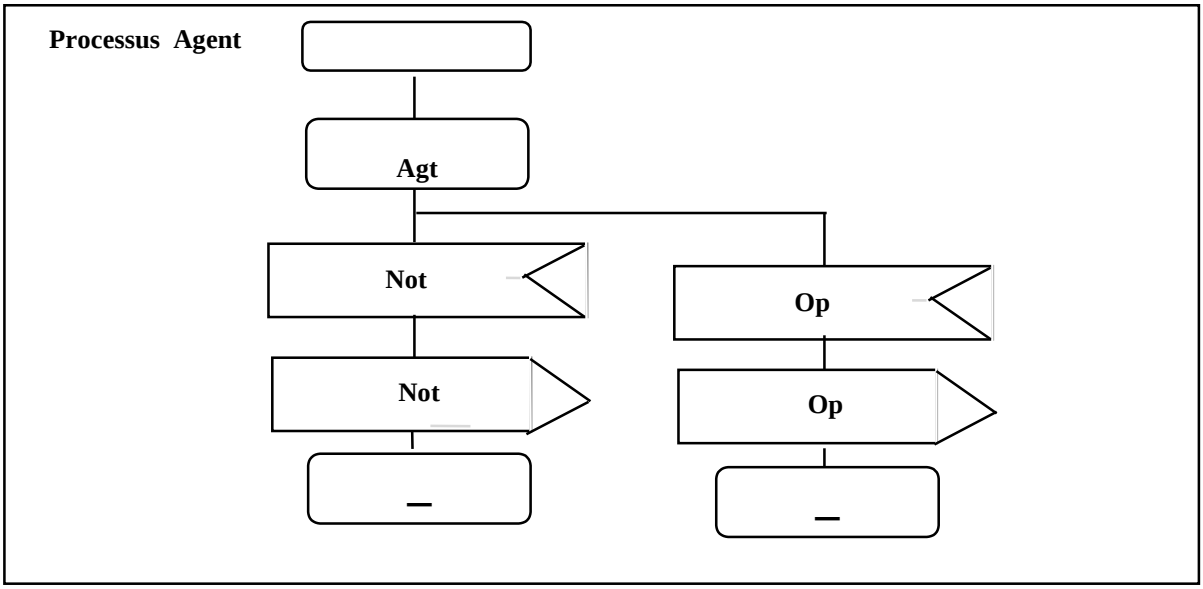

Figure 34. Processus agent

Le processus agent possède un seul état (Agt). Lorsqu'il reçoit une notification (Not) (respectivement opération (Op)), il l'émet en sortie en restant dans le même état.

La figure 35 décrit le processus objet

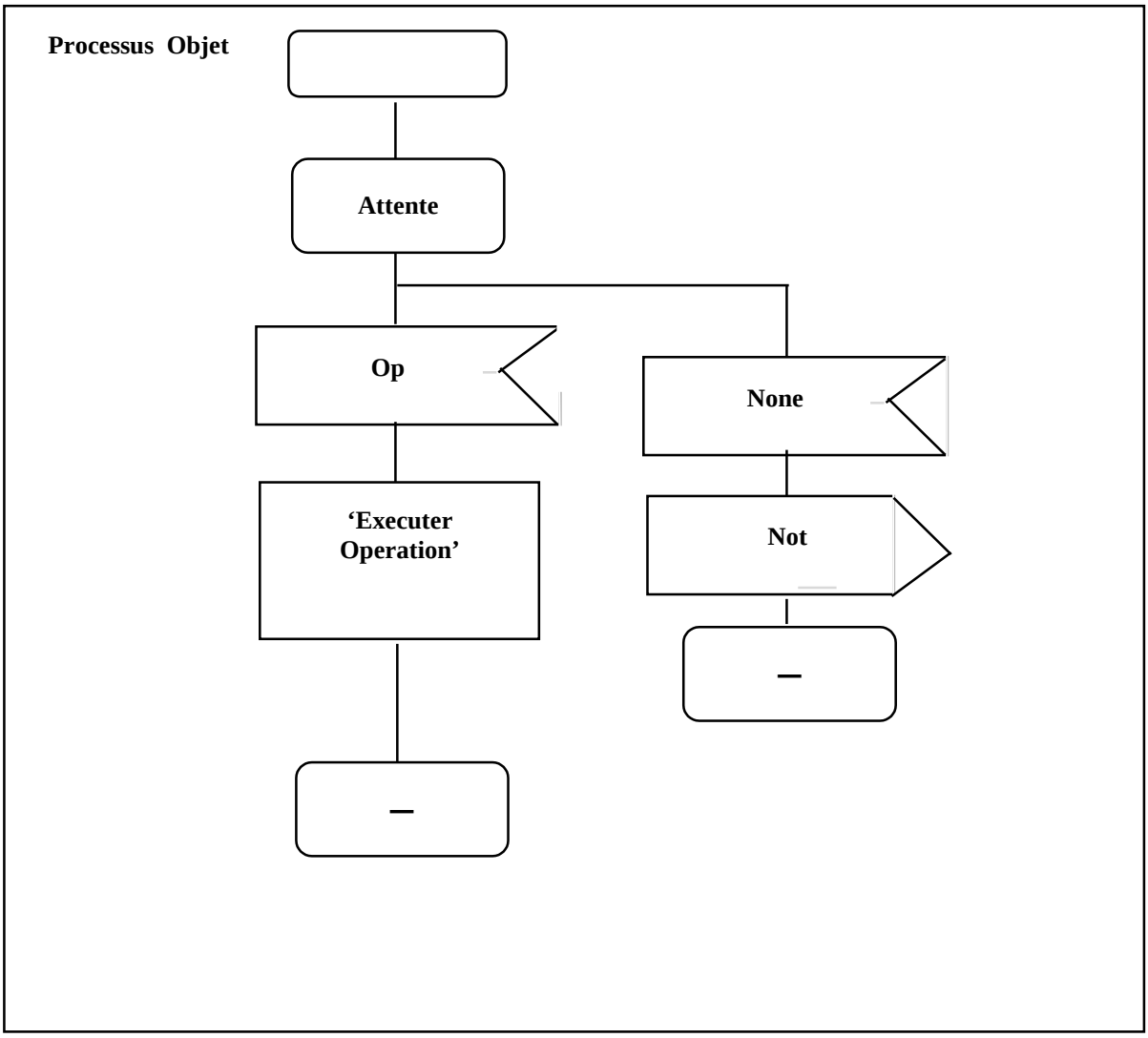

Figure 35. Processus agent

Le processus agent possède lui aussi un seul état (Attente). Lorsqu'il reçoit une opération (Op), il l'exécute et reste dans le même état. Il peut aussi émettre une notification sans rien recevoir en entrée.

IV-4 Orientation Objet de SDL (SDL'92)

SDL '92 contient des concepts pour une structuration orientée objet. Il introduit les notions de paquetage, de super-type, sous-type, spécialisation, héritage. Ainsi, les spécifications en SDL vont pouvoir bénéficier de l'apport du paradigme «orienté objet ».

V Conclusion

Les langages de spécifications formelles ont été d'un grand apport dans la description des systèmes informatiques . Cet apport est dû au fait qu'ils adoptent des techniques de description formelles. En effet, comme on l'a vu dans ce rapport, ESTELLE et SDL sont basés sur les machines à états finis étendues et LOTOS est basé sur l'algèbre des processus. Ces techniques de description formelles permettent des descriptions claires et uniformes.
Ces langages permettent aussi la vérification des spécifications réalisées ainsi des tests et des simulations, permettant aux spécificateurs de s'assurer de leur validité avant de passer aux étapes de conception et implantation.

Références

[1] Kenneth J. Turner, « Using Formal Description Techniques. An Introduction to ESTELLE, LOTOS and SDL», British Library of Cataloguing in Publication Data, 1993.

[2] ISO / IEC 10040 « Information technology - Open Systems Interconnection - Systems Management Overview » ISO / IEC 10040, 1990.

[3] ISO-DIS 9074, ISO / TC97 / SC21 / WG1-FDT / SC-B, « ESTELLE, a formal description technique based on the observationnal behaviour. », 1987.

[4] ISO-DIS 8807, ISO / TC97 / SC21 / WG1-FDT / SC-C, « LOTOS, a formal description technique based on an extended state transition model. », June 1987.

[5] UIT-T.Z.100. « Langage de Description et de Spécification du CCITT », Mars 1993.