Lot 1
Architecture, Intégration

Représentation XML au sein de l'environnement Calife

Description :	Ce document présente la structure des différents documents XML manipulés par la plate-forme Calife. Un modèle d'automates génériques en XML est également présenté.
Auteur(s) :	Bertrand TAVERNIER
Référence :	AVERROES / Sous-Projet 1 / Fourniture 1.1 / V1.0
Statut :	Validé
Version :	1.0

Réseau National des Technologies Logicielles
Projet subventionné par le Ministère de la Recherche et des Nouvelles Technologies

CRIL Technology, France Télécom R&D, INRIA-Futurs, LaBRI (Univ. De Bordeaux - CNRS), LIX (Ecole Polytechnique, CNRS), LORIA, LRI (Univ. Paris Sud - CNRS), LSV (ENS de Cachan – CNRS)

Identification

Type de document : Fourniture 1.1 – Projet RNTL Averroes

Emetteur : CRIL TECHNOLOGY - Systèmes Avancés

Auteurs

Personne	Fonction	Organisme	Visa	Date du visa
B. TAVERNIER	Chef de Projet	CRIL TECHNOLOGY		

Vérification

Personne	Fonction	Organisme	Visa	Date du visa
A. CROIXMARIE	Resp. Dpt	CRIL TECHNOLOGY		

Liste des versions et révisions

Version / Révision	Date	Objet
1.0		Version initiale

SOMMAIRE

1 REPRESENTATION XML DES AUTOMATES 6

1.1 PRINCIPE DE FONCTIONNEMENT DU LANGAGE XML 6
1.1.1 Généralités 6
1.1.2 Règles de grammaire XML 6
1.1.3 Définition de la structure d'un document XML 7
1.1.4 Transformations XSL de documents XML 8

2 REPRESENTATION XML D'UN COMPOSANT 10

2.1 REPRESENTATION XML DES ENTREES-SORTIES D'UN COMPOSANT 10
2.1.1 Représentation de l'environnement 10
2.1.2 Représentation des actions 11
2.2 REPRESENTATION DU COMPORTEMENT D'UN AUTOMATE 12
2.2.1 Représentation XML des prédicats 12
2.2.2 Représentation XML du graphe 15
2.3 REPRESENTATION DU COMPORTEMENT D'UN BLOCK 16
2.3.1 Définition XML des composants 16
2.3.2 Définition XML des liens 16
2.3.3 Définition XML des vecteurs de synchronisation 17
2.4 FORMAT DE REPRESENTATION D'UN PRODUIT SYNCHRONISE 17

3 MODELES D'AUTOMATES 18

3.1 MODELES D'AUTOMATES AU SEIN DE CALIFE V3.0 18
3.2 DEFINITION DU MODELE XML 19
3.2.1 Déclaration de l'environnement général du modèle 19
3.2.2 Déclaration des types de donnée 19
3.2.3 Déclaration des fonctions utilisables 20
3.2.4 Typage de l'arbre de syntaxe abstraite 21
3.3 AJOUT DES OUTILS 22
3.3.1 Définition de variables 22
3.3.2 Actions élémentaires 23
3.3.3 L'interface CalifeEdit 23
3.3.4 Exemple de script XML : la projection vers le model-checker Kronos 24

4 CONCLUSION 26

ANNEXES

TABLE DES FIGURES

Figure 1 : du XML au document 6
Figure 2 : Exemple de fichier XML 7
Figure 3 : DTD du fichier XML de la Figure 2 7
Figure 4 : Représentation d'un composant 10
Figure 5 : DTD associée au nœud Environnement d'un composant 11
Figure 6 : Exemple de représentation XML de l'environnement 11
Figure 7 : DTD associée au nœud Actions 12
Figure 8 : Exemple de nœud Actions 12
Figure 9 : Exemple d'arbre de syntaxe abstraite 13
Figure 10 : DTD des prédicats 13
Figure 11 : Représentation non typé d'une contrainte 14
Figure 12 : Représentation typée d'une contrainte 14
Figure 13 : DTD du graphe d'un automate 15
Figure 14 : Relation grapheur - représentation XML 15
Figure 15 : DTD associé à la définition des composants 16
Figure 16 : DTD associé aux liens des variables 16
Figure 17 : DTD associé aux composantes des vecteurs de synchronisation 17
Figure 18 : Exemple de définition des vecteurs de synchronisation 17
Figure 19 : Environnement du modèles des p-automates 19
Figure 20 : DTD associée à la déclaration des types 20
Figure 21 : Déclaration des types pour les automates temporisés 20
Figure 22 : Grammaire BNF des gardes des automates temporisés 21
Figure 23 : Définition des fonctions utilisables pour les automates temporisés 21
Figure 24 : Vue d'ensemble de l'interface CalifeEdit 24
Figure 25 : Script d'export vers l'outil Kronos 25

INTRODUCTION

Le prototype Calife 2.1.11, développé dans le cadre du projet RNRT du même nom a permis de valider l'utilisation de la technologie XML/XSL pour générer une représentation textuelle de p-automates.

Dans le cadre du développement de la version 3.0 de l'environnement Calife, prévu au titre du projet RNTL Averroes, cet acquis a été réutilisé et étendu pour permettre la représentation d'autres types d'automates (automates temporisés, automates hybrides, automates à compteur,…), à partir de la définition de modèles.

Le présent document s'attachera tout d'abord à décrire la représentation XML d'un automate générique (non typé) et d'un modèle d'automate. La dernière partie du document présentera les mécanismes mis en places dans l'environnement Calife v3.0.

1 REPRESENTATION XML DES AUTOMATES

1.1 Principe de fonctionnement du langage XML

1.1.1 Généralités

XML (eXtensible Markup Language) est un métalangage de balisage créé par le W3C (World Wide Web Consortium, coopération internationnale de nombreuses entreprises et laboratoires de recherche : MIT, INRIA,…,). Celui-ci est né de la volonté de séparer les données de la présentation, pour les fichiers HTML, dont la mise à jour était jusque là très contraignante. On a alors créé le format XML pour représenter les données, et on a adjoint à celui-ci : des fichiers de transformation pour concevoir la présentation de documents (XML Stylesheet Language, XSL) et un fichier de définition de la structure (Document Type Definition, DTD).

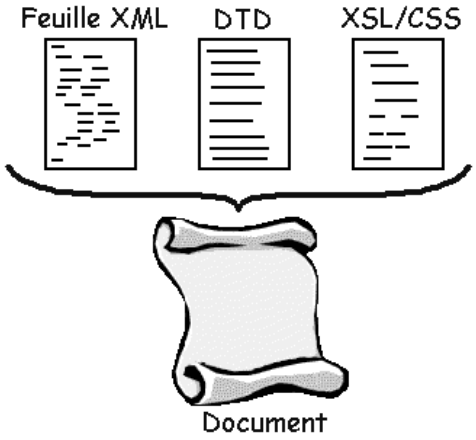


Figure 1 : du XML au document

Même si XML a été, au départ créé dans le but de supplanter HTML, les applications sont beaucoup plus importantes. Il est en effet possible de générer n'importe quelle forme de fichier de donnée, au format texte, et ce à l'aide d'un simple fichier de transformation XSL (sous forme d'un fichier texte également).

1.1.2 Règles de grammaire XML

Un fichier XML est composé :

- D'une entête permettant la déclaration de la version d'XML utilisée, de l'encodage des caractères,… (optionnel)
- De la déclaration de la structure du fichier, au format DTD ou Xschema (optionnel) cf…
- D'un unique élément racine

- De commentaires représentés par des balises de type < !—Contenu du commentaire -->

Un élément XML contient :

- Une balise d'ouverture pouvant éventuellement définir des attributs et leur valeur,
- Le contenu de l'élément, c'est à dire une chaîne de caractère ou une succession d'éléments XML,
- Une balise de fermeture.

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<Graphe>
        <Node Id= "1"  Label="Init"/>
        <Node Id= "2"  Label="Working"/>
        <Node Id= "3"  Label="Finish"/>
        <Edge>
                <From>1</From>
                <To>2</To>
        </Edge>
        <Edge>
                <From>2</From>
                <To>3</To>
        </Edge>
</Graphe>
```

Figure 2 : Exemple de fichier XML

1.1.3 Définition de la structure d'un document XML

XML étant un métalangage, la définition de la structure attendu est nécessaire afin de s'assurer de la cohérence des données manipulées.

Le standard définit la notion de DTD (Document Type Definition) pour figer cette structure.

La DTD d'un document décrit l'ensemble des éléments admissibles au sein du document (< !ELEMENT…), ainsi que les attributs possibles pour chaque élement (< !ATTLIST…).

La définition de la structure reprend le principe des grammaires BNF.

A titre d'exemple, le document XML de la Figure 2 pourrait être associé à la DTD suivante :

```
<!ELEMENT Graphe (Node | Edge)* >
        <!ELEMENT Node EMPTY>
        <!ATTLIST Node Id CDATA #REQUIRED>
        <!ATTLIST Node Label CDATA #IMPLIED>

        <!ELEMENT Edge (From,To)>
                <!ELEMENT From(#PCDATA)/>
                <!ELEMENT To(#PCDATA)/>
```

Figure 3 : DTD du fichier XML de la Figure 2

Les mots clés CDATA et #PCDATA représente une chaîne de caractère pour les attributs et les éléments.

Les mots clés #IMPLIED et #REQUIRED permettent de spécifier le statut (optionnel ou obligatoire) d'un attribut.

1.1.4 Transformations XSL de documents XML

Le standard XML définit le langage XSL (XML Stylesheet Language) permettant d'associer une représentation (HTML, XML ou textuelle) à un document XML.

Un fichier XSL est un fichier XML dont les balises sont interprétées par un processeur XSL (par exemple JAXP ou XALAN) sous la forme de règles associées à chaque élément d'un fichier XML d'entrée.

Une règle (template xsl) est composée :

- d'une expression (dans le langage XPATH) permettant de sélectionner les éléments pour lesquels la règle s'applique
- d'une suite d'actions à appliquer pour réaliser la mise en forme

A titre d'exemple, le document XML de la Figure 2 pourrait être transformé en appliquant les règles du fichier XSL suivant :

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" indent="yes"/>

<xsl:template match="/">
        <xsl:apply-templates/> <!—permet d'appliquer toutes les règles associées -->
</xsl:template>

<xsl:template match="Graphe">
        Liste des transitions du graphe :
        <xsl:for-each select="Edge">
                <xsl:variable name="From" select="From"/>
                <xsl:variable name="To" select="To"/>
                Transition du nœud <xsl:value-of select="../Node[@Id=$From]/@Label"/>
                vers le nœud <xsl:value-of select="../Node[@Id=$To]/@Label"/>.
        </xsl :for-each>
</xsl:template>
```

Le résultat serait alors le suivant :

```
Transition du nœud Init vers le nœud Working.
Transition du nœud Working vers le nœud Finish.
```

Le traitement réalisé par le fichier XSL ci-dessus peut être informellement décrit de la manière suivante :

Pour chaque nœud *Edge* rencontré dans le document XML, on sauvegarde le contenu des balises *From* et *To* dans des variables nommées *$From* et *$To*. On ensuite écrit la chaîne de caractères « Transition du nœud », puis la valeur de l'attribut *Label* (symbolisée par @Label), du nœud *Node* dont l'attribut *Id* est égal à la valeur contenue dans la variable *$From*.

Enfin, on fait de même pour le nœud dont l'attribut *Id* vaut *$To*.

La technologie XML/XSL apporte l'aspect « multi-représentation de document » au sein de l'environnement Calife. La difficulté consiste à définir une structure XML suffisamment générique pour servir de modèle commun aux différents outils interfacés par l'environnement.

2 REPRESENTATION XML D'UN COMPOSANT

L'environnement Calife permet la création de deux types d'automates :

- Les automates simples, représentables sous la forme d'un graphe ;
- Les automates composés par produit synchronisé d'automates.

Par la suite, le terme automate désignera un automate simple et le terme block désignera un produit synchronisé d'automates (simples ou eux-mêmes crées par produit synchronisé). Le terme composant désignera indifféremment un block ou un automate simple.

D'une manière générique, un composant est constitué de 2 parties distinctes :

- Une partie « entrée-sortie », décrivant l'environnement « externe » au composant considéré et des actions permettant de synchroniser ce composant avec d'autres. Cette partie sera représentée en XML d'une manière semblable, qu'il s'agisse d'un block ou d'un automate.
- Une partie « comportement », spécifique au composant considéré et contenant la déclaration des variables « locales ». La représentation XML de la partie comportement d'un automate ou d'un block sera différente.

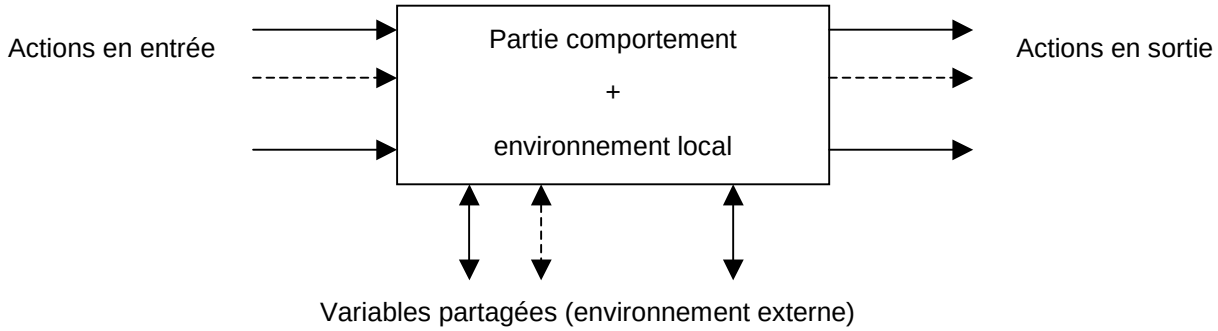


Figure 4 : Représentation d'un composant

2.1 Représentation XML des entrées-sorties d'un composant

Les éléments XML décrits ci-dessous sont communs aux blocks et aux automates.

2.1.1 Représentation de l'environnement

L'environnement d'un composant contient des variables (nœuds *Var*) et des paramètres (nœuds *Param*) auxquels sont associé une notion de portée (local ou externe).

De plus, chaque nœud de type *Var* ou *Param* possède les attributs *Label* et *Type* associés au nom et au type de la variable ou du paramètre considérée.

La DTD associée au nœud environnement est la suivante :

```
<!ELEMENT Environnement (Externe,Local) >
        <!ELEMENT Externe (Var|Param)* >
        <!ELEMENT Local (Var|Param)* >
                <!ELEMENT Var EMPTY>
                <!ATTLIST Var Label CDATA #REQUIRED>
                <!ATTLIST Var Type CDATA #REQUIRED>

                <!ELEMENT Param EMPTY>
                <!ATTLIST Param Label CDATA #REQUIRED>
                <!ATTLIST Param Type CDATA #REQUIRED>
                <!ATTLIST Param Value CDATA #IMPLIED>
```

Figure 5 : DTD associée au nœud Environnement d'un composant

Remarque : l'attribut *Value* du nœud *Param* est utilisé pour spécifier (optionnellement) une valeur à associer à un paramètre local du composant considéré. Dans le cas d'un paramètre externe, il n'est pas possible de spécifier une valeur.

```
<Environnement>
    <Externe>
        <Param Label="Sigma" Type="Time"/>
        <Param Label="Lambda" Type="Time"/>
    </Externe>
    <Local>
        <Var Label="x" Type="Clock"/>
    </Local>
</Environnement>
```

Figure 6 : Exemple de représentation XML de l'environnement

2.1.2 Représentation des actions

Les actions d'un composant permettent de synchroniser celui-ci avec d'autres composants, lors de la construction d'un block.

Les actions définies pour un composant seront les composantes des vecteurs de synchronisation d'un block utilisant ce composant (cf 2.3.3).

Lorsque le composant est un automate, les nœuds <Action> contiennent l'ensemble des labels utilisés dans les transitions de cet automate.

Lorsque le composant est un block, les nœuds <Action> contiennent les différents labels associés aux vecteurs de synchronisation du block (fonction de nommage des vecteurs de synchronisation).

Les actions sont saisies depuis l'interface Calife et peuvent avoir une des formes suivantes :

- <Action> : représente une action sans type
- ?<Action> : représente une action de type « Envoi de message »
- !<Action> : représente une action de type « Réception de message »

La DTD associée au nœuds Actions est la suivante :

```
<!ELEMENT Actions (Action*)>
        <!ELEMENT Action (Comp*)>
        <!ATTLIST Action Label CDATA #REQUIRED>
        <!ATTLIST Action Type (Send|Wait) #IMPLIED>
```

Figure 7 : DTD associée au nœud Actions

Remarque : Les nœuds *Comp* sont spécifiques aux block et seront décrits au paragraphe 2.3.3.

```
<Actions>
    <Action Label="CD" Type="Wait"/>
    <Action Label="begin" Type="Send"/>
    <Action Label="busy" Type="Wait"/>
    <Action Label="fin" Type="Send"/>
</Actions>
```

Figure 8 : Exemple de nœud Actions

2.2 Représentation du comportement d'un automate

Le comportement d'un automate est décrit sous la forme d'un graphe composé de localités (dans lesquelles le temps peut s'écouler) et de transitions discrètes (franchies de manière instantanées).

Aux localités sont associés :

- Un prédicat nommé Invariant permettant de limiter l'écoulement du temps
- Un prédicat nommé Activité (utilisé uniquement pour spécifier certains automates hybrides, cf 3) spécifiant la valeur de la dérivée des horloges en fonction du temps.

Aux transitions sont associés :

- Un prédicat nommé Garde permettant de spécifier si la transition est franchissable ou non.
- Une action qui sera utilisée pour définir les vecteurs de synchronisation
- Un prédicat nommé « Mise à jour » décrivant la nouvelle valuation des variables, une fois la transition franchie.

2.2.1 Représentation XML des prédicats.

Le but de l'utilisation des standards XML et XSL est d'éviter l'écriture de multiples parseurs (1 par outil à interfacer). Il est donc nécessaire de représenter les prédicats sous une forme nous permettant de générer n'importe quelle représentation textuelle.

La solution adoptée utilise le stockage, au format XML, de l'arbre de syntaxe abstraite des prédicats saisis sous l'environnement Calife.

Par exemple, le prédicat *2*x<Sigma && y >= x + Lambda* sera représenté par l'arbre (binaire) de syntaxe abstraite suivant :

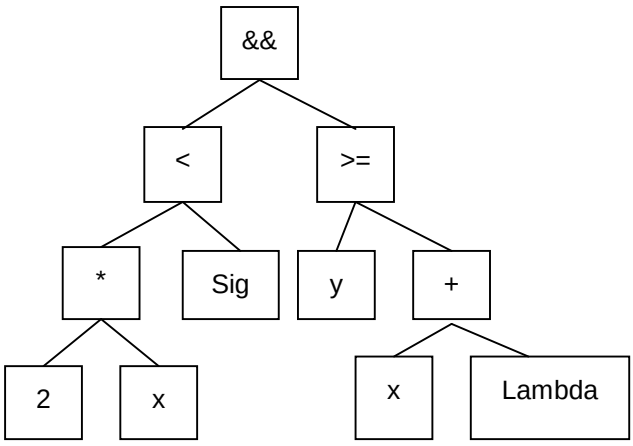


Figure 9 : Exemple d'arbre de syntaxe abstraite

La DTD associée aux prédicats (i.e. aux nœuds *Invariant*, *Activite*, *Garde* et *Updates*) est la suivante :

```
<!ELEMENT Invariant (Contrainte,Activite?)>
<!ATTLIST Invariant PrettyPrint CDATA #REQUIRED>

<!ELEMENT Garde (Contrainte)>
<!ATTLIST Garde PrettyPrint CDATA #REQUIRED>

<!ELEMENT Updates (Update*)>
<!ATTLIST Updates PrettyPrint CDATA #REQUIRED>

        <!ELEMENT Update (FunT|Const)>
        <!ATTLIST Update Type CDATA #REQUIRED>
        <!ATTLIST Update Arg CDATA #REQUIRED>

        <!ELEMENT Activite (FunT|Const)>
        <!ATTLIST Activite Type CDATA #REQUIRED>
        <!ATTLIST Activite Arg CDATA #REQUIRED>

        <!ELEMENT Contrainte (FunT|Const)>
        <!ATTLIST Contrainte Type CDATA #REQUIRED>
        <!ATTLIST Contrainte Arg CDATA #REQUIRED>

        <!ELEMENT FunT (FunT|Const|VAR|VLoc|PLoc|VExt|PExt|OUT)* >
        <!ATTLIST FunT Label CDATA #REQUIRED>
        <!ATTLIST FunT Type CDATA #REQUIRED>
        <!ATTLIST FunT Arg CDATA #REQUIRED>
        <!ELEMENT OUT (FunT|Const|VAR|VLoc|PLoc|VExt|PExt)* >
        <!ATTLIST OUT Type CDATA #REQUIRED>
        <!ATTLIST OUT Arg CDATA #REQUIRED>

                <!ELEMENT Const (#PCDATA) ><!ATTLIST Const Type CDATA #REQUIRED>
                <!ELEMENT VAR (#PCDATA) ><!ATTLIST VAR Type CDATA #REQUIRED>
                <!ELEMENT VLoc (#PCDATA) ><!ATTLIST VLoc Type CDATA #REQUIRED>
                <!ELEMENT PLoc (#PCDATA) ><!ATTLIST PLoc Type CDATA #REQUIRED>
                <!ELEMENT VExt (#PCDATA) ><!ATTLIST VExt Type CDATA #REQUIRED>
                <!ELEMENT PExt (#PCDATA) ><!ATTLIST PExt Type CDATA #REQUIRED>
```

Figure 10 : DTD des prédicats

Tous les nœuds de l'arbre de syntaxe abstraite possèdent un attribut *Type* contenant, lorsque le typage d'une expression est correct, le type de l'expression située sous le nœud considéré.

Les nœuds intermédiaires possèdent un attribut *Arg* contenant le type des arguments et utilisé pour déterminer le type du nœud considéré (cf 3.2.4).

Enfin, les feuilles de cet arbre sont nommées :

- Const pour décrire les constantes utilisées (par exemple *<Const Type='Prop' >True</Const>*)
- VAR pour décrire les variables (ou les paramètres) avant le typage de l'arbre de syntaxe
- VLoc, PLoc, VExt et PExt pour décrire les variables et les paramètres, après typage de l'arbre de syntaxe abstraite.

Les attributs « PrettyPrint » présents aux racines des arbres sont utilisés par l'environnement Calife pour accélérer l'affichage des contraintes (sans avoir besoin de reconstruire les prédicats en « rempilant » l'arbre).

Par exemple, la contrainte de la figure 9 aurait la représentation XML suivante (avant typage):

```
<Contrainte Type="Undef" Arg="Undef">
   <FunT Label="AND" Type="Undef" Arg="Undef">
      <FunT Label="LT" Type="Undef" Arg="Undef">
         <FunT Label="MULT" Type="Undef" Arg="Undef">
            <VAR Type="Undef">2</VAR>
            <VAR Type="Undef">x</VAR>
         </FunT>
         <VAR Type="Undef">Sigma</VAR>
      </FunT>
      <FunT Label="GTE" Type="Undef" Arg="Undef">
         <VAR Type="Undef">y</VAR>
         <FunT Label="PLUS" Type="Undef" Arg="Undef">
            <VAR Type="Undef">x</VAR>
            <VAR Type="Undef">Lambda</VAR>
         </FunT>
      </FunT>
   </FunT>
</Contrainte>
```

Figure 11 : Représentation non typé d'une contrainte

La même contrainte typée aurait pour représentation XML :

```
<Contrainte Type="Prop" Arg="Prop">
   <FunT Label="AND" Type="Prop" Arg="Prop:Prop">
      <FunT Label="LT" Type="Prop" Arg="Clock:Time">
         <FunT Label="MULT" Type="Clock" Arg="nat:Clock">
            <Const Type="nat">2</Const>
            <VLoc Type="Clock">x</VLoc>
         </FunT>
         <PExt Type="Time">Sigma</PExt>
      </FunT>
      <FunT Label="GTE" Type="Prop" Arg="Clock:Clock">
         <VLoc Type="Clock">y</VLoc>
         <FunT Label="PLUS" Type="Clock" Arg="Clock:Time">
            <VLoc Type="Clock">x</VLoc>
            <PExt Type="Time">Lambda</PExt>
         </FunT>
      </FunT>
   </FunT>
</Contrainte>
```

Figure 12 : Représentation typée d'une contrainte

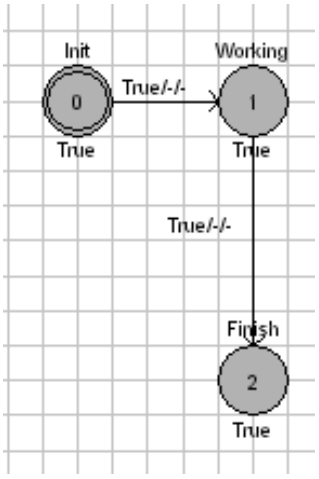
2.2.2 Représentation XML du graphe.

Le graphe associé à un automate simple est décrit à partir de nœuds de type *Localite* ou *Deplacement*. Certaines informations contenues dans la DTD suivante sont purement d'ordre graphique (attributs de type x,y, nœuds de type Vertex,…).

```
<!ELEMENT Localite (Centre,Invariant)>
        <!ATTLIST Localite Label CDATA #REQUIRED>
        <!ATTLIST Localite Id CDATA #REQUIRED>
        <!ATTLIST Localite Initial (true|false) #REQUIRED>
        <!ATTLIST Localite Propriete CDATA #IMPLIED>

                <!ELEMENT Centre EMPTY>
                <!ATTLIST Centre x CDATA #REQUIRED>
                <!ATTLIST Centre y CDATA #REQUIRED>

                <!ELEMENT Invariant (Contrainte,Activite?)>
                <!ATTLIST Invariant PrettyPrint CDATA #REQUIRED>
<!ELEMENT Deplacement (Vertex|TextNode|Garde|Act|Updates)*>
        <!ATTLIST Deplacement Source CDATA #REQUIRED>
        <!ATTLIST Deplacement Target CDATA #REQUIRED>

                <!ELEMENT Vertex EMPTY>
                <!ATTLIST Vertex x CDATA #REQUIRED>
                <!ATTLIST Vertex y CDATA #REQUIRED>
                <!ELEMENT TextNode EMPTY>
                <!ATTLIST TextNode PrettyPrint CDATA #REQUIRED>
                <!ATTLIST TextNode x CDATA #REQUIRED>
                <!ATTLIST TextNode y CDATA #REQUIRED>
                <!ELEMENT Garde (Contrainte)>
                <!ATTLIST Garde PrettyPrint CDATA #REQUIRED>
                <!ELEMENT Act EMPTY>
                <!ATTLIST Act Label CDATA #IMPLIED>
                <!ATTLIST Act Type CDATA #IMPLIED>
                <!ELEMENT Updates (Update*)>
```

Figure 13 : DTD du graphe d'un automate

L'exemple ci-dessous illustre la relation entre le graphe saisi sous l'environnement Calife et sa représentation au format XML. Les représentation des contraintes (gardes, invariants,…) ont été supprimées de la représentation XML :




```
<Localite Label="Init" Id="0" Initial="true" Propriete="">
   <Centre x="840" y="800"/>
</Localite>
<Localite Label="Working" Id="1" Initial="false" Propriete="">
   <Centre x="940" y="800"/>
</Localite>
<Localite Label="Finish" Id="2" Initial="false" Propriete="">
   <Centre x="940" y="960"/>
</Localite>
<Deplacement Source="0" Target="1">
   <TextNode PrettyPrint="True/-/-" x="865" y="796"/>
</Deplacement>
<Deplacement Source="1" Target="2">
   <TextNode PrettyPrint="True/-/-" x="891" y="875"/>
</Deplacement>
```

Figure 14 : Relation grapheur - représentation XML

2.3 Représentation du comportement d'un block

Le comportement d'un block est défini à partir de la donnée :

- des composants à synchroniser ;
- des relations entre les variables externes des différents composants et les variables (internes ou externes) du block (liens) ;
- des vecteurs de synchronisation.

2.3.1 Définition XML des composants

La description de l'ensemble des composants d'un block est réalisée à partir d'un nœud <Components>

Chaque composant est alors décrit par un nœud <Component> possédant les attributs suivants :

- Label : attribut associé au nom du composant (et donc au fichier <Label>.clf)
- Type : attribut pouvant prendre pour valeur Automate | Block
- Index : attribut prenant pour valeur un entier unique ou un paramètre dans le cadre de blocks définissant des familles de composants

```
<!ELEMENT Components (Component*)>
        <!ELEMENT Component (Links)>
        <!ATTLIST Component Label CDATA #REQUIRED>
        <!ATTLIST Component Type CDATA #REQUIRED>
        <!ATTLIST Component Index CDATA #REQUIRED>
```

Figure 15 : DTD associé à la définition des composants

Le nœud « Component » défini ensuite l'ensemble des liens entre les variables (resp. paramètres) externes du composant et les variables du block.

2.3.2 Définition XML des liens

Les liens entre les variables et la paramètres sont définis par un nœud « Links » contenant autant de nœuds « Link » qu'il y a de variables ou paramètres externes au composant à définir.

Chaque nœud « Link » possède les attributs suivants :

- Automate contient le label de la variable ou du paramètre de l'automate lié.
- Block contient le label de la variable ou du paramètre dans le block.

```
<!ELEMENT Links (Link*)>
        <!ELEMENT Link EMPTY>
        <!ATTLIST Link Automate CDATA #REQUIRED>
        <!ATTLIST Link Block CDATA #REQUIRED>
```

Figure 16 : DTD associé aux liens des variables

2.3.3 Définition XML des vecteurs de synchronisation

Les vecteurs de synchronisation permettent de spécifier les transitions d'action possibles pour l'automate construit par produit synchronisé.

La définition XML de ces vecteurs est réalisée à partir du nœud <Action> défini au paragraphe 2.1.2. Celui-ci contient autant de nœuds <Comp> qu'il y a de composants définis dans le block.

Les nœuds <Comp> possèdent les attributs suivants :

- Index : contient l'index du composant associé à la composante du vecteur de synchronisation considéré
- Label : contient de nom d'une action du composant indexé
- Type contient le type de cette action (<Vide> | Send | Wait)

```
<!ELEMENT Comp EMPTY>
<!ATTLIST Comp Label CDATA #REQUIRED>
<!ATTLIST Comp Type CDATA #IMPLIED>
<!ATTLIST Comp Index CDATA #IMPLIED>>
```

Figure 17 : DTD associé aux composantes des vecteurs de synchronisation

Par exemple, les vecteurs de synchronisation (renommés) suivants :

- (!begin, * , ?begin) -> begin
- (?CD, ?CD, !CD) -> CD

auraient pour représentation XML (depuis le nœud Actions du block) :

```
<Action>
        <Action Label="begin">
                <Comp Label="begin" Type="Send" Index="1"/>
                <Comp Label="epsilon" Index="2"/>
                <Comp Label="begin" Type="Wait" Index="3"/>
        </Action>
        <Action Label="CD">
                <Comp Label="CD" Type="Wait" Index="1"/>
                <Comp Label="CD" Type="Wait" Index="2"/>
                <Comp Label="CD" Type="Send" Index="3"/>
        </Action>
</Actions>
```

Figure 18 : Exemple de définition des vecteurs de synchronisation

2.4 Format de représentation d'un produit synchronisé

Ce format est intermédiaire et uniquement utilisé pour réaliser la projection d'un composant vers une représentation textuelle.

La représentation XML d'un produit synchronisé est identique à celle d'un block, mais possède en plus un nœud « Templates » contenant la description XML complète des composants utilisés pour construire le produit synchronisé.

Le nœud « Templates » contient donc des nœuds « Automate » ou « Block ».

3 MODELES D'AUTOMATES

Les spécifications XML apportées au paragraphe précédent permettent de décrire des automates (ou des produits synchronisés d'automates) suivant (pour la sémantique tout du moins) le modèle des automates temporisés « Alur et Dill ». Néanmoins, un grand nombre d'extensions de ce modèle de référence existent (on parle alors d'automates hybrides). Citons par exemple :

- Les extensions sur la forme des contraintes (limitées des conjonctions de la forme $x - y < k$ pour les automates temporisés).
- Les extensions des types de données (variables discrètes pour les automates temporisés étendus, files, compteurs, horloges de dérivée modifiable, …)
- L'utilisation de paramètres formels (pour les p-automates par exemple)
- L'ajout de contraintes sur l'environnement (par exemple une unique horloge globale pour les p-automates simples et étendus)

A ces variantes sont bien associés différents outils (utilisant des structures de représentation de données différentes). Citons par exemple :

- Kronos et CMC pour les automates temporisés « classiques » ;
- Uppaal et OpenKronos pour les automates temporisés étendus (ajout des variables discrètes) ;
- HCMC pour les automates hybrides avec « StopWatch » (horloge dont la dérivée vaut 0 ou 1)
- Hytech pour les automates hybrides paramétrés avec horloges de dérivée constante quelconque.

De surcroît, Il existe un grand nombre d'équivalences entre les modèles. Par exemple :

- Tout automate temporisé est un cas particulier d'automate hybride
- Tout automate temporisé peut être réécrit en un p-automate simple (utilisation de variables pour mémoriser la valeur de l'horloge globale et mesurer ainsi les durées)
- Tout p-automate simple est un cas particulier d'automate hybride

Même si ces modèles sont de plus en plus expressif, il est fortement intéressant de manipuler plusieurs abstractions d'un même système, et ce pour des raisons de complexité (grandement croissante suivant l'expressivité du modèle utilisé) et de décidabilité (car les algorithmes de model-checking sont indécidables pour une grande partie des modèles hybrides).

3.1 Modèles d'automates au sein de Calife v3.0

L'environnement Calife3 permet de spécifier (dans des fichiers XML externes) les extensions souhaitées, ainsi que les outils utilisables et les transformations de modèles réalisables. Ces fichiers nommés « Modèles d'automates » sont composés des 4 éléments suivants :

- La déclaration des types de donnée et de paramètres utilisables (et des trans-typages associés) pour spécifier les extensions sur les types de données.
- La déclaration de fonctions (éventuellement polymorphes) utilisables et de leur signature.

- La déclaration de l'environnement obligatoire de l'automate (par exemple une horloge externe S pour les p-automates)
- La déclaration des outils utilisables et des transformations applicables.

Les deux premiers éléments permettent de spécifier sous la forme d'un système de types, les extensions sur la forme des contraintes et les types utilisables dans le modèle.

3.2 Définition du modèle XML

Le modèle utilisé est indispensable à la création d'un nouveau composant (Block ou Automate). De plus, ce modèles est automatiquement recopié dans le document XML créé, sous le nœud « Modele ».

3.2.1 Déclaration de l'environnement général du modèle

Un modèle XML contient un nœud « Environnement » représentant l'ensemble des variables et des paramètres indispensables à la création d'un nouveau composant selon le modèle.

Le nœud « Environnement » suit la DTD définie au paragraphe 2.1.1.

Par exemple, le modèle associé aux p-automates définit l'environnement suivant :

```
<Environnement>
        <Externe>
                <Var Label="S" Type="Clock"/>
        </Externe>
        <Local/>
</Environnement>
```

Figure 19 : Environnement du modèles des p-automates

De plus, le type *Clock* n'étant pas autorisé dans le modèle des p-automates, cette variable S sera l'unique horloge de tous les composants à base de p-automates.

3.2.2 Déclaration des types de donnée

La déclaration des types de donnée est réalisée à partir d'un nœud <*Types*> contenant des nœuds <*VarType*> pour déclarer un type de variables, <*ParamType*> pour déclarer un type de paramètres et <*ConstType*> pour déclarer des types constants.

Chacun de ces nœuds possède un attribut *Label* associé au nom du type déclaré. De plus, les nœuds de type <*ParamType*> contiennent (de manière optionnelle), un attribut *Value* permettant de spécifier si un paramètre formel de ce type est obligatoirement instancié ou non. Enfin, les nœuds <*ConstType*> contiennent un attribut *Match* définissant, sous la forme d'une expression régulière, la forme d'un terme constant.

De plus, le nœud <Types> contient un certain nombre de nœuds <*Cast*> utilisés pour définir les trans-typages possibles entre les différents types définis.

Les nœuds <Cast> contiennent les attributs :

- *From* : pour spécifier le type à l'origine du trans-typage.
- *To* : pour spécifier le type résultant du trans-typage.

Il est possible d'utiliser dans les nœuds <*Cast*> des types non définis en tant que type de variables ou de paramètres

```
<!ELEMENT Types (VarType|ParamType|ConstType|Cast)* >
        <!ELEMENT VarType EMPTY>
        <!ATTLIST VarType Label CDATA #REQUIRED>
        <!ELEMENT ParamType EMPTY>
        <!ATTLIST ParamType Label CDATA #REQUIRED>
        <!ATTLIST ParamType Value Required #IMPLIED>
        <!ELEMENT ConstType EMPTY>
        <!ATTLIST ConstType Label CDATA #REQUIRED>
        <!ATTLIST ConstType Match CDATA #REQUIRED>
        <!ELEMENT Cast EMPTY>
        <!ATTLIST Cast From CDATA #REQUIRED>
        <!ATTLIST Cast To CDATA #REQUIRED>
```

Figure 20 : DTD associée à la déclaration des types

Remarque : dans l'environnement Calife, un type nommé *Prop* est toujours implicitement défini. Ce type est associé au type des contraintes (gardes, invariants et mises à jour).

Par exemple, les types utilisables dans les automates temporisés « Alur et Dill » pourraient être définis de la manière suivante :

```
<Types >
    <VarType Label="Clock"/>
    <ParamType Label="Z" Value="Required"/>
    <ConstType Label="Z" Match="(-)?[0-9]+"/>
    <Cast From="Z" To="Time"/>
    <Cast From="Clock" To="Time"/>
</Types>
```

Figure 21 : Déclaration des types pour les automates temporisés

3.2.3 Déclaration des fonctions utilisables

Les fonctions utilisables sont déclarées par l'intermédiaire du nœud <*Fonctions*>, contenant des nœuds <*Fun*> déclarant chacun une fonction.

Chaque nœud <*Fun*> contient les attributs suivants :

- Label : contient le nom de la fonction ainsi définie
- Type : contient le (ou les, dans le cadre d'une fonction polymorphe) type résultant de l'application de la fonction
- Arg : contient l'ensemble (ou les ensembles) des types attendus en argument de la fonction. Ces différents types sont séparés par le symbole « : ».

Lors de la déclaration d'une fonction polymorphe, les différents types résultants et ensemble de types en argument sont séparés par le symbole « ; ».

Par exemple, la grammaire BNF suivante (associée aux gardes des automates temporisés) :

```
Z := (-)?[0-9]+
    | Z – Z
    | Z + Z
    | Z * Z

Time :=  Clock
          | Clock – Clock
          | Z * Clock

Garde := Time < Z | Time > Z | Time <= Z | Time >=Z | Time = Z
           | Garde && Garde
```

Figure 22 : Grammaire BNF des gardes des automates temporisés

pourrait être associée à la déclaration des fonctions suivantes :

```
<Fonctions>
          <Fun Label="PLUS" Type="Z" Arg="Z:Z"/>
          <Fun Label="MULT" Type="Z" Arg="Z:Z"/>
          <Fun Label="MOINS" Type="Time;Z" Arg="Clock:Clock;Z:Z "/>
          <Fun Label="LT" Type="Prop" Arg="Time:Z;Z:Time"/>
          <Fun Label="LTE" Type="Prop" Arg=" Time:Z;Z:Time "/>
          <Fun Label="GT" Type="Prop" Arg=" Time:Z;Z:Time "/>
          <Fun Label="GTE" Type="Prop" Arg=" Time:Z;Z:Time "/>
          <Fun Label="EQ" Type="Prop" Arg=" Time:Z;Z:Time "/>
          <Fun Label="AND" Type="Prop" Arg="Prop:Prop"/>
</Fonctions>
```

Figure 23 : Définition des fonctions utilisables pour les automates temporisés

3.2.4 Typage de l'arbre de syntaxe abstraite

Ce paragraphe décrit l'algorithme utilisé pour typer les arbres de syntaxe abstraite générés par les parseurs de l'environnement Calife v3.0.

Le typage de l'arbre est réalisé de la manière suivante :

1. Typage des feuilles de l'arbre : les feuilles de l'arbre, constitués au départ de nœuds <*VAR*> sont renommées en nœuds <*Const*> (lorsque le contenu du nœud suit l'expression régulière saisie dans la déclaration des types constants) , <*VLoc*>, <*PLoc*>, <*VExt*> ou <*PExt*> (suivant les variables et les paramètres déclarés dans l'environnement du composant). L'attribut Type de ces nœuds est complété au cours de cette phase.
2. On propage le type, depuis les feuilles de l'arbre vers le nœud parent et on tente de typer ce nœud en se référant aux définitions de fonctions du modèle. Lorsque plusieurs signatures sont possibles, la signature choisie est celle qui minimise le nombre de trans-typages utilisés.
3. On vérifie que l'arbre est correctement typé (c'est à dire que la racine de l'arbre de syntaxe abstraite est typée *Prop*).

Remarque : les grammaires induites par la définition (dans les modèles) d'un système de type ne doivent pas être ambiguës. La plupart des modèles « classiques » d'automate seront réécrits et mis à la disposition de l'utilisateur. Néanmoins, en laissant à l'utilisateur la possibilité d'écrire ses modèles, celui-ci pourra facilement connecter de nouveaux outils à l'environnement Calife. Par exemple, le model-checker Fast, fonctionnant pour des automates à compteur dont les gardes sont des prédicats de l'arithmétique de Presbuger, a été très facilement connecté à l'environnement Calife.

3.3 Ajout des outils

L'objectif de la définition de modèles est de regrouper un ensemble d'outils pouvant fonctionner sur la même famille d'automates. Par exemple, aux vues des équivalences de modèle présentées en introduction du paragraphe 3, les outils suivants sont utilisables à partir d'un système d'automates temporisés : Kronos, OpenKronos, Uppaal, Hytech, HCMC, …

Les outils cités ci-dessus sont tous des model-checkers, mais il est également possible d'utiliser des assistants de preuve, des générateurs de test temporisés, …

Les modèles d'automate utilisés dans Calife décrivent, à partir d'un nœud nommé <*Exports*>, un certain nombre d'exports (définis par des nœuds <*Export*>) possibles depuis un système défini selon les contraintes de typage exposées ci-dessus.

Ces exports sont réalisés à partir d'un langage de script codé en XML. Les nœuds suivants sont en effet interprétés en actions, réalisées par l'environnement Calife.

3.3.1 Définition de variables

Dans un script XML, la valeur d'une variable est notée ${<Nom de la variable>}.

Un certain nombre de variables sont définies d'une manière indépendante au script : ces définitions concernent les variables suivantes :

- ${CalifeDir} représente le répertoire d'installation de l'environnement Calife.
- ${Projet} contient le nom du projet en cours d'édition.
- ${Current} contient le label du composant sur lequel est réalisé l'export. Ce composant peut-être un block ou un automate.
- ${WorkDir} représente le répertoire de travail de l'application (répertoire contenant les projets).
- Toute variable définie dans le fichier .calife (contenu dans le répertoire $HOME de l'utilisateur) sous la forme Variable = Valeur est accessible dans les scripts sous le nom ${<Variable>}.

De plus, il est possible de définir une variable locale à un script d'export. Cette opération est réalisée par l'intermédiaire d'un nœud XML nommé <*Property*>.

Ce nœud contient obligatoirement les 2 attributs suivants :

- *Name* définit le nom de la nouvelle variable.
- *Value* définit la valeur associée à cette variable. Il est possible dans cet attribut, de faire référence à d'autres variables déjà définies.

Par exemple, pour définir une variable nommé DestDir et pointant vers un répertoire de destination pour générer du code Hytech, la syntaxe XML serait la suivante :

```
<Property Value="${WorkDir}/${Projet}/Hytech" Name="DestDir" />
```

3.3.2 Actions élémentaires

Les nœuds XML suivants sont associés à des actions élémentaires composants un script XML :

Nom du nœud XML	Liste des attributs	Description
Projection	Rules Source (Opt)	Applique la transformation XSL définie par l'attribut <Rules> au fichier XML définit par l'attribut <Source>. Si l'attribut <Source> est manquant, le composant XML est le composant courrant.
Split	Source OutputDir	coupe le fichier <Source> en fonction des ${SPLIT:<filename>} rencontrés dans le fichier. Les fichiers sont créés dans le répertoire <OutputDir>
FileCat	Source Output	concatène les fichiers <Source> dans le fichier <Output>. L'attribut <Source> peut contenir des jockers (*.clf par exemple) pour repérer les fichiers. FileCat peut également être utilisé pour copier un fichier texte.
Delete	Source	supprime les fichiers ou les répertoires passés dans <Source>
Execute	Command Directory (Opt)	exécute la commande <Command> à partir du répertoire <Directory>. Si l'attribut <Directory> est manquant, le répertoire d'exécution est le répertoire ${CalifeDir}
Interactive	Command Directory (Opt)	Fonctionne comme le nœud Execute mais pour une commande attendant des entrées de l'utilisateur. Ces entrées seront réalisée par l'intermédiaire de l'interface CalifeEdit
Message	String (Opt) File (Opt)	Affiche la chaîne <String> puis le contenu du fichier <File> dans la console de l'environnement Calife ou de l'interface CalifeEdit

3.3.3 L'interface CalifeEdit

Cet interface permet d'interagir avec l'utilisateur pendant l'exécution d'un script XML.

Elle est composée de deux éditeurs de texte contenant 2 fichiers passés en paramètres (attributs <Left> et <Right>). Si les fichiers passés en paramètre n'existent pas, ils sont automatiquement crées. Dans le cas contraire, les fichiers sont automatiquement ouverts par l'application.

L'interface CalifeEdit contient également une console permettant de visualiser le résultat de l'exécution d'une commande (par l'intermédiaire d'un nœud <Execute> par exemple), et une ligne permettant de saisir une commande à envoyer à un exécutable (lancé par l'intermédiaire d'un nœud <Interactive>).

L'interface possède enfin les menus suivants :

- le menu « Fichier » permet de sauvegarder les documents édités en cas de modification
- le menu « Actions » contient des items définis, dans le script XML par des nœuds nommés <*ActionItem*>. Ces nœuds contiennent ensuite un script exécuté lorsque les sous-menus sont sélectionnés.

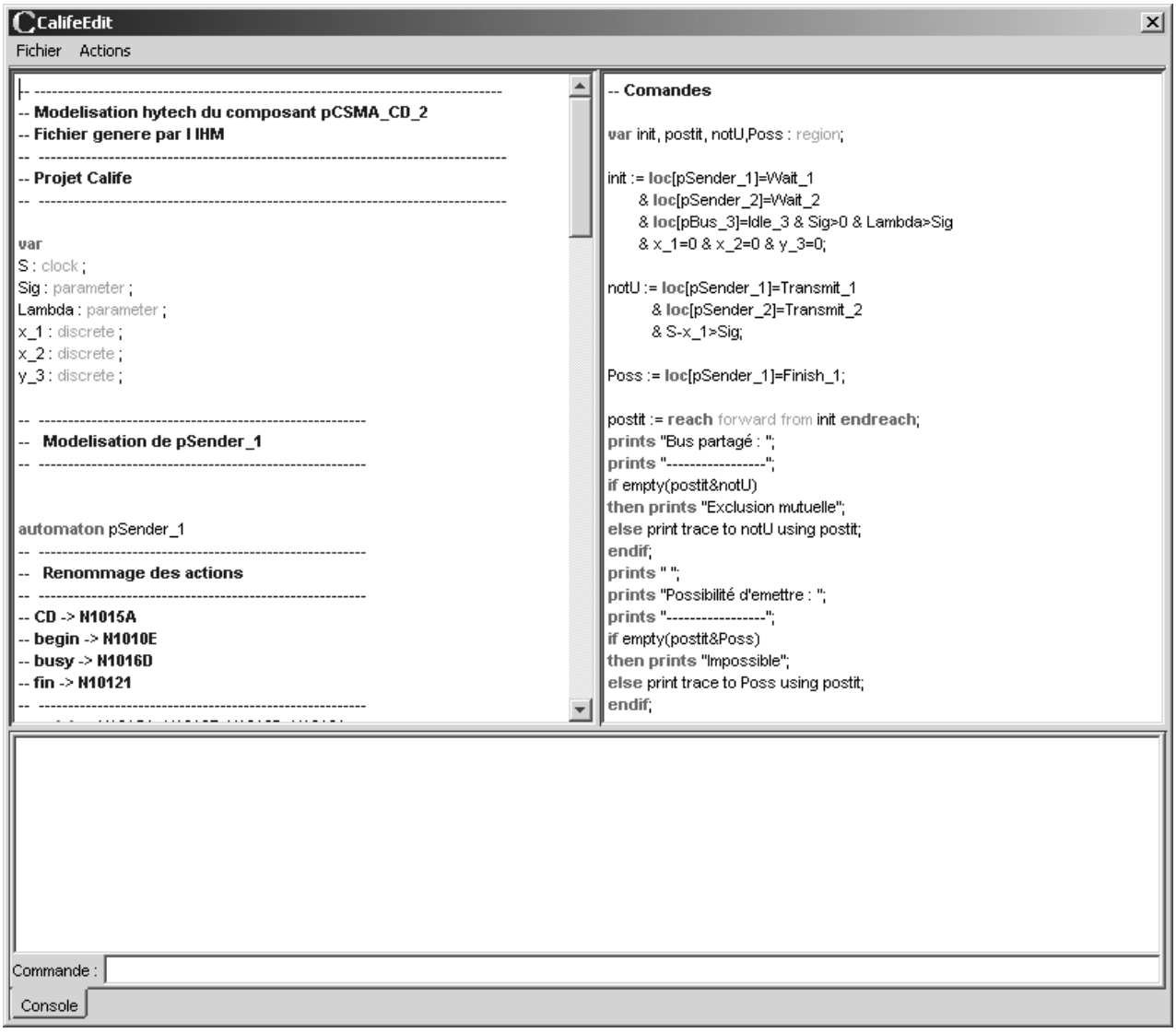


Figure 24 : Vue d'ensemble de l'interface CalifeEdit

3.3.4 Exemple de script XML : la projection vers le model-checker Kronos

Le model-checker Kronos fonctionne à partir d'une description sous la forme d'un fichier par automate.

Pour réaliser une peuve par model-checking, il est tout d'abord nécessaire de construire le produit synchronisé des différents automates composants le système. Ceci est réalisé par la commande « Kronos –out <Nom du produit synchronisé> <Noms des différents automates du système> »

A partir de ce produit synchronisé il est possible de réaliser une exploration de l'espace d'état « vers l'avant » (c'est à dire depuis l'état initial) ou vers l'arrière (à partir de la propriété).

L'export, depuis l'environnement Calife vers l'outils Kronos est réalisé par le script suivant :

```
<Export Label="Kronos" Type="Model-Checker">
   <Property Name="DestDir" Value="${WorkDir}/${Projet}/Kronos"/>
   <Property Name="TempDir" Value="C:/Temp/Kronos"/>
   <Projection Rules="${CalifeDir}/Rules/tg.xsl" Output="${DestDir}/${Current}.tmp"/>

   <!-- Constuction du produit synchronisé -->
   <Split Source="${DestDir}/${Current}.tmp" OutputDir="${TempDir}/Temp"/>
   <Execute Command="kronos -out ${TempDir}/${Current}.tg ${TempDir}/Temp/*.tg"/>
   <CalifeEdit Left="${DestDir}/${Current}.tmp" Right="${WorkDir}/${Projet}/${Current}.tctl" HighLight="Kronos">
     <ActionItem Label="Analyse Avant">
               <Delete Source="${TempDir}/*.eval"/>
               <FileCat Source="${WorkDir}/${Projet}/${Current}.tctl" Output="${TempDir}/${Current}.tctl"/>
              <Execute Command="kronos -forw ${TempDir}/${Current}.tg ${TempDir}/${Current}.tctl"/>
              <Message String="Resultat de l'evaluation Kronos : " File="${TempDir}/${Current}.eval"/>
     </ActionItem>
     <ActionItem Label="Analyse Arriere">
               <Delete Source="${TempDir}/*.eval"/>
               <FileCat Source="${WorkDir}/${Projet}/${Current}.tctl" Output="${TempDir}/${Current}.tctl"/>
             <Execute Command="kronos -back ${TempDir}/${Current}.tg ${TempDir}/${Current}.tctl"/>
              <Message String="Resultat de l'evaluation Kronos : " File="${TempDir}/${Current}.eval"/>
     </ActionItem>
     <ActionItem Label="Accessibilite">
               <Delete Source="${TempDir}/*.eval"/>
               <FileCat Source="${WorkDir}/${Projet}/${Current}.tctl" Output="${TempDir}/${Current}.tctl"/>
              <Execute Command="kronos -reach ${TempDir}/${Current}.tg ${TempDir}/${Current}.tctl"/>
               <Message String="Resultat de l'evaluation Kronos : " File="${TempDir}/${Current}.eval"/>
     </ActionItem>
   </CalifeEdit>
   <Delete Source="${TempDir}"/>
</Export>
```

Figure 25 : Script d'export vers l'outil Kronos

Le script doit être lu de la manière suivante :

- Définition d'une variable DestDir contenant le répertoire de destination des fichiers
- Définition d'une variable TempDir pointant vers un répertoire temporaire
- Application du fichier tg.xsl au composant courant pour générer le fichier ${Current}.tmp
- Le fichier ${Current}.tmp est découpé selon les balises ${SPLIT :<Nom de fichier>} rencontrées (toutes de type <Nom d'un automate>.tg).
- Construction du produit synchronisé
- Ouverture de l'interface CalifeEdit contenant :
    - A gauche le fichier .tmp définissant l'ensemble des automates
    - A droite le fichier .tctl dans lequel sera sauvegardée par propriété à prouver
- L'interface CalifeEdit contient 3 sous-menus Action nommés :
    - Analyse Avant (exécution de kronos avec l'option -forw)
    - Analyse Arrière (option -back)
    - Accessibilité (option -reach)
- A la fermeture de l'interface CalifeEdit, suppression du répertoire temporaire

4 CONCLUSION

Ce document décrit l'ensemble des structures permettant de représenter, au format XML, les différents objets manipulés par l'environnement Calife.

Même si cette structure ne sera pas manipulée par le simple utilisateur de l'environnement, cette description est nécessaire pour l'utilisateur désireux :

- D'écrire ses propres transformations sur des modèles existants (pour par exemple interfacer de nouveaux outils)
- D'écrire ses propres scripts permettant d'exporter les composants saisis sous l'environnement Calife
- De créer de nouveaux modèles actuellement non supportés par l'environnement.

ANNEXES

DTD d'un composant XML

```
<!ELEMENT Block (Modele,Environnement,Components,Actions) >
<!ATTLIST Block Label CDATA #REQUIRED>
<!ATTLIST Block Modif (True|False) #IMPLIED>

<!ELEMENT Automate (Modele,Environnement,Actions,(Localite|Deplacement)*) >
<!ATTLIST Automate Label CDATA #REQUIRED>
<!ATTLIST Automate Modif (True|False) #IMPLIED>

<!ELEMENT Actions (Action*)>
        <!ELEMENT Action (Comp*)>
        <!ATTLIST Action Label CDATA #REQUIRED>
        <!ATTLIST Action Type CDATA #IMPLIED>
            <!ELEMENT Comp EMPTY>
            <!ATTLIST Comp Label CDATA #REQUIRED>
            <!ATTLIST Comp Type CDATA #IMPLIED>
            <!ATTLIST Comp Index CDATA #IMPLIED>

    <!ELEMENT Components (Component*)>
        <!ELEMENT Component (Links)>
        <!ATTLIST Component Label CDATA #REQUIRED>
        <!ATTLIST Component Type CDATA #REQUIRED>
        <!ATTLIST Component Index CDATA #REQUIRED>

            <!ELEMENT Links (Link*)>
                <!ELEMENT Link EMPTY>
                <!ATTLIST Link Automate CDATA #REQUIRED>
                <!ATTLIST Link Block CDATA #REQUIRED>


<!ELEMENT Modele (Types,Fonctions,Environnement,Exports) >
<!ATTLIST Modele Label CDATA #REQUIRED>

<!ELEMENT Types (VarType|ParamType|Cast)* >
<!ATTLIST Types AllowDefinition (True|False) #IMPLIED>

    <!ELEMENT VarType EMPTY>
    <!ATTLIST VarType Label CDATA #REQUIRED>
    <!ELEMENT ParamType EMPTY>
    <!ATTLIST ParamType Label CDATA #REQUIRED>
    <!ELEMENT Cast EMPTY>
    <!ATTLIST Cast From CDATA #REQUIRED>
    <!ATTLIST Cast To CDATA #REQUIRED>

<!ELEMENT Fonctions (Fun*) >
<!ATTLIST Fonctions AllowDefinition (True|False) #IMPLIED>
    <!ELEMENT Fun EMPTY>
    <!ATTLIST Fun Label CDATA #REQUIRED>
    <!ATTLIST Fun Type CDATA #REQUIRED>
    <!ATTLIST Fun Arg CDATA #REQUIRED>

<!ELEMENT Exports (Export*) >
    <!ELEMENT Export (Property
```

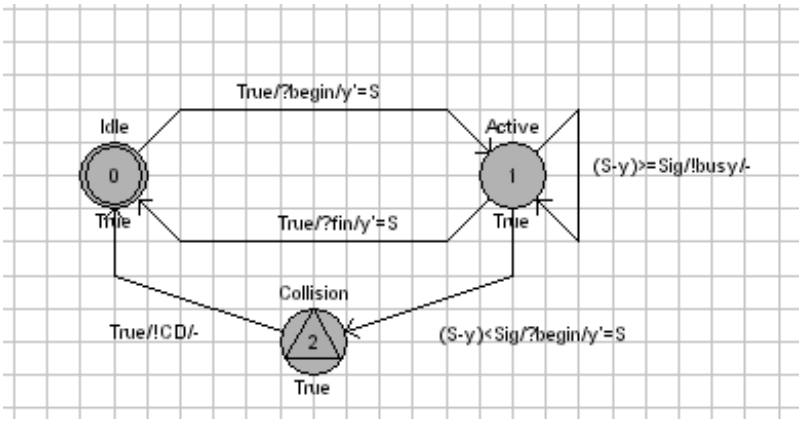
```
                                        |Projection
                                        |Split
                                        |Execute
                                        |Message
                                        |Delete
                                        |CalifeEdit
                                        |FileCat)* >
        <!ATTLIST Export Type CDATA #REQUIRED>
        <!ATTLIST Export Label CDATA #REQUIRED>
                <!ELEMENT CalifeEdit (ActionItem*) >
                <!ATTLIST CalifeEdit Left CDATA #IMPLIED>
                <!ATTLIST CalifeEdit Right CDATA #IMPLIED>
                <!ATTLIST CalifeEdit HighLight CDATA #IMPLIED>
                        <!ELEMENT ActionItem (Projection
                                        |Split
                                        |Execute
                                        |Message
                                        |Delete
                                        |Interactive
                                        |FileCat)* >
                        <!ATTLIST ActionItem Label CDATA #REQUIRED>

                <!ELEMENT Property EMPTY>
                <!ATTLIST Property Name CDATA #REQUIRED>
                <!ATTLIST Property Value CDATA #REQUIRED>

                <!ELEMENT Projection EMPTY>
                <!ATTLIST Projection Source CDATA #IMPLIED>
                <!ATTLIST Projection Rules CDATA #REQUIRED>
                <!ATTLIST Projection Output CDATA #REQUIRED>

                <!ELEMENT Split EMPTY>
                <!ATTLIST Split Source CDATA #REQUIRED>
                <!ATTLIST Split OutputDir CDATA #REQUIRED>

                <!ELEMENT Execute EMPTY>
                <!ATTLIST Execute Command CDATA #REQUIRED>
                <!ATTLIST Execute Directory CDATA #IMPLIED>

                <!ELEMENT Message EMPTY>
                <!ATTLIST Message File CDATA #IMPLIED>
                <!ATTLIST Message String CDATA #IMPLIED>

                <!ELEMENT Delete EMPTY>
                <!ATTLIST Delete Source CDATA #REQUIRED>

                <!ELEMENT Interactive EMPTY>
                <!ATTLIST Interactive Command CDATA #REQUIRED>
                <!ATTLIST Interactive Directory CDATA #IMPLIED>

                <!ELEMENT FileCat EMPTY>
                <!ATTLIST FileCat Source CDATA #REQUIRED>
                <!ATTLIST FileCat Output CDATA #REQUIRED>

<!ELEMENT Environnement (Externe,Local) >
                <!ELEMENT Externe (Var|Param)* >
                <!ELEMENT Local (Var|Param)* >
                        <!ELEMENT Var EMPTY>
                        <!ATTLIST Var Label CDATA #REQUIRED>
                        <!ATTLIST Var Type CDATA #REQUIRED>
                        <!ELEMENT Param EMPTY>
                        <!ATTLIST Param Label CDATA #REQUIRED>
                        <!ATTLIST Param Type CDATA #REQUIRED>
                        <!ATTLIST Param Value CDATA #IMPLIED>
```

```
<!ELEMENT Localite (Centre,Invariant)>
<!ATTLIST Localite Label CDATA #REQUIRED>
<!ATTLIST Localite Id CDATA #REQUIRED>
<!ATTLIST Localite Initial (true|false) #REQUIRED>
<!ATTLIST Localite Propriete CDATA #IMPLIED>

        <!ELEMENT Centre EMPTY>
        <!ATTLIST Centre x CDATA #REQUIRED>
        <!ATTLIST Centre y CDATA #REQUIRED>

        <!ELEMENT Invariant (Contrainte,Activite?)>
        <!ATTLIST Invariant PrettyPrint CDATA #REQUIRED>


                <!ELEMENT Activite (FunT|Const)>
                <!ATTLIST Activite Type CDATA #REQUIRED>
                <!ATTLIST Activite Arg CDATA #REQUIRED>

                <!ELEMENT Contrainte (FunT|Const)>
                <!ATTLIST Contrainte Type CDATA #REQUIRED>
                <!ATTLIST Contrainte Arg CDATA #REQUIRED>

                <!ELEMENT FunT (FunT|Const|VAR|VLoc|PLoc|VExt|PExt|OUT)* >
                <!ATTLIST FunT Label CDATA #REQUIRED>
                <!ATTLIST FunT Type CDATA #REQUIRED>
                <!ATTLIST FunT Arg CDATA #REQUIRED>

                <!ELEMENT OUT (FunT|Const|VAR|VLoc|PLoc|VExt|PExt)* >
                <!ATTLIST OUT Type CDATA #REQUIRED>
                <!ATTLIST OUT Arg CDATA #REQUIRED>

                        <!ELEMENT Const (#PCDATA) >
                        <!ATTLIST Const Type CDATA #REQUIRED>
                        <!ELEMENT VAR (#PCDATA) >
                        <!ATTLIST VAR Type CDATA #REQUIRED>
                        <!ELEMENT VLoc (#PCDATA) >
                        <!ATTLIST VLoc Type CDATA #REQUIRED>
                        <!ELEMENT PLoc (#PCDATA) >
                        <!ATTLIST PLoc Type CDATA #REQUIRED>
                        <!ELEMENT VExt (#PCDATA) >
                        <!ATTLIST VExt Type CDATA #REQUIRED>
                        <!ELEMENT PExt (#PCDATA) >
                        <!ATTLIST PExt Type CDATA #REQUIRED>

<!ELEMENT Deplacement (Vertex|TextNode|Garde|Act|Updates)*>
<!ATTLIST Deplacement Source CDATA #REQUIRED>
<!ATTLIST Deplacement Target CDATA #REQUIRED>

        <!ELEMENT Vertex EMPTY>
        <!ATTLIST Vertex x CDATA #REQUIRED>
        <!ATTLIST Vertex y CDATA #REQUIRED>
        <!ELEMENT TextNode EMPTY>
        <!ATTLIST TextNode PrettyPrint CDATA #REQUIRED>
        <!ATTLIST TextNode x CDATA #REQUIRED>
        <!ATTLIST TextNode y CDATA #REQUIRED>

        <!ELEMENT Garde (Contrainte)>
        <!ATTLIST Garde PrettyPrint CDATA #REQUIRED>

        <!ELEMENT Act EMPTY>
        <!ATTLIST Act Label CDATA #IMPLIED>
        <!ATTLIST Act Type CDATA #IMPLIED>

        <!ELEMENT Updates (Update*)>
        <!ATTLIST Updates PrettyPrint CDATA #REQUIRED>
                <!ELEMENT Update (FunT|Const)>
                <!ATTLIST Update Type CDATA #REQUIRED>
```

<!ATTLIST Update Arg CDATA #REQUIRED>

Exemple de composant XML

Le fichier XML ci-dessous est associé au p-automate suivant, saisi sous l'environnement Calife et typé en suivant l'algorithme décrit au paragraphe 3.2.4.




```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE Automate SYSTEM "../Component.dtd">
<Automate Modif="False" Label="pBus">
  <Modele Label="P-Automates restreints">
    <Types AllowDefinition="False">
      <VarType Label="Time" />
      <VarType Label="Z" />
      <VarType Label="nat" />
      <ParamType Label="Time" />
      <ParamType Label="Z" />
      <ParamType Label="nat" />
      <Cast To="Z" From="nat" />
      <Cast To="Time" From="Z" />
      <Cast To="Clock" From="Time" />
    </Types>

    <Fonctions AllowDefinition="False">
      <Fun Arg="Clock:Clock;Time:Time;Z:Z;nat:nat" Type="Clock;Time;Z;nat" Label="PLUS" />
      <Fun Arg="Clock:Clock;Time:Time;Z:Z;nat:nat" Type="Clock;Time;Z;Z" Label="MOINS" />
      <Fun Arg="Z:Clock;Clock:Z;Z:Z;nat:nat" Type="Clock;Clock;Z;nat" Label="MULT" />
      <Fun Arg="Prop:Prop" Type="Prop" Label="AND" />
      <Fun Arg="Clock:Clock" Type="Prop" Label="LT" />
      <Fun Arg="Clock:Clock" Type="Prop" Label="LTE" />
      <Fun Arg="Clock:Clock" Type="Prop" Label="GT" />
      <Fun Arg="Clock:Clock" Type="Prop" Label="GTE" />
      <Fun Arg="Clock:Clock" Type="Prop" Label="EQ" />
      <Fun Arg="Clock:Clock" Type="Prop" Label="NEQ" />
    </Fonctions>

    <Environnement>
      <Externe>
        <Var Type="Clock" Label="S" />
      </Externe>

      <Local />
    </Environnement>

    <Exports>
      <Export Type="Model-Checker" Label="Hytech">
        <Property Value="${WorkDir}/${Projet}/Hytech" Name="DestDir" />
```

```
        <Projection Output="${DestDir}/${Current}.hy" Rules="${CalifeDir}/Rules/hy-pautomata.xsl" />
        <CalifeEdit HighLight="Hytech" Right="${WorkDir}/${Projet}/${Current}.prop" Left="${DestDir}/${Current}.hy">
          <ActionItem Label="Run Hytech">
            <FileCat Output="${DestDir}/Script.hy"
Source="${DestDir}/${Current}.hy;${WorkDir}/${Projet}/${Current}.prop" />

            <Execute Command="hytech ${DestDir}/Script.hy" />
          </ActionItem>

          <ActionItem Label="Command">
            <Interactive Command="cmd" />
          </ActionItem>
        </CalifeEdit>
      </Export>

      <Export Type="Outil graphique" Label="PostScript">
        <Projection Output="${WorkDir}/${Projet}/${Current}.ps" Rules="${CalifeDir}/Rules/ps.xsl" />

        <Message String="Fichier ${WorkDir}/${Projet}/${Current}.ps généré !" />
      </Export>

      <Export Type="Assistant de Preuve" Label="Coq">
        <Property Value="${WorkDir}/${Projet}/Coq" Name="DestDir" />
        <Projection Output="${DestDir}/${Projet}.tmp" Rules="${CalifeDir}/Rules/coq_module.xsl" />
        <Split OutputDir="${DestDir}" Source="${DestDir}/${Projet}.tmp" />
      </Export>
    </Exports>
  </Modele>

  <Environnement>
    <Externe>
      <Var Type="Clock" Label="S" />
      <Param Label="Lambda" Type="Time" />
      <Param Label="Sig" Type="Time" />
    </Externe>

    <Local>
      <Var Type="Time" Label="y" />
    </Local>
  </Environnement>

  <Actions>
    <Action Label="begin" Type="Wait" />
    <Action Label="fin" Type="Wait" />
    <Action Label="CD" Type="Send" />
    <Action Label="busy" Type="Send" />
  </Actions>

  <Localite Propriete="" Initial="true" Id="0" Label="Idle">
    <Centre x="920" y="820" />

    <Invariant PrettyPrint="True">
      <Contrainte Arg="Undef" Type="Undef">
        <Const Type="Prop">True</Const>
      </Contrainte>
    </Invariant>
  </Localite>

  <Localite Propriete="" Initial="false" Id="1" Label="Active">
    <Centre x="1260" y="820" />

    <Invariant PrettyPrint="True">
      <Contrainte Arg="Undef" Type="Undef">
        <Const Type="Prop">True</Const>
      </Contrainte>
    </Invariant>
  </Localite>
```

```
<Localite Propriete="asap" Initial="false" Id="2" Label="Collision">
  <Centre x="1080" y="1040" />

  <Invariant PrettyPrint="True">
    <Contrainte Arg="Undef" Type="Undef">
      <Const Type="Prop">True</Const>
    </Contrainte>
  </Invariant>
</Localite>

<Deplacement Source="0" Target="1">
  <Vertex x="960" y="760" />

  <Vertex x="1220" y="760" />

  <Garde PrettyPrint="True">
    <Contrainte Arg="Undef" Type="Undef">
      <Const Type="Prop">True</Const>
    </Contrainte>
  </Garde>

  <Act Label="begin" Type="Wait" />

  <Updates PrettyPrint="y'=S">
    <Update Arg="Undef" Type="Undef">
      <FunT Arg="Undef" Type="Time" Label="EQ">
        <OUT Arg="Undef" Type="Undef">
          <VAR Type="Undef">y</VAR>
        </OUT>

        <VAR Type="Undef">S</VAR>
      </FunT>
    </Update>
  </Updates>

  <TextNode PrettyPrint="True/?begin/y'=S" x="1054" y="754" />
</Deplacement>

<Deplacement Source="1" Target="0">
  <Vertex x="1220" y="880" />

  <Vertex x="960" y="880" />

  <Garde PrettyPrint="True">
    <Contrainte Arg="Undef" Type="Undef">
      <Const Type="Prop">True</Const>
    </Contrainte>
  </Garde>

  <Act Label="fin" Type="Wait" />

  <Updates PrettyPrint="y'=S">
    <Update Arg="Undef" Type="Undef">
      <FunT Arg="Undef" Type="Time" Label="EQ">
        <OUT Arg="Undef" Type="Undef">
          <VAR Type="Undef">y</VAR>
        </OUT>

        <VAR Type="Undef">S</VAR>
      </FunT>
    </Update>
  </Updates>

  <TextNode PrettyPrint="True/?fin/y'=S" x="1059" y="873" />
</Deplacement>
```

```
<Deplacement Source="1" Target="2">
  <Vertex x="1260" y="1000" />

  <Garde PrettyPrint="(S-y)&lt;Sig">
    <Contrainte Arg="Undef" Type="Undef">
      <FunT Arg="Undef" Type="Undef" Label="LT">
        <FunT Arg="Undef" Type="Undef" Label="MOINS">
          <VAR Type="Clock">S</VAR>
          <VAR Type="Time">y</VAR>
        </FunT>

        <VAR Type="Undef">Sig</VAR>
      </FunT>
    </Contrainte>
  </Garde>

  <Act Label="begin" Type="Wait" />

  <Updates PrettyPrint="y'=S">
    <Update Arg="Undef" Type="Undef">
      <FunT Arg="Undef" Type="Time" Label="EQ">
        <OUT Arg="Undef" Type="Undef">
          <VAR Type="Undef">y</VAR>
        </OUT>

        <VAR Type="Undef">S</VAR>
      </FunT>
    </Update>
  </Updates>

  <TextNode PrettyPrint="(S-y)&lt;Sig/?begin/y'=S" x="1176" y="1039" />
</Deplacement>

<Deplacement Source="2" Target="0">
  <Vertex x="920" y="1000" />

  <Garde PrettyPrint="True">
    <Contrainte Arg="Undef" Type="Undef">
      <Const Type="Prop">True</Const>
    </Contrainte>
  </Garde>

  <Act Label="CD" Type="Send" />
  <Updates PrettyPrint="" />
  <TextNode PrettyPrint="True/!CD/-" x="917" y="1038" />
</Deplacement>

<Deplacement Source="1" Target="1">
  <Vertex x="1300" y="780" />
  <Vertex x="1300" y="860" />

  <Garde PrettyPrint="(S-y)&gt;=Sig">
    <Contrainte Arg="Undef" Type="Undef">
      <FunT Arg="Undef" Type="Undef" Label="GTE">
        <FunT Arg="Undef" Type="Undef" Label="MOINS">
          <VAR Type="Clock">S</VAR>
          <VAR Type="Time">y</VAR>
        </FunT>
        <VAR Type="Undef">Sig</VAR>
      </FunT>
    </Contrainte>
  </Garde>

  <Act Label="busy" Type="Send" />

  <Updates PrettyPrint="" />
  <TextNode PrettyPrint="(S-y)&gt;=Sig/!busy/-" x="1309" y="818" />
```

```
  </Deplacement>
</Automate>
```