CRIL TECHNOLOGY

Siège & Agence Ile de France
Le Newton
9-11 rue Jeanne Braconnier
92360 MEUDON-LA-FORET
Tél. : 01 40 83 52 52
Fax : 01 40 83 52 53
email : idf@criltechnology.com

Agence Ouest
Espace Nobel - Bât B
Rue Antoine Becquerel
35708 RENNES Cédex 7
Tél. : 02 23 21 11 11
Fax : 02 23 21 11 00
email : ouest@criltechnology.com

Agence Sud-Ouest
2 imp. Henri Pitot
Parc Activité La Plaine
31500 TOULOUSE
Tél. : 05 62 16 79 79
Fax : 05 61 20 47 89
email : sudouest@criltechnology.com

Manuel utilisateur
Calife v3.0
RNTL AVERROES

SOMMAIRE

TABLE DES FIGURES

INTRODUCTION

La plate-forme Calife v3.0 est un environnement développé sous licence GPL (voir annexe 1), permettant la spécification et la validation formelle de systèmes décrits sous la forme d'un produit synchronisé d'automates.

Le présent document est construit de la manière suivante :

- Le chapitre 1 présente pas à pas, la procédure permettant d'installer la plate-forme Calife sous un environnement Windows ou Linux
- Le chapitre 2 propose une présentation rapide des possibilités de l'environnement Calife v3.0
- Le chapitre 3 présente une vue d'ensemble de l'environnement Calife v3.0
- Le chapitre 4 s'attache plus particulièrement à l'éditeur de composants
- Le chapitre 5 présente le mécanisme de modèles d'automates implémenté dans la plate-forme
- Le chapitre 6 expose les fonctionnalités d'export proposées par la plate-forme.

1 INSTALLATION DE LA PLATE-FORME

La plate-forme Calife fonctionne sous les environnements suivants :

- Windows 95/98/NT4/2000/XP sous l'environnement Cygwin (http://www.cygwin.com)
- Linux (Mandrake, RedHat, Suze, Debian)
- Sans doute bien d'autres systèmes compatibles java et gcc...

1.1 Installations préliminaires

La plate-forme Calife est développée en Java (JDK 1.4). Une machine virtuelle Java, ainsi que le compilateur associé (javac) doivent impérativement être installés au préalable et présents dans le contenu de la variable d'environnement PATH (cf. paragraphe 1.1.2.2).

Pour vérifier la version de la machine virtuelle présente sur l'ordinateur, tapez *javac –version* depuis une fenêtre terminal.

Si la version utilisée est inférieure à 1.4, installez une version récente, téléchargeable depuis le site http://www.java.sun.com.

En outre, le mécanisme mis en place pour simuler les automates saisis sous Calife utilise le logiciel élan pour générer du code C. Il est donc nécessaire de disposer de l'ensemble des librairies décrites ci-dessous pour compiler Elan, ainsi que le code généré par celui-ci.

1.1.1 Préparation de l'installation sous Linux, Unix, Mac, …

Vous devez tout d'abord vous assurer que les logiciels suivants sont bien installés sur votre machine et accessibles à l'aide de la variable d'environnement $PATH :

- autoconf
- automake
- bison (ou équivalent)
- flex (ou équivalent)
- gcc
- libtool
- make
- tcsh
- wget

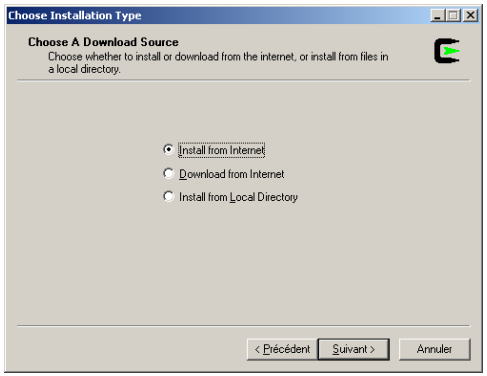
Notons que la plupart de ces logiciels sont présents par défaut sous Linux (et dans tous les cas présents dans le CD de distribution).

1.1.2 Préparation de l'installation sous Windows

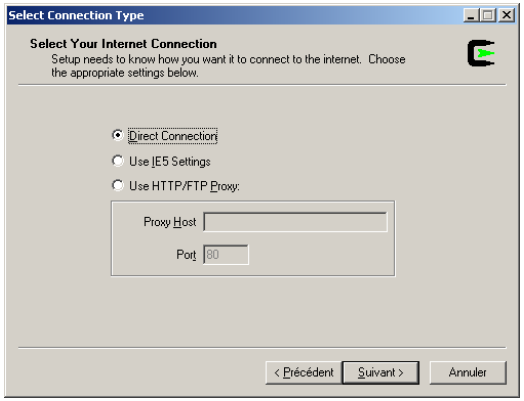
La compilation et l'utilisation d'Elan sous Windows nécessitent l'installation de l'environnement Cygwin dont le programme d'installation « Setup.exe » est disponible sur le site http://www.cygwin.com.

1.1.2.1 Configuration du programme d'installation

- Exécutez le programme Setup.exe téléchargé depuis le site cygwin.com
- Choisissez l'option « Install from Internet »

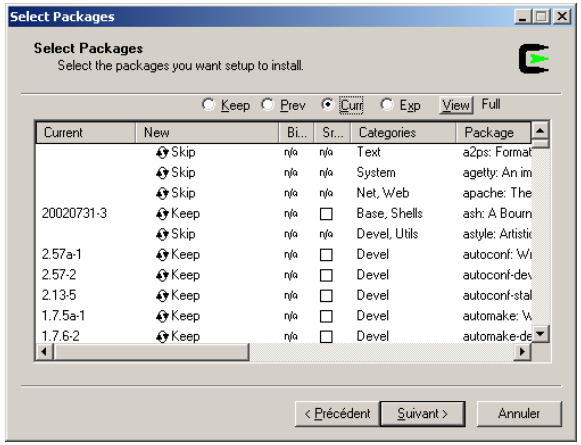


- Choisissez le répertoire où installer cygwin (noté par la suite <Cygwin_Root>) puis un répertoire temporaire dans lequel seront stockés les logiciels téléchargés (« Local package directory »).
- Sur l'écran de configuration de votre connexion Internet, choisissez « Use IE5 Settings » si Internet explorer fonctionne correctement, « Direct Connection » si votre ordinateur n'utilise pas de proxy ou une configuration manuelle si aucune des deux options précédentes ne fonctionnent.



- Choisissez un site miroir à partir duquel seront téléchargés les logiciels sélectionnés par la suite.

- Sur le fenêtre de sélection des paquets, cliquez sur [View] afin de visualiser l'ensemble des paquets logiciels disponibles.



- En conservant les paquets séléctionnés par defaut, ajoutez les paquets suivants :
  - autoconf
  - automake
  - bison
  - flex
  - gcc
  - libtool
  - make
  - tcsh
  - wget
- cliquez sur « Suivant » pour installer les logiciels sélectionnés puis sur « Terminer » pour finaliser l'installation.

1.1.2.2 Configuration de la variable PATH

Pour terminer l'installation de Cygwin, il est nécessaire d'ajouter certains chemins à la variable d'environnement PATH. La méthode est différente suivant la version de Windows utilisée.

- *Sous Windows 95/98 :*

Editez le fichier C:\Autoexec.bat et ajoutez la ligne suivante en fin de fichier :

Set PATH=%PATH%;<Cygwin_Root>\bin

Où <CygWin_Root> est remplacé par le répertoire où vous avez installé Cygwin précédement.

Ensuite, redémarrez l'ordinateur.

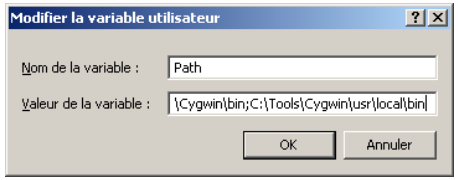
- *Sous Windows NT4/2000/XP :*

Depuis l'icône "Poste de Travail", cliquez avec le bouton droit de la souris et choisissez le menu "Propriétés".

Dans la fenêtre « Propriétés systèmes », choisissez l'onglet « Avancé » et cliquez sur bouton « Variables d'environnement ».

Dans la liste des variables utilisateur, double-cliquez sur la variable path et ajoutez (en remplaçant <Cygwin_Root> par le répertoire où Cygwin est installé) les commandes suivantes à la fin de la liste :

;<Cygwin_Root>\bin



Si la variable Path n'est pas présente dans la liste des variables utilisateurs, créez la en saisissant pour nom « Path » et pour valeur « %Path%;<Cygwin_Root>\bin.

Vous pouvez vérifier que tous les logiciels sont correctement installés en tapant la commande *which gcc* depuis un terminal. La réponse doit alors être */usr/bin/gcc*.

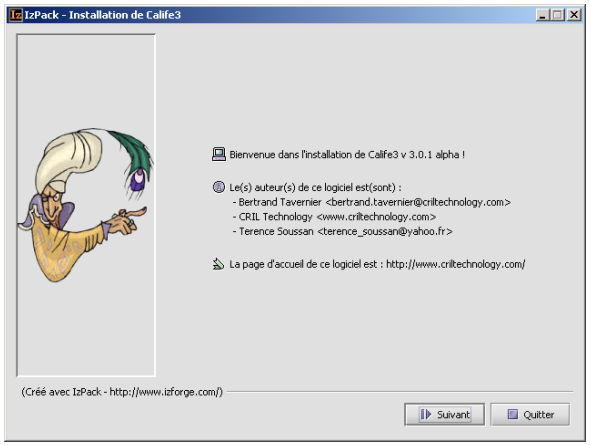
1.2 Installation de l'environnement Calife

La distribution de l'environnement Calife est constituée d'un unique fichier nommé install.jar.

Pour exécuter le programme d'installation, tapez la commande suivante depuis un terminal :

java –jar install.jar

Une fenêtre d'accueil doit alors apparaître :



- Suivez alors les instructions affichées à l'écran pour choisir le répertoire d'installation de l'application, ainsi que le répertoire contenant les fichiers « projets ».
- Sélectionnez ensuite les Packs à installer. Le choix des packs dépend de la configuration de votre machine (et en particulier si Elan est déjà présent sur votre machine). Les packs disponibles sont les suivants :
    - Base : contient tous les éléments nécessaires au bon fonctionnement de la plate-forme Calife seule.
    - Simulateur Elan : contient les fichiers de spécification permettant à Elan de construire un simulateur à partir de la description, depuis Calife, du système
    - Exemples : Contient des exemples de projets au format Calife (CSMA-CD, Train/Gate/Controlleur,…)
    - Sources Calife : Contient tous les fichiers et les makefiles permettant de recompiler Calife.
    - Elan : Contient les fichiers sources et les makefiles du logiciel Elan développé par l'équipe Prothéo du LORIA. L'installation d'Elan sera alors réalisée de manière automatique, sous réserve que les logiciels présentés au paragraphe 1.1.2 soient correctement installés.
- Le programme réalise alors l'installation des packs sélectionnés. Si vous avez sélectionné l'installation du logiciel Elan, sachez que cette opération dure entre 15 et 30 minutes suivant les machines (vous pouvez donc prendre un café en toute sérénité).
- Le programme d'installation propose alors la création de raccourcis (Windows uniquement) vers l'environnement Calife : Cliquez sur le bouton « Créer les raccourcis » pour passer à l'étape suivante.
- Cliquez sur « Quitter » pour quitter le programme d'installation.

1.3 Logiciels complémentaires

Les logiciels qui suivent ne sont pas indispensables au bon fonctionnement de la plate-forme mais permettent la validation formelle (par model-checking ou par preuve interactive) des systèmes décrits sous l'environnement Calife. De plus, tous les logiciels proposés ci-dessous sont interfacés par la plate-forme.

Logiciels recommandés :

- Hytech (http://www-cad.eecs.berkeley.edu/~tah/hytech/)
- Kronos (http://www-verimag.imag.fr/TEMPORISE/kronos/)
- Coq (http://coq.inria.fr/coq-fra.html)
- CMC (http://www.lsv.ens-cachan.fr/~fl/cmcweb.html)

Ces logiciels sont tous gratuitement téléchargeables. Les logiciels Hytech et CMC ne fournissent pas de version Windows, mais peuvent facilement être compilés sous l'environnement Cygwin (http://www.cygwin.com/).

2 PREMIERE PRISE EN MAIN

Le présent chapitre propose une vue d'ensembles des possibilités de la plate-forme Calife. Cette présentation sera réalisée à partir de la modélisation du protocole CSMA-CD utilisé dans l'accès à la couche réseau Ethernet.

2.1 Lancement de l'application

L'environnement Calife est lancé au moyen des scripts de raccourcis Calife.bat (environnement Win32) ou Calife.sh (environnement Unix).

L'application peut également être lancée au moyen de la commande *java –jar Calife.jar* depuis le répertoire d'installation.

2.2 Vue d'ensemble de l'application

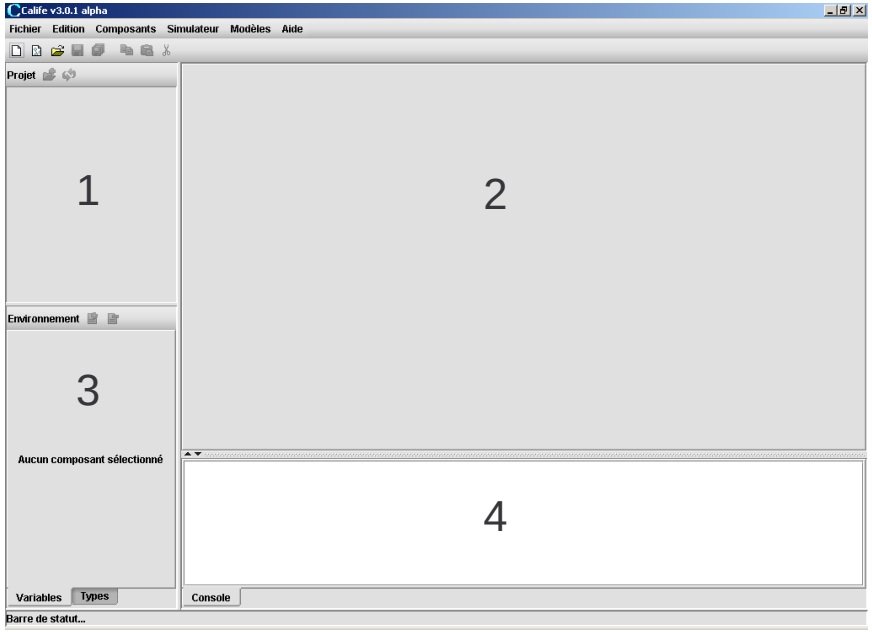


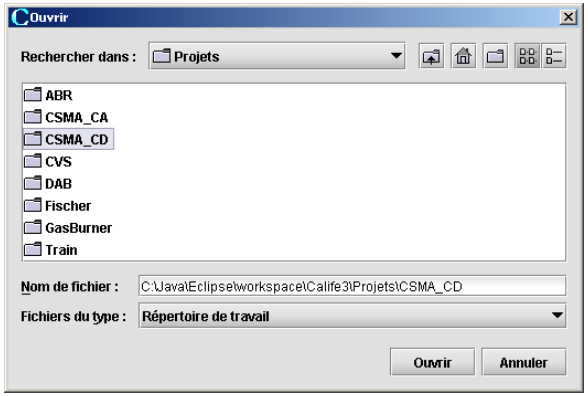
Figure 1 : Vue d'ensemble de l'éditeur Calife

L'éditeur d'automates Calife v3 est composé de 4 zones.

- La zone 1 permet de visualiser l'ensemble des fichiers automates (.clf) composants le projet.
- La zone 2 permet d'éditer un composant sélectionné depuis la zone 1, à partir d'un double-clique.
- La zone 3 permet d'éditer l'environnement (variables et paramètres) du composant sélectionné dans la zone 2.
- La zone 4 est une console affichant divers messages comme les erreurs de déclaration de variables, de typage sur les modèles, …

2.3 Ouverture du projet CSMA-CD

- Dans le menu « Fichier », sélectionnez « Ouvrir Projet ».
- Une boite de dialogue, positionnée dans le répertoire de travail (configuré depuis le fichier .calife) permet de sélectionner un projet existant



- Cliquez sur le répertoire projet « CSMA_CD » puis sur le bouton « Ouvrir »

Les fichiers composant le projet CSMA-CD sont alors présentés dans la partie (1) de l'éditeur :

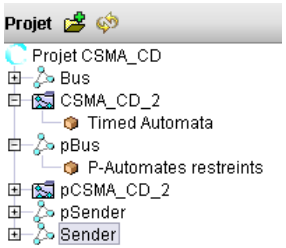


Figure 2 : L'arborescence "Projet"

Le projet CSMA-CD est donc composé de 6 fichiers « .clf » présents dans le répertoire

<Répertoire Projet>/CSMA_CD.

Le projet est constitué de 2 types de composants :

- Les automates simples, symbolisés par l'icône
- Les automates composés par produit synchronisé, symbolisés par l'icône

Les modèles d'automate (cf. paragraphe 5) utilisés par les composants sont visualisables en « dépliant » l'arborescence.

2.4 Visualisation et édition d'un composant

Pour visualiser et éditer un composant existant, il suffit de « double-cliquer » sur son nom, à partir de l'arborescence « Projet ».

Cette action permet d'ajouter un onglet contenant l'éditeur approprié, dans la zone (2) de l'application.

2.4.1 Visualisation et édition d'un automate simple

- Double-cliquer sur le composant « Sender » du projet CSMA_CD permet d'ouvrir un grapheur permettant la visualisation et l'édition de l'automate « Sender » :

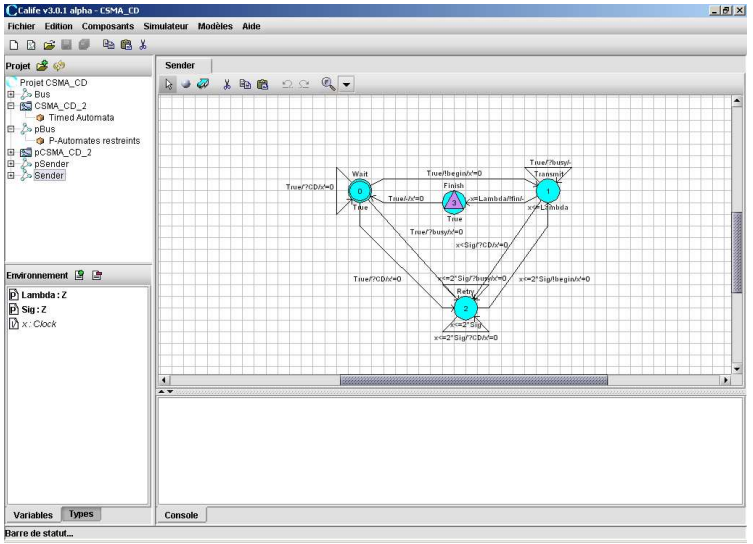


Figure 3 : Edition d'un automate simple

La zone « Environnement » est alors complétée pour faire apparaître l'environnement du composant Sender. Celui-ci est composé :

- De deux paramètres Lamba et Sig (de type Z), en gras, ce qui signifie que ces paramètres sont externes ; c'est à dire que leur valeur sera déterminée par les composants utilisant cet automate.
- D'une variable locale (en italique) x, de type clock.

L'automate Sender est décrit par l'intermédiaire d'un graphe composé de places (comprenant un invariant) et de transitions (composées d'une garde, d'une action et d'une relation de mise à jour).

Les informations associées aux places et aux transitions peuvent être modifiées en sélectionnant (par un clique gauche) la place ou la transition et en réalisant un clic droit.

Une fenêtre permettant l'édition de ces informations s'ouvre alors :

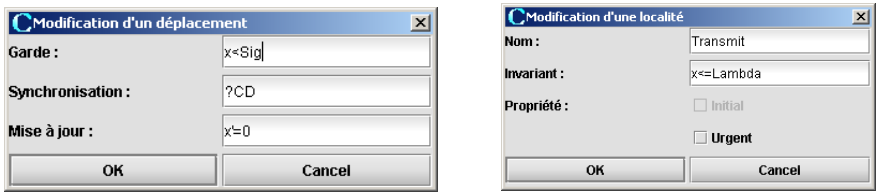


Figure 4 : Edition d'un déplacement ou d'une localité

2.4.2 Visualisation et édition d'un produit synchronisé d'automates

- Double-cliquez sur le composant CSMA_CD_2 pour faire apparaître l'éditeur associé aux produits synchronisés d'automates :

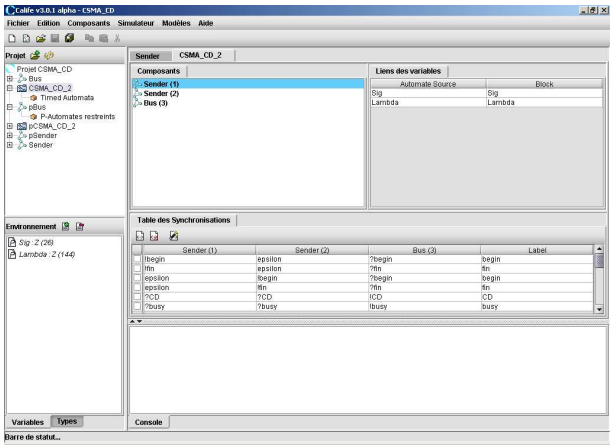


Figure 5 : Edition d'un produit synchronisé

L'environnement du composant CSMA_CD_2 est ici composé (cf zone 3) de 2 paramètres locaux Sig et Lamba, instanciés aux valeurs 26 et 144.

L'éditeur spécifique aux produits synchronisés est composé de 3 zones :

- La zone « Composants » contient l'ensemble des composants constituant le produit synchronisé. Ces composants sont ajoutés par une action de type « glisser-déplacer » depuis l'arborescence « Projet ».
- La zone « Liens des variables » propose, pour chaque composant sélectionné depuis la liste « Composants » les relations entre les variables externes du composant (Sigma et Lamba) et les variables du produit synchronisé (locales ou externes).
- La zone « Table de Synchronisation » permet de visualiser les vecteurs autorisant l'évolution synchronisée des automates composants le produit (cf 4.3.3.3).

2.5 Exports vers les outils de preuve

Les règles permettant l'export vers un outil de preuve sont décrites dans les fichiers modèles, présentés en détail au paragraphe 5.

Le projet CSMA_CD est composé d'automates temporisés et de p-automates. Les outils associés à ces modèles sont bien évidemment différents.

Pour visualiser les exports possibles pour un composant donné, il suffit de cliquer dessus à l'aide du bouton droit, depuis l'arborescence « Projet » et de sélectionner le sous-menu « Export ».

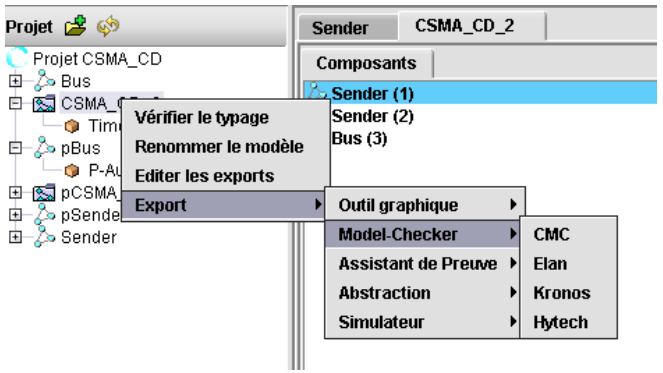


Figure 6 : Export d'un composant

On visualise alors, regroupés par types, les outils vers lesquels le composant sélectionné peut-être exporté.

2.5.1 Export vers Hytech

- Cliquez avec le bouton droit sur le composant CSMA_CD_2.
- Depuis le sous-menu « Export », choisissez la catégorie « Model-Checker », puis l'outils « Hytech ».
- L'environnement Calife réalise alors la construction et le typage du produit synchronisé. Les éventuelles erreurs de compilation sont visualisables dans la zone (4).
- Une nouvelle fenêtre, nommée « CalifeEdit » doit alors apparaître :

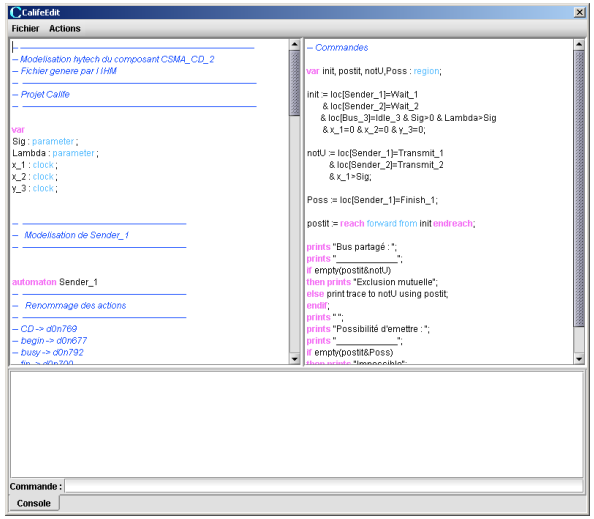


Figure 7 : L'interface "CalifeEdit"

Cette fenêtre est composée de 3 zones de texte :

- La zone à gauche contient la description, en langage Hytech, du système décrit sous l'éditeur. Cette description a été automatiquement générée par l'environnement Calife.
- La zone à droite contient un script de preuve écrit à la main. Ce script est sauvegardé dans le fichier CSMA_CD_2.prop, contenu dans le sous répertoire « Proprietes » du répertoire projet « CSMA_CD ».
- La zone en bas permet de visualiser les resultats fournis par Hytech après exécution.

Pour lancer l'exécution du script Hytech il suffit de sélectionner l'Item « Run Hytech » depuis le menu « Action ».

Remarque : l'enchaînement des actions permettant d'obtenir l'export et d'exécuter le logiciel Hytech est entièrement défini dans le fichier « modèle » associé au composant et sera décrit en détails au paragraphe 5.

2.5.2 Export vers Kronos

L'export vers Kronos peut-être réalisé de la même manière, en choisissant l'item :

« Export » à « Model-checker » à Kronos

La fenêtre contient maintenant :

- Le code Kronos généré automatiquement par l'environnement Calife
- Un fichier contenant les propriétés souhaitées
- Une menu « Action » composé des 3 items « Analyse Avant », « Analyse Arrière » et « Accessibilité »
- Une zone de texte permettant de visualiser le résultat de l'évaluation Kronos.

2.5.3 Simulation du composant CSMA_CD_2

Le composant CSMA_CD_2 peut être simulé depuis l'environnement Calife depuis l'item « Export » à « Simulateur » à « Compiler et Simuler ».

L'environnement Calife réalise alors la construction et la compilation du simulateur, puis ouvre l'interface de simulation suivante :

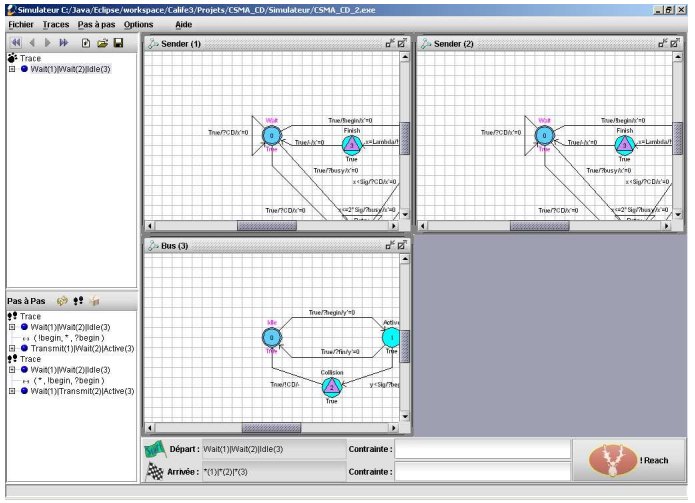
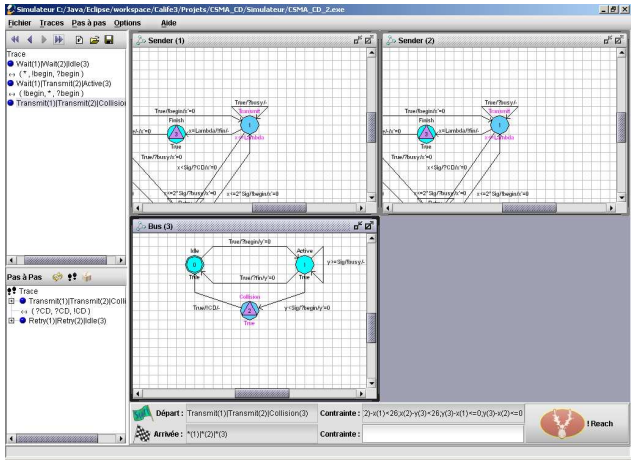


Figure 8 : Vue d'ensemble du simulateur

Cette interface propose :

- Dans la partie centrale, une visualisation des composants du produit synchronisé CSMA_CD_2. L'état au centre de chacun des grapheurs est l'état courant (ici, l'état initial).
- Dans la partie en haut, à gauche, la trace d'exécution construite. Au départ, celle-ci ne contient que l'état initial.

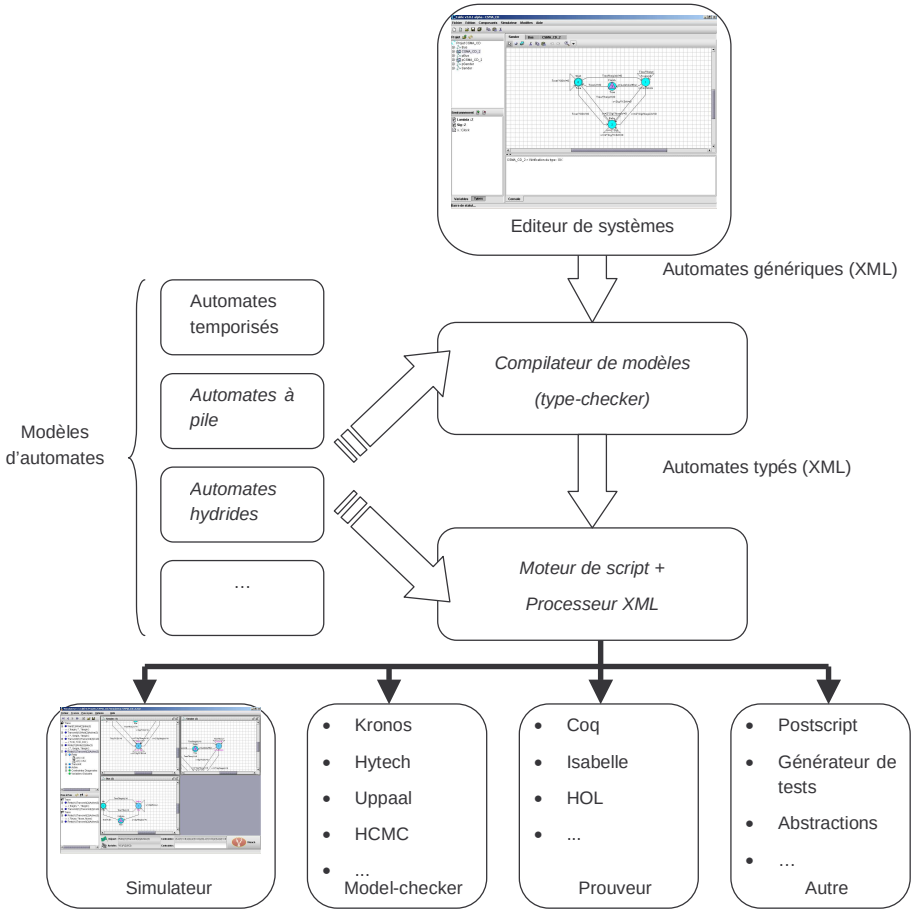
- Dans la partie en bas, à gauche, les états directement successeurs de l'état courrant. Il est possible de faire évoluer le système d'un pas en double cliquant sur une des traces présentes dans cette zone.
- La zone en bas, au centre permet de construire une requête d'accessibilité permettant de construire automatiquement la trace la plus courte amenant à l'état choisi. Pour ce faire, il suffit de cliquer (dans chacun des grapheurs associés aux composants) sur la localité souhaitée, de saisir (optionnellement) dans la zone « Contrainte » une contrainte sur les variables du système et de cliquer sur le bouton « !Reach ». Par exemple,
  - cliquez sur l'état « Collision » du composant « Bus ».
  - celui-ci est alors sélectionné par un carré rouge et la zone de texte « Arrivée » en bas à gauche devient alors : *(1) | *(2) | Collision(3).
  - Cliquez alors sur !Reach pour construire une trace d'exécution permettant d'atteindre l'état Collision pour le composant « Bus ».
  - La trace construite est alors visualisable dans la liste en haut à gauche. Elle est alors jouable à l'aide des boutons pour avancer d'un pas et pour reculer d'un pas.



  - L'état final de cette trace contient bien l'état Collision pour le bus et deux états quelconques (symbolisés par * dans la requête construite) pour les deux composants de type Sender.
  - Retournez à l'état initial de la trace, par l'intermédiaire du bouton
  - Sélectionnez de nouveau l'état « Collision » et dans la zone Contrainte, ajoutez la contrainte *x(1)>26*
  - Cliquez sur le bouton « !Reach »
  - Le simulateur doit alors renvoyer le message « Etat inatteignable ! ». Le simulateur a en effet réalisé une exploration exhaustive pour finalement conclure à un état inatteignable par model-checking.

3 VUE D'ENSEMBLE DE LA PLATE-FORME CALIFE

3.1 Schéma d'ensemble de la plate-forme Calife



La plate-forme Calife est constituée de 3 modules principaux :

- Un éditeur de système permettant de modéliser, de manière graphique, un système sous la forme de produits synchronisés d'automates. Le formalisme utilisé est celui des automates génériques présentés au paragraphe suivant.
- Un compilateur de modèle permettant de vérifier la bonne formation des contraintes (gardes, invariants, …) en regard du modèle sur lequel est construit le système (automates temporisés, hydrides,…). Le compilateur de modèle vérifie également la bonne déclaration des variables, des paramètres,…
- Un moteur de scripts permettant l'exécution de scripts associés à l'exportation des modèles vers les différents outils interfacés par la plate-forme.

La plate-forme Calife s'oriente donc vers 2 types d'utilisateurs :

- Le « super-utilisateur » définit des modèles d'automate et des scripts d'exports (Fichiers Modèles) dans le but d'interfacer de nouveaux outils.
- L' utilisateur définit des systèmes sous l'éditeur de systèmes et exporte ses systèmes sous les différents outils associés au modèle utilisé pour réaliser des preuves.

3.2 Automates génériques Calife

Les automates spécifiés sous l'éditeur de système suivent un formalisme très générique, inspiré des p-automates généralisés définis dans le cadre du projet Calife. Un automate générique peut se résumer à un p-automate sans la contrainte liée à l'utilisation d'une unique horloge universelle.

La définition ci-dessous n'est absolument pas formelle. En particulier, la sémantique des automates génériques est laissée libre au « super-utilisateur » à quelques contraintes près. Ce modèle d'automate définit surtout une syntaxe de spécification et un modèle de composition permettant de construire un système sous la forme d'un produit synchronisé.

3.2.1 Contexte des automates génériques

3.2.1.1 Contexte des types

Les automates génériques Calife sont définis dans un contexte de types quelconques.

Les types peuvent être définis comme :

- des types de variables (Types appartenant à l'ensemble *Ti*)
- des types de paramètres (Types appartenant à l'ensemble *Pi*)
- des types de constantes (Types appartenant à l'ensemble *Ci*)
- des types intermédiaires utilisés dans les signatures de fonctions (définition implicite)

Un unique type noté « Prop » est systématiquement et implicitement défini. Ce type est associé au type des prédicats (gardes, invariants,…).

On définit également un ensemble de fonctions de trans-typages permettant de définir, de manière hiérarchique, des équivalences entre les types.

3.2.1.2 Contexte des paramètres

On définit un ensemble de noms de paramètres. Ces paramètres sont de types $\in$ *Pi*.

A un paramètre, peut-être associée de manière optionnelle ou non (suivant les modèles), une constante d'un type compatible.

Les paramètres définis pour un automate générique sont associés à une notion de portée (local ou externe). Cette information sera utilisée lors de la composition des automates et de la génération de code.

3.2.1.3 Contexte des fonctions

Le contexte d'un automate générique définit un ensemble de signatures de fonctions. La sémantique de ces fonctions est définie par le « super-utilisateur » (en regard des outils interfacés).

Les fonctions introduites peuvent être polymorphes (c'est à dire avoir plusieurs signatures).

Une fonction notée « AND » doit impérativement être définie. Sa signature est la suivante :

AND : Prop -> Prop -> Prop

Cette fonction consomme 2 arguments de type *Prop* pour produire un résultat de type *Prop*. La sémantique associée est bien évidemment celle du « et » logique. Cette fonction est utilisée pour construire un produit synchronisé d'automates génériques. Elle est pretty-printée « && » dans l'éditeur de systèmes.

Une fonction notée « OUT » est implicitement définie. Sa signature est la suivante :

Pour tout type $\in$ Vi, OUT : Vi -> Vi

Etant donné une variable x de type V, le terme OUT(x), de type V représente la même variable « en sortie de transition ». Dans l'éditeur de systèmes, cette fonction est pretty-printée « ' » (i.e. OUT(x)=0 ó x'=0).

3.2.2 Automates génériques Calife

Le contexte ci-dessus définit un système de types sur lequel sont construits l'ensemble des prédicats définis dans les automates génériques. Un prédicat est bien formé si il est typable en « Prop », aux vues du système de types.

Dans ce contexte, un automate générique est défini (de manière informelle) par un graphe orienté.

Les transitions sont associées à :

- Un prédicat nommé Garde (devant être vrai pour que la transition soit tirable),
- Un label utilisé pour synchroniser l'automate,
- Une relation de mise à jour exprimée sous la forme d'un prédicat utilisant la fonction *OUT*.

Les localités du graphe sont associées à :

- Un prédicat nommé Invariant (toujours vrai pour pouvoir atteindre ou rester dans l'état)
- Un prédicat nommé Activité (utilisant généralement une fonction dont la sémantique est celle d'une dérivation par rapport au temps).

L'automate générique peut évoluer de deux manières différentes :

- Soit par écoulement du temps dans une localité. L'écoulement du temps est généralement défini par le prédicat Activité (ou par la sémantique de certains types) et restreint par le prédicat Invariant.
- Soit par tirage d'une transition discrète (Transition d'Action).

3.3 Composition d'automates génériques

Le produit synchronisé d'automates génériques (nommé *Block* dans l'éditeur de systèmes) est déduit de la donnée :

- des composants à synchroniser, indexés par des entiers,
- d'une relation entre les variables et les paramètres externes au composant et les variables internes (ou externes) du block,
- d'une table de synchronisation, composée de vecteurs de synchronisation et d'une fonction de nommage de ces vecteurs.

Un block contient ses propres variables et ses propres paramètres (internes ou externes). Les relations entre les variables (resp. paramètres) externes d'un composant et celles du block permettent d'effectuer un renommage des variables du composant à synchroniser.

Un vecteur de synchronisation est un vecteur de taille n (où n est le nombre de composants à synchroniser) où la $k^{ième}$ composante contient :

- soit un label introduit dans les transitions du composant indexé par k
- soit le label « epsilon » (associé à une transition virtuelle, sans effet pour le composant k)

La fonction de nommage des vecteurs de synchronisation permet de définir l'ensemble des labels de l'automate construit par produit synchronisé.

D'un point de vue sémantique, une transition d'action de l'automate synchronisé est tirable, si au niveau de chaque automate, une transition d'action (de label A_i) est tirable et que le vecteur $(A_1,\ldots,A_n)$ existe dans la table de synchronisation (en considérant que la transition « epsilon » est toujours tirable).

4 EDITION DE PROJETS

4.1 Construction d'un projet

Un projet, au sein de l'environnement Calife, est constitué d'un sous-répertoire du répertoire ${WorkDir}. Ce dernier, racine de l'ensemble des projets Calife, est défini au moment de l'installation du logiciel. Il peut être modifié à tout moment, en éditant le fichier « .calife » situé dans le répertoire « HOME » de l'utilisateur.

Un répertoire « Projet » contiendra l'ensemble des fichiers « .clf » définissant les différents composants du projet. Par la suite, ce répertoire contiendra également tous les fichiers créés lors de l'exportation d'un composant.

Remarque : le répertoire ${WorkDir} contient impérativement le fichier « Component.dtd ». Si ce fichier est absent, il est automatiquement recréé au démarrage du logiciel.

4.1.1 Création d'un nouveau projet

La création d'un nouveau projet est réalisée par l'intermédiaire du menu ***« Fichier à Nouveau Projet… »*** ou par l'icône .

Une fenêtre de dialogue apparaît pour saisir le nom du nouveau projet. Cette fenêtre indique également le répertoire associé au projet créé :

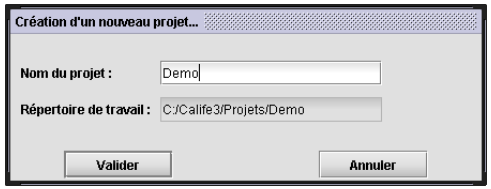


Figure 9 : Création d'un nouveau projet

Lorsque le nom du répertoire saisi est valide, le répertoire Projet est créé et automatiquement ouvert par l'éditeur de systèmes.

4.1.2 Ouverture d'un projet existant

L'ouverture d'un projet existant est réalisée par l'intermédiaire du menu ***« Fichier à Ouvrir Projet… »*** ou par l'icône .

Une fenêtre de dialogue apparaît alors pour saisir le nom du répertoire Projet :

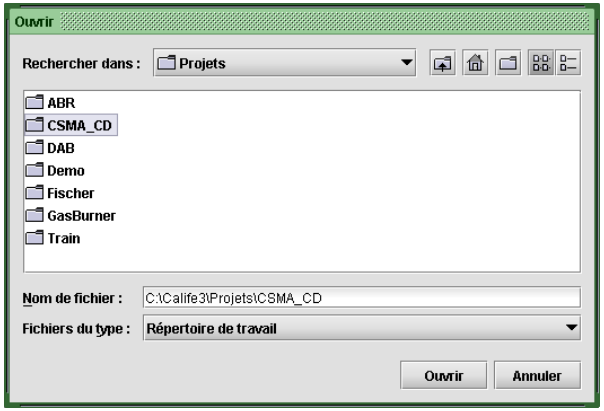


Figure 10 : Ouverture d'un projet existant

La sélection du projet à ouvrir est réalisée en cliquant sur le nom du projet depuis la liste proposée puis sur le bouton ***« Ouvrir »***.

4.1.3 Panneau de visualisation du contenu d'un projet

Ce panneau est situé en haut, à gauche de l'éditeur :

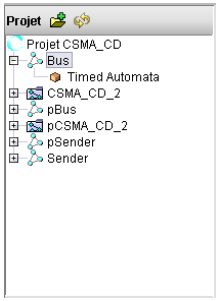


Figure 11 : Visualisation d'un projet

Il permet de visualiser l'ensemble des composants du projet (associés à des fichiers .clf situés dans le répertoire projet).

Le panneau projet permet :

- De sélectionner à l'aide d'un double-click sur son nom, un composant à éditer.
- D'insérer, à l'aide d'une action de type glisser-déplacer, un composant dans un block
- De recopier un composant existant d'un autre projet vers le projet actif (bouton)

- De rafraîchir le contenu de l'arborescence Projet (bouton) pour vérifier si de nouveaux composants ont été insérés dans le répertoire projet.
- De vérifier le typage d'un composant à la vue du système de type défini dans le modèle. Cette action est réalisée à l'aide du menu contextuel apparaissant en cliquant à l'aide du bouton droit sur le nom d'un composant.
- D'exporter un composant. Cette action est réalisée à l'aide du menu contextuel apparaissant en cliquant à l'aide du bouton droit sur le nom d'un composant.

4.2 Création d'un composant

La création d'un nouveau composant est réalisée par l'intermédiaire du menu ***« Composant à Nouveau Composant … »*** ou par l'icône .

Une fenêtre de dialogue apparaît alors pour saisir :

- Le nom du nouveau composant. Ce nom détermine le nom du fichier « .clf ». Il n'est donc pas possible, au sein du même projet, de créer deux composants ayant le même nom.
- Le modèle sur lequel ce composant est construit. L'éditeur propose tous les modèles valides contenus dans le répertoire ***Modeles*** du répertoire d'installation de Calife v3.
- Le type de composant (Automate ou Block)

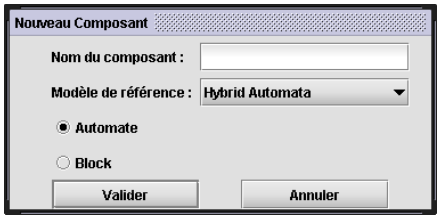


Figure 12 : Création d'un nouveau composant

Le menu « Composants » permet de réaliser des actions spécifiques à la gestion des composants. En particulier, ce menu propose les actions suivantes :

- Création d'un composant,
- Importation d'un composant existant dans le projet courrant,
- Suppression d'un composant,
- Renommage d'un composant,
- Vérification du typage d'un composant,
- Changement du modèle de référence d'un composant
- Edition, lorsque le modèle le permet, de certains éléments du système de type.

4.3 Edition d'un composant

D'un point de vue général, l'édition d'un composant (qu'il s'agisse d'un automate ou d'un block) est réalisée en ouvrant l'éditeur associé à celui-ci.

Cette action se fait par l'intermédiaire d'un double-click sur le nom du composant dans l'arborescence « Projet ». Un onglet dont le nom est celui du composant est alors ajouté dans la partie centrale de l'éditeur. Si cet onglet était déjà présent dans la partie centrale, il est alors sélectionné.

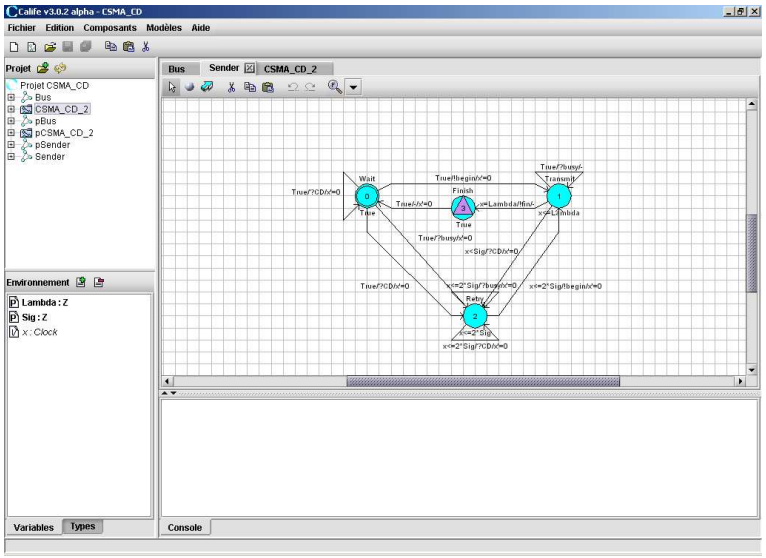


Figure 13 : Edition d'un composant

Remarque : les onglets associés aux éditeurs peuvent être fermés par l'intermédiaire de la croix apparaissant au passage de la souris à côté du nom du composant sur l'onglet.

4.3.1 Edition de l'environnement

Le panneau *Environnement* situé en bas à gauche, est toujours associé au composant actif dans la partie éditeur à onglets (partie centrale).

Ce panneau permet de visualiser d'une part l'ensemble des variables et des paramètres définis pour le composant (onglet ***Variables***), d'autre part le système de types associé (onglet ***Types***).

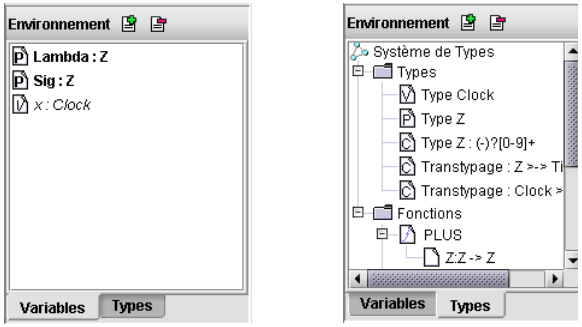


Figure 14 : Visualisation de l'environnement

L'onglet *Variables* permet la création, la modification ou la suppression de variables et de paramètres.

4.3.1.1 *Création d'une variable (resp. paramètre)*

La création d'une nouvelle variable est réalisée par l'intermédiaire de l'icône située en haut du panneau *Environnement*.

Une fenêtre de dialogue, permettant la saisie du nom, du type et des attributs de la variable (resp. paramètre) s'ouvre alors :

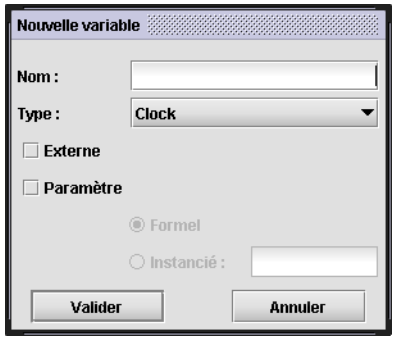


Figure 15 : Création d'une nouvelle variable

Dans cette fenêtre, la liste déroulante associée au type est différente pour une variable ou un paramètre (case *Paramètre* cochée).

La case à cocher *Externe* permet de spécifier la portée du paramètre ou de la variable (Local/Externe).

Lorsque la case *Paramètre* est cochée et que le paramètre est local (case *Externe* non cochée), il est possible d'associer une valeur au paramètre en sélectionnant *Instancié* et en saisissant la valeur dans la zone de saisie.

4.3.1.2 Modification d'une variable existante

La modification d'une variable existante est réalisée en double-cliquant sur le nom de la variable depuis la liste des variables.

Une fenêtre de dialogue, identique à la précédente est alors ouverte.

4.3.1.3 Suppression d'une variable existante

La suppression d'une variable (resp. un paramètre) existante est réalisée en sélectionnant la variable dans la liste et en cliquant sur l'icône .

4.3.2 Construction d'un automate

La construction d'un automate est réalisée de manière graphique, par l'intermédiaire d'un grapheur associé au composant :

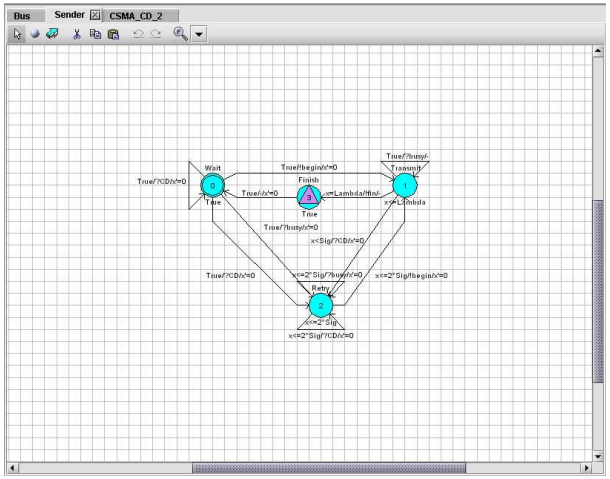


Figure 16 : Grapheur associé au composant Sender

Le grapheur propose les fonctionnalités suivantes :

- Fonctions de Sélection. Ces fonctions sont actives en se positionnant en mode *Sélection*, par l'intermédiaire de l'icône de la barre d'outils. Les sélections peuvent alors être réalisées :
  - En cliquant sur un élément du graphe (Localité, Texte, Nœud intermédiaire d'une transition)
  - En dessinant un rectangle à l'aide de la souris, par une action de type glisser-déplacer.
  - En double-cliquant sur un des éléments d'une transition (Texte ou Nœud intermédiaire). Dans ce cas, toute la transition est alors sélectionnée.

- Fonctions de Zoom : 3 modes de zoom sont disponibles par l'intermédiaire de la liste déroulante à droite sur la barre d'outils. Le type de zoom sélectionné est réalisé en cliquant sur le bouton sélectionné dans la liste déroulant. Les 3 fonctionnalités de zoom sont :
  - Augmenter le zoom par l'icône
  - Diminuer le zoom par l'icône
  - Choisir l'échelle la plus appropriée par l'icône
- Fonctions d'éditions avancées : les fonctionnalités sont les suivantes :
  - Copier le contenu d'une sélection par l'icône ou *CRTL+C*.
  - Couper la sélection par l'icône ou *CRTL+X* (si une sélection contient une transition partiellement sélectionnée, celle-ci sera alors supprimée).
  - Coller le contenu du presse-papier par l'icône ou *CRTL+V*. Cette opération n'est possible que si une sélection a été préalablement copiée ou coupée.
  - Défaire des modifications par l'icône
  - Refaire des modifications défaites par l'icône

4.3.2.1 Création et modification d'une localité.

La création d'une localité est réalisée en se plaçant au préalable dans le mode « Insertion de Localité », par l'intermédiaire de l'icône .

Il suffit alors de cliquer sur une zone libre du graphe pour faire apparaître la fenêtre de dialogue suivante :

Figure 17 : Création d'une localité

Il est alors possible de :

- spécifier un nom pour la localité. Un nom du type e<numéro> sera défini par défaut .
- saisir un prédicat associé à l'invariant. La grammaire du prédicat dépend du système de type défini dans le modèle. Néanmoins, certains opérateurs sont pretty-printés au niveau du parseur et automatiquement traduits sous la forme de fonctions associées (définies ou non…). Cette remarque concerne les fonctions suivantes (définies dans tous les modèles fournis avec la plate-forme) :
  - Fonctions « AND » pretty-printé « && »
  - Fonction « OR » pretty-printé « || »
  - Fonctions de comparaison « LT », « LTE », « GT », « GTE », « EQ », « NEQ » respectivement pretty-printés « < », « <= », « > », « >= », « = », « != ».

- Fonctions usuelles « PLUS », « MOINS », « MULT » pretty-printées « + », « -», « * »

- saisir un prédicat associé à l'activité dans la localité (optionnel). Dans ce cas, le prédicat associé à l'invariant sera suivi du symbole « ; » puis du prédicat associé à l'activité. Par exemple, l'automate « gas burner » (construit sur le modèle des automates hybrides, fournit avec la plate-forme) utilise ce genre de constructions :

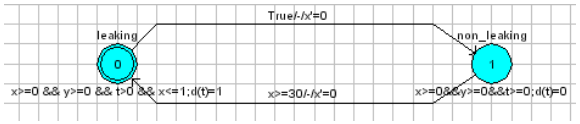


Figure 18 : Exemple de localités (avec prédicat Activité)

- saisir certaines propriétés liées au nœud (état initial, état urgent).

Remarque : il n'est possible de sélectionner qu'un état initial. Cette sélection est donc impossible lorsque un état initial est déjà défini dans l'automate.

La localité ainsi créée peut alors être déplacée sur le graphe en la sélectionnant puis en réalisant une action de type glisser-déplacer.

La modification des informations associées à une localité existante peut-être réalisée en sélectionnant la localité (par un simple ou double-click) et en cliquant à l'aide du bouton droit.

Une fenêtre de dialogue, identique à celle décrite ci-dessus s'ouvre alors.

4.3.2.2 Création et modification d'un déplacement (transition d'action).

La création d'un déplacement est réalisée en passant au préalable dans le mode « Insertion de déplacements », par l'intermédiaire de l'icône  .

Il suffit alors de cliquer sur la localité de départ du déplacement. En déplaçant la souris, il est possible de visualiser l'arc en cours de création. On peut alors insérer des nœuds intermédiaires pour fragmenter l'arc. Le fait de cliquer sur une localité d'arrivée entraîne l'ouverture de la fenêtre de dialogue suivante :

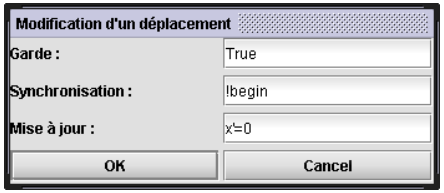


Figure 19 : Création d'un déplacement

Il est alors possible de saisir :

- Le prédicat associé à la garde
- Un label utilisé par la suite pour synchroniser le composant (optionnel)
- Une relation de mise à jour sous la forme de prédicats séparés par le symbole « ; »

Remarque : les labels de synchronisation peuvent commencer par le symbole « ! » ou « ? ». Cette information n'a pas de sémantique précise, mais sera utilisée par certains algorithmes de création automatique de la table de synchronisation (cf. paragraphe 4.3.3.3).

La modification des informations associées à un déplacement existant peut-être réalisée en sélectionnant le déplacement complet (par double-click) et en cliquant à l'aide du bouton droit.

Une boite de dialogue, identique à celle décrite ci-dessus s'ouvre alors.

4.3.3 Construction d'un block

La construction d'un block (produit synchronisé) est réalisée par l'intermédiaire d'un panneau de saisie des informations nécessaires (composants, liens et table de synchronisation) :

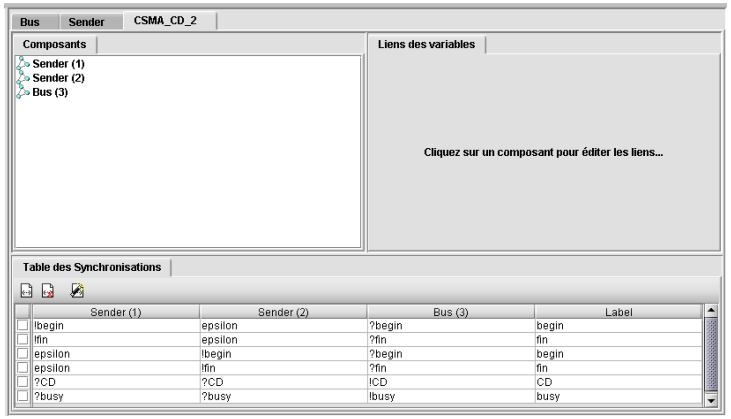


Figure 20 : Panneau de construction d'un block

4.3.3.1 Spécification des composants

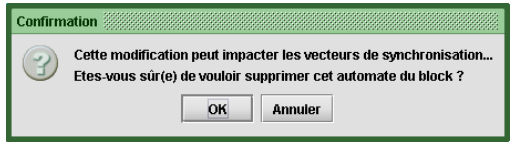
Les composants constituant le block sont visibles dans la liste présente sous l'onglet « Composants ».

Il est possible d'ajouter des composants dans cette liste en réalisant une action de type glisser-déplacer, depuis un composant de l'arborescence *Projet* vers cette liste.

Un index numérique est alors automatiquement attribué au composant et une colonne est ajoutée à la table de synchronisation.

Si celle-ci contient déjà des vecteurs de synchronisation, les composantes ajoutées sont positionnées sur le label epsilon.

Il est possible de supprimer un composant de la liste en le sélectionnant et en appuyant sur la touche *Suppr.* du clavier. Un message de confirmation apparaît alors :

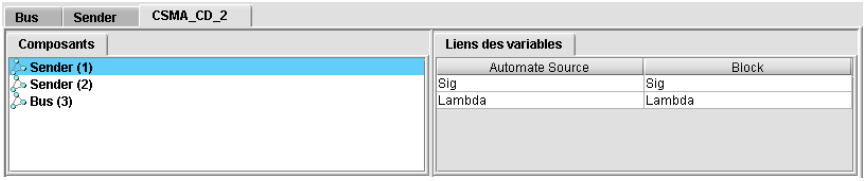


Cliquez sur OK pour continuer ou Annulez.

Remarque : la colonne de la table de synchronisation associée au composant supprimé sera également supprimée en cas de confirmation.

4.3.3.2 Edition des liens

Les liens spécifiques à un composant apparaissent lorsque celui-ci est sélectionné depuis la liste « Composants » :



Le panneau « Liens des variables » permet de visualiser à gauche les variables « externes » du composant sélectionné et à droite la variable du block liée à la variable de l'automate.

La variable du block peut être modifiée en cliquant sur la cellule. Une liste déroulante propose alors toutes les variables du block. Il est également possible de choisir « -- Valeur Constante --» dans cette liste. Une boite de dialogue permet alors de saisie la valeur choisie.

Remarque : A l'insertion d'un nouveau composant dans le block, l'éditeur réalise alors un lien automatique entre les variables et les paramètres portant le même nom.

4.3.3.3 Edition de la table de synchronisation

L'éditeur associé à la table de synchronisation se présente sous la forme d'un tableau. Chaque colonne est associée à un composant, à l'exception de la dernière colonne utilisée pour saisir le nom du vecteur de synchronisation (cf. fonction de nommage au paragraphe 3.3).

Chaque ligne de ce tableau peut donc être lue comme un vecteur de synchronisation présenté au paragraphe 3.3.

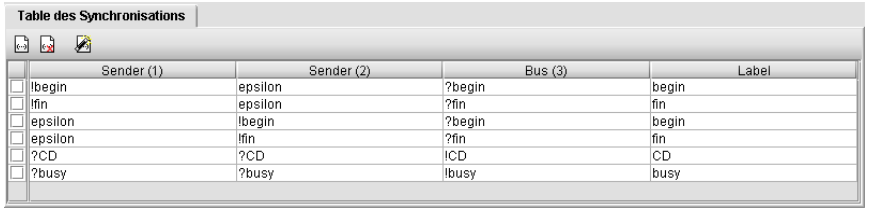


Figure 21 : Edition de la table de synchronisation

La barre d'outils située en haut du panneau permet :

- d'insérer un nouveau vecteur de synchronisation à l'aide de l'icône
- de supprimer les vecteurs sélectionnés à l'aide de la case à cocher (colonne la plus à gauche), à l'aide de l'icône
- d'appeler l'assistant de création automatique de tables de synchronisation à l'aide de l'icône

Lorsqu'un nouveau vecteur de synchronisation est créé, celui-ci est initialisé avec toutes ses composantes à la valeur « epsilon » et « Action » comme nom.

Cliquer sur une composante du vecteur permet de faire apparaître une liste déroulante contenant l'ensemble des labels introduits dans les transitions de l'automate. Il est alors possible de sélectionner, pour chaque composant, les labels synchronisés entre eux.

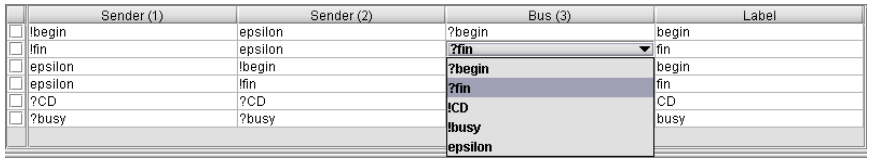


Figure 22 : Edition d'une composante du vecteur

Le label de la synchronisation peut être modifié en saisissant dans la case le nouveau label du vecteur.

4.3.3.4 Tables de synchronisation automatiques

L'assistant de création de table de synchronisation apparaît par l'intermédiaire de l'icône :

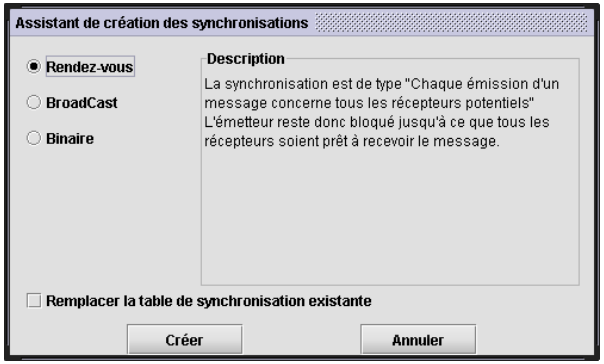


Figure 23 : Assistant de création des synchronisations

L'assistant propose alors un certain nombre d'algorithmes de création de tables de synchronisation :

- La synchronisation par rendez-vous : le principe consiste à synchroniser chaque label commençant par le symbole « ! » puis une chaîne de caractères avec tous les labels des autres composants contenant un label de la forme « ? » puis la même chaîne de caractères.

- La synchronisation par broadcast, déduite de la synchronisation par rendez-vous : par opposition à la synchronisation par rendez-vous où tous les composants contenant le label ?Action sont concernés, la synchronisation par broadcast va créer tous les sous-vecteurs concernant de 0 à n composants contenant le label ?Action. Ce type de synchronisation permet de modéliser l'envoi de messages non-bloquants pour l'émetteur.
- La synchronisation binaire est une synchronisation ne concernant que 2 composants. On synchronise donc chaque label de type !Action de chaque composant avec chaque label de type ?Action des autres composants.

Exemple de synchronisation sur CSMA_CD :

CSMA_CD est construit à partir de 2 types d'automates :

- L'automate Sender dont les actions sont les suivantes : !begin, !fin, ?CD et ?busy
- L'automate Bus dont les actions sont les suivantes : ?begin, ?fin, !CD et !busy

Le block « Système est construit de 2 automates Sender (indexés 1 et 2) et d'un automate Bus (Il est possible de généraliser ce système à n Sender et un unique Bus).

L'algorithme de synchronisation par rendez-vous crée alors la table suivante :

	Sender (1)	Sender (2)	Bus (3)	Label
☐	!begin	epsilon	?begin	begin
☐	!fin	epsilon	?fin	fin
☐	epsilon	!begin	?begin	begin
☐	epsilon	!fin	?fin	fin
☐	?CD	?CD	!CD	CD
☐	?busy	?busy	!busy	busy

On remarque que la synchronisation entre !begin et ?begin engendre 2 vecteurs (n dans le cas généralisé), alors que la synchronisation entre !CD et ?CD n'engendre qu'un vecteur (comme dans le cas généralisé).

L'algorithme de synchronisation par broadcast crée la table suivante :

	Sender (1)	Sender (2)	Bus (3)	Label
☐	?busy	?busy	!busy	busy
☐	!fin	epsilon	epsilon	fin
☐	epsilon	!begin	epsilon	begin
☐	epsilon	!fin	epsilon	fin
☐	!fin	epsilon	?fin	fin
☐	epsilon	!begin	?begin	begin
☐	?CD	epsilon	!CD	CD
☐	epsilon	epsilon	!busy	busy
☐	?CD	?CD	!CD	CD
☐	?busy	epsilon	!busy	busy
☐	epsilon	?CD	!CD	CD
☐	epsilon	?busy	!busy	busy
☐	!begin	epsilon	?begin	begin
☐	epsilon	epsilon	!CD	CD
☐	!begin	epsilon	epsilon	begin
☐	epsilon	!fin	?fin	fin

La synchronisation entre !CD et ?CD a engendré 4 vecteurs :

- 1 vecteur concerne tous les composants (i.e. ?CD, ?CD, !CD) ce vecteur est identique à celui déduit par l'algorithme par Rendez-Vous

- 1 vecteur ne concerne aucun composant (i.e. epsilon,epsilon, !CD) il correspond au cas ou aucun composant ne reçoit le message
- 2 vecteurs concernent uniquement 1 des deux composants

5 MODELES D'AUTOMATES

L'objectif des automates génériques présentés au paragraphe 3.2.2 est de laisser la possibilité d'inclure, dans de nouveaux modèles, des extensions au modèle des automates temporisés « Alur et Dill ». On parle alors d'automates hydrides. 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 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).

5.1 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.

5.2 Définition du modèle XML

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

5.2.1 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.

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

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

Figure 24 : 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.

5.2.2 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 25 : 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 26 : Déclaration des types pour les automates temporisés

5.2.3 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
```


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>
```

5.3 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.

5.3.1 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 nommée 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" />
```

5.3.2 Actions élémentaires

Les nœuds XML suivants sont associés à des actions élémentaires composant 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 courant.
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 réalisées 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
CalifeEdit	Left (Opt) Right(Opt) HighLight(Opt)	Crée un éditeur CalifeEdit (cf. 0) avec le contenu du fichier <Left> en partie gauche et du fichier <Right> en partie droite. Les éditeurs de texte utilisent la coloration syntaxique définie dans le fichier xml <HighLight> (cf. 5.3.4).

5.3.3 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 créés. 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.

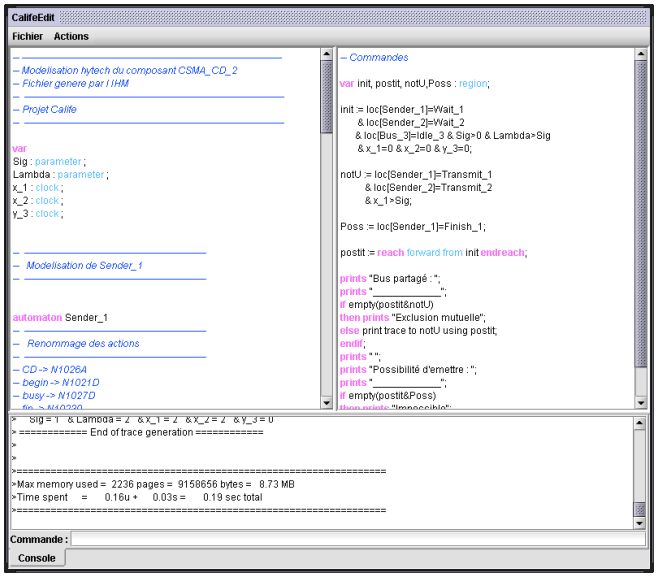


Figure 27 : Vue d'ensemble de l'interface CalifeEdit

5.3.4 Coloration syntaxique

La coloration syntaxique utilisée par les éditeurs de texte du composant CalifeEdit est définie par l'intermédiaire de fichiers XML externes.

Celle-ci peut donc être personnalisée, en fonction du code et des outils utilisés.

Un fichier définissant la coloration syntaxique du composant CalifeEdit doit impérativement suivre la DTD suivante :

```
<!ELEMENT HighLighter (Styles?,KeyWords?,Comments?) >
<!ATTLIST HighLighter Label CDATA #REQUIRED>

        <!ELEMENT Styles (Style*) >
                <!ELEMENT Style EMPTY>
                <!ATTLIST Style Label CDATA #REQUIRED>
                <!ATTLIST Style Color CDATA #REQUIRED>
                <!ATTLIST Style Italic CDATA #REQUIRED>
                <!ATTLIST Style Bold CDATA #REQUIRED>
                <!ATTLIST Style Police CDATA #IMPLIED>

        <!ELEMENT KeyWords (KeyWord*) >
                <!ATTLIST KeyWords Separator CDATA #IMPLIED>
                <!ELEMENT KeyWord EMPTY>
                <!ATTLIST KeyWord Label CDATA #REQUIRED>
                <!ATTLIST KeyWord Style CDATA #REQUIRED>

        <!ELEMENT Comments (BlockComment|LineComment)* >
                <!ELEMENT BlockComment EMPTY>
                <!ATTLIST BlockComment Start CDATA #REQUIRED>
                <!ATTLIST BlockComment End CDATA #REQUIRED>
                <!ATTLIST BlockComment Style CDATA #REQUIRED>

                <!ELEMENT LineComment EMPTY>
                <!ATTLIST LineComment Start CDATA #REQUIRED>
                <!ATTLIST LineComment Style CDATA #REQUIRED>
```

Figure 28 : DTD d'un fichier de coloration syntaxique

Le fichier XML peut donc définir :

- des balises <Style> contenant les attributs suivants :
  - Label : définit le nom du style,
  - Color : définit la couleur du style, de la forme R:G:B (par exemple 245:115:245),
  - Italic : définit l'attribut « Italique » de la police de caractères (les valeurs possibles étant « True » ou « False »),
  - Bold : définit l'attribut « Gras » de la police (« True » | « False »),
  - Police (optionnel) : définit la police de caractère à utiliser. Les valeurs possibles sont des noms de polices Java.
- Des balises <KeyWord> contenant les attributs suivants :
  - Label : définit le mot clé à colorer,
  - Style : définit le style à appliquer au mot clé.
- Des balises <BlockComment> servant à définir des zones de commentaires (avec un style unique), contenant les attributs suivants :
  - Start : définit le symbole débutant une zone de commentaires,
  - End : définit le symbole finissant la zone de commentaires,
  - Style : définit le style à appliquer à la zone de commentaire.

- Des balises <LineComment> servant à définir des lignes de commentaire (équivalent à un nœud <BlockComment> avec pour attribut End un caractère de retour chariot), contenant les attributs suivants :
  - Start : définit le symbole débutant un commentaire de ligne,
  - Style : définit le style à appliquer au commentaire.

L'attribut *Separator* du nœud <KeyWords> permet de spécifier, de manière optionnelle, les caractères de séparation des mots (en plus de l'espace et du retour-chariot).

Exemple de fichier de coloration syntaxique :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<!DOCTYPE HighLighter SYSTEM "HighLighter.dtd">
<HighLighter Label="Elan">
        <Styles>
                <Style Label="Strong" Color="245:115:245" Italic="False" Bold="True"/>
                <Style Label="Light" Color="76:185:255" Italic="False" Bold="False"/>
                <Style Label="Asap" Color="245:115:245" Italic="True" Bold="False"/>
                <Style Label="Comment" Color="17:73:255" Italic="True" Bold="False"/>
        </Styles>
        <KeyWords Separator="-().;:">
                <KeyWord Label="specification" Style="Strong"/>
                <KeyWord Label="Environment" Style="Strong"/>
                <KeyWord Label="Synchronisation" Style="Strong"/>
                <KeyWord Label="end" Style="Strong"/>
                <KeyWord Label="widthFirst" Style="Strong"/>
                <KeyWord Label="ReachInOneStep" Style="Strong"/>
                <KeyWord Label="ReachableIn" Style="Strong"/>
                <KeyWord Label="stepsFrom" Style="Strong"/>
                <KeyWord Label="Clocks" Style="Strong"/>
                <KeyWord Label="Variables" Style="Strong"/>
                <KeyWord Label="States" Style="Light"/>
                <KeyWord Label="Labels" Style="Light"/>
                <KeyWord Label="Automata" Style="Strong"/>
                <KeyWord Label="Invariant" Style="Light"/>
                <KeyWord Label="Transitions" Style="Light"/>
                <KeyWord Label="ASAP" Style="Asap"/>
                <KeyWord Label="nil" Style="Light"/>
        </KeyWords>
        <Comments>
                <BlockComment Start="/*" End="*/" Style="Comment"/>
                <LineComment Start="//" Style="Comment"/>
        </Comments>
</HighLighter>
```

Figure 29 : Exemple de fichier de coloration syntaxique

5.3.5 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 preuve par model-checking, il est tout d'abord nécessaire de construire le produit synchronisé des différents automates composant 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 30 : 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 la 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.