



# STACS

Spécification et Test Abstraites et Compositionnels de Systèmes

## Eclipse Transition Systems

Référence	
Auteur responsable	Pascal Poizat (LaMI)
Partenaires	CEA, Thales Com., Ligeron SA
Date	22/11/2005 (révisé le 06/11/2006)
Type	PDF
Identifiant	
Version	0.2
Statut	Fourniture contractuelle
Groupe	Spécifications des extensions d'outils liées aux tests structurés
Diffusion	RNRT
Résumé	Sans restriction
Mots clefs	voir page suivante
	LTS, STS, Eclipse, plugin, produit synchronisé, vecteurs de synchronisation, glu, Agatha



# Eclipse Transition Systems\*

Pascal Poizat

LaMI UMR 8042 CNRS - Université d'Evry Val d'Essonne, Genopole  
Pascal.Poizat@lami.univ-evry.fr

**Résumé** La vérification de systèmes passe par la prise en compte de leurs aspects dynamiques. Pour cela, les modèles mis en œuvre sont typiquement des systèmes de transition étiquetés (LTS ou IOLTS). Les problèmes liés à l'explosion du nombre d'états de ces modèles peuvent trouver des solutions dans l'*abstraction* des systèmes de par l'utilisation d'extensions *symboliques* des LTS (STS, IOSTS, EIOLTS). Par ailleurs, la prise en compte de la structuration des systèmes en architectures de composants interagissants devrait permettre de vérifier les composants de façon *compositionnelle*, c'est-à-dire sans procéder à la mise à plat du système avant leur vérification. Le développement de ces techniques d'abstraction symbolique et de compositionnalité dans le cadre du test orienté modèle est l'objectif du projet RNRT STACS. Dans ce contexte, différents travaux basés sur différentes sémantiques de produits entre STS ont été développés au LaMI. Notre objectif est la définition d'un environnement (plateforme) dans lequel expérimenter l'implantation de ces travaux sémantiques. Pour cela, en nous basant sur les travaux antérieurs du projet CLAP, nous avons développé un *plugin* pour Eclipse permettant la prise en compte de différents modèles de type LTS ou STS ainsi qu'une forme de produit très général. Cet environnement est prévu pour être extensible lui-même et des mécanismes de transformation des STS au formats Agatha et DOT ont été implantés à des fins de démonstration.

## 1 Introduction

Il est maintenant reconnu que la vérification des systèmes logiciels passe par la prise en compte de leurs aspects dynamiques. Un exemple simple tiré de l'ingénierie logicielle basée composants (CBSE) est celui de deux composants interagissants compatibles au niveau de leur interface statique (signatures des services requis et fournis, décrits à l'aide d'un IDL dans le cadre de CCM ou des EJB par exemple). Ces deux composants peuvent cependant se révéler incompatibles d'un point de vue protocolaire, ce qui peut conduire à un blocage général du système [9]. La description des composants et des systèmes doit donc mettre en œuvre à la fois des descriptions statiques et des descriptions dynamiques.

La spécification de la dynamique des systèmes se fait à l'aide de langages de spécification tels que les algèbres de processus [13]. Au final ce sont en général les modèles de ces langages, les systèmes de transitions étiquetés (LTS) qui servent de support à la vérification des systèmes. L'une de leurs limitations,

---

\* Ce travail a fait l'objet d'un financement via le projet RNRT STACS (label 2002).

sinon la principale, est le fait qu'ils conduisent à un problème d'explosion du nombre d'états (et de transitions) des modèles en présence de données échangées entre les processus (value passing), de données encapsulées par les processus ou encore de nombre non borné de composants interagissants. Pour résoudre les deux premiers problèmes, la prise en compte des données de façon symbolique dans les LTS a conduit à la définition de modèles appelés systèmes ou graphes de transitions symboliques (STG ou STS) [14,8,11]. Ces systèmes se retrouvent aussi dans la littérature sous d'autres noms (*e.g.*, IOSTS, EIOLTS, STSA) en fonction par exemple de la prise en compte ou non d'une différenciation entre entrées et sorties.

Une autre façon de résoudre l'explosion du nombre d'états semble être de tirer parti du caractère structuré des systèmes en ensemble de composants interagissants. C'est l'objectif des techniques dites compositionnelles dans lesquelles la construction de la sémantique globale du système (c'est-à-dire la mise à plat d'une architecture de sous-systèmes ou composants en un modèle global) se fait par exemple à *la volée*, au fur et à mesure de la vérification du système.

Le développement de techniques de test orienté modèle, dans le cadre de systèmes décrit à l'aide de systèmes de transitions, est l'objet du projet RNRT STACS (partenaires : Thalès Communications, CEA, LaMI UMR 8042 CNRS, Ligeron S.A.). Dans ce cadre, des techniques reposant sur l'abstraction (symbolique sur les données, STS) et sur la compositionnalité sont développées [16,1,20,17]. En amont du développement d'une seconde version de l'outil Agatha [7], une plateforme permettant la définition de différents types de STS ainsi que l'implantation de différents algorithmes de produit de composants décrits à l'aide de STS, permettrait l'expérimentation plus rapide des développements formels. C'est dans ce cadre qu'ETS (Eclipse Transition Systems) a été conçu. Il s'inscrit dans le cadre de la contribution STACS intitulée " Spécifications des extensions d'outils liées aux tests structurés ".

## 2 ETS

ETS est basé sur des travaux antérieurs du projet CLAP [12] et de son extension dédiée à l'animation de diagrammes d'états UML étendus avec données formelles, xCLAP [4,3]. ETS est destiné à l'expérimentation d'algorithmes de produits mais aussi d'analyse sur les LTS et les STS. Pour cela, un modèle très général a été défini. Il permet la prise en compte de propriétés appelées *features* au niveau des états et des transitions. Ces features peuvent être simplement des noms d'états ou des étiquettes de transition, mais plus loin, sont extensibles. Il est ainsi par exemple possible d'associer aux états des valeurs de résultats de test (*Accept*, *Pass*, *Fail*) ou encore des probabilités aux transitions. Un algorithme de produit très général a été implanté. Cet algorithme étend le produit synchronisé [2] implanté dans l'outil MEC de plusieurs façons. Premièrement, il prend en compte les features et leur extensibilité. Il permet aussi la conservation du caractère structuré des spécifications au niveau du produit. En effet, le résultat d'un produit est un STS dont les états et les transitions sont structurés eux-

mêmes sous forme d'arbre (on peut alors parler de *tree-STS* ou *t-STS*). L'intérêt de la conservation de cette information est évident par exemple dans le cas d'un produit entre un système global (un *t-STS*) et d'un objectif de test (un *STS*). Le résultat étant lui-même un *t-STS*, il est possible de projeter un test global (trace du résultat, suite de transitions arbres) sur les différents composants du système (trace constituée d'états simples). Le produit d'ETS doit aussi plus facilement permettre le raisonnement compositionnel en ce que, le résultat d'un produit ayant la même structure qu'un système simple, il est possible de parler de *composant composite*. À notre connaissance, seuls les travaux récents d'E. Madelaine [5] prennent en compte aussi cet aspect. Le produit général d'ETS généralise aussi par ailleurs les produits développés au LaMI dans [1,20] en ce qu'il se base sur une définition explicite des recollements (d'états ou de transitions), ce qui est un héritage du produit synchronisé à base de vecteurs de synchronisation. C'est un point important (et qui a été validé) en ce qu'il s'agissait là de notre objectif premier.

## 2.1 Pourquoi Eclipse ?

Plus qu'un environnement de programmation Java (ce pour quoi il est souvent pris), Eclipse est une plateforme de développement, d'échange et de transformation de modèles. Pour cela il se base sur un métamodèle légèrement différent de celui d'UML (MOF) appelé Eclipse Modelling Framework (EMF). Eclipse est une plateforme extensible. Ceci se fait par la définition de composants additionnels appelés *plugins*. La philosophie principale d'Eclipse est que les plugins soient eux-mêmes, si possible, extensibles. Dans le cadre qui nous intéresse cela veut dire un plugin pour *STS* qui puisse facilement être étendu par un plugin tiers de façon à traduire ces *STS* dans un format externe et assurer leur vérification. Ces possibilités d'échange sont selon nous un point clé du développement et de l'utilisation d'outils formels. Dans le cadre d'un Travail d'Etude et de Recherche (TER) nous avons validé cette approche sur des *LTS*. Nous avons défini un plugin simple pour *LTS* puis dans un second plugin nous avons étendu le premier de façon à permettre la traduction des *LTS* en algèbre de processus (CCS) ainsi que leur vérification : une formule de logique temporelle est demandée, le *LTS* est traduit puis un outil externe - CWB - est appelé pour réaliser la vérification de modèle ; le résultat final de cette vérification est finalement présenté dans une fenêtre d'Eclipse.

Eclipse est la base de nombreux projets open-source y compris dans la communauté du génie logiciel. On peut par exemple citer les plugins ACME pour les modèles d'architecture logicielle (ADL) ou OSATE pour le langage AADL. De plus en plus de plugins formels voient aussi le jour, supportés par des bourses Eclipse.

## 2.2 Principes

**LTS, produits et vecteurs.** Pour présenter les principes d'ETS nous nous baserons (pour simplifier) sur les *LTS*. Les principes s'appliquent aux *STS* dont

différentes définitions de structure et de produit pourront être trouvées par exemple dans [15,20,17].

La description de la dynamique des systèmes se fait grâce à un langage (comme une algèbre de processus) dont la sémantique est donnée par exemple par traduction dans des structures appelées LTS.

**Definition 1 (LTS).** *Un système de transitions étiquettées (LTS) est un tuple  $(A, S, I, F, T)$  où  $A$  est un ensemble d'événements (alphabet, parfois disjoint en alphabet d'entrée et alphabet de sortie),  $S$  un ensemble d'états,  $I \in S$  un état initial,  $F \subseteq S$  un ensemble d'états finaux (permettant de différencier des états de terminaison correcte d'un état de blocage), et  $T \subseteq S \times A \times S$  la relation de transition.*

La communication entre processus (algèbres de processus) passe par différentes règles de concurrence et de communication. Ces règles sont différentes selon l'algèbre de processus considérée. CCS par exemple impose la communication binaire alors qu'en LOTOS la communication est n-aire. Arnold à montré [2] que le produit d'automate permettait d'exprimer la concurrence et la communication entre processus.

La base est le produit libre qui représente l'évolution parallèle et indépendante de deux processus (nous nous restreignons ici à des définitions binaires mais il est possible de passer à n processus). L'événement  $\epsilon$  (stuttering step) dénote un événement interne (ne rien faire).

**Definition 2 (Produit libre).** *Le produit libre de deux LTS  $L_i = (A_i, S_i, I_i, F_i, T_i)$ ,  $i \in 1..2$ , est le LTS  $(A, S, I, F, T)$  tel que :*

- $A \subseteq A_1 \times A_2$ ,
- $S \subseteq S_1 \times S_2$ ,
- $I = (I_1, I_2)$ ,
- $F \subseteq \{(s_1, s_2) \in S \mid s_1 \in F_1 \wedge s_2 \in F_2\}$ ,
- $T$  est défini par les règles suivantes :
  - $r1$  : si  $(s_1, s_2) \in S$ ,  $(s_1, a, s'_1) \in T_1$  et  $(s_2, b, s'_2) \in T_2$   
alors  $(s'_1, s'_2) \in S$  et  $((s_1, s_2), (a, b), (s'_1, s'_2)) \in T$
  - $r2$  : si  $(s_1, s_2) \in S$  et  $(s_1, a, s'_1) \in T_1$   
alors  $(s'_1, s_2) \in S$  et  $((s_1, s_2), (a, \epsilon), (s'_1, s_2)) \in T$
  - $r3$  : si  $(s_1, s_2) \in S$  et  $(s_2, b, s'_2) \in T_2$   
alors  $(s_1, s'_2) \in S$  et  $((s_1, s_2), (\epsilon, b), (s_1, s'_2)) \in T$

Ce produit dénote des systèmes qui évoluent librement. Il existe une variante sans  $r1$  (produit libre asynchrone).

Le produit synchronisé permet de modéliser la communication (plus exactement la synchronisation ou rendez-vous) entre processus. Il se base sur une équivalence entre événements qui peut s'instancier par une correspondance entre un événement  $(a)$  et son complémentaire  $(\bar{a})$  comme dans les algèbres de processus type CCS, par une correspondance entre événements de même nom et dans une liste de synchronisations autorisées comme dans les algèbres de processus type CSP ou LOTOS, ou encore par une correspondance entre événements de

même nom mais l'un étant d'entrée et l'autre de sortie comme dans les modèles à base d'IOLTS ou IOSTS. Nous présentons ici la version proche de CCS.

**Definition 3 (Produit synchronisé).** *Le produit synchronisé de deux LTS  $L_i = (A_i, S_i, I_i, F_i, T_i)$ ,  $i \in 1..2$ , est le LTS  $(A, S, I, F, T)$  tel que :*

- $A \subseteq A_1 \times A_2$ ,
- $S \subseteq S_1 \times S_2$ ,
- $I = (I_1, I_2)$ ,
- $F \subseteq \{(s_1, s_2) \in S \mid s_1 \in F_1 \wedge s_2 \in F_2\}$ ,
- $T$  est défini par les règles suivantes :
  - les règles r2 et r3
  - r4 : si  $(s_1, s_2) \in S$ ,  $(s_1, a, s'_1) \in T_1$  et  $(s_2, \bar{a}, s'_2) \in T_2$   
alors  $(s'_1, s'_2) \in S$  et  $((s_1, s_2), (a, \bar{a}), (s'_1, s'_2)) \in T$

Il existe une acceptation de produit synchronisé (synchronisation pure, forcée) qui consiste à supprimer les règles r2 et r3. Seules les synchronisations peuvent alors se passer.

Notons que ces produits ne permettent pas la prise en compte de synchronisations entre événements de noms différents. Pour cela, l'utilisation de vecteurs de synchronisation permet de spécifier les synchronisations légales (nous parlerons aussi de recollement dans la suite puisque le sens à leur donner n'est pas forcément une communication). Ces vecteurs présentés à l'origine dans [2] ont été étendus dans [19] pour prendre en compte par exemple la communication asynchrone et l'échange de données entre diagrammes d'états étendus. Nous donnons ici nos définitions.

**Definition 4 (Vecteur).** *Un vecteur pour un ensemble  $Id$  indexé de composants  $L_i = (A_i, S_i, I_i, F_i, T_i)$ ,  $i \in Id$ , est un tuple  $(e_i)$ ,  $i \in Id$  avec  $e_i \in A_i \cup \{\epsilon\}$ . Notons qu'un vecteur peut être donné implicitement (pas d'ordre obligatoire comme dans un tuple, pas d' $\epsilon$ ) comme un ensemble  $Id$  indexé d'éléments d'alphabet  $\{i.e_i\}$ ,  $i \in Id$  avec  $e_i \in A_i$ . La traduction d'un format de vecteur à l'autre est triviale.*

**Definition 5 (Produit synchronisé ouvert avec vecteur).** *Le produit synchronisé ouvert avec vecteur de deux LTS  $L_i = (A_i, S_i, I_i, F_i, T_i)$ ,  $i \in 1..2$  avec un ensemble de vecteurs  $V$ , est le LTS  $(A, S, I, F, T)$  tel que :*

- $A \subseteq A_1 \times A_2$ ,
- $S \subseteq S_1 \times S_2$ ,
- $I = (I_1, I_2)$ ,
- $F \subseteq \{(s_1, s_2) \in S \mid s_1 \in F_1 \wedge s_2 \in F_2\}$ ,
- $T$  est défini par les règles suivantes :
  - r4V : si  $(s_1, s_2) \in S$ ,  $(s_1, a, s'_1) \in T_1$ ,  $(s_2, b, s'_2) \in T_2$  et  $\exists(a, b) \in V$   
alors  $(s'_1, s'_2) \in S$  et  $((s_1, s_2), (a, b), (s'_1, s'_2)) \in T$ ,
  - r2V : si  $(s_1, s_2) \in S$ ,  $(s_1, a, s'_1) \in T_1$  et  $\bar{A}(a, \_ ) \in V$   
alors  $(s'_1, s_2) \in S$  et  $((s_1, s_2), (a, \epsilon), (s'_1, s_2)) \in T$
  - r3V : si  $(s_1, s_2) \in S$ ,  $(s_2, b, s'_2) \in T_2$  et  $\bar{A}(\_ , b) \in V$   
alors  $(s_1, s'_2) \in S$  et  $((s_1, s_2), (\epsilon, b), (s_1, s'_2)) \in T$

**Features, glu et arbres.** Le produit d'ETS se base sur une généralisation des vecteurs de synchronisation (la glu). Le metamodelle de la glu est présenté en figure 1.

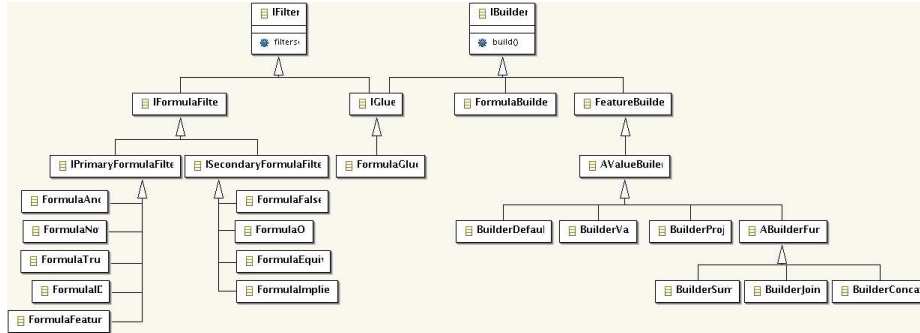


Fig. 1. Metamodelle de la glu

Les particularités de cette extension des vecteurs de synchronisation sont :

1. Transitions **et** états portent des propriétés. Le terme général pour ces propriétés est *feature*. Habituellement il s'agit de noms pour les états et d'étiquettes pour les transitions (propriétés prédéfinies). Il est cependant possible d'en définir de nouvelles.
2. Une glu décrit les recollements (leur légalité et leur valeur) entre états et entre transitions. La légalité d'un recollement est l'objet de la partie *filter* d'une glu. La valeur (ou résultat) d'un recollement est l'objet de la partie *builder* d'une glu.
3. Une glue est donc d'abord faite d'un filter qui précise les recollements légaux. Si on associe aux états une feature couleur et aux transitions une feature étiquette, il est par exemple possible de spécifier que les états de couleur bleue ne peuvent se recoller qu'avec les états de couleur rouge et que par ailleurs, les transitions d'étiquette *a* ne peuvent se recoller qu'avec les transitions d'étiquette *b* ou celles d'étiquette *c*. Cette glu peut être donnée sous forme de vecteurs étendus ou sous forme d'une formule de logique [18].
4. Un builder décrit l'effet des recollements. Il n'est plus implicite (construction de tuples dans les produits). Il est bien sûr possible d'utiliser des builders prédéfinis (construction d'un tuple d'étiquettes de transition comme dans les vecteurs de base), mais aussi d'utiliser des builders spécifiques à certaines features (une somme de couleur étant définie pour la feature couleur permettant que l'état correspondant au recollement d'un état bleu et d'un état rouge soit valué à violet) voire d'en définir des nouveaux en construisant de nouvelles classes.
5. Le résultat d'un produit est un composant de même type que celui des composants de départ (le principe composite=composant des ADL). Par ailleurs

les informations sur les systèmes recollés sont conservées lors du recollement. Pour cela nous nous basons sur le patron Composite en l’appliquant à des arbres dont les branches portent les identifiants des composants d’un produit et les feuilles les éléments de ces composants qui ont été recollés par le produit. Cette structure se retrouve aussi dans la syntaxe des glu. En reprenant notre exemple d’états coloriés, nous trouverions une formule de glu de type :

$\langle c1.[couleur=bleu], c2.[couleur=rouge] \rangle : [couleur=SUM]$

(nous simplifions la syntaxe pour l’exemple, voir la documentation utilisateur [6] pour la syntaxe réelle et complète).

En prenant deux composants  $c$  et  $d$  avec respectivement une transition  $s \xrightarrow{a} s'$  et une transition  $t \xrightarrow{b} t'$ , et une glu  $\langle c.[label=a], d.[label=b] \rangle : [label=e]$ , nous obtenons une transition globale  $[(s, t), \{c : s, d : t\}] \xrightarrow{[e, \{c:a, d:b\}]} [(s', t'), \{c : s', d : t'\}]$ .  $[(s, t), \{c : s, d : t\}]$  représente un état arbre. Sa feature nom a été constituée par mise en tuple des noms des états recollés. Par ailleurs il conserve l’information sur les états recollés.  $[e, \{c : a, d : b\}]$  représente l’étiquette d’une transition arbre. Son étiquette visible est celle spécifiée lors du recollement ( $e$ ). Par ailleurs, on conserve l’information sur les transitions recollées. Encore une fois nous avons ici simplifié la syntaxe exacte des glu et builder pour l’exemple.

Il faut noter que les algorithmes développés au LaMI induisent deux formes de glu (avec  $A_1$  et  $A_2$  les alphabets des deux composants à recoller et  $c_1$  et  $c_2$  leurs identifiants dans l’architecture du système) :

- $\forall a \in A_1 \cup A_2 . \langle c1.[label=a], c2.[label=\bar{a}] \rangle : [label=\tau]$
- $\forall a \in A_1 \cap A_2 . \langle c1.[label=a], c2.[label=\bar{a}] \rangle : [label=a]$

Il est possible d’ajouter des features et de prendre en compte différentes sémantiques de communication (produit libre, synchronisé, avec vecteurs, produits LaMI, ...) sans changer une ligne au produit ou la syntaxe de la glu. Seule la glu change. Notre produit est donc bien plus général que celui de [2] et de l’outil MEC et que ceux du LaMI.

Le principe du produit est basé sur un parcours en largeur d’abord (un état initial global est construit puis on avance dans la construction du LTS global), plus efficace dans le cadre de la construction d’un système global complet. Ceci est l’une des améliorations d’ETS par rapport à CLAP qui construisait d’abord un produit libre (potentiellement très gros) puis filtrait (supprimait) les états et transitions qui ne respectaient pas leur glu et enfin supprimait les parties du LTS résultat non atteignables depuis l’état initial global. On peut d’une certaine façon parler dans le cas d’ETS d’un parcours *build-and-filter* par opposition à un parcours *build-then-filter* pour CLAP. Il est évident que la stratégie de base d’ETS devrait être remise en cause par exemple s’il s’agit de vérifier à la volée une propriété de logique temporelle (par réfutation de sa négation). Le model-checker SPIN utilise par exemple dans ce cadre un parcours de construction en profondeur d’abord. Par ailleurs, la conservation de la structure d’arbre empêche (ou réduit considérablement) la possibilité de réduction comportementale d’un LTS (réduction observationnelle, bissimulation). Il reste dans ce cas la possibilité

de mettre à plat un t-LTS en “oubliant” les informations sur les composants recollés (l’opération est implantée dans ETS).

### 2.3 Entrés-sorties

L’éditeur graphique de STS (basé sur EMF et GEF) n’est actuellement pas terminé (en raison de la prise en compte du caractère extensible des features et de la structure d’arbre des t-STs) mais une version simplifiée permet de dessiner des modèles simples. Les aspects sémantiques présentés plus haut sont par ailleurs pour l’instant séparés du modèle interne Eclipse (EMF). L’éditeur graphique permet la lecture et la sauvegarde de STS dans le format de sérialisation XML lié au métamodèle d’Eclipse, EMF.

Nous avons par ailleurs développé deux fonctions de transformation de modèles (STS vers le format Agatha et STS vers DOT), et une fonction de lecture (parser format Agatha vers STS). Les colles explicites entre composants sont basées sur un format textuel qui nous est propre. Un objectif est la prise en compte des descriptions protocolaires des composants (STS) et des aspects architecturaux (types et instances de composants, colles) au sein du plugin Eclipse dédié à AADL, OSATE.

### 2.4 Crédits et métriques

Le développement d’ETS repose sur :

- TER CLAP, 1998-1999, Université de Nantes  
encadrement : M. Allemand et P. Poizat, étudiants : G. Nedelec et G. Salaün
- TER xCLAP, 2002-2003, Université de Nantes  
encadrement : C. Attiogbé et G. Salaün, étudiants : A. Auverlot, C. Cailler, M. Coriton, V. Gruet et M. Noël  
(voir <http://www.inrialpes.fr/vasy/Gwen.Salaun/>)
- TER Eclipse et LTS, 2004-2005, Université d’Evry  
encadrement : P. Poizat, étudiants : Th. Esnouf et L. Liger
- stage d’été de L. Liger et S. Beauche (actuellement en M2 recherche à l’Université Paris VI).

Les métriques au premier septembre 2005 étaient de : 12 paquetages, 178 classes et interfaces (ce nombre s’explique par le respect du patron Factory d’EMF). La javadoc est complète. L’environnement de développement est Eclipse 3.1, EMF 2.0.1, GEF 3.1, UML 1.1.0 et antlr. Nous avons par ailleurs beaucoup utilisé la version complète 2.1.0 d’Eclipse UML (Omondo) et nous remercions la société Omondo pour avoir bien voulu nous fournir une licence complète gratuite d’utilisation.

Une documentation plus technique (avec des exemples et la syntaxe) d’ETS pourra être trouvée dans le manuel utilisateur d’ETS [6].

### 3 Conclusions

ETS est principalement un prototype dédié à l'expérimentation de travaux et fondements théoriques développés dans le cadre du projet RNRT STACS. Il est aussi utilisé dans un outil d'adaptation orientée modèles de composants logiciels, Adaptor [10]. En cela, son objectif n'est pas de concurrencer des outils plus performants basés sur des représentations efficaces des systèmes de transitions tels que CADP (Vérification LOTOS), MEC (Produit synchronisé), FC2TOOLS (Réseaux d'automates) ou même STG (Test de STS). Il faut noter que le développement d'ETS repose principalement sur un TER et un stage d'été. Nos perspectives à court terme (TER) sont le nettoyage du plugin (structuration des paquets), la terminaison de la partie graphique et l'intégration de la partie sémantique (produits, import-export Agatha) avec la partie graphique. Nos perspectives à moyen terme sont la prise en compte de l'architecture des systèmes (intégration avec le plugin OSATE pour AADL) ainsi que l'implantation d'algorithmes d'analyse basés sur l'abstraction partielle d'architectures de composants [16,17]. Nous envisageons aussi l'ajout de fonctions d'import-export avec différents outils (CADP, SPIN).

### Références

1. M. Aiguier, C. Gaston, P. Le Gall, D. Longuet, and A. Touil. A Temporal Logic for Input Output Symbolic Transition Systems. In *Proc. of the Asia-Pacific Software Engineering Conference (APSEC'2005)*, pages 43–50, 2005.
2. A. Arnold. *Finite Transition Systems*. International Series in Computer Science. Prentice-Hall, 1994.
3. C. Attiobé, P. Poizat, and G. Salaün. A Formal and Tool-Equipped Approach for the Integration of State Diagrams and Formal Datatypes. submitted, 2005.
4. C. Attiobé, P. Poizat, and G. Salaün. Integration of Formal Datatypes within State Diagrams. In *Proc. of the International Conference on Fundamental Approaches to Software Engineering (FASE'2003)*, volume 2621 of *Lecture Notes in Computer Science*, pages 341–355. Springer Verlag, 2003.
5. T. Barros, L. Henrio, and E. Madelaine. Behavioural Models for Hierarchical Components. In *Proc. of the SPIN Workshop on Model Checking of Software*, volume 3639 of *Lecture Notes in Computer Science*, pages 154–168. Springer Verlag, 2005.
6. S. Beauche. Manuel de l'utilisateur ETS. Disponible à l'URL <http://www.lami.univ-evry.fr/~sbeauche/>, Septembre 2005.
7. C. Bigot, A. Faivre, J.-P. Gallois, A. Lapitre, D. Lugato, J.-Y. Pierron, and N. Rappin. Automatic Test Generation with AGATHA. In *Proc. of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2003)*, volume 2619 of *Lecture Notes in Computer Science*, pages 591–596. Springer Verlag, 2003.
8. M. Calder, S. Maharaj, and C. Shankland. A Modal Logic for Full LOTOS Based on Symbolic Transition Systems. *The Computer Journal*, 45(1) :55–61, 2002.
9. C. Canal, J. M. Murillo, and P. Poizat. Software Adaptation : an Introduction. *L'Objet*, 12(1) :9–31, 2006.

10. C. Canal, P. Poizat, and G. Salaün. Synchronizing Behavioural Mismatch in Software Composition. In *Proc. of the International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'2006)*, volume 4037 of *Lecture Notes in Computer Science*, pages 63–77. Springer Verlag, 2006.
11. C. Choppy, P. Poizat, and J.-C. Royer. A Global Semantics for Views. In T. Rus, editor, *Proc. of the 8th International Conference on Algebraic Methodology And Software Technology (AMAST'00)*, volume 1816 of *Lecture Notes in Computer Science*, pages 165–180. Springer Verlag, 2000.
12. C. Choppy, P. Poizat, and J.-C. Royer. The Korrigan Environment. *Journal of Universal Computer Science*, 7(1) :19–36, 2001. Special issue on Tools for System Design and Verification.
13. H. Garavel. Défense et illustration des algèbres de processus. In *Actes de l'Ecole d'été Temps Réel ETR 2003*, 2003.
14. M. Hennessy and H. Lin. Symbolic Bisimulations. *Theoretical Computer Science*, 138(2) :353–389, 1995.
15. B. Jeannot, Th. Jérón, V. Rusu, and E. Zinovieva. Symbolic Test Selection Based on Approximate Analysis. In *Proc. of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'2005)*, Lecture Notes in Computer Science, pages 349–364. Springer Verlag, 2005.
16. O. Maréchal, P. Poizat, and J.-C. Royer. Checking Asynchronously Communicating Components using Symbolic Transition Systems. In *Proc. of the International Symposium on Distributed Objects and Applications (DOA'2004)*, volume 3291 of *Lecture Notes in Computer Science*, pages 1502–1519. Springer Verlag, 2004.
17. P. Poizat, J.-C. Royer, and G. Salaün. Bounded Analysis and Decomposition for Behavioural Descriptions of Components. In *Proc. of the International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS'2006)*, volume 4037 of *Lecture Notes in Computer Science*, pages 33–47. Springer Verlag, 2006.
18. P. Poizat and G. Salaün. Formal Coordination of Communicating Entities described with Behavioural Interfaces. Technical Report 120-2005, LaMI - CNRS et Université d'Evry Val d'Essonne, 2005. available at <http://www.lami.univ-evry.fr/~poizat/>.
19. G. Salaün and P. Poizat. Interacting Extended State Diagrams. In *Proc. of the International Workshop on the Semantic Foundations of Engineering Design Languages (SFEDL'2004)*, volume 115 of *Electronic Notes in Theoretical Computer Science*, pages 49–57, 2005.
20. A. Touil, C. Gaston, and P. Le Gall. Automatic Generation of Symbolic Test Purposes. In *Proc. of the 2nd Model Design and Validation Workshop (MoDeVa'2005)*, 2005.