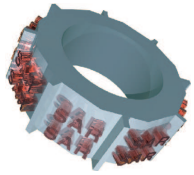
Master UPMC Sciences et technologies, mention informatique
Spécialité Systèmes et Applications Réparties

Réalisation Assistée d'Applications Réparties



TME de prise en main de ***M***eta***S***cribe

Fabrice.Kordon@lip6.fr

1er octobre 2009

Table des matières

1 Avant-propos

Cette section vous donne des informations nécessaires pour installer l'environnement de travail de ce TME dans la salle machine de la spécialité SAR ou sur le compte de votre machine personnelle.

1.1 Utiliser *M*etaScribe sur votre compte dans la salle de TME de SAR

Pour réaliser ce TME, vous devrez vous munir des fichiers et documents suivants :

- `poly-metascribe.pdf` en version imprimée ou pdf, qui contient la documentation de référence de *M*etaScribe (pour tout ce qui est syntaxe du langage, une lecture préliminaire est recommandée). Ce document est disponible sur `http://www.lip6.fr/metascribe-doc/MS_doc.pdf`

Pour construire l'environnement vous permettant de réaliser ce TME, il faut suivre les étapes suivantes :

1. Créez un répertoire dans lequel vous installerez les données du TME.
2. Dans le fichier `Makefile` associé aux données du TME, vérifiez que la définition de la variable `CONTAINER_TDIR` vaut bien `/Vrac/MetaScribe/executables`[1] sur les PC/Linux ou `/Users/admin/MetaScribe/executables` sur les iMac[2] ; pour cela, vous pouvez avoir à décommenter une ligne.

Vous êtes maintenant prêts à travailler et effectuer les questions suivantes.

1.2 Installer et utiliser *M*etaScribe sur votre machine personnelle

*M*etaScribe est développé en Ada et génère des programmes en Ada. Pour l'installer, vous devez au préalable vous assurer qu'un compilateur Ada est installé sur votre machine. Pour savoir si tel est le cas, vérifiez que la commande `which gnatmake` rende un chemin de répertoire.

Vous n'avez pas besoin d'être `root` pour procéder à l'installation, être sur votre compte suffit. Pour réaliser ce TME, vous devrez vous munir des fichiers et documents suivants :

- `MetaScribePackage-<`*version*`>.tgz` qui contient la distribution en cours de l'outil *M*etaScribe, vous obtiendrez ce fichier via l'URL suivante : `http://move.lip6.fr/METASCRIBE/download.html`,
- `poly-metascribe.pdf` en version imprimée ou pdf, qui contient la documentation de référence de *M*etaScribe (pour tout ce qui est syntaxe du langage, une lecture préliminaire est recommandée).

Pour construire l'environnement vous permettant de réaliser ce TME, il faut suivre les étapes suivantes :

1. Créez un répertoire dans lequel vous installerez l'environnement.
2. Dans ce répertoire, décompressez[3] le fichier `MetaScribeKit-<` *version* `>.tgz` et suivez les instructions contenues dans le fichier `READ_ME`. A l'issue de ces instructions, vous devriez trouver dans ce répertoire, deux autres répertoires : `package-tme` qui contient les sources de *M*etaScribe que vous devrez conserver et `executables` qui contient les exécutables produits à l'issue de l'installation.

1.3 Les données de ce TME

Vous devez récupérer le fichier `TME-MS-pem.tgz` qui contient les fichiers de base vous permettant de réaliser ce TME. Si vous êtes chez-vous, mettez vous dans le répertoire ou vous trouvez le répertoire `package-MS`. Si vous êtes dans la salle machine de la spécialité SAR, alors positionnez-vous dans le répertoire ou vous souhaitez travailler.

Dans ce répertoire, décompressez[4] le fichier `TME-MS-pem.tgz`. Cela produira un répertoire supplémentaire : `RAAR-MS-prise-en-main`.

Si vous êtes dans la salle machine de la spécialité SAR, l'outil *M*etaScribe est déjà installé sur le serveur de fichiers. Vous devez décommenter la ligne adéquate dans le fichier `Makefile` présent dans le répertoire `RAAR-MS-prise-en-main`.

Déplacez-vous dans ce dernier répertoire `RAAR-MS-prise-en-main`. Vous êtes maintenant prêts à travailler et effectuer les questions suivantes.

[1] On suppose que vous avez installé une fois *M*etaScribe dans ce répertoire

[2] *M*etaScribe est préinstallé sur les Imac.

[3] Commande `tar xzfv MetaScribeKit-<` *version* `>.tgz`

[4] Commande `tar xzfv TME-MS-pem.tgz`

1.4 l'exemple de test par défaut

Pour traiter cet exercice, nous vous fournissons un jeu de test simple correspondant au réseau de Petri de la Figure 1. Le code MSM décrivant cet exemple est donné en annexe.

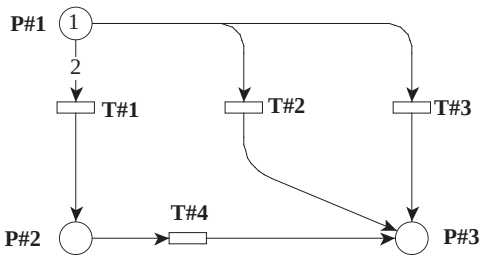


FIG. 1 – Schéma du réseau de Petri fourni comme premier jeu de test.

Il va sans dire qu'il est vivement conseillé que vous établissiez une batterie d'exemples complémentaires pour tester votre moteur de transformation. Si vous me les envoyez (schéma Macao + fichier MSM), je veux bien les mettre à la disposition de tous sur le site web de SAR.

ATTENTION : n'utilisez que des caractères ISO sans accents pour décrire des fichiers MSM, MSF, MSSM et MSST ; le parseur de ***M***eta***S***cribe ne les supporte pas.

2 Mécanique de production d'un moteur de transformation

Nous vous proposons dans un premier temps de «jouer» avec le système. Pour cela, vous pouvez consulter les fichier `.msf` qui correspondent au méta-modèle d'un réseaux de Petri place/transition[5]. Il vous est également donné des exemples de modèle dans ce formalisme dans les fichiers `.msm` que vous pouvez consulter également.

Regardez la structure du patron sémantique (fichiers `.mssm`) et du patron syntaxique (fichiers `.msst`).

Tapez ensuite la commande suivante :

`make rules-q1`[6]

Vous devriez obtenir un ensemble de fichiers Ada que vous pouvez consulter. Vous verrez qu'ils sont la traduction parfaite des différentes règles écrites dans les langages MSSM et MSST.

Tapez alors la commande suivante :

`make tengine-q1`

Cette commande lance la compilation de ce premier moteur de transformation. Si tout se passe bien, vous devez trouver à l'issue de cette commande le fichier exécutable `p_t_nets_statistics`.

Vous devez ensuite invoquer le moteur de transformation pour des modèles donnés. La commande vous est donnée dans le Makefile. Pour l'exemple présenté en cours, L'invocation est accessible via la commande suivante :

`make test-q1` Pour toutes information supplémentaire, référez-vous au manuel d'utilisation qui vous est fourni.

A l'issue de ces manipulations, vous devriez savoir générer un moteur de transformation à partir d'un ensemble de fichiers décrivants un patron sémantique et un patron syntaxique.

Question 2.1 :

A partir de l'analyse des règles et arbres définis dans les fichiers `rdp_sem.rules.mssm` et `rdp_sem.trees.mssm`, indiquez la structure de l'arbre de syntaxe abstraite produit par le patron sémantique **P_T_NETS**.

Dans les parties suivantes, vous allez devoir coder certains éléments d'un patron sémantique et syntaxique.

[5] Pour ceux qui n'ont pas suivi l'UE MSR, ce n'est pas grave, la courte présentation de cette notation dans le cours précédant ce TME suffit pour ce que nous souhaitons réaliser.

[6] lors d'essais suivants, il est recommandé de lancer la commande `make superclean` au préalable.

3 Modification d'un patron syntaxique

L'objectif de cette partie est de modifier le patron syntaxique qui vous est proposé sans toucher au patron sémantique.

Le patron sémantique par défaut (analysé en cours) produit un fichier listant les places et les transition d'un réseau de Petri. Le résultat est le suivant (pour le fichier `rdp-1-main.msm`) :

```
---------------------------------
Voici le contenu du modele test_model
---------------------------------
Il y a 3 places, les voici :
   Place P#1
   Place P#2
   Place P#3
Il y a 4 transitions, les voici :
   Transition T#1
   Transition T#2
   Transition T#3
   Transition T#4
```

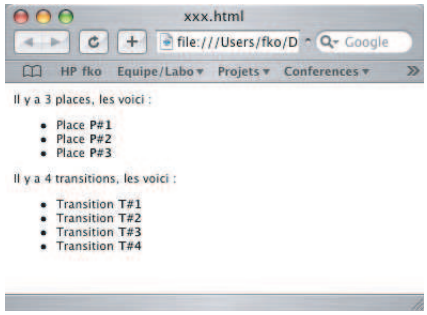


FIG. 2 – Résultat attendu pour le nouveau patron syntaxique.

Nous souhaitons produire un fichier HTML correspondant au résultat donné en Figure 2. Nous devons donc générer le code suivant (toujours pour le fichier `rdp-1-main.msm`) :

```
<html><body>
<p>Il y a 3 places, les voici :</p><ul>
<li>Place <b>P#1</b></li>
<li>Place <b>P#2</b></li>
<li>Place <b>P#3</b></li>
</ul>
<p>Il y a 4 transitions, les voici :</p><ul>
<li>Transition <b>T#1</b></li>
<li>Transition <b>T#2</b></li>
<li>Transition <b>T#3</b></li>
<li>Transition <b>T#4</b></li>
</ul>
</body></html>
```

Question 3.1 :

Réalisez le patron syntaxique permettant de produire le fichier HTML voulu.

Il vous est demandé de modifier le patron syntaxique en conséquence. Si vous souhaitez récupérer les commandes prévues à cet effet pour le `Makefile`, il vous faudra respecter les conventions suivantes :

– le patron doit se nommer `STAT_HTML`.

– les fichiers qui le décrivent doivent se nommer `rdp_html.main.msst` et `rdp_html.annex.msst`.

Ces conventions vous permettent de faire ce qui suit :

`make q2` (réalise un "superclean" puis les trois commandes ci après)

`make rules-q2` (génère le moteur de transformation)

`make tengine-q2` (compile le moteur de transformation)

`make test-q2` (exécute le moteur compilé pour l'exemple trivial vu en cours)

NB : vous ne devez pas toucher au patron sémantique à ce stade.

Important : mode debug de *M*eta*S*cribe

Vous pouvez activer le mode debug de *M*eta*S*cribe afin de tracer certaines règles en positionnant la variable `DEBUG` derrière les actions `q2` ou `rules-q2`. La syntaxe à adopter est donnée directement en invoquant *M*eta*S*cribe sans paramètres, comme indiqué ci après[7] :

```
$ ../executables/meta_scribe
MetaScribe (1.1b11) by F.Kordon, May 2003 (3.0-3.0-3.0)
================================================
= Laboratoire d'Informatique de Paris 6,       =
= Universite P. & M. Curie, 4 place Jussieu,   =
= 75252 Paris Cedex 05, France                 =
================================================
usage : meta_scribe [-t]{-Txxx}[-v version]
                    formalism_file semantic_pattern_file syntactic_pattern_file
        -t for debug mode for all rules
        -95e to enable Ada-95 exception name handling (error propagation clearer)
        -Txxx to put semantic or syntactic entity xxx in trace mode
        -a to set the author of the transformation engine
        -v to set a version number to the generated transformation engine
```

Ainsi, `make q2 DEBUG="-t"` active la trace de toutes les règles. Similairement, `make q2 DEBUG=-TWORK_ON_PLACES` n'active les traces que pour la règle `WORK_ON_PLACES`. Vous pouvez utiliser l'option -T plusieurs fois pour activer les traces sur différentes règles sémantiques ou syntaxiques.

Lorsque vous activez les traces sur une règle (ici sémantique), *M*eta*S*cribe génère du code supplémentaire pour tracer les paramètre d'appel (s'il y en a) ainsi que les valeurs rendues par les règles (s'il y a lieu). Vous trouvez ci après un exemple pour la règle sémantique `WORK_ON_PLACES` qui n'a aucun paramètre d'appel et rend un arbre de syntaxe abstraite (arbre sémantique). Si une règle tracée effectue des affichages, alors ces derniers sont reproduits fidèlement.

```
--> WORK_ON_PLACES (SMR)
no parameters
return value:
[P_T_NETS_RID_LIST_OF_PLACES - 3 - ]
   [P_T_NETS_RID_OBJECT_NAME - P#1 - ]
   [P_T_NETS_RID_OBJECT_NAME - P#2 - ]
   [P_T_NETS_RID_OBJECT_NAME - P#3 - ]
<-- WORK_ON_PLACES (SMT)
```

4 Modification d'un patron sémantique (et syntaxique)

L'objectif de cette partie est de travailler à la fois les aspects sémantiques et syntaxiques du moteur de transformation. Vous devrez donc modifier les deux types de patrons.

Par rapport aux parties précédentes, nous souhaitons traiter les informations relatives aux arcs du réseau de Petri et les afficher, au format HTML, de manière similaire à ce qui est indiqué en Figure 3. Pour cela, il nous faudra procéder par étapes. Les questions suivantes ont pour objectif de vous guider.

Question 4.1 :

Enrichissez la structure de l'arbre de syntaxe abstraite déterminée en question 1.1 afin d'y intégrer le support des nouvelles informations à produire.

Identifiez sur cet arbre de syntaxe abstraite les nouvelles constructions syntaxiques qu'il faut déclarer dans le patron sémantique.

À partir de cette information, quelles nouvelles règles syntaxique faudra-t-il introduire dans le patron syntaxique ?

[7] On suppose que vous êtes dans le répertoire du TME et que l'installation s'est faite conformément à ce qui a été indiqué en préambule.

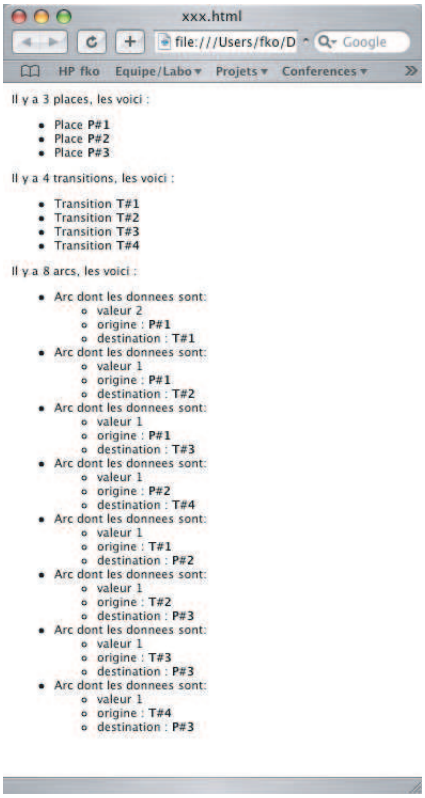


FIG. 3 – Résultat attendu avec traitement des informations relatives aux arcs.

Question 4.2 :

Élaborez un nouveau patron sémantique et un nouveau patron syntaxique permettant de produire le code HTML suivant (pour l'exemple considéré).

```
<html><body>
<p>Il y a 3 places, les voici :</p><ul>
<li>Place <b>P#1</b></li>
<li>Place <b>P#2</b></li>
<li>Place <b>P#3</b></li>
</ul>
<p>Il y a 4 transitions, les voici :</p><ul>
<li>Transition <b>T#1</b></li>
<li>Transition <b>T#2</b></li>
<li>Transition <b>T#3</b></li>
<li>Transition <b>T#4</b></li>
</ul>
<p>Il y a 8 arcs, les voici :</p><ul>
<li>Arc dont les donnees sont:<ul><li>
valeur 2</li><li>origine :
<b>P#1</b></li><li>destination :
<b>T#1</b></li></ul>
</li>
<li>Arc dont les donnees sont:<ul><li>
valeur 1</li><li>origine :
<b>P#1</b></li><li>destination :
<b>T#2</b></li></ul>
</li>
<li>Arc dont les donnees sont:<ul><li>
valeur 1</li><li>origine :
<b>P#1</b></li><li>destination :
<b>T#3</b></li></ul>
</li>
<li>Arc dont les donnees sont:<ul><li>
```

```
valeur 1</li><li>origine :
<b>P#2</b></li><li>destination :
<b>T#4</b></li></ul>
</li>
<li>Arc dont les donnees sont:<ul><li>
valeur 1</li><li>origine :
<b>T#1</b></li><li>destination :
<b>P#2</b></li></ul>
</li>
<li>Arc dont les donnees sont:<ul><li>
valeur 1</li><li>origine :
<b>T#2</b></li><li>destination :
<b>P#3</b></li></ul>
</li>
<li>Arc dont les donnees sont:<ul><li>
valeur 1</li><li>origine :
<b>T#3</b></li><li>destination :
<b>P#3</b></li></ul>
</li>
<li>Arc dont les donnees sont:<ul><li>
valeur 1</li><li>origine :
<b>T#4</b></li><li>destination :
<b>P#3</b></li></ul>
</li>
</ul>
</body></html>
```

Si vous souhaitez récupérer les commandes prévues à cet effet pour le Makefile, il vous faudra respecter les convention suivantes :

- le patron sémantique doit se nommer `PT_WITH_ARCS`.
- les fichiers qui décrivent le patron sémantique doivent se nommer `rdp_sem2.main.mssm`, `rdp_sem2.trees.mssm` et `rdp_sem2.rules.mssm` (vous conserverez la structure du patron sémantique existant pour plus de facilité).
- le patron syntaxique doit se nommer `STAT_HTML2`.
- les fichiers qui décrivent le patron syntaxique doivent se nommer `rdp_html2.main.msst` et `rdp_html2.annex.msst` (vous conserverez la structure du patron syntaxique précédent pour plus de facilité).

Ces conventions vous permettent de faire ce qui suit :

`make q3` (réalise un "superclean" puis les trois commandes ci après)

`make rules-q3` (génère le moteur de transformation)

`make tengine-q3` (compile le moteur de transformation)

`make test-q3` (exécute le moteur compilé pour l'exemple trivial vu en cours)

Pour la mise au point, vous utiliserez les même mécanismes que ce décrits pour la question 2.2.

5 Produire un fichier au format dot

L'objectif de cette question est de départager les ex-aequo ;-)

Construisez un moteur de transformation qui produit un fichier au format de l'outil `dot`, des AT&T Labs (ex Bell-Labs). Vous trouverez toutes les informations nécessaire à l'URL suivante :

`http://www.research.att.com/sw/tools/graphviz/refs.html`.

L'idée est que l'on puisse ainsi utiliser l'outil `dot` pour produire un dessin dans les formats `eps`, `jpg` ou `gif` du réseau de Petri considéré.

Question 5.1 :

Pour me faciliter le test de vos travaux, ***le patron sémantique devra se nommer PT_WITH_ARCS2 et le patron syntaxique devra se nommer TO_DOT***. Ainsi, le fichier exécutable correspondant au moteur de transformation se nommera `pt_with_arcs2_to_dot`. Cela me permettra de passer mes jeux de tests en batterie et donc de m'assurer que votre travail se comporte correctement pour chacun d'entre vous.

Vous rendrez le travail correspondant à cette partie pour le **Samedi 13 Novembre 2004** en respectant le protocole suivant :

– Si l'authentification fonctionne, vous utiliserez delivery_builder pour rendre votre travail.
– Dans le cas contraire, vous me construirez une archive au format tar/gz en vous mettant dans le répertoire ou vous avez travaillé et en tapant la commande suivante :
`tar livraison_`$<$ *mon* $-$ *nom* $>$`.tgz czv *.msf *.msm *.mssm *.msst Makefile`
Où $<$ *mon* $-$ *nom* $>$ est votre nom (afin que je puisse vous attribuer une note). Ce fichier contiendra ainsi l'ensemble des fichiers me permettant de produire le moteur de transformation et un fichier `Makefile` me permettant de construire le moteur de transformation que vous devrez rendre.

Annexe A : code MSM du modèle donné en exemple

```
MAIN_FILE
FORMALISM ( 'TST_FORM_RDPO' ) ;
// Attributs globaux
WHERE (ATTRIBUTE net_name => 'test_model') ;

// Places

NODE 'place_1' IS place
   WHERE (ATTRIBUTE name => 'P#1',
  ATTRIBUTE marking => 2) ;
NODE 'place_2' IS place
   WHERE (ATTRIBUTE name => 'P#2') ;
NODE 'place_3' IS place
   WHERE (ATTRIBUTE name => 'P#3') ;

// Transitions

NODE 'trans_1' IS transition
   WHERE (ATTRIBUTE name => 'T#1') ;
NODE 'trans_2' IS transition
   WHERE (ATTRIBUTE name => 'T#2') ;
NODE 'trans_3' IS transition
   WHERE (ATTRIBUTE name => 'T#3') ;
NODE 'trans_4' IS transition
   WHERE (ATTRIBUTE name => 'T#4') ;

// Les arcs

LINK 'p1_to_t1' is ARC
WHERE (ATTRIBUTE value => 2)
RELATE 'place_1' TO 'trans_1';
LINK 't1_to_p2' is ARC
WHERE (ATTRIBUTE value => 1)
RELATE 'trans_1' TO 'place_2';
LINK 'p1_to_t2' is ARC
WHERE (ATTRIBUTE value => 1)
RELATE 'place_1' TO 'trans_2';
LINK 'p1_to_t3' is ARC
WHERE (ATTRIBUTE value => 1)
RELATE 'place_1' TO 'trans_3';
LINK 'p2_to_t4' is ARC
WHERE (ATTRIBUTE value => 1)
RELATE 'place_2' TO 'trans_4';
LINK 't2_to_p3' is ARC
WHERE (ATTRIBUTE value => 1)
RELATE 'trans_2' TO 'place_3';
LINK 't3_to_p3' is ARC
WHERE (ATTRIBUTE value => 1)
RELATE 'trans_3' TO 'place_3';
LINK 't4_to_p3' is ARC
WHERE (ATTRIBUTE value => 1)
RELATE 'trans_4' TO 'place_3';
```

Annexe B : Métamodèle de réseaux de Petri exprimé en MSF

Fichier principal

```
main_file
formalism ('TST_FORM_RDPO');

// ======================================================
// Les entites d'un RdP Ordinaire

entity_list
   PLACE : node,
   TRANSITION : node,
   ARC : link;

// ======================================================
// Les attributs globaux d'un RdP Ordinaire

global_attributes
   attribute string  : NET_NAME;
end;

// ======================================================
// Pas d'operateurs car on n'a pas d'attribut de type
// expression.

construction_list (NONE);

// ======================================================
// Les Fichiers annexes

file_list
   file ('rdp_form.annex.msf');
```

Fichier annexe

```
annex_file
// ======================================================
// Les places

node (PLACE) is
   attribute_list
      attribute string  : NAME;
      attribute integer : MARKING;
   end ;
   connectability_list
      with ARC
         direction out ,
         maximum none ;
      with ARC
         direction in ,
         maximum none ;
   end ;
end PLACE ;

// ======================================================
// Les transitions

node (TRANSITION) is
   attribute_list
      attribute string : NAME;
   end ;
   connectability_list
      with ARC
         direction out ,
         maximum none ;
      with ARC
         direction in ,
         maximum none ;
   end ;
```

```
end TRANSITION ;

// ======================================================
// Les arcs

link (ARC) is
   attribute_list
      attribute integer : VALUE ;
   end ;
end ARC ;
```