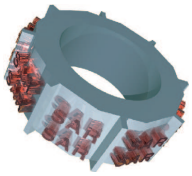
Master UPMC Sciences et technologies,
mention informatique
Spécialité *Systèmes et Applications Réparties*

Réalisation Assistée d'Applications Réparties



Projet - écriture d'un générateur de code

Fabrice.Kordon@lip6.fr

5 novembre 2012

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.

Utiliser *M*eta*S*cribe sur votre compte dans la salle de TME de SAR

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

- `MS_doc.pdf` en version imprimée ou pdf, qui contient la documentation de référence de ***M***eta***S***cribe (pour tout ce qui est syntaxe du langage, une lecture préliminaire est recommandée). Ce document est disponible sur `http://move.lip6.fr/software/METASCRIBE/manual.html`

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.

Installer et utiliser *M*eta*S*cribe sur votre machine personnelle

Meta***S***cribe 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***eta***S***cribe, vous obtiendrez ce fichier via l'URL suivante : `http://move.lip6.fr/software/METASCRIBE/download.html`,
- `MS_doc.pdf` en version imprimée ou pdf, qui contient la documentation de référence de ***M***eta***S***cribe (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***eta***S***cribe que vous devrez conserver et `executables` qui contient les exécutables produits à l'issue de l'installation.

1.1 Les données de ce TME

Vous devez récupérer le fichier `RAAR-proj-MS.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 `RAAR-proj-MS.tgz`. Cela produira un répertoire supplémentaire : `RAAR-env-projet`. Vous devez positionner la variable d'environnement référençant les exécutables de ***M***eta***S***cribe das le `Makefile`

Déplacez-vous dans ce dernier répertoire `CAR-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***eta***S***cribe dans ce répertoire
2. ***M***eta***S***cribe est préinstallé sur les Imac.
3. Commande `tar xzfv MetaScribeKit-<` *version* `>.tgz`
4. Commande `tar xzfv RAAR-proj-MS.tgz`

1.2 Exemples de test fournis

Pour traiter cet exercice, nous vous fournissons deux exemples simples correspondant aux réseaux de Petri des Figure 1 et 2. Le code **MSM** décrivant ces exemples est donné en annexe. Nous vous les montrons sous la forme de réseaux de Petri mais vous devez les comprendre comme indiqué à droite des fihures (*i.e.* comme des automates communicants de manière synchrone ou asynchrone).

La Figure 1 montre un modèle impliquant deux processus qui se synchronisent en accédant à une donnée. Si la donnée est prise par l'un d'eux, l'autre ne peut s'exécuter et vice-versa. La synchronisation se fait au moyen d'un `COMM_PLACE` dans le formalisme d'automates communicants fourni dans le sujet.

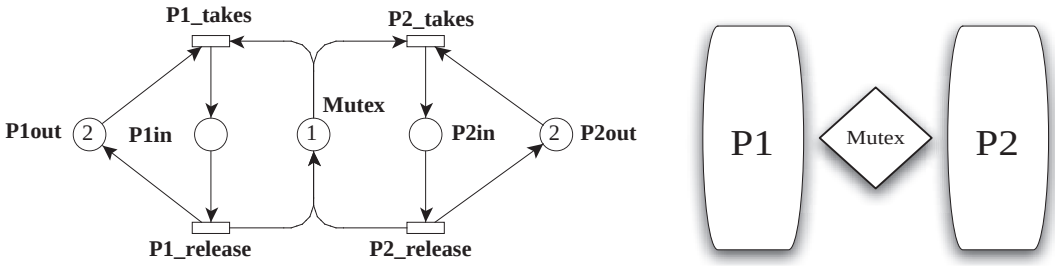


FIGURE 1 – Schéma du modèle *anlzed-pn-1* fourni comme premier jeu de test.

La Figure 2 montre un second modèle de synchronisation et le schéma suivant la sémantique d'automates communicants (à droite. Les deux processus se synchronisent via une barrière matérialisée par un `SYNC_TRANS` dans le formalisme d'automates communicants fourni dans le sujet.

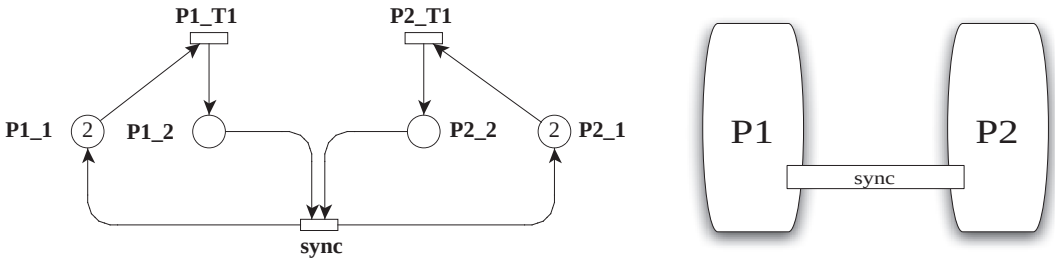


FIGURE 2 – Schéma du modèle *anlzed-pn-2* fourni comme second jeu de test.

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 Objectifs du projet

L'objectif de ce projet est de vous faire écrire un générateur de programmes à partir d'un réseaux de Petri place/transition pré-analysé (c'est-à-dire que le partitionnement a déjà été effectué pour vous). Pour cela, vous devrez vous mettre en équipe de 6 pour faire ce travail (7 autorisés à titre exceptionnel car vous êtes plus de 30 étudiants à suivre l'UE RAAR). Vous devrez ainsi vous organiser en équipes et vous structurer pour mener ce projet.

Le travail s'accompagnera d'un rapport (rédigé avec LaTeX)qui brièvement rappeler :

– vos objectifs,
– l'architecture gros grain de votre prototype ainsi que les principaux choix (qu'ils soient liés à la classe de langages utilisés ou au langage en lui-même),
– l'architecture gros grain de votre générateur de programmes et la manière dont vous avez structuré vos règles dans les différents patrons,
– votre technique de validation (les modèles que vous avez validés),
– une synthèse et des remarques conclusives.

Inutile d'en «faire des tonnes», vous devez plutôt être efficaces et donner une compréhension générale de la manière dont vous avez fait ce travail. Nous vous recommandons vivement de suivre le plan-type qui est proposé dans le modèle de rapport que nous vous fournissons.

3 À propos des fichiers fournis

Cette section vous donne des informations utiles sur l'utilisation des fichiers que nous vous fournissons dans l'archive `RAAR-proj-MS.tgz`.

3.1 Conventions à respecter

Pour vous faciliter la tâche, nous vous fournissons un fichier `Makefile` dont les entrées principales sont les suivantes :

– `rules-generator` qui invoque ***M***eta***S***cribe pour produire les sources du generateur de code et transforme le code au format voulu par le compilateur `gnat`,
– `tengine-generator` qui compile le code généré par ***M***eta***S***cribe,
– `test-generator` qui exécute le programme produit par ***M***eta***S***cribe puis compilé avec `gnat`,
– `generator` qui invoque successivement les trois règles précédentes (c'est l'entrée par défaut),
– `showinput1` et `showinput2` qui permettent de visualiser le contenu des descriptions **MSM** des exemples fournis dans la distribution,
– `rapport.pdf` qui compile le rapport que vous devez fournir,
– `clean`, `bigclean`, `superclean` qui sont trois niveau de nettoyage des données générées.

Vous pouvez (devrez ;-) modifier et ajouter des entrées à ce `Makefile` mais je vous recommandons vivement respecter les conventions suivantes :

1. la commande `make` (sans paramètre) doit permettre de construire votre générateur de programmes,
2. Vous devez compléter dans le Makefile l'entrée `test-pour-fk` en respectant la convention donnée ci-dessous.

L'entrée `test-pour-fk` me permettra de traiter des exemples en sachant que le modèle <*nom*> sera contenu dans un fichier <*nom*>`-main.msm` et que l'on supposera que le «main» du code produira un exécutable (dans le langage voulu) s'appelant <*nom*>. Cette entrée procèdera à la génération des programmes dans le langage choisi, la compilation des programmes produits puis l'exécution de ce programme. Le non respect de ces conventions pourraient m'indisposer lors de la notation en me compliquant la tâche d'évaluation de votre travail ;-). Cette entrée s'invoquera comme suit :

```
make test-pour-fk NAME=toto
```

Où `toto` est le nom du modèle (qui sera par convention stocké dans le fichier `toto-main.msm`). Le fichier exécutable produit doit alors se nommer `toto`.

Nous vous fournissons également une capsule pour les fichiers **MSSM** (patron sémantique) et **MSST** (patron syntaxique). Il vous est vivement recommandé de ne changer ni le nom des patrons sémantiques et syntaxique, ni le nom des fichier principaux (`generator-main.mssm` et `to-language-main.msst`) sous peine de risquer que je ne puisse tester correctement votre travail (cela risque également de m'indisposer au moment de la notation ;-).

3.2 Le 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 [5] :

```
$ ../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, la commande suivante :

```
make q2 DEBUG="-t"
```

active la trace de toutes les règles.

Similairement, la commande :

```
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)
```

3.3 Structuration de la livraison

Vous rendrez le travail correspondant à ce projet pour le **Dimanche 4 Décembre 2011** en utilisant delivery_builder.

Pour que la correction se fasse de la manière la plus pratique, je vous demande de respecter **scrupuleusement** les consignes suivantes :

- votre livraison incluera à la racine le fichier `equipe.csv` indiquant (à raison de un par ligne) chacun des membres du projet. Le format d'une ligne sera
  <NOM> ; *<prenom>* ; *<dossier>* ; *<email>* ;
- vous suivrez rigoureusement les conventions décrites en section 3.1, page 4 ;
- Votre jeu-de-test étendu (conseillé) sera déposé dans un répertoire `JDT` contenant les fichiers MSM.
- Pour chaque élément de votre jeu de tests étendu, vous dessinerez le modèle RTdP P/T en Macao et mettrez le tout dans une archive .dmg. Vous fournirez également pour chaque modèle une version pdf. Cela me permettra de visualiser simplement votre jeu de teste et de décider de sa pertinence.
- vous laisserez un fichier `READ_ME.txt` décrivant toutes les caractéristiques propres à votre projet.

5. 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.

Annexe A : code MSM des modèles donnés en exemple

fichier `anlzed-pn-1.msm`

```
main_file
formalism ( 'ANLZED_PN' ) ;
// Attributs globaux
where (attribute NAME => 'duo_mutex') ;

// Places de communications

node 'cp1' is COMM_PLACE
   where (attribute NAME => 'mutex',
          attribute NB_IN_TOKENS => 1);

// Premier processus

node 'proc1' is PROCESS
   where (attribute NAME => 'processus_1',
          attribute INSTANCES => sy_node (INSTANCE_LIST:
                                    sy_node (ONE_INSTANCE:
                                       sy_leaf ('P1out')),
                                    sy_node (ONE_INSTANCE:
                                       sy_leaf ('P1out'))
                                       ));


node 'p1p1' is STT_PLACE
   where (attribute NAME => 'P1out');

node 'p1p2' is STT_PLACE
   where (attribute NAME => 'P1in');

node 'p1t1' is LOC_TRANS
   where (attribute NAME => 'P1_Takes');

node 'p1t2' is LOC_TRANS
   where (attribute NAME => 'P1_release');

link 'p1bl1' is BELONG
where (none)
relate together STT_PLACE:'p1p1', STT_PLACE:'p1p2',
   LOC_TRANS:'p1t1', LOC_TRANS:'p1t2', PROCESS:'proc1';

link 'p1l1' is ARC
where (none)
relate STT_PLACE:'p1p1' to LOC_TRANS:'p1t1';

link 'p1l2' is ARC
where (none)
relate STT_PLACE:'p1p2' to LOC_TRANS:'p1t2';

link 'p1l3' is ARC
where (none)
relate LOC_TRANS:'p1t2' to STT_PLACE:'p1p1';

link 'p1l4' is ARC
where (none)
relate LOC_TRANS:'p1t1' to STT_PLACE:'p1p2';

link 'commp1l1' is COMM_ARC
where (none)
relate LOC_TRANS:'p1t1' to COMM_PLACE:'cp1';

link 'commp1l2' is COMM_ARC
where (none)
relate LOC_TRANS:'p1t2' to COMM_PLACE:'cp1';


// Second processus
```

```
node 'proc2' is PROCESS
   where (attribute NAME => 'processus_2',
          attribute INSTANCES => sy_node (INSTANCE_LIST:
                                    sy_node (ONE_INSTANCE:
                                       sy_leaf ('P2out')),
                                    sy_node (ONE_INSTANCE:
                                       sy_leaf ('P2out'))
                                       ));


node 'p2p1' is STT_PLACE
   where (attribute NAME => 'P2out');

node 'p2p2' is STT_PLACE
   where (attribute NAME => 'P2in');

node 'p2t1' is LOC_TRANS
   where (attribute NAME => 'P2_Takes');

node 'p2t2' is LOC_TRANS
   where (attribute NAME => 'P2_release');

link 'p2bl1' is BELONG
where (none)
relate together STT_PLACE:'p2p1', STT_PLACE:'p2p2',
   LOC_TRANS:'p2t1', LOC_TRANS:'p2t2', PROCESS:'proc2';

link 'p2l1' is ARC
where (none)
relate STT_PLACE:'p2p1' to LOC_TRANS:'p2t1';

link 'p2l2' is ARC
where (none)
relate STT_PLACE:'p2p2' to LOC_TRANS:'p2t2';

link 'p2l3' is ARC
where (none)
relate LOC_TRANS:'p2t2' to STT_PLACE:'p2p1';

link 'p2l4' is ARC
where (none)
relate LOC_TRANS:'p2t1' to STT_PLACE:'p2p2';

link 'commp2l1' is COMM_ARC
where (none)
relate LOC_TRANS:'p2t1' to COMM_PLACE:'cp1';

link 'commp2l2' is COMM_ARC
where (none)
relate LOC_TRANS:'p2t2' to COMM_PLACE:'cp1';
```

fichier `anlzed-pn-2.msm`

```
main_file
formalism ( 'ANLZED_PN' ) ;
// Attributs globaux
where (attribute NAME => 'duo_mutex') ;

// Places de communications

node 'ts1' is SYNC_TRANS
   where (attribute NAME => 'sync');

// Premier processus

node 'proc1' is PROCESS
   where (attribute NAME => 'processus_1',
          attribute INSTANCES => sy_node (INSTANCE_LIST:
                                    sy_node (ONE_INSTANCE:
```

```
                                    sy_leaf ('P1_1')),
                                 sy_node (ONE_INSTANCE:
                                    sy_leaf ('P1_2'))
                                    ));


node 'p1p1' is STT_PLACE
   where (attribute NAME => 'P1_1');

node 'p1p2' is STT_PLACE
   where (attribute NAME => 'P1_2');

node 'p1t1' is LOC_TRANS
   where (attribute NAME => 'P1_T1');

link 'p1bl1' is BELONG
where (none)
relate together STT_PLACE:'p1p1', STT_PLACE:'p1p2',
   LOC_TRANS:'p1t1', SYNC_TRANS:'ts1', PROCESS:'proc1';

link 'p1l1' is ARC
where (none)
relate STT_PLACE:'p1p1' to LOC_TRANS:'p1t1';

link 'p1l2' is ARC
where (none)
relate STT_PLACE:'p1p2' to SYNC_TRANS:'ts1';

link 'p1l3' is ARC
where (none)
relate SYNC_TRANS:'ts1' to STT_PLACE:'p1p1';

link 'p1l4' is ARC
where (none)
relate LOC_TRANS:'p1t1' to STT_PLACE:'p1p2';


// Second processus

node 'proc2' is PROCESS
   where (attribute NAME => 'processus_2',
          attribute INSTANCES => sy_node (INSTANCE_LIST:
                                    sy_node (ONE_INSTANCE:
                                       sy_leaf ('P2_1')),
                                    sy_node (ONE_INSTANCE:
                                       sy_leaf ('P2_2'))
                                       ));


node 'p2p1' is STT_PLACE
   where (attribute NAME => 'P2_1');

node 'p2p2' is STT_PLACE
   where (attribute NAME => 'P2_2');

node 'p2t1' is LOC_TRANS
   where (attribute NAME => 'P2_T1');

link 'p2bl1' is BELONG
where (none)
relate together STT_PLACE:'p2p1', STT_PLACE:'p2p2',
   LOC_TRANS:'p2t1', SYNC_TRANS:'ts1', PROCESS:'proc2';

link 'p2l1' is ARC
where (none)
relate STT_PLACE:'p2p1' to LOC_TRANS:'p2t1';

link 'p2l2' is ARC
where (none)
relate STT_PLACE:'p2p2' to SYNC_TRANS:'ts1';
```

```
link 'p2l3' is ARC
where (none)
relate SYNC_TRANS:'ts1' to STT_PLACE:'p2p1';

link 'p2l4' is ARC
where (none)
relate LOC_TRANS:'p2t1' to STT_PLACE:'p2p2';
```

Annexe B : Métamodèle de réseaux de Petri analysé exprimé en MSF

Il s'agit d'une présentation analysée de ces réseaux de Petri qui sont ici vus comme des machines à états communicantes de manière synchrone ou asynchrone.

Fichier principal (`anlzed-pn-main.msf`)

```
main_file
formalism ('ANLZED_PN');

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

entity_list
   PROCESS    : node,
   STT_PLACE  : node,
   COMM_PLACE : node,
   LOC_TRANS  : node,
   SYNC_TRANS : node,
   BELONG     : link,
   ARC        : link,
   COMM_ARC   : link;

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

global_attributes
   attribute string  : NAME;
end;

// ======================================================
// Les operateurs pour decrire les instances

construction_list (INSTANCE_LIST,
   ONE_INSTANCE);

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

file_list
   file ('anlzed-pn-node.msf'),
   file ('anlzed-pn-link.msf');
```

Fichier `anlzed-pn-node.msf` (annexe)

```
annex_file

// ======================================================
node (PROCESS) is
   attribute_list
      attribute string     : NAME;
  attribute expression : INSTANCES;
   end;
   connectability_list
      with BELONG
         direction non_oriented,
         maximum none;
   end ;
end PROCESS;

// ======================================================
node (STT_PLACE) is
   attribute_list
      attribute string : NAME;
   end;
   connectability_list
      with BELONG
         direction non_oriented,
         maximum 1;
      with ARC
```

```
         direction in,
         maximum none;
      with ARC
         direction out,
         maximum none;
   end;
end STT_PLACE;

// =======================================================
node (COMM_PLACE) is
   attribute_list
      attribute string  : NAME;
      attribute integer : NB_IN_TOKENS;
   end;
   connectability_list
      with COMM_ARC
         direction in,
         maximum none;
      with COMM_ARC
         direction out,
         maximum none;
   end;
end COMM_PLACE;

// =======================================================
node (LOC_TRANS) is
   attribute_list
      attribute string : NAME;
   end;
   connectability_list
      with BELONG
         direction non_oriented,
         maximum 1;
      with ARC
         direction in,
         maximum 1;
      with ARC
         direction out,
         maximum 1;
      with COMM_ARC
         direction in,
         maximum none;
      with COMM_ARC
         direction out,
         maximum none;
   end;
end LOC_TRANS;

// =======================================================
node (SYNC_TRANS) is
   attribute_list
      attribute string : NAME;
   end;
   connectability_list
      with BELONG
         direction non_oriented,
         maximum none;
      with ARC
         direction in,
         maximum none;
      with ARC
         direction out,
         maximum none;
   end;
end SYNC_TRANS;
```

Fichier `anlzed-pn-link.msf` (annexe)

```
annex_file
```

```
// ========================================================
link (BELONG) is
   attribute_list
      none
   end;
end BELONG;

// ========================================================
link (ARC) is
   attribute_list
      none
   end;
end ARC;

// ========================================================
link (COMM_ARC) is
   attribute_list
      attribute integer : VALUE;
   end;
end COMM_ARC;
```