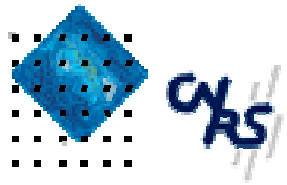
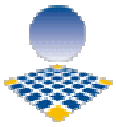
Du Modèle à l'exécution : Traduction Automatique d'un Réseau de Petri Interprété en Langage VHDL

David Andreu, Nicolas Bruchon, Thierry Gil



LIRMM



UM II

Du modèle à l'exécution :

Traduction automatique d'un réseau de Petri interprété en langage VHDL

D. Andreu, N. Bruchon, T. Gil

Rapport de Recherche LIRMM n° 04008

Juillet 2004

Contexte

La complexité sans cesse croissante des systèmes numériques impose le recours à des formalismes et outils facilitant tant leur conception que leur réalisation. Les réseaux de Petri (RdP) sont un formalisme particulièrement intéressant, et très utilisé, pour la spécification de systèmes complexes ; son pouvoir de description permet d'exhiber parallélisme, choix, synchronisation,.. Ils permettent également une conception itérative par raffinement (de places ou de transition) ou par composition de modèles partiels (sous-réseaux, hiérarchisation). Leur fondement théorique permet tant une analyse comportementale (vivacité, finitude, séquence répétitives, etc.) qu'une étude de performance. De multiples travaux ont témoigné de leur intérêt et de leur utilisation, aussi bien pour la conception que pour l'analyse et l'évaluation de performance [DAI].

Dès lors que l'on spécifie le comportement d'un système ayant des interactions avec son environnement, via des entrées (capteurs, signaux, etc.) et des sorties (actionneurs), le modèle est enrichi d'une interprétation traduisant ces interactions. Il s'agit de réseaux de Petri interprétés, i.e. non autonomes. Le temps est également une dimension importante, tant dans la conception que dans la réalisation de systèmes complexes ; les RdP temporisés et les RdP temporels sont deux classes de réseaux de Petri intégrant explicitement cette dimension temporelle.
En conséquence, le modèle que nous adressons dans le cadre de ces travaux est le réseau de Petri généralisé interprété temporel. La dimension temporelle permet tant la spécification de temporisations (attente, retard) que de chiens de garde (watchdog).

La phase préliminaire de spécification à l'aide de ce modèle formel permet de vérifier le respect de certaines propriétés (vivacité, finitude, invariants, etc.). Garantir que l'implantation (aspect matériel) du modèle spécifié préserve ces propriétés nécessite de concevoir une technique de synthèse automatique, ou plutôt une technique de traduction automatique dans un langage permettant la synthèse. Notre projet repose sur l'implantation dans des composants électroniques programmables (FPGA par exemple), nous avons donc retenu comme langage cible le VHDL. L'intérêt d'une traduction automatique des réseaux de Petri dans un langage d'implantation est évidente … et pratiquée déjà depuis quelques années [MAR98].

L'objectif de ce rapport est donc de présenter notre approche de traduction automatique d'un réseau de Petri généralisé interprété temporel en VHDL.

1 Réseaux de Petri, composants et VHDL

Le langage VHDL est un langage de description de structures électroniques numériques. L'utilisation d'une description VHDL a deux avantages :

- la portabilité des descriptions VHDL, c'est à dire la possibilité de cibler une description dans n'importe quel composant programmable (à condition bien sur que le composant supporte la description en terme de ressources logiques), et la possibilité d'utiliser l'outil de synthèse et de simulation que l'on veut.
- la conception de haut niveau, c'est à dire la possibilité de décrire le comportement d'une structure sous forme d'instructions et non pas d'équations logiques.

En description VHDL, on distingue deux façons de décrire une structure:

- la description comportementale : le comportement de la structure est décrit par un ensemble d'instructions séquentielles (instructions de conditions, instructions de boucle,...).
- la description structurelle : dans ce formalisme, la structure est définie par une hiérarchie de composants connectés entre eux. Le composant au sens VHDL est défini par le couple *entity*/*architecture*. L'*entity* représente la vue externe du composant, elle spécifie l'interface du composant en termes de signaux d'entrées/sorties. L'*architecture* représente la vue interne du composant, elle décrit le comportement du composant. A partir de la définition des composants de base de la structure, il est possible de décrire complètement la structure par instanciation des composants.

Différentes approches de description d'un réseau de Petri en langage VHDL ont été proposées [MAR98] :

- celles basées sur la génération des équations logiques équivalentes, en passant parfois par une traduction dans un langage intermédiaire comme le langage CONPAR par exemple [FER97].
- la traduction directe du modèle en s'appuyant sur son graphe d'état équivalent, le réseau de Petri devant souvent être sauf (i.e. similaire à une Machine à Etats Finis).
- la traduction indirecte du modèle, avec passage par un modèle intermédiaire comme le STG[1] par exemple [MAR98].
- la traduction du modèle par décomposition en blocs structurels élémentaires (séquence, convergences et divergences en ET et en OU). Cette approche exploite l'orientation composant du langage mais s'avère délicate à automatiser dès lors que la structure du graphe imbrique des blocs structurels dits élémentaires.

Notre proposition consiste également à exploiter l'orientation composant du langage en définissant 2 composants (génériques) structurels de base : un composant place et un composant transition. Néanmoins la différence repose sur l'intégration des arcs au sein du composant transition ; ce dernier intégrant les arcs amont et aval, il intègre l'interconnexion entre les composants de base et supporte donc les liens structurels entre les nœuds du modèle de réseau de Petri que sont les places et les transitions.

Décrivons le modèle de réseau de Petri utilisé afin d'identifier les entités constituantes.

[1] Signal Transition Graph

1.1 Description du modèle

Le modèle considéré est le réseau de Petri généralisé interprété temporel (Figure 1). Le lecteur intéressé trouvera une description formelle des réseaux de Petri dans [MUR89][DAV89].
Dans ce modèle, l'interprétation repose sur :

- des conditions de franchissement associées aux transitions. Tant que la condition n'est pas vraie, la transition n'est pas franchissable. Une même condition peut être associée à plusieurs transitions.
- des actions impulsionnelles (que l'on appellera aussi fonctions pour lever l'ambiguïté avec les actions continues), associées aux transitions et exécutées lors de leur tir. Ces actions peuvent être associées à plusieurs transitions, mais elles n'en restent pas moins impulsionnelles (i.e. elles ne doivent être exécutées qu'une seule fois lors du tir de la transition).
- des actions continues, associées aux places et activées tant qu'au moins l'une des places concernées est marquée (une même action pouvant être associée à plusieurs places). Les actions associées aux places doivent être « booléennes » (Active / Non_active), au sens indépendantes du nombre de jetons dans les places.

Le modèle étudié est temporel ; les transitions ont donc potentiellement une fenêtre de temps de tir associée. Cette fenêtre de temps est décrite par un intervalle [Tmin ;Tmax] : Tmin et Tmax définissant respectivement la date de tir au plus tôt et au plus tard. L'origine du temps relatif, auquel se réfèrent les bornes de l'intervalle, est lié à la date de sensibilisation de la transition concernée. Cette transition n'est franchissable qu'après Tmin et qu'avant Tmax, si la condition associée est vérifiée.

Le modèle est généralisé, ce qui sous-entend qu'il s'agit d'un réseau de Petri pondéré. La pondération est associée aux arcs. Le poids associé aux arcs entrants d'un transition définit le nombre de jetons nécessaire dans les places amont de cette transition (i.e. les nœuds « source » des arcs). Le poids associé aux arcs sortants de la transition définit quant à lui le nombre de jetons à déposer dans les places aval de cette transition (i.e. les nœuds « cible » des arcs).

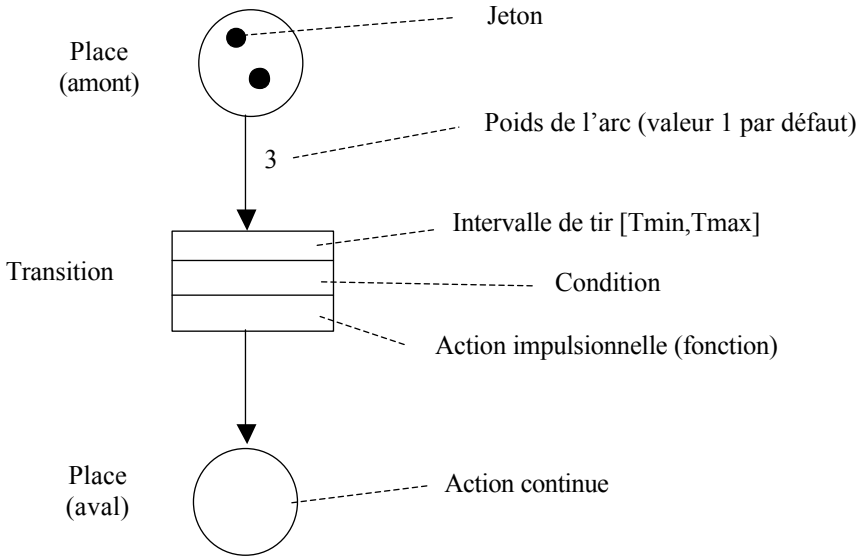
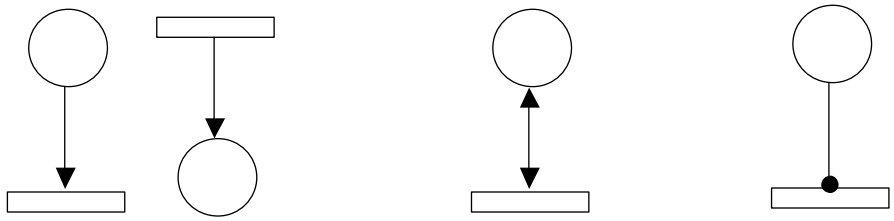


Figure 1 : éléments constitutifs d'un RdP généralisé interprété temporel

Nous distinguerons trois types d'arcs :

- les arcs « classiques » place-transition (sigle : PT) ou transition-place (sigle : TP) (Figure 2-a),
- les arcs dits de « test » (sigle : TST), qui sont des arcs bi-directionnels reliant dans les deux sens une place et une transition (Figure 2-b). Ces arcs ont nécessairement un poids unique, i.e. un poids identique qu'il s'agisse du sens place-transition ou du sens transition-place. Un arc de test agrège les deux arcs PT et TP entre la place et la transition, diminuant de fait la dimension des interfaces des composants (cf. section 1.3).
- Les arcs inhibiteurs (sigle : INH) (Figure 2-c) qui permettent de représenter le « test à zéro » du marquage d'une place, phénomène dont la modélisation fait sinon recours aux places complémentaires (induisant un graphe plus complexe). Ils restreignent les possibilités d'analyse (seulement pris en compte dans les méthodes basées sur le graphe des marquages accessibles) mais permettent une description plus concise. Aucun poids n'est associé à ce type d'arc.

Nous reviendrons sur ces arcs lors des considérations relatives à la traduction automatique du modèle. Dans l'immédiat précisons la particularité de l'arc de test et de l'arc inhibiteur.



-a- -b- -c-

Figure 2 : les différents types d'arcs

L'arc de test, tel que nous le considérons dans notre formalisme, exprime l'influence d'une place sur une transition, sans que cette dernière n'ait d'influence sur la place. En effet, si la place est vide la transition concernée n'est pas franchissable (cf. section 1.2) mais le tir de la transition ne modifie pas le marquage de cette place ; ce tir induit le retrait d'autant de jetons de cette place qu'il en dépose. De fait, nous considérerons dans l'établissement des règles d'évolution, que ce tir est sans influence sur le marquage de la place concernée. Notons immédiatement les conséquences de ce fonctionnement sur les transitions temporelles, i.e. les transitions auxquelles sont associées une fenêtre de tir. Sur une transition dite temporelle, le temps est relatif à la date de sensibilisation, i.e. à la date de marquage des places amont. Si une des places amont est démarquée puis remarquée cela ré-initialise le temps relatif, i.e. re-initialise le comptage du temps sur la transition concernée. Le fait que pour un arc de test, la place amont n'est pas démarquée puis remarquée, le comptage du temps n'est pas affecté.

L'arc inhibiteur quant à lui, est similaire à l'arc de test au sens où il exprime l'influence d'une place sur une transition sans influence réciproque. Si cette place est vide, la transition concernée est franchissable (cf. section 1.2) mais le tir de la transition ne modifie pas le marquage de la place.

Les règles d'évolution de l'état de ce graphe, i.e. de l'évolution du marquage de ce modèle, sont naturellement dépendantes de ses particularités : le type d'arc, la pondération, l'interprétation et le temps.

1.2 Règles d'évolution du modèle

L'évolution du marquage du modèle est basée sur le franchissement des transitions. Ce franchissement comprend plusieurs phases : le test de sensibilisation, le test de validation, le test de franchissement et enfin le tir effectif… l'action impulsionnelle associée (appelée fonction) est alors exécutée.

Les différentes étapes sont donc :

a- Une transition T est *sensibilisée* si et seulement si :
   - les places amont d'arcs entrant PT ou TST ont un marquage supérieur ou égal au poids de l'arc,
   - et les places amont d'arc INH ont un marquage nul.

b- Une transition temporelle T est *validée* si et seulement si :
   - Elle est sensibilisée depuis au moins T_min et au plus T_max (fenêtre de temps associée). Rappelons que le temps relatif est compté à la date de la sensibilisation.

c- Une transition T est *franchissable* si et seulement si :
   - Elle est validée et sa condition associée est vraie

d- Dès lors qu'une transition T est franchissable, son tir se compose de :
   - pour chaque arc amont PT, retrait de poids jetons dans la place amont,
   - pour les arcs TST et INH, aucune influence,
   - pour chaque arc aval TP, ajout de poids jetons dans la place aval.

e- Le tir induit également l'exécution de la fonction (au sens action impulsionnelle) associée à T.

Le fonctionnement du réseau de Petri sera **synchrone**, au sens où son implantation sera pilotée par une horloge. Ce mode de fonctionnement permet d'éviter des aléas de fonctionnement (régimes transitoires), cependant il introduit un problème car le modèle est par essence asynchrone (i.e. chaque transition est tirée indépendamment des autres). Il est donc nécessaire d'être attentif aux divergences en OU. En effet, si plusieurs transitions aval à une place sont franchissables en même temps, il faut qu'une seule des transitions soit franchie et par conséquent qu'une seule soit franchissable. On évitera ce problème en imposant à l'utilisateur d'assurer l'**exclusion mutuelle des conditions d'une divergence en OU**, à l'image d'un modèle synchrone comme le Grafcet [DAV89].

1.3 Décomposition du modèle en composants interconnectés

Le réseau de Petri que nous venons de décrire doit être décomposé en un graphe de composants interconnectés par leurs interfaces (ports) ; cette approche veut tirer profit de l' « orientation composant » du VHDL, composants interconnectables par leurs interfaces (entrées/sorties du composant spécifiées par l'*entity*). Nous allons donc exposer la décomposition proposée dans cette optique. Il ne s'agit pas encore rigoureusement de composants VHDL mais simplement de composants « conceptuels », i.e. décomposition à partir de laquelle seront définis les composants VHDL.

1.3.1 Les principes de la décomposition

Deux composants fondamentaux s'imposent : la place et la transition. Leur interconnexion sur le modèle est traduite par les arcs orientés. Nous considérons ces arcs comme le support des flux d'entrées/sorties, i.e. flux de jetons, entre les composants ; ce support exprimant en terme

de propagation, i.e. de flux de jetons, tant des contraintes (la pondération des arcs entrants d'une transition) que des conséquences (la pondération des arcs sortants d'une transition).

Les flux de jetons résultants (entrants et sortants) sont alors issus des interconnexions de la place avec ses transitions aval et amont. Le « pivot » est la transition. Chaque transition amont (resp. aval) apporte (resp. retire) un flux de jetons ; l'ensemble des flux entrants (resp. sortants) d'une place est alors exprimé par un vecteur d'entrée (resp. de sortie) dont la dimension correspond au nombre de transitions amont (resp. aval) de la place. Chaque transition apportant (retirant) potentiellement un nombre différent de jetons, directement spécifié par le poids de l'arc correspondant, il est nécessaire d'associer un poids à chaque flux.
Le type d'arc est également important puisqu'il influe sur la propagation du flux ; un arc inhibiteur et un arc de test ne spécifient pas les mêmes contraintes et reposent sur des règles de propagation différentes (cf. section 1.2).

Le modèle retenu est le réseau de Petri interprété ; cette interprétation décrit l'interaction entre le modèle et son environnement. Cette interprétation associée au modèle, à savoir les conditions, les fonctions et les actions, est décrite en dehors des composants de base place et transition. En effet, ces entités peuvent être associées à plusieurs composants, i.e. plusieurs transitions peuvent avoir la même condition associée ou la même fonction associée, et plusieurs places peuvent déclencher une même action. Par conséquent, ces entités (condition, fonction, action) sont considérées comme des blocs fonctionnels « liés » aux composants et non pas contenus dans les composants. Cela permet une « mutualisation » de leur description ; par exemple toute condition associée à plusieurs transitions ne sera décrite qu'une seule fois et ne sera évaluée qu'une seule fois. Selon cette orientation, nous considérons même qu'une transition est un produit de termes dont chaque terme est potentiellement utilisable dans plusieurs conditions ; nous minimisons ainsi l'implantation du modèle dans la mesure où chaque terme ne sera implanté (et évalué) qu'une seule fois. La description VHDL exposera clairement cette séparation.

1.3.2 La transition

Le composant transition est schématiquement représenté sur la Figure 3.

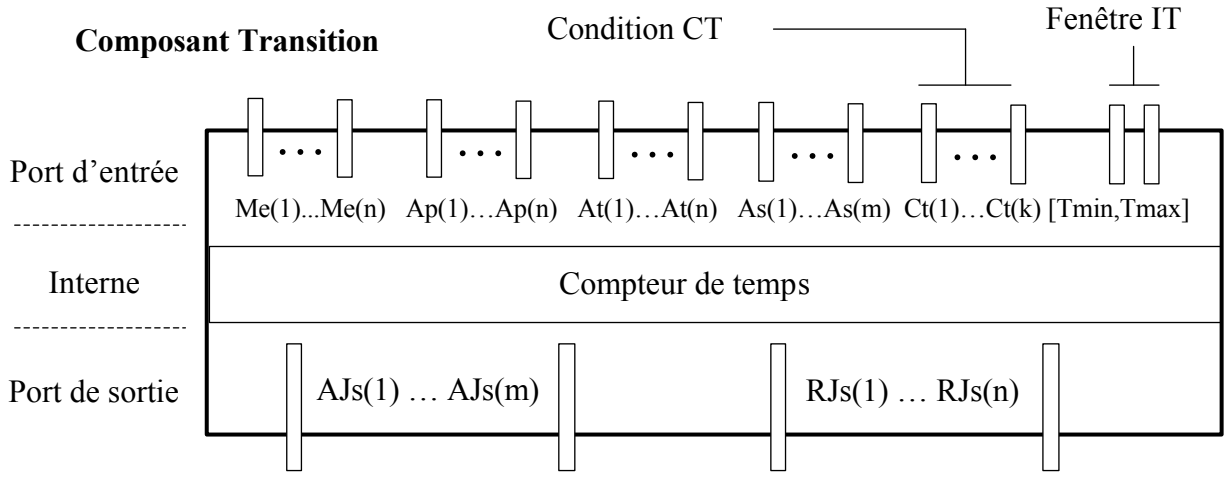


Figure 3 : représentation schématique du composant transition

Il est structuré autour de :

- Un port d'entrée constitué :
  - d'un vecteur *Me* de dimension *n*, avec *n* étant le nombre d'arcs entrants. Ce vecteur supporte donc la connexion avec les places amont. Chaque élément du vecteur *Me(i)* désigne la source de jeton (flux entrant) de l'arc considéré, i.e. il apporte directement le marquage de la place amont désignée.
  - un vecteur *Ap* de dimension *n*, avec *n* étant le nombre d'arcs entrants. Ce vecteur exprime les contraintes sur les flux entrants. Chaque élément du vecteur *Ap(i)* exprime le poids associé à l'arc entrant désigné, i.e. il exprime directement la contrainte sur le marquage de la place amont désignée par *Me(i)*.
  - un vecteur *At* de dimension *n*, avec *n* étant le nombre d'arcs entrants. Ce vecteur exprime le type des arcs entrants. Chaque élément du vecteur *At(i)* exprime le type associé à l'arc entrant désigné, i.e. il permet d'exprimer la règle de propagation sur cet arc.
  - un vecteur *As* de dimension *m*, avec *m* étant le nombre d'arcs sortants. Ce vecteur supporte donc la connexion avec les places aval. Chaque élément du vecteur *As(i)* exprime directement le poids de l'arc sortant désigné, duquel sera déduit le nombre de jetons à déposer dans la place cible. Notons que l'arc étant nécessairement de type TP, ce type n'apparaît pas.
  - d'un vecteur *Ct* de dimension *k*, avec *k* étant le nombre de termes de la condition associée à la transition. La condition peut en effet être décrite comme un produit de termes ; la section 2.4 expose plus en détail ce point.
  - un intervalle de temps IT (cf. section 1.1), décrit par deux bornes T_{min} et T_{max}.
- Une zone « interne » comprenant :
  - un compteur de temps, le temps est un temps relatif, i.e. il s'agit du temps écoulé depuis la sensibilisation de la transition.
- Un port de sortie constitué de deux vecteurs :
  - Un premier vecteur *AJs* (AjoutJetons) de dimension *m*, avec *m* étant le nombre d'arcs sortants. Ce vecteur supporte donc la connexion avec les places aval, en terme de jeton à ajouter (déposer) à ces places. Chaque élément du vecteur *AJs(i)* transporte le flux de jeton sortant (produit) correspondant à l'arc désigné, i.e. il exprime directement le nombre de jetons à déposer dans la place cible désignée.
  - Un second vecteur *RJs* (RetraitJetons) de dimension *n*, avec *n* étant le nombre d'arcs entrants. Ce vecteur supporte donc la connexion avec les places amont, en terme de jetons à retirer de ces places. Chaque élément du vecteur *RJs(i)* transporte le flux de jeton entrant (consommé) correspondant à l'arc désigné, i.e. il exprime directement le nombre de jetons à retirer de la place source désignée.

Ce composant est par ailleurs potentiellement en relation avec :

- une (ou plusieurs) fonction(s) FT (cf. section 2.5)

Le composant transition supporte aussi les règles d'évolution du modèle au sens de l'évaluation et du tir de cette transition. Nous aborderons plus en détail ce comportement du composant ultérieurement.

1.3.3 La place

La place est également traduite en un composant sur lequel s'exerce la propagation de flux (de jetons) induite par le tir des transitions auxquelles il est connecté. La propagation de flux résulte par un flux de jeton à ajouter et/ou à retirer.

Le composant place, schématiquement représenté sur la Figure 4, est structuré autour de :

- Un port d'entrée comprenant :
  - le flux entrant de jetons, en terme de jetons à ajouter. Il est décrit par un vecteur *AJe* (AjoutJetons) de dimension *n*, avec *n* étant le nombre d'arcs entrants. Ce vecteur supporte donc la connexion avec les transitions amont. Chaque élément du vecteur *AJe(i)* apporte le flux de jeton entrant correspondant à l'arc désigné, i.e. il apporte directement les jetons issus du tir de la transition amont désignée par l'arc.
  - le flux sortant de jetons, en terme de jetons à retirer. Il est décrit par un vecteur *RJe* (RetraitJetons) de dimension *k*, avec *k* étant le nombre d'arcs sortants. Ce vecteur supporte donc la « retro-connexion » avec les transitions aval. Chaque élément du vecteur *RJe(i)* retire le flux de jeton sortant correspondant à l'arc désigné, i.e. il retire directement les jetons issus du tir de la transition aval désignée par l'arc.
- Une zone « interne » comprenant son état (le marquage), décrit en terme de jetons présents dans la place.
- Un port de sortie via lequel il « extériorise » son état courant, i.e. port qui supporte donc la « connexion-avant » avec les transitions aval. Cela est décrit par un entier *Ms*, image du marquage (de dimension 1 car indépendant du nombre d'arcs sortants).

Ce composant est par ailleurs potentiellement en relation avec :

- une (ou plusieurs) action(s) AC (cf. section 2.6).

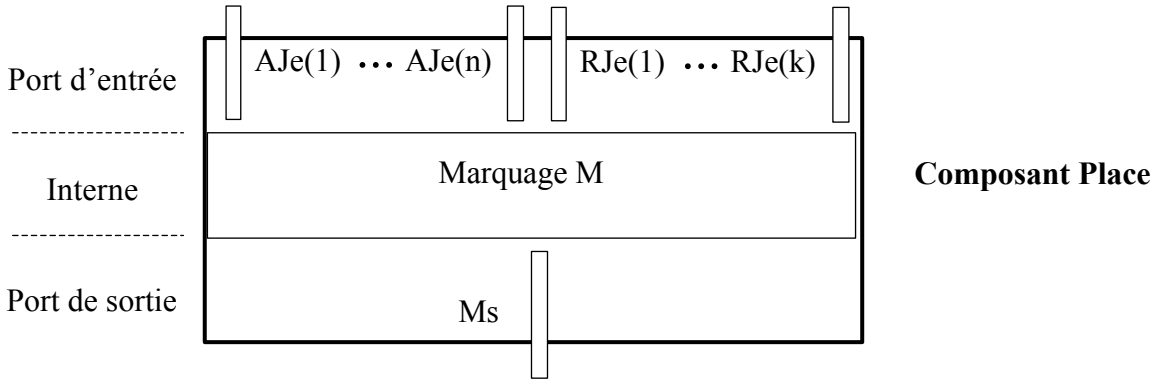


Figure 4 : représentation schématique du composant place

A l'instar du composant transition, le composant place supporte les règles d'actualisation de son propre état.

1.3.4 Illustration

La Figure 5 illustre, sur un extrait simple de réseau de Petri (Figure 5-a), l'interconnexion résultante entre ces composants (Figure 5-b) ; seuls les flux de jetons sont représentés, i.e. l'interconnexion de ports.

NB : la représentation est simplifiée. Par exemple les poids et types d'arcs, les conditions, etc. ne sont pas mentionnés. Une illustration plus précise sera donnée dans la description VHDL.

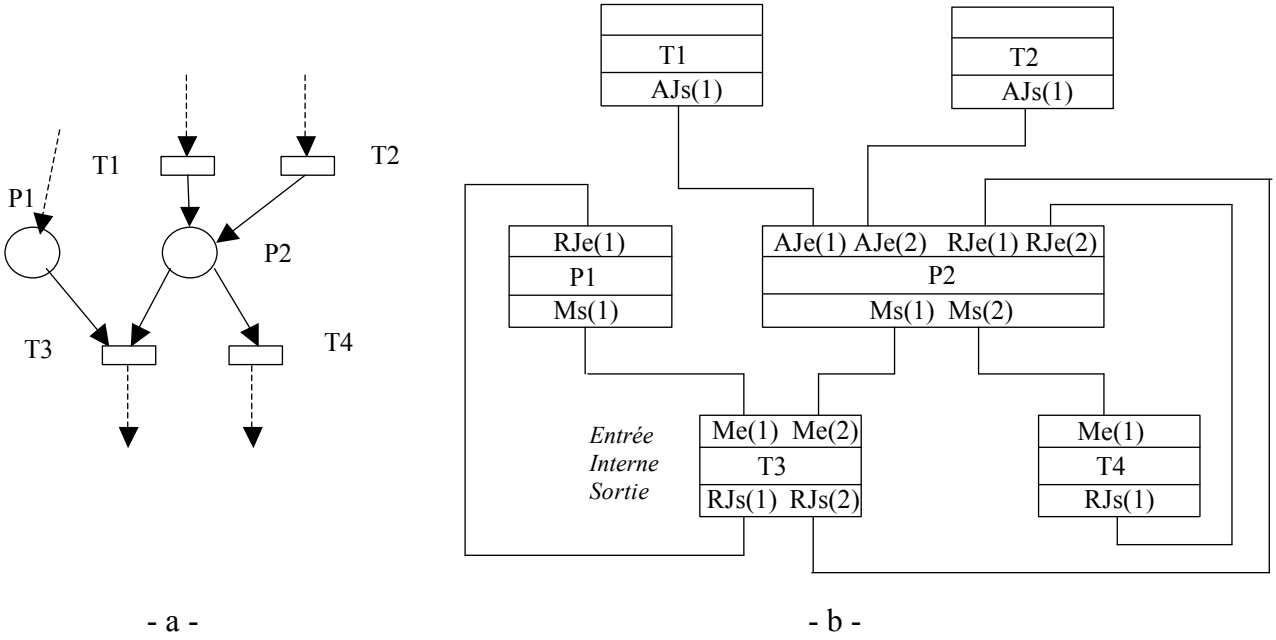


Figure 5 : illustration de l'interconnexion de ces composants « conceptuels »

Revenons sur le partitionnement des règles d'évolution du modèle, entre le composant place et le composant transition.

1.4 Partitionnement des règles d'évolution du modèle

Tel que nous venons de le décrire, les règles d'évolution sont réparties entre les deux types de composant afin justement de respecter l'approche composant, i.e. que chacun assure l'évolution de son propre état. Ce partitionnement attribue au composant transition la responsabilité de l'évaluation de la « franchissabilité » et du tir des transitions, et la responsabilité de l'évolution de l'état du graphe (marquage de la place) au composant place. Le problème qui se pose alors est celui de la synchronisation des différentes phases constituant l'évolution d'un modèle réseau de Petri généralisé interprété temporel (cf. section 1.2). Toutes ces phases ne peuvent avoir lieu simultanément ; notre implantation étant synchrone, cela signifie que toutes ces phases ne peuvent être directement déclenchées par simple occurrence de l'horloge (sur front d'horloge).

En effet, ces phases doivent être séquencées :

- Sur un état stable (i.e. une distribution stable des jetons dans les places), on évalue la possibilité de franchissement des transitions… même si une ou plusieurs transitions venaient à être tirées cela n'induit aucune évolution effective du marquage (i.e. de l'état des places) mais le positionnement de l'ajout et du retrait de jetons conséquent au tir. Rappelons que toutes transitions en conflit (divergence

en OU) doivent être mutuellement exclusives, i.e. seule l'une d'entre elles doit être franchissable à l'instant t.

- Suite à cette évaluation du changement d'état du graphe, l'évolution du marquage peut avoir lieu … il s'agit d'appliquer l'ajout et le retrait de jetons déterminé à la phase précédente. L'évaluation du franchissement des transitions est alors interdite tant que l'état n'est pas à nouveau stable, i.e. tant que le marquage des places n'a pas été actualisé.
- L'interaction entre le modèle et son environnement ne doit pas être omise. Cette interaction comprend :
  - l'évaluation des conditions associées aux transitions (qui peut nécessiter la lecture de la valeur d'un capteur par exemple). Elle doit être effectuée avant l'évaluation du franchissement des transitions.
  - l'exécution d'actions sur l'environnement, via les actions impulsionnelles (appelées fonctions) associées aux transitions, et les actions continues (appelées actions) associées aux places (qui peut résulter par l'enclenchement d'un actionneur par exemple). Les fonctions doivent être exécutées suite au tir des transitions, et les actions après l'actualisation du marquage.

Les différentes phases sont alors séquencées comme suit :

(1) actualisation du marquage (génération de l'état stable) et évaluation des conditions,
(2) actualisation des actions et évaluation du franchissement des transitions,
(3) exécution des fonctions.

L'implantation étant synchrone nous utilisons une horloge (signal clk), et nous introduisons un signal (signal selec) qui, dérivé de cette horloge sans être synchronisé (diviseur de fréquence par deux), permet d'établir une distinction entre ces phases (cf. Figure 6). L'évolution complète du modèle s'effectue par conséquent en deux cycles d'horloge ; la fréquence maximale du circuit résultant (celle permettant une exécution cohérente est différentes phases) est donc à diviser par deux.

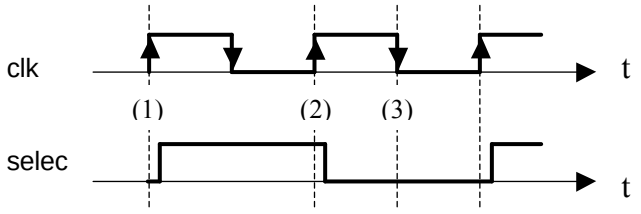


Figure 6 : les différentes phases de l'évolution du modèle

2 La traduction en VHDL

Notre traduction en VHDL va naturellement s'appuyer sur la décomposition en deux composants génériques structurels de base qui sont donc un composant place et un composant transition (ce dernier intégrant les arcs amont et aval).

2.1 Etude du composant place

Dans l'esprit du composant place conceptuel évoqué section 1.3, la représentation schématique du composant VHDL place est donnée sur la Figure 7.

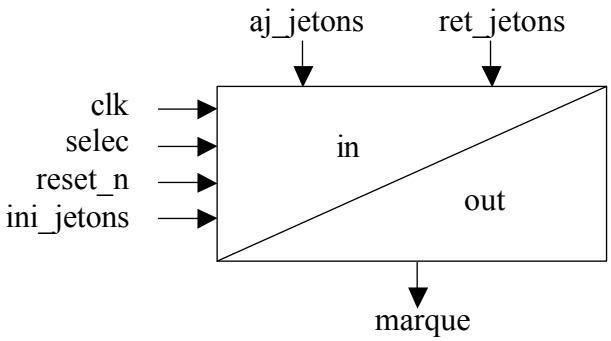


Figure 7 : représentation schématique du composant VHDL place

Avec :

- Entrées
  - *clk* l'entrée d'horloge
  - *selec* le signal permettant le séquencement des phases d'évolution du modèle
  - *reset_n* le signal d'initialisation
  - *ini_jetons* le nombre initial de jetons dans la place (marquage initial)
  - *aj_jetons* (vecteur aj_jetons) le flux de jetons à ajouter suite au tir des transitions amont
  - *ret_jetons* (vecteur ret_jetons) le flux de jetons à retirer suite au tir des transitions aval
- Interne
  - *marquage* le marquage de la place
- Sorties
  - *marque* le marquage propagé aux transitions aval

Le composant place se comporte en fait comme un simple additionneur/soustracteur synchrone (signal d'horloge clk) et commandé par le signal selec (cf. section 1.4). A la sortie marque est affecté le résultat de marquage + aj_jetons – ret_jetons.

La description VHDL de l'interface du composant générique place[2] est donc :

```
entity place is
generic (       nb_entrees      : integer := 1;
                nb_sorties      : integer := 1;
                max_poids_ent   : integer range 0 to 255 := 1;
                max_poids_sort: integer range 0 to 255 := 1;
                marque_max_p : integer:= marque_max_net);
port (  clk             : in std_logic;
        selec           : in std_logic;
        reset_n         : in std_logic;
        ini_jetons      : in integer range 0 to marque_max_p;
        aj_jetons       : in array (nb_entrees-1 downto 0) of integer range 0 to max_poids_ent;
        ret_jetons      : in array (nb_sorties-1 downto 0) of integer range 0 to max_poids_sort;
        marque          : out integer range 0 to marque_max_p);
end place;
```

[2] entity sur laquelle s'appuie le component place

Notons que :

- la dimension des vecteurs d'interconnexion est dynamiquement fixée en fonction de la structure du réseau de Petri, et par conséquent « optimisée ». La partie generic de l'interface comprend entre autres les paramètres :
  - nb_entrees et nb_sorties, qui indiquent respectivement le nombre d'arcs entrants et sortants (à 1 par défaut),
  - max_poids_ent et max_poids_sort, qui indiquent respectivement le flux maximal de jetons entrant (pour chaque arc entrant) et de jetons sortant (pour chaque arc sortant).
- la limite sur le marquage d'un place marque_max_p est également un paramètre du composant (fixé à marque_max_net par défaut). Dans un but d'optimisation, de dimensionnement de ces signaux au nombre de bits strictement nécessaire pour leur codage, la limite sur le marquage de chaque place peut être déterminée à partir du graphe des marquages accessibles. A défaut, il est possible de limiter le marquage de toutes les places à la borne maximale du réseau lui-même, nécessairement k-borné (constante marque_max_net).

La description VHDL du comportement du composant générique place est :

```
architecture a_place of place is
signal marquage : integer range 0 to marque_max_p;
begin
  process(clk, reset_n)
        variable sum_aj : integer range 0 to marque_max_p;
        variable sum_ret : integer range 0 to marque_max_p;
  begin
        if ( reset_n = '0' ) then marquage <= ini_jetons;  -- initialisation du marquage
        elsif rising_edge(clk) then
                if selec = '0' then -- phase d'actualisation du marquage
                        sum_aj := 0;
                        sum_ret := 0;
                        for i in 0 to nb_entrees-1 loop -- jetons à ajouter
                                sum_aj := sum_aj + aj_jetons(i);
                        end loop;  -- i
                        for j in 0 to nb_sorties-1 loop -- jetons à retirer
                                sum_ret := sum_ret + ret_jetons(j);
                        end loop;  -- j
                        marquage <= marquage + sum_aj - sum_ret;
                end if;
        end if;
  end process;
  marque <= marquage; -- positionnement du marquage (propagé)
end a_place;
```

L'actualisation du marquage est effectuée dans la phase selec = 0.
Le signal marquage permet de modifier le marquage à l'intérieur même de la place avant de l'affecter à la sortie marque (i.e. le marquage est positionné à la fin du process). C'est cette sortie marque qui est reliée à une entrée de chaque transition aval.

2.2 Etude du composant transition

Dans l'esprit du composant transition conceptuel évoqué section 1.3, la représentation schématique du composant VHDL transition est donnée sur la Figure 8.

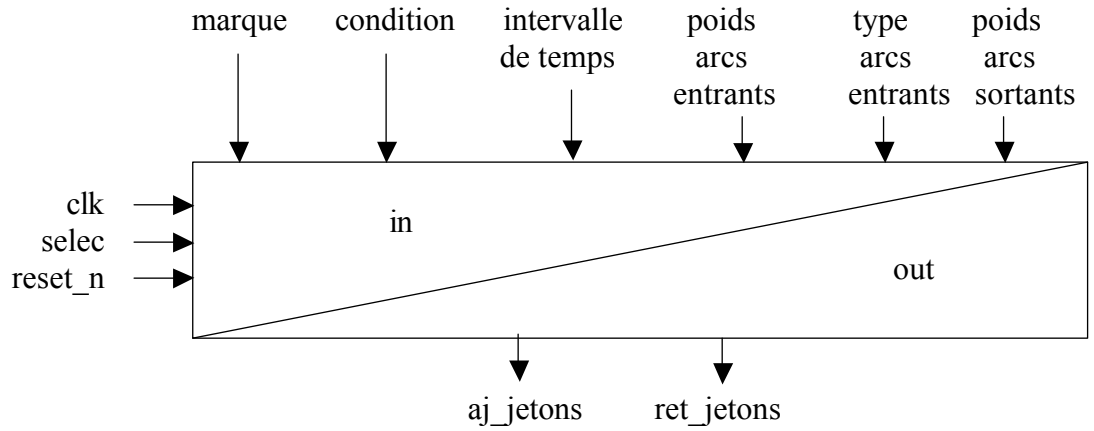


Figure 8 : représentation schématique du composant VHDL transition

Avec :

- Entrées
  - *clk* l'entrée d'horloge
  - *selec* le signal permettant le séquencement des phases d'évolution du modèle
  - *reset_n* le signal d'initialisation
  - *marque* (vecteur Me), le marquage des places amont

  condition (vecteur c_t), les termes de la condition associée
  - *intervalle de temps IT* (2 entrées : t_min,t_max), les bornes de la fenêtre de tir
  - *poids des arcs entrants* (vecteur Ap), le poids des arcs entrants
  - *type* (vecteur At), le type des arcs entrants
  - *poids des arcs entrants* (vecteur As), le poids des arcs sortants
- Interne
  - *Compteur de temps*, le temps est un temps relatif, i.e. il s'agit du temps écoulé depuis la sensibilisation de la transition. Il est compté en unité de temps, l'unité étant dépendante de la fréquence d'horloge ; le comptage évolue tous les deux tops d'horloge puisque le cycle d'évolution s'effectue en deux top d'horloge (cf. section 1.4). La fenêtre de tir doit donc être spécifiée en nombre de top, avec la conversion suivante pour revenir au temps effectif : une unité correspond à deux périodes d'horloge.
- Sorties
  - *aj_jetons* (vecteur ret_amont, RJs) le flux de jetons à ajouter aux places aval, suite au tir
  - *ret_jetons* (vecteur aj_aval, AJs) le flux de jetons à retirer aux places amont, suite au tir

La description VHDL de l'interface du composant générique transition temporelle[3] est :

```
entity transition_temp is
  generic (       nb_bits_temps  : integer range 0 to 255 := 16;
                  nb_entrees     : integer range 0 to 255 := 1;
                  nb_sorties     : integer range 0 to 255 := 1;
                  nb_termes      : integer range 0 to 255 := 1;
                  max_poids_ent  : integer range 0 to 255 := 1;
                  max_poids_sort : integer range 0 to 255 := 1);
  port (  clk                    : in std_logic;
```

[3] entity sur laquelle s'appuie le component transition_temporelle

```
        selec           : in std_logic;
        reset_n         : in std_logic;
        c_t             : in std_logic_vector (nb_termes-1 downto 0);
        t_min,t_max     : in integer range 0 to (2**nb_bits_temps-1);
        Ap              : in array (nb_entrees-1 downto 0) of integer range 0 to max_poids_ent;
        At              : in vec_At(nb_entrees-1 downto 0);
        Me              : in array (nb_entrees-1 downto 0) of integer range 0 to marque_max_net;
        As              : in array (nb_sorties-1 downto 0) of integer range 0 to max_poids_sort;
        ret_amont       : out array (nb_entrees-1 downto 0) of integer range 0 to max_poids_ent;
        aj_aval         : out array (nb_sorties-1 downto 0) of integer range 0 to max_poids_sort);
end transition_temp;
```

avec le type vec_At défini par :

```
type vec_At is array (natural range <>) of std_logic_vector(1 downto 0);
```

dont les combinaisons codant le type d'arc sont :

- "00" arc de type PT
- "01" arc de type TST
- "10" arc de type INH
- "11" arc de type TP (inutilisé car implicite)

Notons que :

- comme pour la place, la dimension des vecteurs d'interconnexion est dynamiquement fixée en fonction de la structure du réseau de Petri, et par conséquent « optimisée ». La partie generic de l'interface comprend entre autres les paramètres :
  - nb_entrees et nb_sorties, qui correspondent respectivement au nombre d'arcs entrants et sortants (à 1 par défaut),
  - max_poids_ent et max_poids_sort, qui indiquent respectivement le flux maximal de jetons entrants (poids maximal des arcs entrants) et de jetons sortants (poids maximal des arcs sortants).
- la limite sur le temps est fixée en terme de dimension du compteur (nombre de bits pour le compteur et les bornes de l'intervalle de temps). La partie generic de l'interface comprend le paramètre nb_bits_temps, qui permet d'ajuster ce dimensionnement pour chaque transition (fixé à 16 par défaut).
- les bornes de l'intervalle de temps (fenêtre de tir) sont membres de l'interface du composant, permettant ainsi soit de les fixer comme « constantes » (i.e. valeurs fixées dès la traduction en VHDL) soit de les modifier dynamiquement (i.e. valeurs pouvant être modifiées d'un tir à l'autre[4]).
- la condition associée à la transition est décrite sous la forme d'un produit de terme (cf. section 2.4). La partie generic de l'interface comprend le paramètre nb_termes, qui correspond au nombre de termes constituant la condition (à 1 par défaut). Tous les termes sont combinés par un ET logique.

La description VHDL du comportement du composant générique transition temporelle est :

```
architecture a_transition_temp of transition_temp is

begin
  process(clk,reset_n)
    variable sensibilisation : std_logic;
    variable condition : std_logic;
```

[4] valeurs fixées par le biais d'une fonction

```
    variable cpt : integer range 0 to (2**nb_bits_temps-1);
    begin
        if reset_n = '0' then
            cpt := 0;
            for i in 0 to nb_entrees-1 loop -- reset du flux de jetons consommé
                ret_amont(i) <= 0;
            end loop;  -- i
            for j in 0 to nb_sorties-1 loop -- reset du flux de jetons produit
                aj_aval(j) <= 0;
            end loop;  -- j
        elsif rising_edge(clk) then
            if selec = '1' then -- phase d'évaluation du franchissement de la transition
                for i in 0 to nb_entrees-1 loop -- reset du flux de jetons consommé
                    ret_amont(i) <= 0;
                end loop;  -- i
                for j in 0 to nb_sorties-1 loop -- reset du flux de jetons produit
                    aj_aval(j) <= 0;
                end loop;  -- j
                sensibilisation := '1';
                for k in 0 to nb_entrees-1 loop -- test sensibilisation
                    if (At(k) = "00" or At(k) = "01") then -- si arc PT ou TST
                        if (Me(k) < Ap(k)) then sensibilisation := '0';
                        end if;
                    else
                        if At(k) = "10" then -- si arc INH
                            if Me(k) /= 0 then sensibilisation := '0';
                            end if;
                        end if;
                    end if;
                end loop;  -- k
                if (sensibilisation = '1') then cpt := cpt + 1; -- incrementation compteur temps
                else cpt := 0; -- reset compteur temps
                end if;
                if (cpt < t_min -1 or cpt > t_max -1) then sensibilisation := '0';
                end if;
                if (sensibilisation = '1') then
                    condition:='1';
                    for g in 0 to nb_termes-1 loop -- test condition
                        condition := condition and c_t(g);
                    end loop;  -- g
                    if (condition = '1') then
                        for l in 0 to nb_entrees-1 loop -- flux de jetons consommé
                            if At(l)="00" then ret_amont(l) <= Ap(l);
                            end if;
                        end loop;  -- l
                        for m in 0 to nb_sorties-1 loop -- flux de jetons produit
                            aj_aval(m) <= As(m);
                        end loop;  -- m
                    end if;
                end if;
            end if;
        end if;
    end process;
end a_transition_temp;
```

L'évaluation du franchissement de la transition est effectuée dans la phase selec = 1.
La variable sensibilisation est utilisée pour le test de sensibilisation de la transition, qui consiste à vérifier si les places amont sont suffisamment marquées. Elle est alors exploitée pour déterminer si le temps local, i.e. le compteur de temps de la transition, doit être incrémenté

(transition sensibilisée) ou mis à zéro (transition non sensibilisée). Cette même variable sensibilisation est alors utilisée pour exprimer si la transition reste validée après la prise en compte de la fenêtre de tir, i.e. d'une part elle est sensibilisée et d'autre part le temps local est dans la fenêtre de tir autorisée. Enfin, si elle est validée et que sa condition associée est vraie, la transition est alors franchissable ; les flux de jetons amont (à retirer) et aval (à déposer) sont alors positionnés.
On peut constater dans la description du composant que seuls les arcs PT et TP ont une influence sur le marquage des places amont et aval à la transition.

Pour des raisons d'optimisation nous avons également défini le composant transition non temporelle ; une transition temporelle est effectivement plus complexe (et plus coûteuse en terme d'implantation) puisqu'elle intègre un compteur de temps. Notre traducteur automatique utilisera donc l'une ou l'autre selon le type de transition.

La description VHDL de l'interface du composant générique transition non temporelle[5] est :

```
entity transition is
generic (       nb_entrees      : integer := 1;
                nb_sorties      : integer := 1 ;
                nb_termes       : integer range 0 to 255 := 1;
                max_poids_ent   : integer range 0 to 255 := 1;
                max_poids_sort: integer range 0 to 255 := 1);
port (  clk             : in std_logic;
        selec           : in std_logic;
        reset_n         : in std_logic;
        c_t             : in std_logic_vector (nb_termes-1 downto 0);
        Ap              : in array (nb_entrees-1 downto 0) of integer range 0 to max_poids_ent;
        Me              : in array (nb_entrees-1 downto 0) of integer range 0 to marque_max_net;
        As              : in array (nb_sorties-1 downto 0) of integer range 0 to max_poids_sort;
        At              : in vec_At(nb_entrees-1 downto 0);
        ret_amont       : out array (nb_entrees-1 downto 0) of integer range 0 to max_poids_ent;
        aj_aval         : out array (nb_sorties-1 downto 0) of integer range 0 to max_poids_sort);
end transition;
```

La description VHDL du comportement du composant générique transition non temporelle est :

```
architecture a_transition of transition is
begin
  process(clk, reset_n)
    variable sensibilisation : std_logic;
    variable condition : std_logic;
    begin
        if reset_n = '0' then
            for i in 0 to nb_entrees-1 loop -- reset du flux de jetons consommé
                ret_amont(i) <= 0;
            end loop;  -- i
            for j in 0 to nb_sorties-1 loop -- reset du flux de jetons produit
                aj_aval(j) <= 0;
            end loop;  -- j
        elsif rising_edge(clk) then
            if selec = '1' then  -- phase d'évaluation du franchissement de la transition
                for i in 0 to nb_entrees-1 loop  -- reset du flux de jetons consommé
                    ret_amont(i) <= 0;
```

[5] entity sur laquelle s'appuie le component transition

```
                        end loop;  -- i
                        for j in 0 to nb_sorties-1 loop  -- reset du flux de jetons produit
                                aj_aval(j) <= 0;
                        end loop;  -- j
                        sensibilisation := '1';
                        for k in 0 to nb_entrees-1 loop  -- test sensibilisation
                                if (At(k) = "00" or At(k) = "01") then -- si arc PT ou TST
                                        if (Me(k) < Ap(k)) then sensibilisation := '0';
                                        end if;
                                else
                                if At(k) = "10" then -- si arc INH
                                                if Me(k) /= 0 then sensibilisation := '0';
                                                end if;
                                        end if;
                                end if;
                        end loop;  -- k
                        if (sensibilisation = '1') then
                                condition:='1';
                                for g in 0 to nb_termes-1 loop  -- test condition
                                        condition := condition and c_t(g);
                                end loop;  -- g
                                if (condition = '1') then
                                        for l in 0 to nb_entrees-1 loop -- flux de jetons consommé
                                                if At(l)="00" then ret_amont(l) <= Ap(l);
                                                end if;
                                        end loop;  -- l
                                        for m in 0 to nb_sorties-1 loop -- flux de jetons produit
                                                aj_aval(m) <= As(m);
                                        end loop;  -- m
                                end if;
                        end if;
                end if;
        end if;
   end process;
end a_transition;
```

2.3 Instanciation et interconnexion des éléments de base

Pour illustrer l'instanciation et l'interconnexion des instances, prenons l'exemple simple d'une convergence en ET décrite sur la Figure 9. Pour chaque élément de base (places et transitions), une instance du composant correspondant est générée ; les instances doivent ensuite être mappées et interconnectées entre elles. Elles le sont par l'intermédiaire de signaux, automatiquement générés, nommés et numérotés ; un exemple est donné sur la Figure 9.

Le composant VHDL décrivant la place P2 est une instance du composant place :

```
Component place
generic ( nb_entrees : integer := 1;
          nb_sorties : integer := 1;
          max_poids_ent : integer range 0 to 255 := 1;
          max_poids_sort : integer range 0 to 255 := 1;
          marque_max_p : integer:= marque_max_net);
port (    clk : in std_logic;
          selec : in std_logic;
          reset_n : in std_logic;
          ini_jetons : in integer range 0 to marque_max_p;
          aj_jetons : in array (nb_entrees-1 downto 0) of
integer range 0 to max_poids_ent;
```

```
P2 : place
 generic map ( nb_entrees => 2,
               nb_sorties => 2,
               max_poids_ent => 4,
               max_poids_sort => 3,
        marque_max_p => marque_max_net)
 port map(
          clk          =>clk,
          selec        =>s_selec,
          reset_n      =>reset_n,
          ini_jetons   =>s_ini_P2,
          aj_jetons(0) => s_aj_atp_12,
```

ret_jetons : in array (nb_sorties-1 downto 0) of integer range 0 to max_poids_sort; marque : out integer range 0 to marque_max_p); end component;	aj_jetons(1) => s_aj_atp_7, ret_jetons(0) => s_ret_apt_6, ret_jetons(1) => s_ret_apt_9, marque =>s_marque_P2);

Tableau 1 : Instanciation de la place P2

component transition
generic (nb_entrees : integer := 1;
nb_sorties : integer := 1 ;
nb_termes : integer range 0 to 255 := 1;
max_poids_ent : integer range 0 to 255 := 1;
max_poids_sort : integer range 0 to 255 := 1);
port (clk : in std_logic;
selec : in std_logic;
reset_n : in std_logic;
c_t : in std_logic_vector (nb_termes-1 downto 0);
Ap : in array (nb_entrees-1 downto 0) of integer range 0 to max_poids_ent;
Me : in array (nb_entrees-1 downto 0) of integer range 0 to marque_max_net;
As : in array (nb_sorties-1 downto 0) of integer range 0 to max_poids_sort;
At : in vec_At(nb_entrees-1 downto 0);
ret_amont : out array (nb_entrees-1 downto 0) of integer range 0 to max_poids_ent;
aj_aval : out array (nb_sorties-1 downto 0) of integer range 0 to max_poids_sort);
end component;
	T3 : transition
	generic map (
	nb_entrees => 2,
	nb_sorties => 1,
	nb_termes => 1,
	max_poids_ent => 3,
	max_poids_sort => 2)
	port map(
	clk =>clk,
	selec =>s_selec,
	c_t =>cond_vraie,
	reset_n =>reset_n,
	Me(0) => s_marque_P1,
	Me(1) => s_marque_P2,
	Ap(0) => s_Ap_apt_4,
	Ap(1) => s_Ap_apt_6,
	At(0) => s_At_apt_4,
	At(1) => s_At_apt_6,
	ret_amont(0)=> s_ret_apt_4,
	ret_amont(1)=> s_ret_apt_6,
	As(0) => s_As_atp_14,
	aj_aval(0) => s_aj_atp_14);

Tableau 2 : Instanciation de la transition T3 (non temporelle)

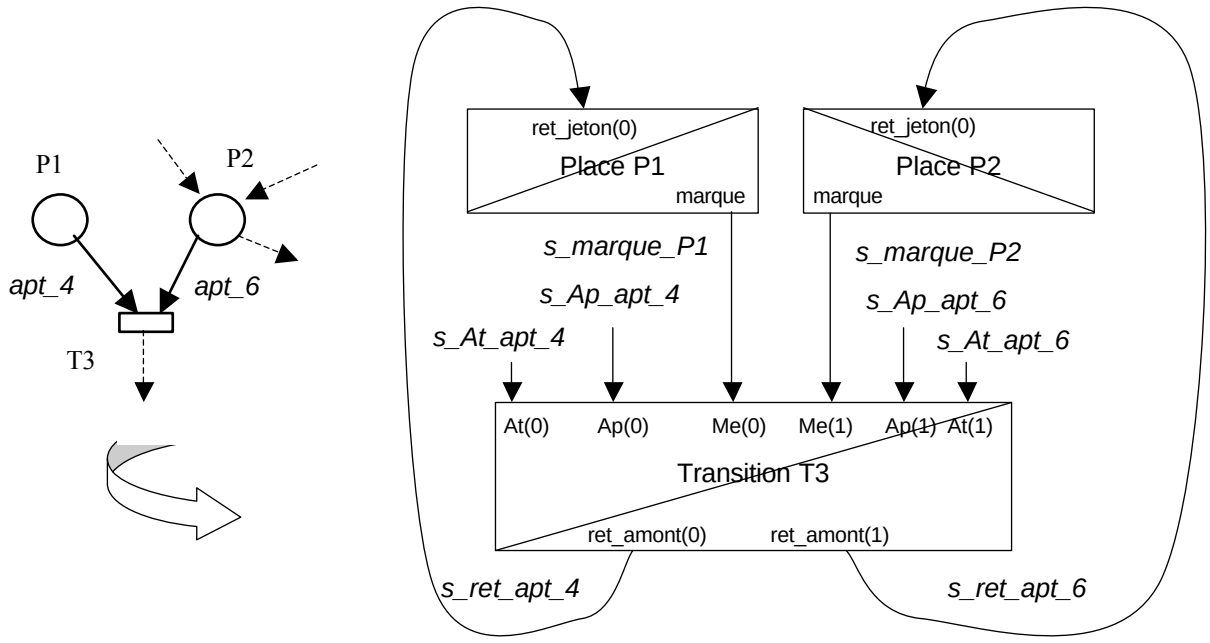


Figure 9 : signaux interconnectant les places P1 et P2 à la transition T3

Tous les signaux indiqués sur la Figure 9 sont automatiquement générés, dont :

- *s_marque_[nomplace]*, avec pour exemple *s_marque_P1* le signal associé au marquage de la place P1,
- *s_ini_[nomplace]*, avec pour exemple *s_ini_P1* le signal associé à l'initialisation de la place P1,
- *s_At_[nomarc]_[numéroarc]*, avec pour exemple *s_At_apt_4* le signal indiquant le type de l'arc (*s_At*), concernant l'arc *apt_4* reliant la place P1 à la transition T3. Ce nom est composé à partir de son type (*apt* : arc place transition) et d'une numérotation automatique (*4*).
- *s_Ap_[nomarc]_[numéroarc]*, avec pour exemple *s_Ap_apt_4* le signal indiquant le poids de l'arc (*s_Ap*), concernant l'arc *apt_4* reliant la place P1 à la transition T3.
- *s_ret_[nomarc]_[numéroarc]*, avec pour exemple *s_ret_apt_4* le signal indiquant le nombre de jetons retirés de la place amont de la transition (*s_ret*), place désignée par l'arc concerné (ici l'arc *apt_4* reliant la place P1 à la transition T3).

Tous les signaux sont automatiquement générés lors de la traduction, selon un principe similaire ; ceci est totalement transparent pour l'utilisateur.

L'interconnexion des composants se résume alors à l'affection de ces signaux aux entrées/sorties des composants lors du mapping. Par exemple, le marquage de la place P1 est propagé à la transition T3 via le signal *s_marque_P1* qui connecte la sortie marque de P1 et l'entrée Me(0) de T3.

2.4 Les conditions

Pour des considérations de modularité, de réutilisation et de minimisation de l'implantation, la condition est une entité décrite en dehors du composant transition. Plusieurs transitions peuvent en effet avoir la même condition, ou avoir des termes communs dans leurs conditions respectives. Par conséquent, il est possible dans notre approche de décomposer une condition en produit de termes, chacun d'entre eux étant réutilisable. Pour une transition donnée, les termes qui la composent sont connectés à son port d'entrée Ct ; la transition effectue le produit de ces termes pour déterminer si la condition correspondante est vraie (cf. section 2.2). Précisons qu'il est tout à fait possible de décrire simplement une condition pour chaque transition, sans se préoccuper de cette « décomposition ». D'ailleurs, en l'absence de condition associée, une transition porte une condition « mono-terme », fixée à 1 (condition toujours vraie).

Les conditions ou les termes qui les composent, directement écrits en VHDL, sont tous regroupés au sein d'un même process VHDL dont l'activation est effectuée avant l'évaluation du franchissement des transitions (cf. section 1.4).

```
conditions : process (clk)
  begin
     if rising_edge (clk) then
          if s_selec = '0' then
                    -- exemple d'évaluation un terme
                    if (s_requete="010") then terme1 <= '1';
                    else terme1 <= '0';
                    end if;
                    … -- autres termes
          end if;
    end if;
end process;
```

2.5 Les fonctions

A l'instar des conditions, les fonctions peuvent également être décomposées en fonctions élémentaires réutilisables pour différentes transitions ; plusieurs fonctions élémentaires (tout simplement des fonctions) peuvent être associées à une même transition.
Les fonctions, directement écrites en VHDL, sont toutes regroupées au sein d'un même process VHDL dont l'activation est effectuée suite au tir des transitions (cf. section 1.4).
Une fonction doit être exécutée si et seulement si au moins l'une des transitions auxquelles elle est associée, est franchie. Pour déterminer si une transition est franchie, il suffit de comparer le flux sortant qu'elle génère (i.e. sa sortie aj_aval) avec le poids de l'arc sortant correspondant (i.e. son entrée As). L'égalité, qui n'est à tester que sur un arc de sortie (car si elle est vraie pour un des arcs de sortie d'un transition elle est vraie pour tous ses arcs de sortie), signifie que la transition vient d'être franchie dans le cycle courant, la sortie aj_aval étant remise à 0 avant chaque évaluation du tir.

```
fonctions : process (clk)
  begin
  if clk'event and clk='0' then
          if reset_n = '1' then
                  if s_selec = '0' then
                          -- exemple simple de fonction
                          if (s_As_atp_13 = s_aj_atp_13) then CO <="0010110";
                          end if;
                          … -- autres fonctions
                  end if;
          end if;
  end if;
end process;
```

avec, donnés ici pour exemple :

- s_aj_atp_13 correspondant au signal connecté à la sortie aj_aval(i) d'une transition vers l'entrée d'une place.
- s_As_atp_13 correspondant au signal connecté au poids de l'arc sortant As(j) concerné (i.e. reliant cette transition à cette place).

2.6 Les actions

Il en est de même pour les actions, i.e. une même action peut être associée à différentes places et une place peut avoir plusieurs actions associées ; les actions, directement décrites en VHDL, sont uniquement booléennes.
Les actions sont toutes regroupées au sein d'un même process VHDL dont l'activation est effectuée après l'actualisation du marquage (marquage stable), i.e. lors de l'évaluation du franchissement des transitions (cf. section 1.4). L'activation d'une action est directement dépendante du marquage des places concernées : si l'une des places est marquée alors l'action est activée.

```
actions : process (clk)
   begin
     if (clk'event and clk='1') then
          if reset_n = '1' then
                  if s_selec = '1' then
                          -- exemple d'action simple (sur place binaire)
                          if (s_marque_ON = '1' ) then led <= '1';
                          else led <= '0';
                          end if;
                          … -- autres actions
```

```
                end if;
        end if;
    end if;
end process;
```

avec, donnés ici pour exemple :

- s_marque_ON le signal associé à la sortie marque de la place ON (i.e. marquage stable de la place).
- led le signal affecté par l'action.

2.7 Génération du signal de distinction des phases d'évolution

La génération du signal selec, dédié à la distinction des phases d'évolution de l'état du modèle (cf. section 1.4), est effectuée via un simple process VHDL.

```
selec : process (clk)
begin
  if reset_n = '0' then s_selec <= '0';  -- initialisation
  elsif rising_edge(clk) then s_selec <= not (s_selec);   -- commutation
  end if;
end process;
```

2.8 Constitution du fichier VHDL décrivant le composant rezo

Le code VHDL ci-dessous décrit l'ensemble du fichier VHDL automatiquement généré lors de la traduction d'un réseau de Petri généralisé interprété temporel : il décrit donc le composant rezo équivalent au réseau de Petri traduit. L'attribution du nom des signaux est automatique.
Ce fichier, décrit à partir d'extraits VHDL d'un exemple, se structure en plusieurs parties :

- Insertion des librairies et packages utilisés, dont celui contenant nos différents types de vecteur.

```
library ieee;
library work;
use ieee.std_logic_1164.all;
use ieee.std_logic_arith.all;
use ieee.std_logic_unsigned.all;
use work.petri.all;  -- comprenant la déclaration de   constant : marque_max_net : integer := 6 ;
```

- Déclaration de l'interface de l'entité rezo.

Notons que les signaux d'entrées/sorties du réseau de Petri sont automatiquement déclarés dans l' interface, permettant l'insertion (interconnexion) du composant rezo VHDL avec le reste de l'architecture du système. Logiquement, les signaux d'horloge clk et d'initialisation reset_n font partie de cette interface.

```
entity rezo is
  port(clk : in std_logic;
       ext_s_M_out : out std_logic_vector(6 downto 0);
       reset_n : in std_logic;
       … );
end rezo;
```

- Déclaration de l'architecture de l'entité rezo.

```
architecture a_rezo of rezo is
```

- Déclaration de tous les signaux utilisés au sein de l'entité rezo.

Ces nombreux signaux sont dédiés à l'interconnexion des instances des composants de base place et transition, à l'interconnexion des conditions avec les transitions, à l'interconnexion des signaux internes avec ceux de son interface, etc. (cf. section 2.3).

Précisons que, dans cet exemple, de nombreux signaux sont déclarés de type std_logic, car nous avons utilisé nos composants génériques place et transition binaires (non présentés dans cet article).

```
-- signaux généraux nécessaires à l'évolution du réseau
        signal s_selec : std_logic;
…
-- signaux de marquage des places
        signal s_marque_OFF : std_logic;        -- s_marque_[nomplace]
…
-- signaux d'initialisation du marquage des places
        signal s_ini_OFF : std_logic;           -- s_ini_[nomplace]
…
-- signaux d'ajout de jetons dans les places (arcs TP)
        signal s_aj_atp_6 : std_logic;          -- s_aj_[type_arc]_[numero_arc_automatique]
…
-- signaux de retrait de jetons dans les places (arc PT ou TST ou INH)
        signal s_ret_apt_4 : std_logic;         -- s_ret_[type_arc]_[numero_arc_automatique]
        signal s_ret_atst_7 : std_logic;
        signal s_ret_ainh_0 : std_logic;
…
-- signaux relatifs au poids des arcs entrants (arc PT ou TST ou INH)
        signal s_Ap_apt_7 : std_logic;          -- s_Ap_[type_arc]_[numero_arc_automatique]
        signal s_Ap_atst_5 : std_logic;
        signal s_Ap_ainh_0 : std_logic;
…
-- signaux relatifs au type des arcs entrants d'une transition (arc PT ou TST ou INH)
        signal s_At_apt_7 : std_logic_vector(1 downto 0);  --s_At_[type_arc]_[numero_arc_automatique]
        signal s_At_atst_4 : std_logic_vector(1 downto 0);
        signal s_At_ainh_2 : std_logic_vector(1 downto 0);
…
-- signaux relatifs au poids des arcs sortants d'une transition (arc TP)
        signal s_As_atp_1 : std_logic; --s_As_[type_arc]_[numero_arc_automatique]
…
-- signaux relatifs au temps dans une transition (une seule transition temporelle dans cet exemple)
        signal t_min_delay : integer range 0 to 65535;
        signal t_max_delay : integer range 0 to 65535;

-- signaux relatifs aux conditions (termes) associées aux transitions
        signal cond_vraie : std_logic;          -- terme « condition vraie » (valeur par défaut)
        signal cond1 : std_logic;               -- nom (ou cond[numéro]) (terme quelconque)
…
 -- variables locales, i.e. variables utilisées en interne au composant rezo (réseau de Petri)
        signal CPT : std_logic_vector(23 downto 0);
…
 -- signaux internes des sorties du composant, (i.e. de connexion avec l'interface du composant rezo)
        signal s_M_out : std_logic_vector(6 downto 0);
…
```

- Déclaration de tous les composants utilisés au sein de l'entité rezo (cf. section 2).

```
-- déclaration des composants génériques (transition temporelle et non temporelle, place...)
component transition_temporelle
…
end component;
…
```

- Description du process d'initialisation

```
begin

-- description du process d'initialisation du marquage des places, du poids et du type des arcs

  process(reset_n)
    begin
      if (reset_n='0') then
            -- initialisation du terme "cond_vraie" (terme par défaut)
            cond_vraie <= '1';
            -- initialisation du marquage des places
            s_ini_ON <= '1';
            s_ini_execute <= '0';
            …
            -- initialisation du poids des arcs entrants (PT, TST)
            s_Ap_apt_7 <= '1';
            s_Ap_apt_4 <= '1';
            …
            -- initialisation du type des arcs entrants(PT,TST, INH)
            s_At_apt_7 <= "00";
            s_At_atst_5 <= "01";
            s_At_ainh_0 <= "10";
            …
            -- initialisation du poids des arcs sortants TP
            s_As_atp_1 <= '1';
            s_As_atp_14 <= '1';
            …
       end if;
  end process;
```

- Déclaration des instances (mappées)

```
 -- instanciation des différents composants place
ON : place
 generic map (
            nb_entrees => 4,
            nb_sorties => 3,
            max_poids_ent => 3,
            max_poids_sort => 2,
            marque_max_p := marque_max_net)
 port map(
            clk           =>clk,
            selec         =>s_selec,
            reset_n       =>reset_n,
            ini_jetons    =>s_ini_ON,
            aj_jetons(0)  => s_aj_atp_6,
            aj_jetons(1)  => s_aj_atp_7,
            aj_jetons(2)  => s_aj_atp_12,
            aj_jetons(3)  => s_aj_atp_17,
            ret_jetons(0) => s_ret_apt_4,
            ret_jetons(1) => s_ret_apt_5,
            ret_jetons(2) => s_ret_apt_16,
            marque        =>s_marque_ON);
…
-- instanciation des différents composants transition
 delay : transition_temporelle
 generic map (
            nb_entrees => 1,
            nb_sorties => 1,
```

```
            nb_termes =>1,
            max_poids_ent => 2,
            max_poids_sort => 3)
 port map(
            clk          =>clk,
            selec        =>s_selec,
            c_t          =>cond15,
            reset_n      =>reset_n,
            t_min        => t_min_delay,
            t_max        => t_max_delay,
            Me(0)        => s_marque_execute,
            Ap(0)        => s_Ap_apt_7,
            At(0)        => s_At_apt_7,
            ret_amont(0) => s_ret_apt_7,
            As(0)        => s_As_atp_1,
            aj_aval(0)   => s_aj_atp_1);
…
```

- Déclaration des process dédiés aux actions, aux fonctions, aux conditions et à la génération du signal selec.

```
-- process des actions
actions : process (clk)
   begin
     if (clk'event and clk='1') then
            if reset_n = '1' then
                    if s_selec = '1' then
                            if (s_marque_ON = '1' ) then led <= '1';
                            else led <= '0';
                            end if;
                            … -- autres actions
                    end if;
            end if;
     end if;
   end process;

-- process des fonctions
fonctions : process (clk)
  begin
  if clk'event and clk='0' then
            if reset_n = '1' then
                    if s_selec = '0' then
                            if (s_Ais_atp_13 = s_aj_atp_13 or s_Ais_atp_15 = s_aj_atp_15)
                            then CO<="00000000";
                            end if;
                            … -- autres fonctions
                    end if;
            end if;
  end if;
end process;

-- process permettant l'évaluation des conditions
conditions : process (clk)
  begin
  if rising_edge (clk) then
            if s_selec = '0' then
                    if (s_requete="010") then cond1 <= '1';
                    else cond1 <= '0';
                    end if;
                    … -- autres conditions
            end if;
```

```
  end if;
end process;

-- process de génération du signal s_selec

selec : process (clk)
begin
  if reset_n = '0' then s_selec <= '0';
  elsif rising_edge(clk) then s_selec <= not (s_selec);
  end if;
end process;
```

- Déclaration de l'affectation des signaux internes aux sorties, i.e. des liens entre les variables internes manipulées et celles exposées via l'interface du composant rezo.

```
-- affectation des signaux internes sur les sorties
ext_s_M_out <= s_M_out;
…

end a_rezo;
```

3 Le traducteur automatique

Le traducteur automatique que nous avons conçu et réalisé s'inscrit au sein d'un environnement comprenant :

- *un éditeur graphique de RdP généralisé interprété temporel*

Il permet la saisie du modèle au niveau structurel : places (Figure 10), transitions (*Figure 11*) et arcs. Il impose lors de cette saisie le respect de contraintes structurelles comme par exemple l'alternance place-transition, l'utilisation d'un arc inhibiteur seulement en entrée d'une transition. Par ailleurs, il est imposé (mais ces contraintes sont supprimables) :

- qu'une transition n'ait pas comme seul arc amont, un arc de test,
- qu'une transition n'ait pas comme seul arc amont, un arc inhibiteur.

L'éditeur permet la description et la réutilisation de sous-réseaux. Cette possibilité de décrire des sous-réseaux favorise d'une part une conception et une description modulaires, et d'autre part la réutilisabilité de parties du modèle. Le modèle global peut être construit par assemblage de sous-réseaux (Figure 12, Figure 13), chaque sous-réseau pouvant être « instancié » plusieurs fois (Figure 14).

- *un éditeur des variables manipulées par le modèle*

Ces variables (Figure 15) sont soit des variables d'entrée ou de sortie (ou d'entrée/sortie), soit des variables internes (locales). Ces variables supportent notamment les interactions entre le composant rezo, i.e. le bloc VHDL traduisant le modèle, et les autres éléments du système contrôlé par le réseau de Petri. Elles sont donc automatiquement rapportées dans l'interface du composant rezo afin de pouvoir l'interconnecter avec les éléments externes (RAM, registres, etc.). Le type de la variable doit être indiqué par l'utilisateur (std_logic, integer, …).

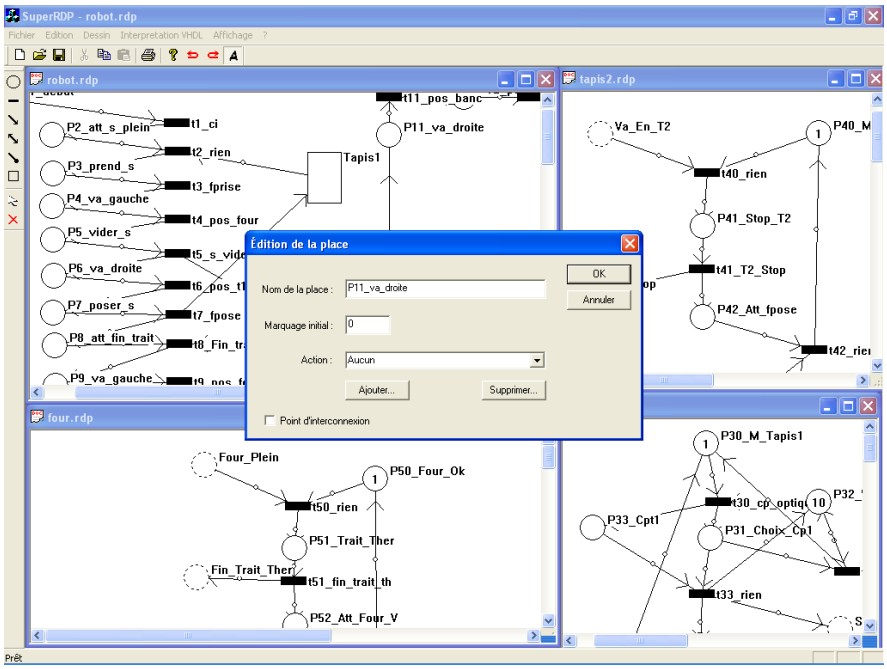


Figure 10 : Définition d'une place

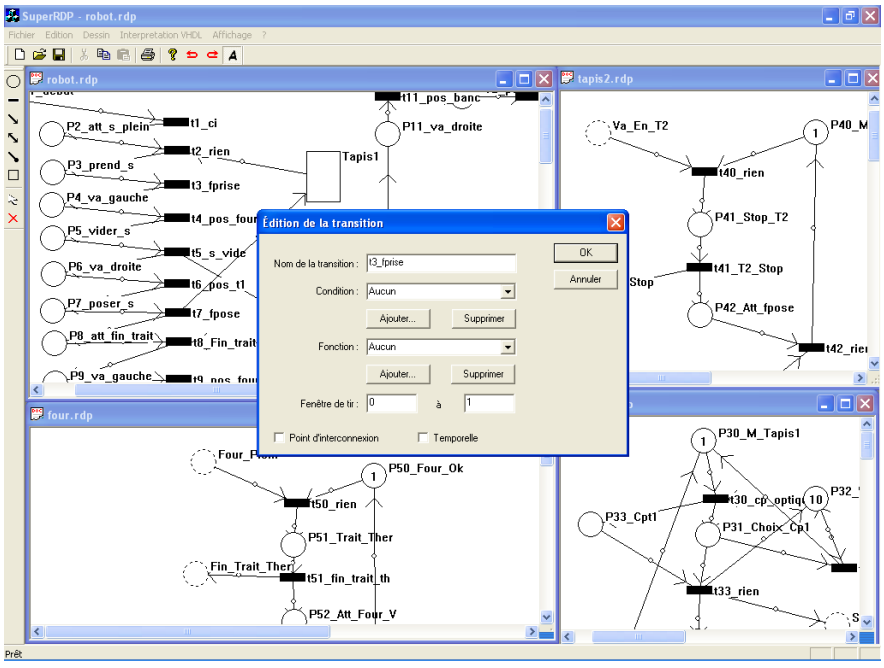


Figure 11 : Définition d'une transition

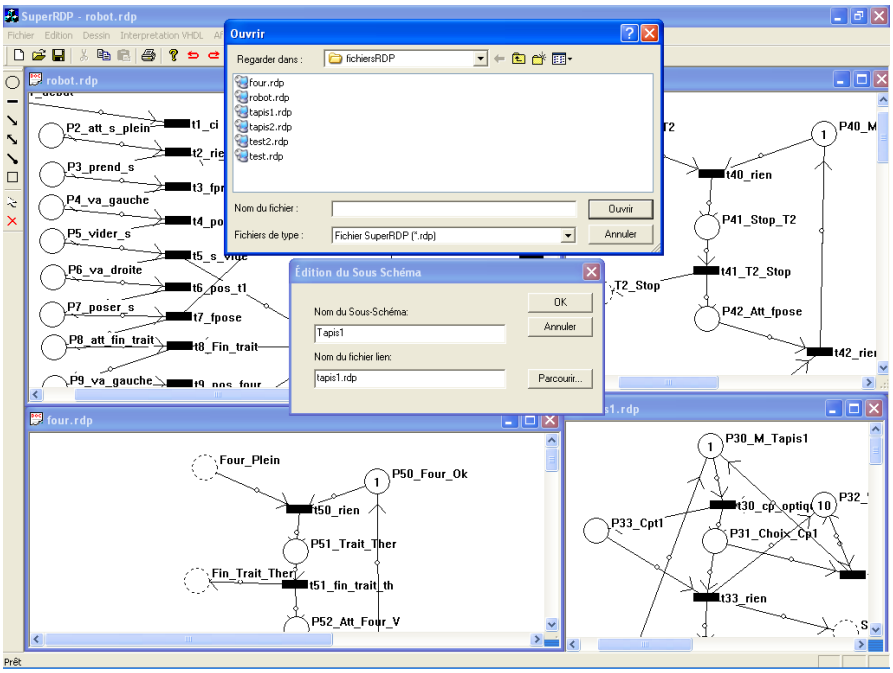


Figure 12 :Insertion d'un sous-réseau

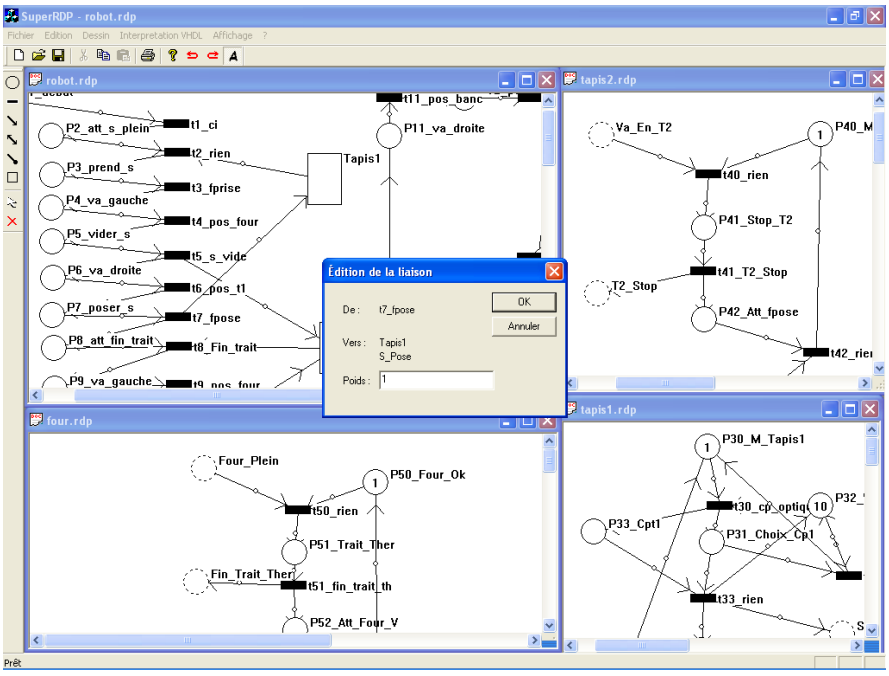


Figure 13 : Edition de l'interconnexion (arc) entre un nœud du réseau principal et un nœud d'un sous-réseau

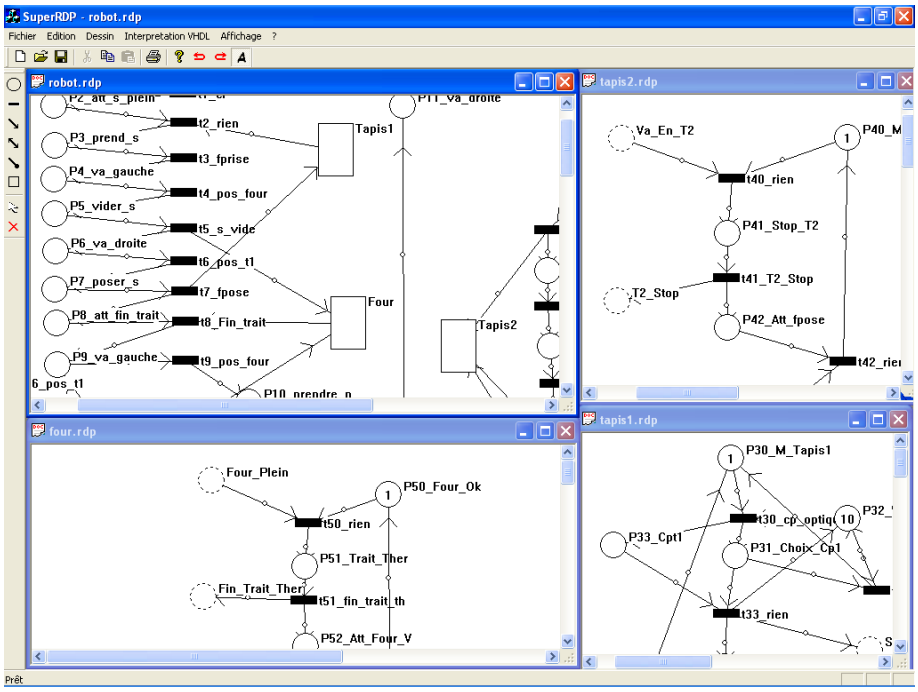


Figure 14 : Réseau (principal) incluant des sous-réseaux (instanciés)

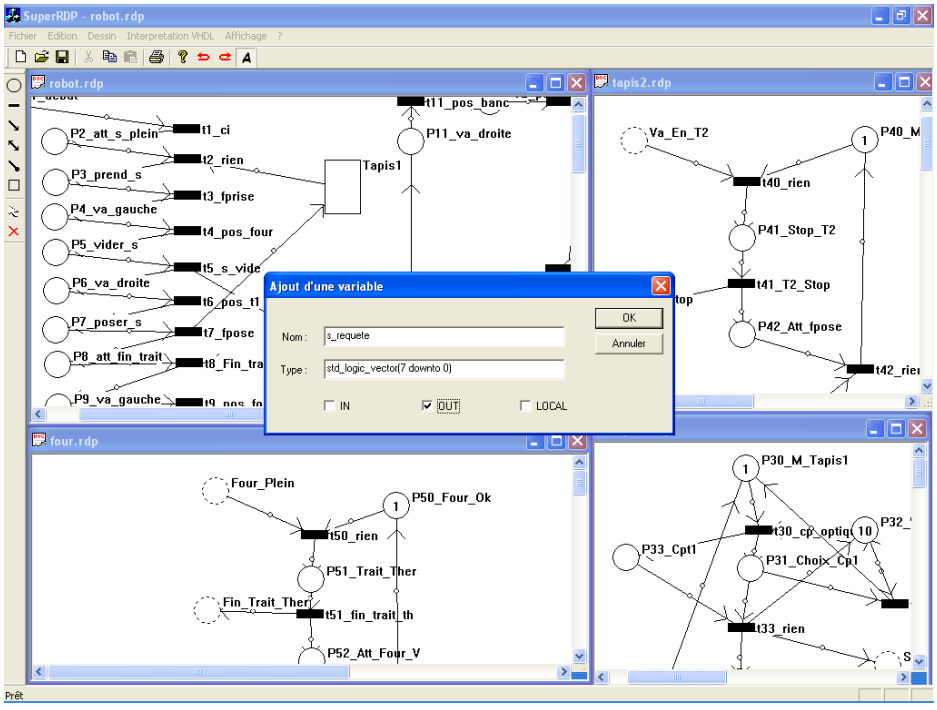


Figure 15 : Définition d'une variable

- *un éditeur de l'interprétation associée au modèle*

Cette interprétation comprend les conditions, les actions et les fonctions ; elles doivent être décrites en VHDL. L'interprétation peut directement porter sur les variables d'entrée, de sortie ou locales décrites ci-dessus (Figure 16, Figure 17).

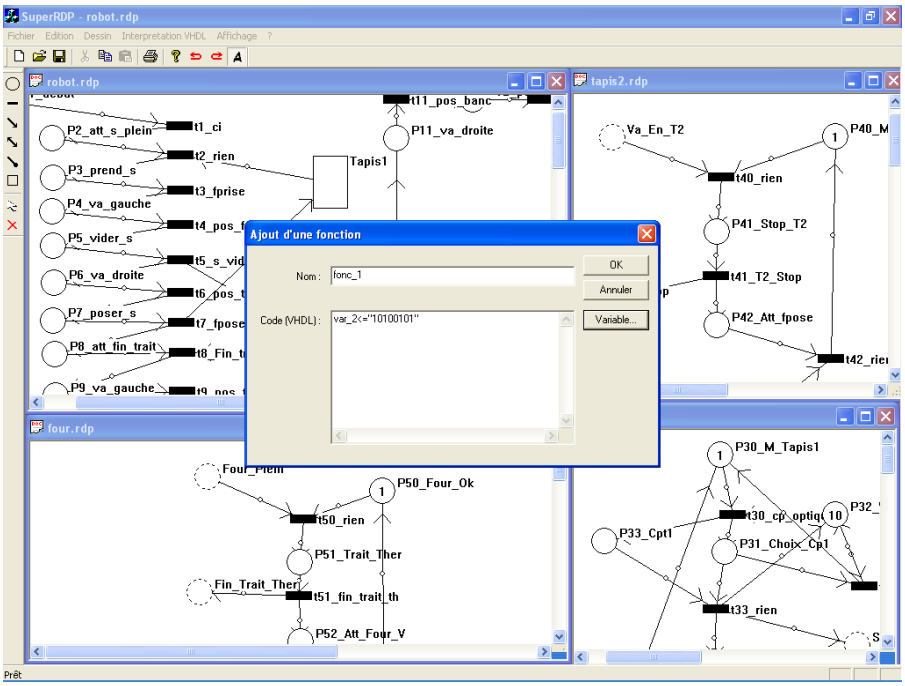


Figure 16 : Définition d'une fonction

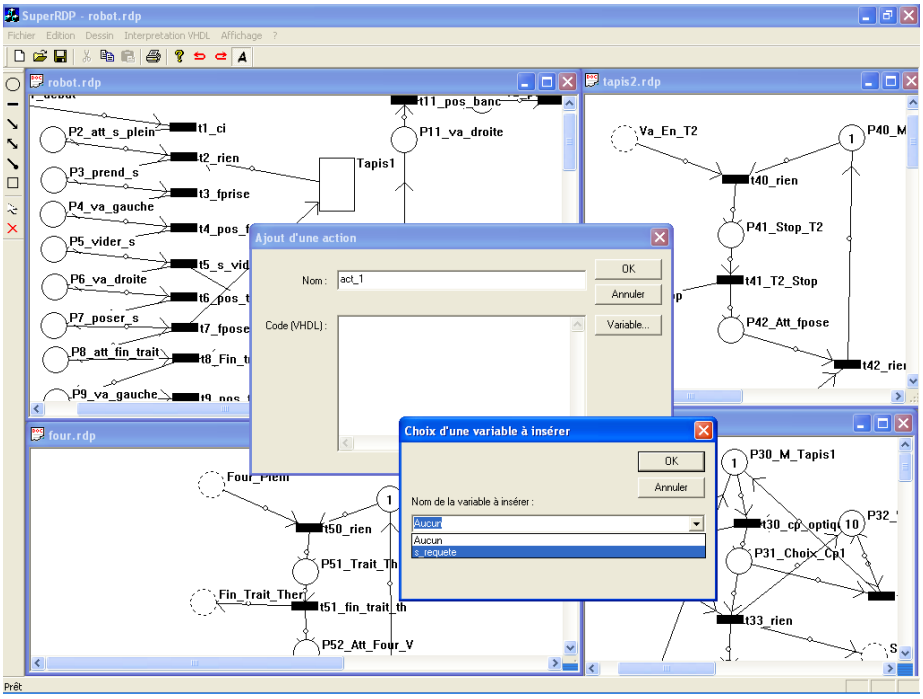


Figure 17 : Ajout d'une variable dans la liste des variables affectées par l'action

- *un traducteur automatique*

A partir de ces éléments descriptifs (structure, interprétation, variables) et des composants génériques présentés (place, transition) le traducteur génère le bloc VHDL équivalent, i.e. le composant rezo.

- *un analyseur de réseaux de Petri (en cours de développement)*

L'analyseur comporte un ensemble d'algorithmes permettant d'obtenir les propriétés générales du modèle (vivacité, finitude, …), son graphe des marquages accessibles, ainsi que les invariants de place et de transition.

4 Conclusion

Nous avons présenté une nouvelle approche pour la traduction automatique d'un réseau de Petri généralisé interprété temporel en VHDL. Cette approche consiste à exploiter l'orientation composant du langage en définissant 2 composants structurels de base : un composant place et un composant transition. Le composant transition intégrant les arcs amont et aval, il intègre l'interconnexion entre les composants de base et supporte donc les liens structurels entre les nœuds du modèle de réseau de Petri que sont les places et les transitions. Les composant VHDL définis sont des composants génériques, dimensionnés au plus juste en fonction des caractéristiques du modèle : nombre de places amont et aval de chaque transition, nombre de transitions amont et aval de chaque place, limite sur le marquage des places, poids des arcs, transition temporelle ou non, etc.

Le composant VHDL « rézo » automatiquement généré, tant au niveau de son architecture que de son interface (entity), est exécuté de manière synchrone. Le partitionnement des phases d'évolution du modèle entre les différents composants de base le constituant, a été exposé.

Nous avons également développé un outil logiciel permettant la saisie graphique tant modèle réseau de Petri que de son interprétation associée. Cet éditeur permettant de décrire des sous-réseaux, favorise d'une part une conception et une description modulaires, et d'autre part la réutilisabilité de parties du modèle.

Ces travaux ont été utilisés avec succès dans différents projets.

Parallèlement à la poursuite de ces travaux, notamment sur l'éditeur (intégration d'un analyseur), nous étudions d'une part les mécanismes d'interconnexion (fusions de places et de transitions) et d'agrégation (macro-place), et d'autre part la traduction automatique du modèle de réseau de Petri, selon une approche (d'exécution) asynchrone.

5 Références

[DAI] http://www.daimi.au.dk/PetriNets/research/

[DAV89] R. David, H. Alla, « Du Grafcet aux réseaux de Petri ». Traité des nouvelles technologies, Edition Hermès, 1989.

[FER97] J.M. Fernandes, M. Adamski, A.J. Proença « *VHDL generation from hierarchical Petri net specifications of parallel controlers* », IEE Proc. Part E, Computers and Digital Techniques, 144(2), 127-137, 1997.

[MAR98] Norian Marranghello, « *Digital systems synthesis from Petri net description* », Technical Report DAIMI/PB-530, march 1998.

[MUR89] A. Murata. « *Petri nets: Properties analysis and applications* ». In IEEE, volume 77, pages 541-580, Avril 1989.