

UNIVERSITÉ PAUL SABATIER

Modèle Temporel avancé

- TP : SYSTÈME DE TRAITEMENT AUTOMATISÉ -

Auteurs :

Lucien RAKOTOMALALA
David TOCAVEN

Encadrant :

Pauline RIBOT
Euriell LE CORRONC Michel COMBACAU

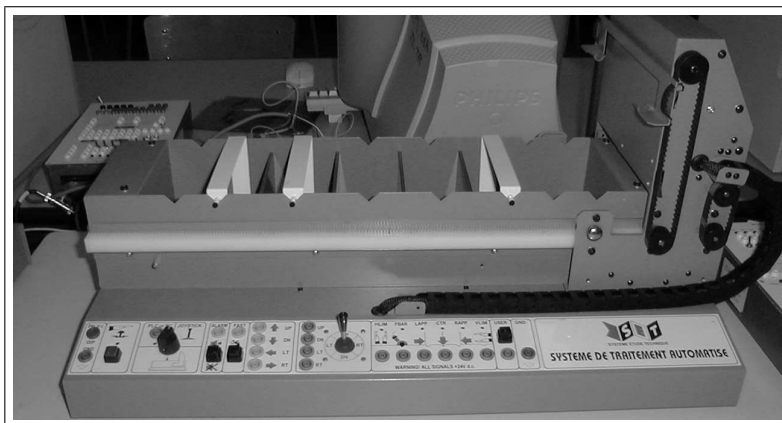


Table des matières

Introduction	1
1 Modélisation et analyse de la réalisation d'une opération	2
1.1 Modèle réseau de Petri temporel d'une opération	2
1.2 Estimation de Prise/Pose et Avance/Reculé	3
1.3 Analyse du modèle	3
2 Modélisation et analyse des premières opérations de chaque pièce	5
2.1 Réseau de PETRI Temporels de commande de deux opérations	5
2.2 Analyse du modèle avec TINA	7
2.3 Mise au point des Intervalles Temporels	8
3 Modélisation des séquences d'opérations des deux pièces	9
3.1 Analyse d'ordonnancement	9
3.2 Réseau de Petri de commande	10
3.3 Analyse par graphe de classes	10
3.4 Implémentation	10
4 Conclusion	11
 Annexes	 13
Mesures de temps	13
4.1 Mesure du temps de déplacement et de saisie/dépôt d'une pièce	13
4.2 Mesure du temps	19
Analyse TINA	26
4.1 Analyse d'accessibilité du modèle 2 opérations	26

Introduction

1 | Modélisation et analyse de la réalisation d'une opération

Nous allons dans un premier temps réaliser une modélisation par réseau de Petri temporel de la réalisation d'une opération. Cette modélisation sera générique à la réalisation de toute opération O_i . Ensuite, nous réaliserons un code C qui permet d'estimer les durées des différentes opérations. Finalement, nous analyserons le réseau de Petri à l'aide de *TINA 2.8.4*.

1.1 Modèle réseau de Petri temporel d'une opération

Nous avons, pour modélisation générique d'une opération, considéré que le chariot de déplacement se trouve en bas. Voici le réseau de Petri temporel (voir figure 1.1) :

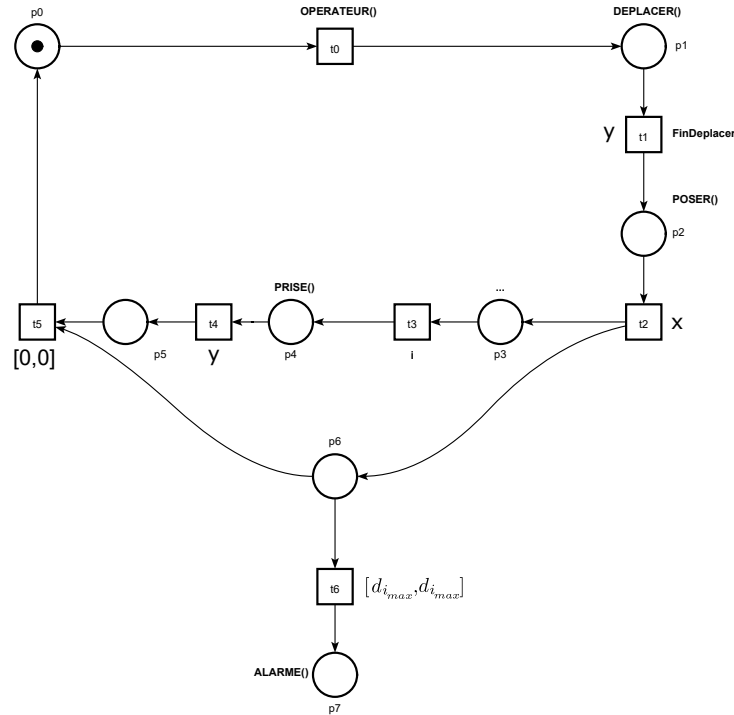


FIGURE 1.1 – Modèle réseau de Petri générique d'une opération.

Nous considérerons que l'action *DEPLACER()* et l'événement *FinDeplacer* correspondent à, respectivement, *AVANCER()* et *FinAvancer* si le chariot est à droite de l'emplacement de l'opération o_i ou *RECULER()* et *FinReculer* si celui-ci est à gauche.

Le marquage initial est constitué d'un unique jeton sur la place p_0 . Ce jeton, une fois que la transition t_0 est sensibilisée et tirée (pour cela il faut que l'événement *OPERATEUR()* est eu lieu), est en p_1 . Le chariot se déplace tant qu'il y a un jeton en p_1 . Le jeton reste en p_1 jusqu'à ce que le chariot arrive à destination, c'est-à-dire que *FinDeplacer* se déclenche. Lorsque cet événement se produit, la transition t_1 est sensibilisée et tirée (le déplacement prend un temps y qui est représenté sur la transition t_1). Ensuite, un jeton marque la place

p_2 ce qui déclenche l'action $POSER()$. Cette action prend un temps x et celui-ci est représenté sur la transition t_2 . Une fois le temps x écoulé, la transition t_2 est tiré et les places p_3 et p_6 sont marqués d'un jeton chacun.

À partir de cet état, il y a deux jetons dans le réseau : un permet de décrire le comportement du chariot et un autre, celui qui marque p_6 , permet de déclencher l'alarme si la pièce qui subit l'opération o_i n'est pas reprise avant le temps maximal de l'opération. En effet, le jeton présent en p_6 , au bout d'un temps $d_{i_{max}}$, va être consommé par la transition t_6 et un jeton va marquer p_7 . Ceci déclenchera l'action $ALARME()$. Il faut donc que le jeton présent en p_3 arrive en p_5 en moins de $d_{i_{max}}$ unités de temps pour que l'alarme ne se déclenche pas. De cette façon, le tir de la transition t_5 , qui nécessite et consomme un jeton en p_6 et un jeton en p_5 , empêchera l'alarme de sonner et permettra d'effectuer une nouvelle opération (retour au marquage initial). La place p_3 à un événement ..., cela représente la possibilité d'effectuer n'importe(s) quelle(s) action(s) et de revenir à l'emplacement de l'opération o_i , de façon à ce que l'action en p_4 , $PRISE()$, de durée y , permette de récupérer la pièce. La transition t_3 est marquée de la temporisation i . Celle-ci représente le temps de(s) action(s) de la place p_4 et/ou un temps d'attente afin que l'on récupère la pièce à la fin de l'opération i . Ainsi, si l'on souhaite que l'alarme ne sonne pas, il faut que $i + y < d_{i_{max}}$.

1.2 Estimation de Prise/Pose et Avance/Reculé

Voir annexe 4, page 13.

Nous avons maintenant besoin d'identifier le temps de $AVANCER()$ (égal à celui de $RECULER()$) que l'on appelait précédemment y et de $PRISE()$ (équivalent à celui de $POSE()$) appelait y . Pour cela, nous avons créé, à partir d'un code fourni, un code permettant de mesurer les temps x et y . Pour mesurer le temps d'une action, nous avons stocké le temps du PC à l'instant du début de l'action, puis nous avons stocké le temps à la fin de celle-ci et avons affiché la soustraction des deux temps sur le terminal. Nous avons déterminé que :

$$x = \begin{bmatrix} 1 & 1 \end{bmatrix} \text{ seconde} \quad (1.1)$$

$$y = \begin{bmatrix} 3 & 3 \end{bmatrix} \text{ secondes} \quad (1.2)$$

$$(1.3)$$

1.3 Analyse du modèle

Grâce aux mesures précédentes, nous avons pus remplacer x et y par des valeurs temporelles sur le modèle générique. Nous avons choisi arbitrairement les valeurs de $d_{i_{min}} = 13$ secondes et $d_{i_{max}} = 16$ secondes, respectivement le temps minimal de l'opération et le temps maximal de l'opération o_i . Ainsi, nous avons la condition suivante qui doit être respectée $i < d_{i_{max}} - y$, soit $i < 13$ secondes. Donc il "reste" moins de 13 secondes afin de réaliser d'autres opérations. Nous allons fixer une temporisation $i = \begin{bmatrix} 12 & 12 \end{bmatrix}$ secondes pour étudier le réseau.

Figure 1.2, voici le nouveau réseau de Petri temporel. Une analyse à l'aide de TINA nous a permis de

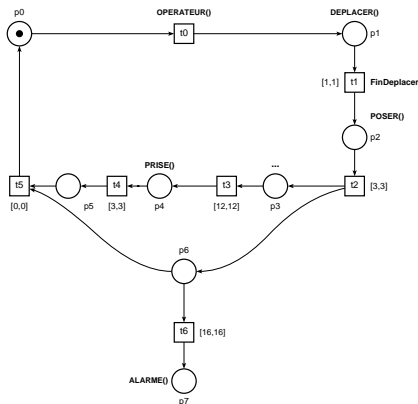


FIGURE 1.2 – Modèle réseau de Petri générique d'une opération avec les temps estimés

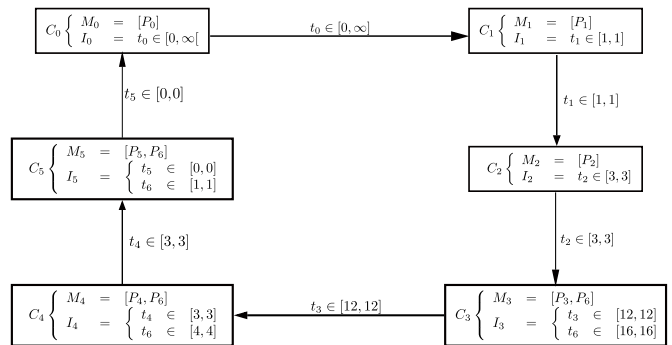


FIGURE 1.3 – Diagramme de classe.

déterminer les différentes classes du réseau. L'automate est présenté figure 1.3. Nous avons put aussi extraire les propriétés suivantes grâce à TINA.

- Le RdPT (Réseau de Petri Temporisé) n'est pas vivant.
- La transition t_6 est non vivante.
- LE RdPT est borné.

2 | Modélisation et analyse des premières opérations de chaque pièce

Maintenant que nous connaissons un modèle valide pour une opération ainsi que les temps nécessaires au déplacement du chariot sur l'axe vertical et horizontal, nous allons pouvoir commencer à modéliser le travail du *STA* sur deux opérations.

Nous allons, dans un premier temps, effectuer une modélisation en RdP Temporels d'une commande de deux opérations suite à quoi, nous en effectuerons une analyse grâce à une version de *TINA* identique que dans le chapitre 1. Nous utiliserons cette analyse pour déterminer les intervalles d'attentes et le meilleur ordonnancement possible pour ne pas déclencher l'alarme.

2.1 Réseau de PETRI Temporels de commande de deux opérations

A partir du modèle générique établi en 1.1, nous avons obtenu, pour la commande de opérations O_1 et O_2 le modèle RdP Temporels en figure 2.1.

Dans ce réseau, nous pouvons identifier tout d'abord la ressemblance avec le modèle générique (en figure 1.1) : les places p_4 et p_{12} sont les représentations de la place p_2 dans le modèle générique. Elles seront donc suivi, dans les modèle Temporels que nous analyseront, d'une transition qui contient le temps des opérations *Poser*. Il en va de même pour les places p_2 , p_{10} , p_5 et p_{13} qui contiennent l'opération *Prendre*, elles seront suivi d'une transition contenant une intervalle de temps y .

Séparation du modèle Nous pouvons séparer ce modèle complexe en deux ensembles de places :

- l'ensemble $P_1 = \{p_2, p_3, p_4, p_5, p_6, p_7, p_{20}, p_{18}\}$ est utilisé pour emmener la pièce $p1$ du bac $e1$ (son bac initial) vers le bac $e3$ dans lequel elle subit l'opération O_2 . Cet ensemble est lié avec les deux places p_{20} et p_{18} qui modélisent l'alarme liée à l'opération O_1 .
- l'ensemble $P_2 = \{p_{10}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}\}$ modélise le transport de la pièce $p2$ de $e2$ (son bac initial) vers $e3$, bac dans lequel elle subit l'opération $O1$, puis du transport de $e5$ vers $e7$ (Pour prévoir les prochains RdP). L'alarme de l'opération est enclenchée par la place p_{17} , qui est lié au reste de l'ensemble P_2 par la place p_{19} .

Liaison entre les ensembles Les places p_9 , p_{21} et p_{23} , places qui se situent entre les deux ensembles P_1 et P_2 , sont utilisées pour effectuer les passages entre les 2 ensembles. De même, nous avons deux places p_{22} et p_{24} qui font la liaison entre les deux ensembles, à l'attention que celles ci ne servent pas à amener la chariot d'une pièce à l'autre mais à le faire attendre le temps nécessaire avant la fin d'une opération et donc la récupération d'une pièce.

Nous notons aussi les transitions t_{15} et t_{20} qui sont les représentations de la transition t_5 dans le modèle générique. Elles permettent dans ce contexte de synchroniser la prise de la pièce et l'arrêt du compteur de l'alarme. Comme dans le modèle générique, ces transitions ont un intervalle de temps $[0; 0]$, cela oblige les jetons arrivant dans les places en amont à être immédiatement tiré. Ainsi, le compteur de l'alarme est stoppé après Ou.t. que la pièce ait été retiré de son emplacement.

Ordonnancement Il est aussi à noter que nous avons choisi d'effectuer l'opération de la pièce $p1$ avant celle de $p2$ et cela pour des raisons temporelles. Nous avons remarqué, dans une étude préliminaire à la construction de ce réseau, que cet ordonnancement était possible alors qu'une inversion l'ordre du traitement des pièces donne obligatoirement le retentissement d'une alarme (notamment dans la suite du TP).

Nous avons maintenant à notre disposition une commande à appliquer sur les *STA*. Toutefois, avant de passer à l'implémentation, nous allons utiliser l'outil *TINA* pour analyser le modèle ainsi obtenu.

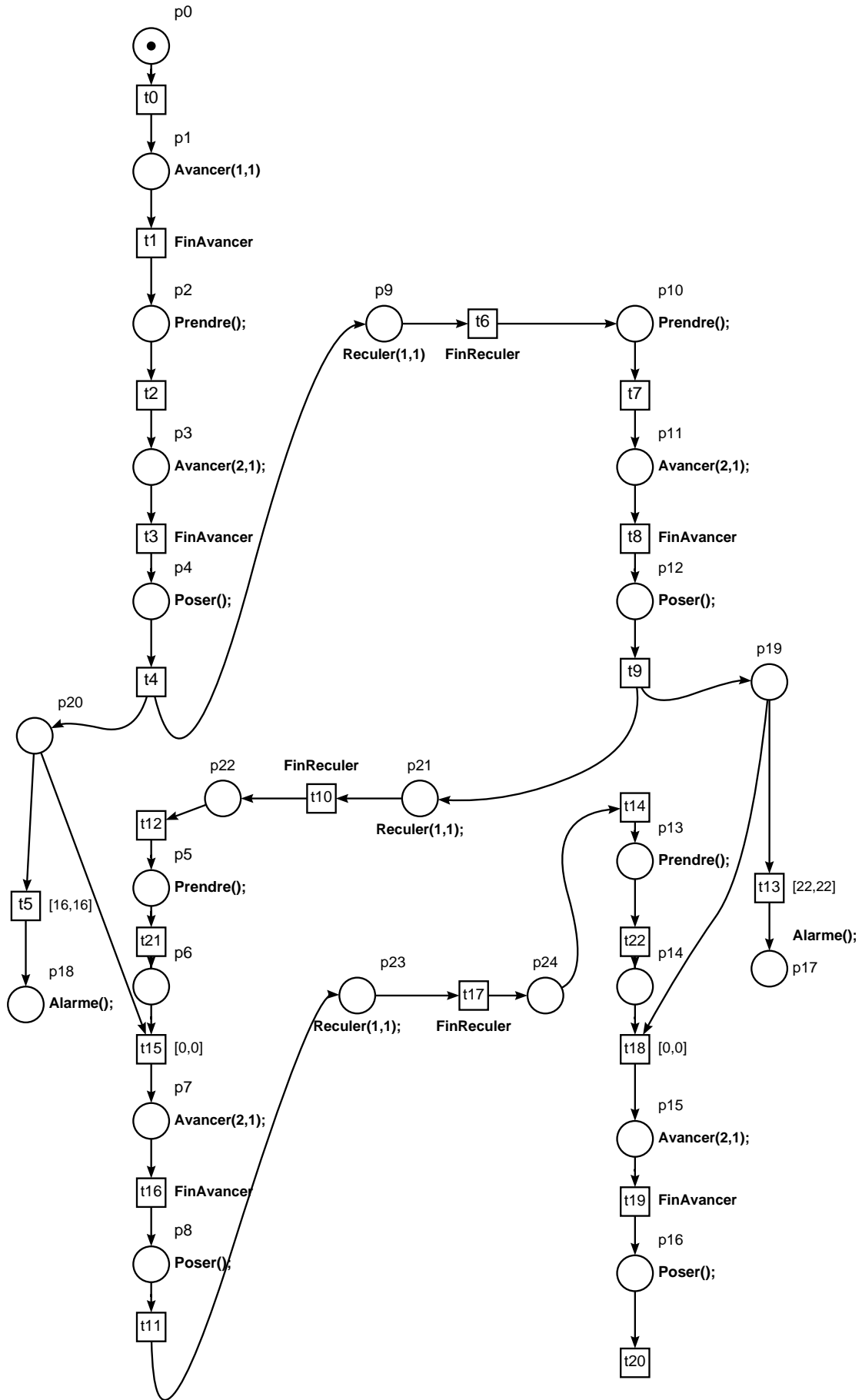


FIGURE 2.1 – Réseau de PETRI Temporels pour la commande de 2 opérations

2.2 Analyse du modèle avec TINA

L'analyse temporelle de réseau nécessite un réseau tel que nous vous présentons en figure 2.2. Dans ce type de réseau, nous avons choisi de ne plus utiliser des noms sur les évènements mais plutôt des intervalles de temps. Ces intervalles ont été établi dans la section 1.2.

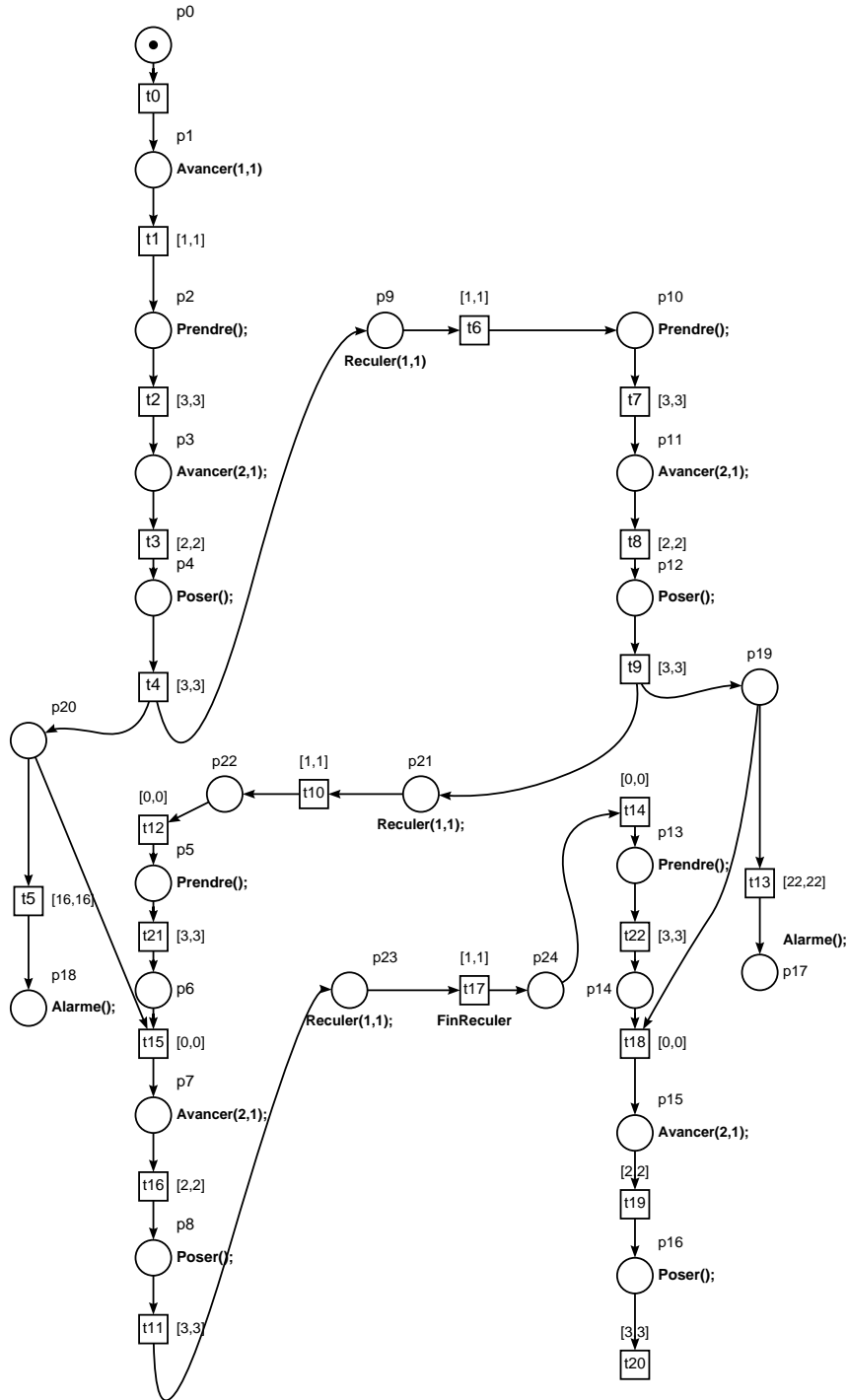


FIGURE 2.2 – Réseau de PETRI Temporels pour l'analyse de 2 opérations

Avec la version de *TINA* adéquate, nous avons réalisé la construction d'un graphe des classes accessibles. Ce graphe comporte 22 classes que nous n'avons pas représenté graphiquement, cependant vous pouvez trouver le rapport d'analyse du logiciel en annexe 4.2.

Vivacité du réseau Dans cette analyse, nous nous référons dans un premier temps à cette bonne propriété. En effet, le résultat obtenu (disponible à la ligne 244) nous indique que les 2 transitions t_5 et t_{13} ne sont jamais franchies. Ceci signifie que les places p_{18} et p_{17} ne sont jamais marquées, car le seul moyen existant pour qu'un jeton soit dans ces places est en franchissant respectivement cette transition. Donc, l'alarme n'est jamais activée.

Graphe des classes Une analyse des places accessibles à partir du graphe des classes nous confirme ce que la vivacité du réseau nous avait indiqué : les places p_{18} et p_{17} ne sont jamais marquées.

Cette analyse nous apporte aussi une information que nous pourrions utiliser dans la prochaine section. Il s'agit de l'intervalle temporelle restante avant le franchissement des transitions t_5 et t_{13} avec lequel il est possible de connaître l'intervalle temporelle passée dans l'opération O_1 et O_2 , respectivement. Nous pouvons observer aux lignes 153 et 194 ces intervalles, elles sont de $[3; 3]$ pour t_3 et de $[9; 9]$ pour t_{13} .

Intervalles de temps des opérations Intéressons nous maintenant au temps d'exécution des opérations. En regardant les intervalles de temps que doivent respecter O_1 et O_2 , nous pouvons connaître le minimum et le maximum de temps admis pour les opérations. De plus, nous avons à notre disposition le temps passé dans les opérations à l'aide des transitions t_3 et t_{13} , en regardant dans le graphe des classes les intervalles de temps durant lesquelles elles peuvent être franchies.

Donc, si l'intervalle temporelle des transitions atteint $d_{i_{max}} - d_{i_{min}}$ u.t, alors la pièce doit être sortie du bac. Nous allons donc, dans la prochaine section, faire en sorte que les dernières intervalles temporelle atteinte pour t_3 et t_{13} soit comprises entre : $[0; d_{i_{max}} - d_{i_{min}}]$.

2.3 Mise au point des Intervalles Temporels

Avec la conclusion précédente, nous avons pu déterminer l'ajustement nécessaire. Toutefois, nous ne pouvons pas caler toutes les intervalles d'un seul coup, nous devons les placer l'une après l'autre, dans l'ordre dans lequel les opérations sont lancées.

Opération O_1 A partir du graphe des classes établi à partir du modèle 2.2, nous avons le dernier intervalle temporelle atteint par la transition t_5 . $d_{1_{min}} = 15$ $d_{2_{min}} = 20$

3

Dans cette partie, l'objet de l'étude se porte sur la réalisation de 6 opérations, 3 par pièces (les opérations o_1, o_3, o_5 pour p_1 et o_2, o_4, o_6 pour p_2). Il faut, dans un premier temps, trouver une séquence optimale telle que toutes les opérations soient effectuées en un temps minimum sans déclencher l'alarme. Ensuite, nous devons réaliser une commande en réseau de Petri temporel pour effectuer les opérations puis l'analyser à l'aide d'un graphe des classes. Dans un dernier temps, nous avons réalisé la commande en langage C.

3.1 Analyse d'ordonnement

Nous avons décidé d'étudier l'ordonnancement optimal par un chronogramme car nous avons trouvé l'approche graphique plus intuitive au vu du faible nombre d'opérations. Nous avons réalisé cet ordonnancement sur Excel en prenant pour échelle *une case égale une seconde*. Le chronogramme (figure 3.1) est coupé en trois parties pour des raisons de visibilité : une ligne est réservée aux opérations liées à la pièce p_2 , une seconde aux actions du charriot et une dernière aux opérations de p_1 .

[illegible]

FIGURE 3.1 – Capture du tableur de calcul de l'ordonnancement.

Nous avons positionné les opérations au plus tôt, en vérifiant qu'elles n'empêchent pas de récupérer les pièces avant leurs alarmes respectives. Par exemple, l'opération o_5 pourrait commencer à partir de la 57^e seconde mais il serait impossible de retirer les deux pièces avant leurs alarmes donc nous avons décalé o_5 de façon à ce que, ni la pose, ni la prise de la pièce p_1 ne perturbe la prise et la pose de p_2 . De plus, nous prenons pour hypothèse que les deux dernières opérations o_5 et o_6 n'ont pas de date récupération maximale et peuvent rester respectivement en e_7 et en e_8 car ces deux emplacements représentant le bac de sortie. Si nous avions pris le parti de les ramener à leurs emplacements initiaux, cela n'aurait pas été possible avec l'ordonnancement actuel, il aurait fallu organiser les opérations différemment de façon à avoir le temps de ramener les pièces au début ce qui nécessite beaucoup de temps à cause de la distance de déplacement importante.

3.2 Réseau de Petri de commande

Nous avons transformé la séquence d'actions de la ligne *Chariot* de l'ordonnancement précédent (voir figure 3.1) en un réseau de Petri temporel.

3.3 Analyse par graphe de classes

3.4 Implémentation

4 | Conclusion

Annexes

Annexe 1 - Mesures de temps

4.1 Mesure du temps de déplacement et de saisie/dépôt d'une pièce

```

1
2 /* ===== */
3 /*      Squelette de programme pour la mise en oeuvre de MEF synchronisees      */
4 /*      Systeme de Traitement Automatique                                     */
5 /* ===== */
6
7 /* COMPILATION : gcc -Wall -o STA sta-macsed_rdp.c -lpci_dask -lsta */
8
9 /* Exemple qui realise ce RdP (1 jeton initialement dans p0):
10
11 p0 -----> | -----> p1 -----> | -----> p2 -----> | ---
12             operateur                FinAvancer
13             Avancer(3,1)                Prendre()
14
15 -----> p3 -----> | -----> p4 -----> | -----> p5 --- | -----> p6
16             [ 5 ; 5 ]                FinReculer
17 ne rien faire !                Reculer(2,1)                Poser()                Fin du RdP !
18
19 */
20
21 #include <stdio.h>
22 #include <unistd.h>                // pour sleep()
23 #include <dask.h>                  // pour Release_Card()
24 #include <signal.h>                // pour deroutement de CTRL C
25 #include <entreesortie_sta.h>
26 #include <sta_mef.h>
27
28 // -----
29 #define NBPLACES 7
30 // -----
31
32 void Avancer (int , int);
33 void Reculer (int , int);
34 void Prendre (void);
35 void Poser (void);
36
37 /* variables externes */
38 short int idcard , stop;
39
40 /* declaration globale */
41 int FinAvancer=0;
42 int FinReculer=0;
43
44 /* etudiant */
45 time_t  ti ,
46 tf;

```



```

47
48 // les entrees
49 int appG, ctr, appD, presence, lim_hor, lim_ver, operateur;
50
51 // les sorties
52 int v_acc, haut, bas, gauche, droite, alarme;
53
54 int main(void)
55 {
56
57 /* Declarations variables */
58
59 // -----
60 int p[NBPLACES]; // places "presentes"
61 int ps[NBPLACES]; // places "suivantes"
62 int i;
63 double fintempo; // variables utilisees pour
64 time_t tempol; // la tempo
65 // -----
66
67 // initialisation des ports
68 init_io();
69 //initialisation bac : ramene le systeme de transport en bas a droite
70 //ne doit pas etre sur les capteurs en haut ou a gauche
71 //init_bac();
72 printf("init faite \n");
73     sortie(V_ACC,0);
74     sortie(HAUT, 0);
75     sortie(BAS,0);
76     sortie(GAUCHE,0);
77     sortie(DROITE,0);
78     sortie(ALARME,0);
79
80 /* Initialisation variables */
81
82 // -----
83 p[0] = 1;
84 ps[0] = 1;
85 for(i=1;i<NBPLACES;i++){
86     p[i] = 0;
87     ps[i] = 0;
88 }
89 fintempo=0;
90 // -----
91 ti = 0;
92 tf = 0;
93
94 /***** Ajout etudiant*/
95 //Poser(); // Position itniale en bas
96 /******
97 while(1)
98 {
99     /******
100     /* Lecture des entrees */
101     /******
102
103     appG = entree(APPG);
104     ctr = entree(CTR);
105     appD = entree(APPD);
106     presence = entree(PRESENCE);
107     lim_hor = entree(LIM_HOR);

```

```

108     lim_ver = entree(LIM_VER);
109     operateur = entree(OPERATEUR);
110
111
112 // -----
113     fintempo = difftime(time(NULL),tempo1);
114 // -----
115
116     /* allongement du cycle programme */
117     // usleep(50);
118
119 // -----
120
121     /* blocs F */ // Description des transitions possibles
122
123     if(p[0]==1){
124         printf("p0 \n");
125         if(operateur==1){ // marquage initial, attente d'un appui sur operateur
126             ps[0]--;
127             ps[1]++;
128             ti = time(NULL);
129         }
130     }
131
132     if(p[1]==1){
133 //         printf("p1 \n"); // FinAvancer est a 1 lorsque le capteur ctr est a 1
134         if(FinAvancer==1){ // et qu'on a parcouru toutes les encoches demandees
135             ps[1]--; // Exemple: 3 encoches parcourues si Avancer(2,1)
136             ps[2]++;
137             tf = time(NULL);
138             printf("Avance Init --> e1 : %f \n",difftime(tf,ti));
139             Avancer(0,0); // Remise a zero de la machine a etats definie dans la
                           // fonction Avancer
140
141
142             ti = time(NULL);
143         }
144     }
145
146     if(p[2]==1){
147
148     /*
149 //         printf("p2 \n"); // Passage de e1 vers e2
150         if(FinAvancer==1){
151             ps[2]--;
152             ps[3]++;
153             tf = time(NULL);
154             printf("Avance e1 --> e2 : %f \n",difftime(tf,ti));
155             Avancer(0,0); // Remise a zero de la machine a etats definie dans la
                           // fonction Avancer
156             // ti = time(NULL); Inutile dans ce cas la
157         } */
158     }
159
160     if(p[3]==1){
161 //         printf("p3 \n"); // Passage de e2 vers e3
162         if(FinAvancer==1){
163             ps[3]--;
164             ps[4]++;
165             Avancer(0,0); // Remise a zero de la machine a etats definie dans la
                           // fonction Avancer

```

```

166
167     ti = time(NULL);
168 }
169
170 }
171
172 if(p[4]==1){
173 //     printf("p4 \n"); // Passage de e3 vers e4
174     if(FinAvancer==1){
175         ps[4]--;
176         ps[5]++;
177         tf = time(NULL);
178         printf("Avance e3 --> e4 : %f \n",difftime(tf,ti));
179         Avancer(0,0); // Remise a zero de la machine a etats definie dans la
                        fonction Avancer
180         ti = time(NULL);
181     }
182 }
183
184 if(p[5]==1){
185 //     printf("p5 \n"); // Prendre un objet
186     ps[5]--;
187     ps[6]++;
188 }
189
190 if(p[6]==1) // --> fin du Reseau de Petri
191 { // Poser un objet
192 //     printf("p5 \n"); // Prendre un objet
193     ps[6]--;
194     //ps[0]++;
195 }
196
197 /* blocs M */ // Franchissement des transitions
198
199 /* if(p[3]==0 && ps[3]==1)
200 { // du coup, lancement des actions liees au franchissement des transitions
201     tempo1 = time(NULL); // (ici, on recupere le "temps actuel" au moment ou
                        on arrive dans la place p3
202 }
203 // pour pouvoir ensuite le comparer avec le temps actuel
                        recupere a chaque passage
204 */ // dans la boucle while -> cf. lecture des entrees et ligne
                        "if(fintempo>=5)"
205
206 for(i=0;i<NBPLACES;i++){ // et actualisation des etats presents
207     p[i] = ps[i];
208 }
209 // -----
210
211 /*-----*/
212 /* Ecriture des sorties */
213 /*-----*/
214
215 // -----
216
217 /* blocs G */ // gestion des sorties en fonction du marquage mis a jour
218
219 if(p[1]==1){
220     Avancer(1,1);
221 }
222 // if(p[2]==1){

```

```

223 //      Avancer (1,1);
224 //      }
225
226      if (p[3]==1){
227          Avancer (1,1);
228      }
229
230      if (p[4]==1){
231          Avancer (1,1);
232      }
233      if (p[2]==1){
234          ti = time(NULL);
235          Poser ();
236      tf = time(NULL);
237      printf ("Poser: %f \n", difftime (tf , ti));
238      }
239
240      if (p[6]==1){
241          ti = time(NULL);
242          Poser ();
243      tf = time(NULL);
244      printf ("Poser: %f \n", difftime (tf , ti));
245      }
246
247
248 // -----
249
250      sortie (V_ACC,0);
251      // sortie (HAUT, haut); // actions haut/bas gereses dans les fonctions
252      // sortie (BAS, bas); // Prendre et Poser ci-dessous
253      sortie (GAUCHE, gauche);
254      sortie (DROITE, droite);
255      sortie (ALARME,0);
256      /**COMMENTAIRE eTUDIANT*/
257 //printf ("haut: %d — bas : %d — gauche : %d — droite : %d |n", haut, bas,
    gauche, droite);
258 }
259
260      sortie (V_ACC,0);
261      sortie (HAUT, 0);
262      sortie (BAS,0);
263      sortie (GAUCHE,0);
264      sortie (DROITE,0);
265      sortie (ALARME,0);
266
267 return 0;
268 }
269
270 void Prendre (void)
271 {printf ("prise d'un objet\n");
272  sortie (HAUT, 1);
273  while (entree (LIM_VER));
274  //printf ("prise d'un objet2 |n");
275  usleep (10);
276  while (!entree (LIM_VER));
277  //printf ("prise d'un objet3 |n");
278  sortie (HAUT, 0);
279  }
280
281 void Poser (void)
282 {printf ("pose d'un objet\n");

```

```

283  sortie (BAS,1);
284  while (entree(LIM_VER));
285  while (!entree(LIM_VER));
286  sortie (BAS,0);
287  }
288
289 void Avancer (int nb, int start)
290 {
291     static int Etat = 0;
292     static int i = 1;
293     switch (Etat)
294     {
295         case 0 : if (start)
296                 {
297                     Etat = 1;
298                     i=nb;
299                 }
300                 break;
301         case 1 : if (! ctr)
302                 {
303                     Etat = 2;
304                 }
305                 break;
306         case 2 : if (ctr &&(i==1))
307                 {
308                     Etat = 3;
309                     FinAvancer=1;
310                 }
311                 else if (ctr &&(i!=1))
312                 {
313                     Etat=1; i--;
314                 }
315                 break;
316         case 3 : if (! start)
317                 {
318                     Etat = 0;
319                     FinAvancer = 0;
320                 }
321                 break;
322     }
323
324 gauche = ((Etat==1)|| (Etat==2));
325 //printf ("EtatAvancer : %d — gauche : %d\n",Etat, gauche);
326 }
327
328 void Reculer (int nb, int start)
329 {
330     static int Etat = 0;
331     static int i = 1;
332     switch (Etat)
333     {
334         case 0 : if (start)
335                 {
336                     Etat = 1;
337                     i=nb;
338                 }
339                 break;
340         case 1 : if (! ctr)
341                 {
342                     Etat = 2;
343                 }

```

```

344         break;
345     case 2 : if (ctr &&(i==1))
346     {
347         Etat = 3;
348         FinReculer=1;
349     }
350     else if (ctr &&(i!=1))
351     {
352         Etat=1; i--;
353     }
354     break;
355     case 3 : if (! start)
356     {
357         Etat = 0;
358         FinReculer = 0;
359     }
360     break;
361 }
362 droite = ((Etat==1)|| (Etat==2));
363 }

```

Mesure de x , le temps pour avancer d'un emplacement et de y , le temps pour poser ou prendre une pièce.

4.2 Mesure du temps

```

1
2 /* ===== */
3 /*      Squelette de programme pour la mise en oeuvre de MEF synchronisees      */
4 /*      Systeme de Traitement Automatique                                     */
5 /* ===== */
6
7 /* COMPILATION : gcc -Wall -o STA sta-macsed_rdp.c -lpci_dask -lsta */
8
9 /* Exemple qui realise ce RdP (1 jeton initialement dans p0):
10
11 p0 -----> | -----> p1 -----> | -----> p2 -----> | ---
12             operateur                FinAvancer
13                               Avancer(3,1)                Prendre()
14
15 -----> p3 -----> | -----> p4 -----> | -----> p5 --- | -----> p6
16             [ 5 ; 5 ]                FinReculer
17 ne rien faire !                Reculer(2,1)                Poser()                Fin du RdP !
18
19 */
20
21 #include <stdio.h>
22 #include <unistd.h>                // pour sleep()
23 #include <dask.h>                  // pour Release_Card()
24 #include <signal.h>                // pour deroutement de CTRL C
25 #include <entreesortie_sta.h>
26 #include <sta_mef.h>
27
28 // -----
29 #define NBPLACES 7
30 // -----
31
32 void Avancer (int , int);
33 void Reculer (int , int);
34 void Prendre (void);
35 void Poser (void);
36

```

```

37 /* variables externes */
38 short int idcard, stop;
39
40 /* declaration globale */
41 int FinAvancer=0;
42 int FinReculer=0;
43
44 /* etudiant */
45 time_t ti,
46 tf;
47
48 // les entrees
49 int appG,ctr, appD,presence, lim_hor,lim_ver, operateur;
50
51 // les sorties
52 int v_acc, haut, bas, gauche, droite, alarme;
53
54 int main(void)
55 {
56
57 /* Declarations variables */
58
59 // -----
60 int p[NBPLACES]; // places "presentes"
61 int ps[NBPLACES]; // places "suivantes"
62 int i;
63 double fintempo; // variables utilisees pour
64 time_t tempol; // la tempo
65 // -----
66
67 // initialisation des ports
68 init_io();
69 //initialisation bac : ramene le systeme de transport en bas a droite
70 //ne doit pas etre sur les capteurs en haut ou a gauche
71 init_bac();
72 printf("init faite \n");
73 sortie(V_ACC,0);
74 sortie(HAUT, 0);
75 sortie(BAS,0);
76 sortie(GAUCHE,0);
77 sortie(DROITE,0);
78 sortie(ALARME,0);
79
80 /* Initialisation variables */
81
82 // -----
83 p[0] = 1;
84 ps[0] = 1;
85 for(i=1;i<NBPLACES;i++){
86 p[i] = 0;
87 ps[i] = 0;
88 }
89 fintempo=0;
90 // -----
91 ti = 0;
92 tf = 0;
93
94 while(1)
95 {
96 /******
97 /* Lecture des entrees */

```

```

98      /*******/
99
100     appG = entree (APPG);
101     ctr = entree (CTR);
102     appD = entree (APPD);
103     presence = entree (PRESENCE);
104     lim_hor = entree (LIM_HOR);
105     lim_ver = entree (LIM_VER);
106     operateur = entree (OPERATEUR);
107
108
109 // -----
110 fintempo = difftime (time (NULL) ,tempo1);
111 // -----
112
113     /* allongement du cycle programme */
114     // usleep (50);
115
116 // -----
117
118     /* blocs F */    // Description des transitions possibles
119
120     if (p[0]==1){
121         printf ("p0 \n");
122         if (operateur==1){ // marquage initial , attente d'un appui sur operateur
123             ps[0]--;
124             ps[1]++;
125
126         }
127     }
128
129     if (p[1]==1){
130         printf ("p1 \n"); // FinAvancer est a 1 lorsque le capteur ctr est a 1
131         if (FinAvancer==1){ // et qu'on a parcouru toutes les encoches demandees
132             ps[1]--; // Exemple: 3 encoches parcourues si Avancer (2,1)
133             ps[2]++;
134
135             Avancer (0,0); // Remise a zero de la machine a etats definie dans la
                             // fonction Avancer
136
137         }
138     }
139 }
140
141     if (p[2]==1){
142         printf ("p2 \n");
143
144         ps[2]--;
145         ps[3]++;
146     }
147
148     if (p[3]==1){
149         printf ("p3 \n");
150
151         if (fintempo>=5){ // attente de 5 secondes avant de passer a la place 4
152             ps[3]--;
153             ps[4]++;
154         }
155     }
156
157     if (p[4]==1){

```



```

158     printf("p4 \n"); // FinAvancer est a 1 lorsque le capteur ctr est a 1
159     if(FinReculer==1){ // et qu'on a parcouru toutes les encoches demandees
160         ps[4]--; // (exemple : 2 encoches parcourues si on a ecrit
161             Reculer(2,1) )
162         ps[5]++;
163         Reculer(0,0); // Remise a zero de la machine a etats definie dans la
164             fonction Reculer
165     }
166
167     if(p[5]==1){
168         printf("p5 \n");
169         ps[5]--;
170         ps[6]++;
171         if (ti == 0)
172         {
173             //ti = time(NULL);
174         }
175     }
176
177     if(p[6]==1) // --> fin du Reseau de Petri
178     {
179         if(tf == 0)
180         {
181             //tf = time(NULL);
182             //printf("P6 : poser : %f \n", difftime(tf, ti));
183         }
184     }
185
186     /* blocs M */ // Franchissement des transitions
187
188     if(p[3]==0 && ps[3]==1)
189     { // du coup, lancement des actions liees au franchissement des transitions
190         tempol = time(NULL); // (ici, on recupere le "temps actuel" au moment ou
191             on arrive dans la place p3
192     } // pour pouvoir ensuite le comparer avec le temps actuel
193         recupere a chaque passage
194         // dans la boucle while -> cf. lecture des entrees et ligne
195             "if(fintempo>=5)"
196
197     for(i=0;i<NBPLACES;i++){ // et actualisation des etats presents
198         p[i] = ps[i];
199     }
200 // -----
201
202     /* Ecriture des sorties */
203     /* -----
204 // -----
205
206     /* blocs G */ // gestion des sorties en fonction du marquage mis a jour
207
208     if(p[1]==1){
209         if (ti == 0)
210         {
211             ti = time(NULL);
212             Avancer(8,1);
213         }

```

```

214     if (p[2]==1){
215         Prendre();
216     }
217
218     if (p[4]==1){
219         Reculer(2,1);
220     }
221
222     if (p[5]==1){
223         Poser();
224     }
225 // -----
226
227     sortie(V_ACC,0);
228     // sortie(HAUT, haut); // actions haut/bas gereses dans les fonctions
229     // sortie(BAS,bas); // Prendre et Poser ci-dessous
230     sortie(GAUCHE, gauche);
231     sortie(DROITE, droite);
232     sortie(ALARME,0);
233
234 printf ("haut: %d — bas : %d — gauche : %d — droite : %d \n",haut,bas,
gauche, droite);
235 }
236
237     sortie(V_ACC,0);
238     sortie(HAUT, 0);
239     sortie(BAS,0);
240     sortie(GAUCHE,0);
241     sortie(DROITE,0);
242     sortie(ALARME,0);
243
244 return 0;
245 }
246
247 void Prendre (void)
248 {printf ("prise d'un objet1\n");
249     sortie(HAUT, 1);
250     while (entree(LIM_VER));
251     printf ("prise d'un objet2\n");
252     usleep(10);
253     while (!entree(LIM_VER));
254     printf ("prise d'un objet3\n");
255     sortie(HAUT, 0);
256 }
257
258 void Poser (void)
259 {printf ("pose d'un objet\n");
260     sortie(BAS,1);
261     while (entree(LIM_VER));
262     while (!entree(LIM_VER));
263     sortie(BAS,0);
264 }
265
266 void Avancer (int nb, int start)
267 {
268     static int Etat = 0;
269     static int i = 1;
270     switch (Etat)
271     {
272     case 0 : if (start)
273         {

```

```

274         Etat = 1;
275         i=nb;
276     }
277     break;
278     case 1 : if (! ctr)
279     {
280         Etat = 2;
281     }
282     break;
283     case 2 : if (ctr &&(i==1))
284     {
285         Etat = 3;
286         FinAvancer=1;
287         tf = time(NULL);
288         printf("Avvvvvvvvvvvvvvancer : %f \n",difftime(tf,ti));
289     }
290     else if (ctr &&(i!=1))
291     {
292         Etat=1; i--;
293     }
294     break;
295     case 3 : if (! start)
296     {
297         Etat = 0;
298         FinAvancer = 0;
299     }
300     break;
301 }
302
303 gauche = ((Etat==1)|| (Etat==2));
304 printf ("EtatAvancer : %d — gauche : %d\n",Etat , gauche);
305 }
306
307 void Reculer (int nb, int start)
308 {
309     static int Etat = 0;
310     static int i = 1;
311     switch (Etat)
312     {
313     case 0 : if (start)
314     {
315         Etat = 1;
316         i=nb;
317     }
318     break;
319     case 1 : if (! ctr)
320     {
321         Etat = 2;
322     }
323     break;
324     case 2 : if (ctr &&(i==1))
325     {
326         Etat = 3;
327         FinReculer=1;
328     }
329     else if (ctr &&(i!=1))
330     {
331         Etat=1; i--;
332     }
333     break;
334     case 3 : if (! start)

```

```

335         {
336             Etat = 0;
337             FinReculer = 0;
338         }
339         break;
340     }
341 droite = ((Etat==1) || (Etat==2));
342 }

```

Mesure du temps de traversé de bout en bout.

Annexe 2 - Analyse TINA

4.1 Analyse d'accessibilité du modèle 2 opérations

```
.5
Tina version 2.8.4 — 10/27/06 — LAAS/CNRS

mode -W
4 INPUT NET —————

parsed net {reseau-AnalyseIII-2}

25 places , 23 transitions

net {reseau-AnalyseIII-2}
tr t0 p0 -> p1
tr t1 [1,1] p1 -> p2
14 tr t10 [1,1] p21 -> p22
tr t11 [3,3] p8 -> p23
tr t12 [0,0] p22 -> p5
tr t13 [22,22] p19 -> p17
tr t14 [0,0] p24 -> p13
tr t15 [0,0] p20 p6 -> p7
tr t16 [2,2] p7 -> p8
tr t17 : FinReculer [1,1] p23 -> p24
tr t18 [0,0] p14 p19 -> p15
tr t19 [2,2] p15 -> p16
24 tr t2 [3,3] p2 -> p3
tr t20 [3,3] p16 ->
tr t21 [3,3] p5 -> p6
tr t22 [3,3] p13 -> p14
tr t3 [2,2] p3 -> p4
tr t4 [3,3] p4 -> p20 p9
tr t5 [16,16] p20 -> p18
tr t6 [1,1] p9 -> p10
tr t7 [3,3] p10 -> p11
tr t8 [2,2] p11 -> p12
34 tr t9 [3,3] p12 -> p19 p21
pl p0 (1)
pl p1 : {Avancer(1,1)}
pl p10 : {Prendre();}
pl p11 : {Avancer(2,1);}
pl p12 : {Poser();}
pl p13 : {Prendre();}
pl p15 : {Avancer(2,1);}
pl p16 : {Poser();}
pl p17 : {Alarme();}
44 pl p18 : {Alarme();}
pl p2 : {Prendre();}
pl p21 : {Reculer(1,1);}
pl p23 : {Reculer(1,1);}
pl p3 : {Avancer(2,1);}
pl p4 : {Poser();}
pl p5 : {Prendre();}
pl p7 : {Avancer(2,1);}
pl p8 : {Poser();}
pl p9 : {Reculer(1,1)}
54 0.000 s

REACHABILITY ANALYSIS —————
```

```

bounded

22 classe(s), 21 transition(s)

CLASSES:
64 class 0
    marking
    p0
    domain
    0 <= t0

    class 1
    marking
    p1
74    domain
    1 <= t1 <= 1

    class 2
    marking
    p2
    domain
    3 <= t2 <= 3

    class 3
84    marking
    p3
    domain
    2 <= t3 <= 2

    class 4
    marking
    p4
    domain
    3 <= t4 <= 3
94

    class 5
    marking
    p20 p9
    domain
    16 <= t5 <= 16
    1 <= t6 <= 1

    class 6
    marking
104    p10 p20
    domain
    15 <= t5 <= 15
    3 <= t7 <= 3

    class 7
    marking
    p11 p20
    domain
    12 <= t5 <= 12
114    2 <= t8 <= 2

    class 8
    marking
    p12 p20
    domain
    10 <= t5 <= 10
    3 <= t9 <= 3

    class 9
124    marking
    p19 p20 p21
    domain
    1 <= t10 <= 1
    22 <= t13 <= 22
    7 <= t5 <= 7

    class 10
    marking
    p19 p20 p22
134    domain

```

```

0 <= t12 <= 0
21 <= t13 <= 21
6 <= t5 <= 6

class 11
    marking
    p19 p20 p5
    domain
    21 <= t13 <= 21
144 3 <= t21 <= 3
    6 <= t5 <= 6

class 12
    marking
    p19 p20 p6
    domain
    18 <= t13 <= 18
    0 <= t15 <= 0
    3 <= t5 <= 3
154

class 13
    marking
    p19 p7
    domain
    18 <= t13 <= 18
    2 <= t16 <= 2

class 14
    marking
164 p19 p8
    domain
    3 <= t11 <= 3
    16 <= t13 <= 16

class 15
    marking
    p19 p23
    domain
    13 <= t13 <= 13
174 1 <= t17 <= 1

class 16
    marking
    p19 p24
    domain
    12 <= t13 <= 12
    0 <= t14 <= 0

class 17
184    marking
    p13 p19
    domain
    12 <= t13 <= 12
    3 <= t22 <= 3

class 18
    marking
    p14 p19
    domain
194 9 <= t13 <= 9
    0 <= t18 <= 0

class 19
    marking
    p15
    domain
    2 <= t19 <= 2

class 20
204    marking
    p16
    domain
    3 <= t20 <= 3

class 21
    marking

```

domain

214

REACHABILITY GRAPH:

```
0 -> t0 in [0,w[/1
1 -> t1 in [1,1]/2
2 -> t2 in [3,3]/3
3 -> t3 in [2,2]/4
4 -> t4 in [3,3]/5
5 -> t6 in [1,1]/6
6 -> t7 in [3,3]/7
224 7 -> t8 in [2,2]/8
8 -> t9 in [3,3]/9
9 -> t10 in [1,1]/10
10 -> t12 in [0,0]/11
11 -> t21 in [3,3]/12
12 -> t15 in [0,0]/13
13 -> t16 in [2,2]/14
14 -> t11 in [3,3]/15
15 -> t17 in [1,1]/16
16 -> t14 in [0,0]/17
234 17 -> t22 in [3,3]/18
18 -> t18 in [0,0]/19
19 -> t19 in [2,2]/20
20 -> t20 in [3,3]/21
21 ->
```

0.000 s

LIVENESS ANALYSIS

244 not live

```
1 dead classe(s), 1 live classe(s)
2 dead transition(s), 0 live transition(s)
```

dead classe(s): 21

dead transition(s): t5 t13

STRONG CONNECTED COMPONENTS:

254

```
21 : 0
20 : 1
19 : 2
18 : 3
17 : 4
16 : 5
15 : 6
14 : 7
13 : 8
264 12 : 9
11 : 10
10 : 11
9 : 12
8 : 13
7 : 14
6 : 15
5 : 16
4 : 17
3 : 18
274 2 : 19
1 : 20
0 : 21
```

SCC GRAPH:

```
21 -> t0/20
20 -> t1/19
19 -> t2/18
18 -> t3/17
284 17 -> t4/16
16 -> t6/15
15 -> t7/14
```



```

14 -> t8/13
13 -> t9/12
12 -> t10/11
11 -> t12/10
10 -> t21/9
9 -> t15/8
8 -> t16/7
294 7 -> t11/6
6 -> t17/5
5 -> t14/4
4 -> t22/3
3 -> t18/2
2 -> t19/1
1 -> t20/0
0 ->

```

```

0.000 s

```

```

304 ANALYSIS COMPLETED _____

```

Analyse d'accessibilité du modèle 2 opérations