Chapitre 2 : Les techniques de spécification.

1. Introduction

Tout produit complexe à construire doit d'abord être spécifié ; par exemple un pont de 30 mètres de long, supportant au moins 1000 tonnes, construit en béton, etc. Ces spécifications peuvent être considérées comme un *contrat* entre un client (la collectivité qui veut réaliser le pont) et un producteur (l'entreprise de génie civil).

En informatique, le client et le producteur peuvent être différents selon les phases du cycle de vie.

La *spécification des besoins* ou *spécification des exigences* ('requirements') est un contrat entre les futurs utilisateurs et les concepteurs. Elle concerne les caractéristiques attendues (exigences fonctionnelles et non fonctionnelles : efficacité, taille, sûreté, sécurité, portabilité, etc.). Elle intervient pendant la phase d'Analyse des besoins et se rédige en langue naturelle.

La *spécification d'un système* est un contrat entre les futurs utilisateurs et les concepteurs. Elle concerne *la nature des fonctions offertes*, *les comportements souhaités*, *les données nécessaires*, etc. Elle intervient pendant la phase d'Analyse du système.

La *spécification d'une architecture de système* est un contrat entre les concepteurs et les réalisateurs. Elle intervient pendant la phase de Conception générale. Elle définit l'architecture en modules de l'application à réaliser.

La *spécification technique* d'un module, d'un programme, d'une structure de données ou d'un type de données est un contrat entre le programmeur qui l'implante et les programmeurs qui l'utilisent. Elle intervient pendant la phase de Conception détaillée.

De manière générale une spécification décrit les caractéristiques attendues (le *quoi*) d'une implantation (le *comment*).

Dans cette partie nous traitons essentiellement de la spécification d'un système en termes de fonctions, de données, de comportement.

Il est souhaitable qu'une spécification soit claire, non ambiguë et compréhensible. Les descriptions en langue naturelle manquent souvent de précision.
Les spécifications doivent aussi être cohérentes (pas de contradictions) et complètes.
La complétude peut prendre deux formes :

'interne', c'est à dire que tous les concepts utilisés sont clairement spécifiés,

et 'externe', par rapport à la réalité décrite. Cette seconde forme est quelque peu illusoire dans la pratique. On ne peut pas en général spécifier tous les détails ou tout le 'monde' qui entoure un système.

2. Les styles de spécification

Il y a deux critères de classification orthogonaux :

la formalité : on distingue des spécifications informelles (en langue naturelle), semi formelles (souvent graphiques, dont la sémantique est plus ou moins précise), formelles (quand la syntaxe et la sémantiques sont définies formellement par des outils mathématiques).

le caractères opérationnel ou déclaratif : les spécifications opérationnelles décrivent le *comportement désiré* par un modèle qui permet d'une certaine manière de le simuler; par opposition, les spécifications déclaratives décrivent seulement les *propriétés désirées*.

3. Des techniques de spécification pour les phases d'analyse

3.1. Les spécifications en langue naturelle

Elles sont très souples, conviennent pour tous les aspects, sont très facilement communicables à des non spécialistes. Mais elles manquent de structuration, de précision et sont difficiles à analyser.
Des efforts peuvent être faits pour les structurer (spécifications standardisées) : chapitres, sections, items, justifications ('rationale'), etc.

Exemple :

3.5.1 Adding nodes to a design

3.5.1.1 To add a node, the user selects the appropriate node type from the entity type menu. He or she then moves the mouse so that the cursor is placed within the drawing area. On entering the drawing area, the cursor shape should change.
Rationale:
The editor mut know the node type so that it can draw the correct shape and invoke the appropriate checks for that node. The cursor shape change indicates that the editor is in 'node drawing mode'.

3.5.1.2 The user moves the cursor to the approximate node position and any mouse button is pressed. This should cause the node symbol, in a standard size set up by the symbol definer, to appear surrounding the cursor. It may then be dragged by mo ving the mouse, with the button depressed, to its required position. Releasing the mouse button fixes the node position and highlights the node.
Rationale:
The user is the best person to decide where to position a node on the diagram. This approach gives the user direct control.

3.5.1.3 If the entity type is such that its symbol may be varied in size, the fact that the symbol size is variable should be indicated as part of the node highlighting.

Specification: ECLIPSE/WORKSTATION_TOOLS/DE/FS. Section 3.5.1

3.2. Les spécifications techniques dans des langages spécialisés ou des langages informatiques de haut niveau

Des langages semi formels spécialisés pour spécifier des systèmes ont été proposés. Ils comportent des sections et champs prédéfinis, ce qui force à une certaine structuration.

Certains utilisent aussi des langages de haut niveau comme des 'pseudo codes' pour décrire les fonctionnalités attendues.

Exemple : spécification en ADA de l'exemple précédent.

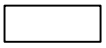
```
-- the procedure Add_to_design describes the actions
-- which should take place when a new node is added
-- to the design.
procedure Add_to_design is
begin
    -- Select the type of node to be placed on the design
    -- A menu of known nodes types is available and the
    -- selection is made from this.
```

```
Node_type := Select_node_type(Node_type_menu);
-- the cursor should now be moved to the drawing area
loop
        Cursor_position := Track_cursor;
        exit when Inside_drawing_area (Cursor_position);
end loop;
-- when the cursor is in the drawing area, change its shape
-- to a circle
Change_cursor_shape (Circle);
-- find the position where the node is to be inserted. This is
-- indicated by pressing the mouse button
loop
        Position := track_cursor;
        exit when Mouse_button_pressed;
end loop;
-- draw the shape to be inserted
Draw_shape (Node_type, Position);
-- the user may drag the shape to another position. Its
-- final position is when the mouse button is released
loop
        Position := Track_cursor;
        Draw_outline (Position);
        Draw_shape (Node_type, PosItion);
        exit when Mouse_button_released;
end loop;
-- Once a node has been placed on a diagram, enter its
-- details in the design database
Add_new_node_to_design_database (Node_type, Position);
-- if the symbol size can be changed, indicate this
-- by drawing stretch blobs on the symbol otherwise
-- highlight it in some way.
if Node_symbol_is_stretchy (Node_type) then
        Draw_stretch_blobs (Node_type, Position);
else
        Highlight_node (Node_type, Position);
end if;

end Add_to_design;
```

3.3. Les diagrammes de flots de données (DFD)

Il s'agit d'une technique semi-formelle et opérationnelle. Les DFD décrivent des collections de données manipulées par des fonctions. Les données peuvent être persistantes (dans des stockages) ou circulantes (flots de données).
La représentation graphique classique distingue :

les *fonctions* par des cercles 
les *stockages* par des boîtes ouvertes 
les *flots* par des flèches
les *entités externes* par des rectangles 

Au niveau le plus abstrait, on peut se contenter des entités à l'interface ('acteurs') et des flots qu'ils s'échangent, sans décomposition en fonctions. On parle alors de '*diagramme de contexte*'. En faisant apparaître les fonctions et en les *raffinant* de plus en plus, on obtient des DFD à différents niveaux d'abstraction.

La figure 1 donne un exemple de DFD, concernant la sélection des réponses à un appel d'offre. Il s'agit du diagramme de contexte.

Le diagramme suivant (Fig. 2) est un premier raffinement du diagramme de contexte.

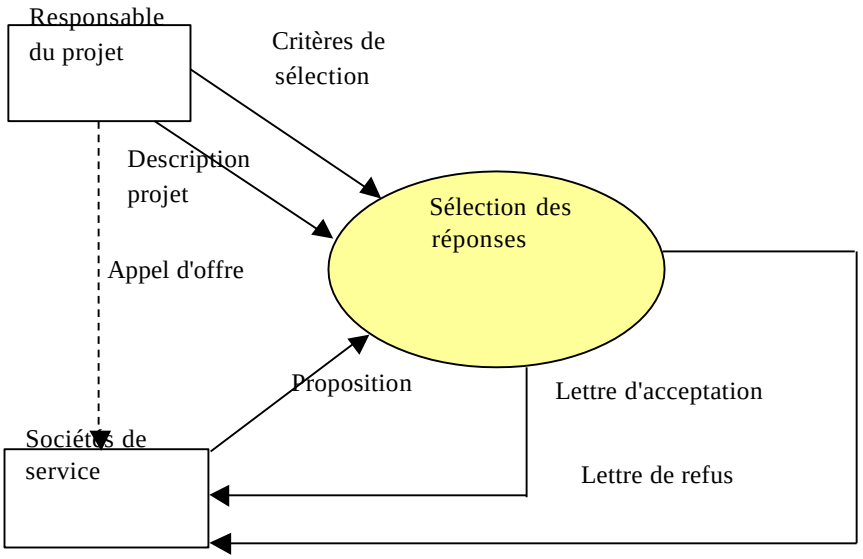


Fig.1. Exemple de diagramme de contexte

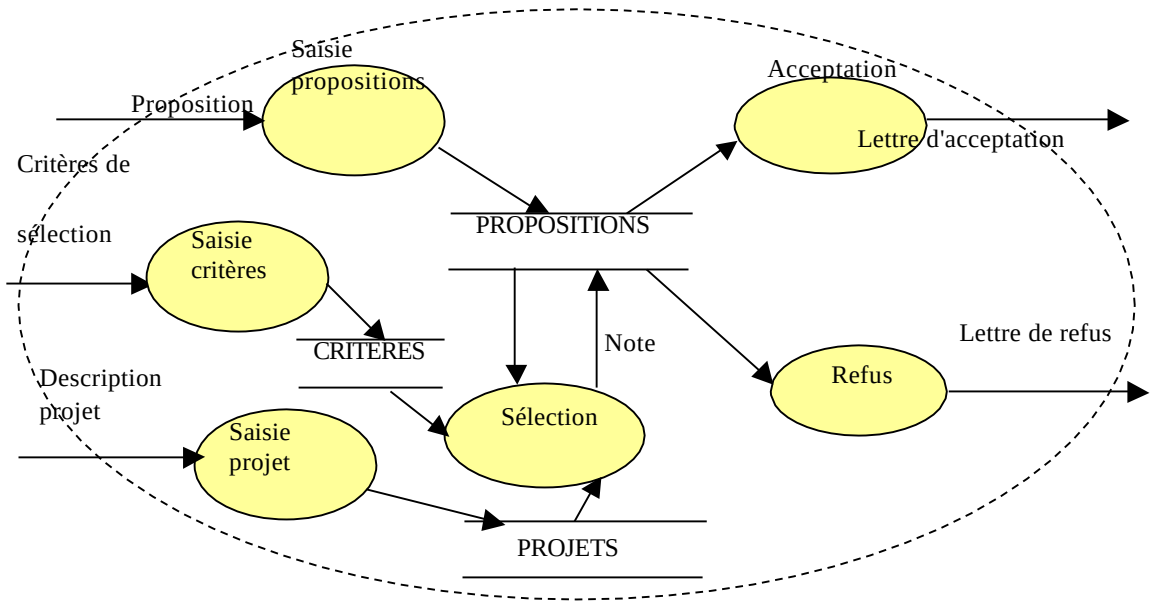
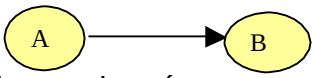


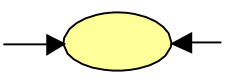
Fig.2. Raffinement du DFD précédent (les flèches entrantes et sortantes doivent être les mêmes)

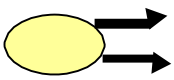
Les DFD sont une notation semi formelle, car la sémantique des fonctions n'est pas spécifiée précisément (que signifie exactement 'Evaluation' ?), ni les aspects liés au contrôle (ou séquencement des opérations).
Exemple : plusieurs interprétations sont possibles pour le DFD élémentaire suivant



A produit une donnée et attend que B la traite pour en produire une autre, ou A et B sont des processus autonomes avec un tampon entre eux.

Les DFD peuvent être analysés à le recherche de formes 'pathologiques', comme le 'trou noir', le 'générateur spontané', etc.





Pour ces raisons les DFD sont soit complétés par d'autres spécifications, soit étendus : flots de contrôle, tampons, etc. Ils connaissent un très grand succès pour spécifier les fonctions d'un système à cause de leur *simplicité et de leur facilité de compréhension par des non informaticiens*.

3.4. Les machines à états finis

C'est une technique formelle et opérationnelle très largement répandu pour décrire les aspects *liés au contrôle*. Il consiste en :

un ensemble fini d'*états* (S),
un ensemble fini d'*entrées* (I),
une fonction de *transition* t : S x I -> S ; c'est une fonction partielle. Une certaine entrée dans un certain état fait passer à un autre état.

Graphiquement une machine à états finis est représenté par un graphe (appelé *diagramme d'états*) dont les noeuds sont les états. Un arc nommé a va de s1 à s2 si et seulement si t(s1,a)=s2. La figure 3 donne comme exemple, le diagramme d'états d'un appel téléphonique.

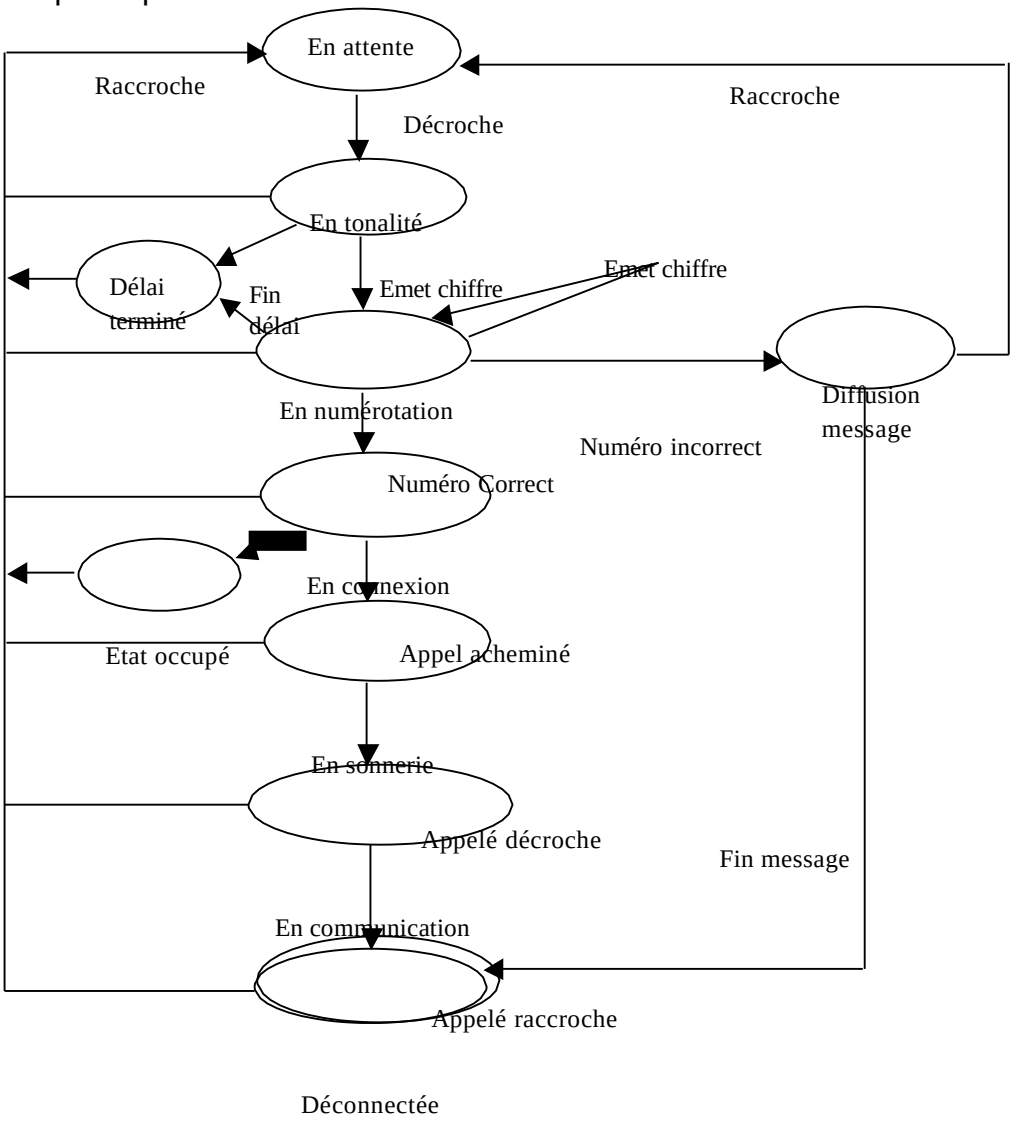


Fig.3. Diagramme d'états d'une ligne téléphonique.

Le modèle est souvent enrichi de concepts supplémentaires : un état initial (s_0 S),

des états finaux (F S), notés souvent par un double cercle, des signaux de *sortie* O (t : SxI -> SxO), notés souvent par <e,s> sur les arcs (l'entrée e dans un certain état fait passer à un autre état avec production de la sortie s).

Les limitations de ce modèle sont évidentes. Dans les systèmes *complexes* le nombre des états peut être énorme et la modélisation complète irréalisable. La seule possibilité est d'abstraire de nombreux détails. Par ailleurs, il s'agit d'un modèle *synchrone* ; c'est à dire qu'à tout instant *un état global unique doit être défini et une seule transition peut survenir*. Il est donc très mal adapté à la description de systèmes asynchrones, où plusieurs composants évoluent en parallèle de manière assez autonome, comme le montre l'exemple suivant du producteur/consommateur.

Exemple : modélisation un système producteur consommateur avec un tampon à deux places ; la modélisation du producteur (2 états), du consommateur (2 états) et du tampon (3 états) séparément sont simples (cf. Fig.4). La modélisation complète nécessite un grand nombre d'états (2 x 2 x 3 = 12 états -cf. Fig.5); on parle *d'explosion combinatoire* dans les cas complexes ; un état <1, P2, C1> signifie que le tampon est dans l'état 1 (contient 1 message), le producteur dans l'état P2 (il vient de produire) et le consommateur dans l'état C1 (il vient de consommer).

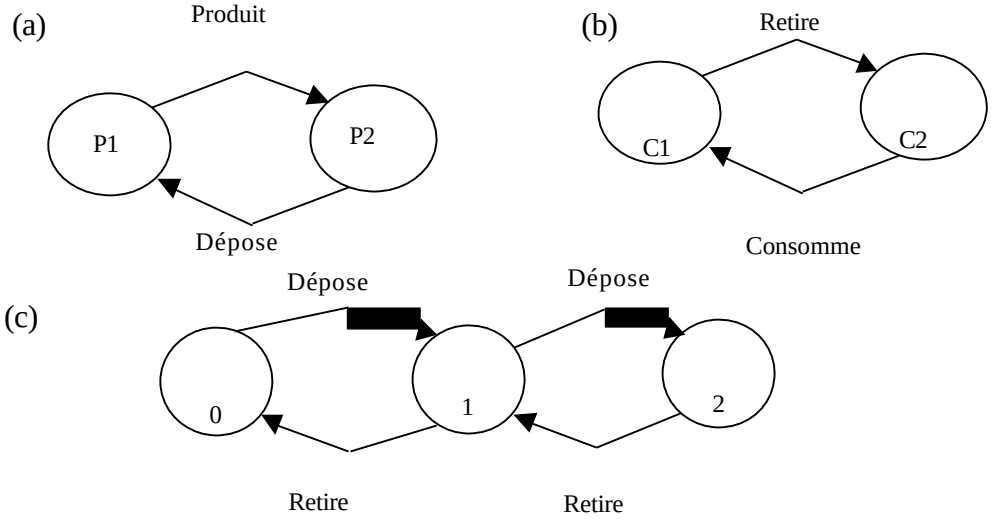


Fig.4. Modélisations séparées du producteur(a), du consommateur(b), du tampon(c)

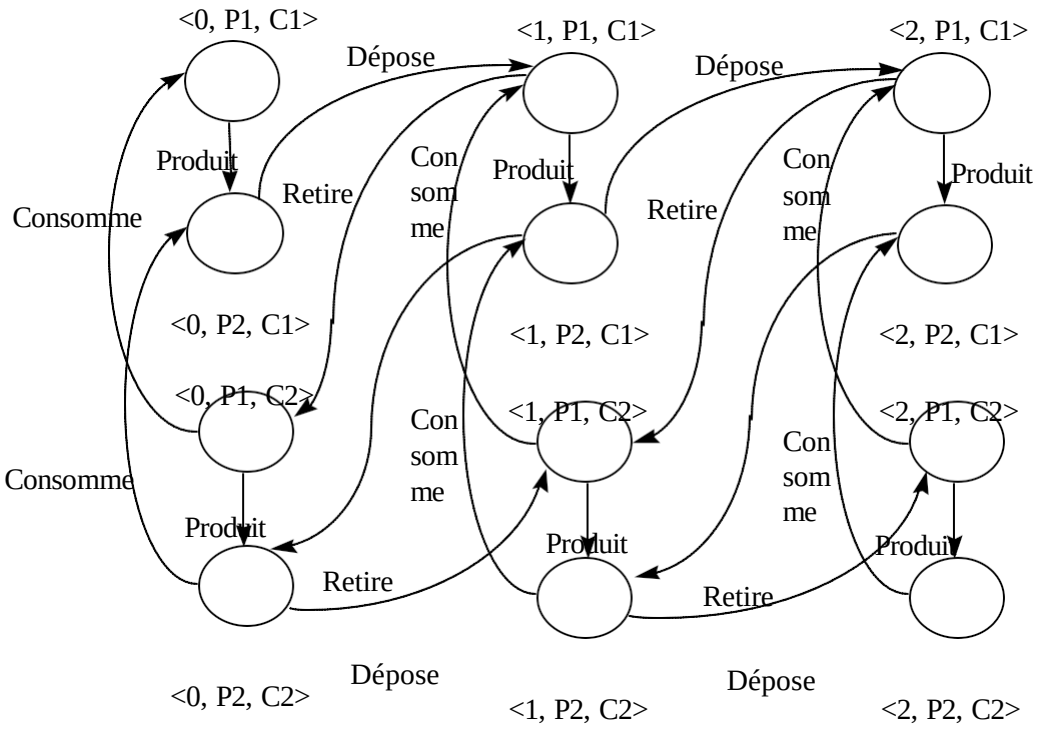


Fig.5. Modélisation complète du producteur consommateur

Il faut noter que des modèles de machines à états finis structurées ont été proposés qui contournent cette difficulté. C'est le cas en particulier des 'statecharts' de Harel (repris dans UML). Les machines à état finis sont très utilisées pour la spécification du comportement des composants élémentaires des systèmes.

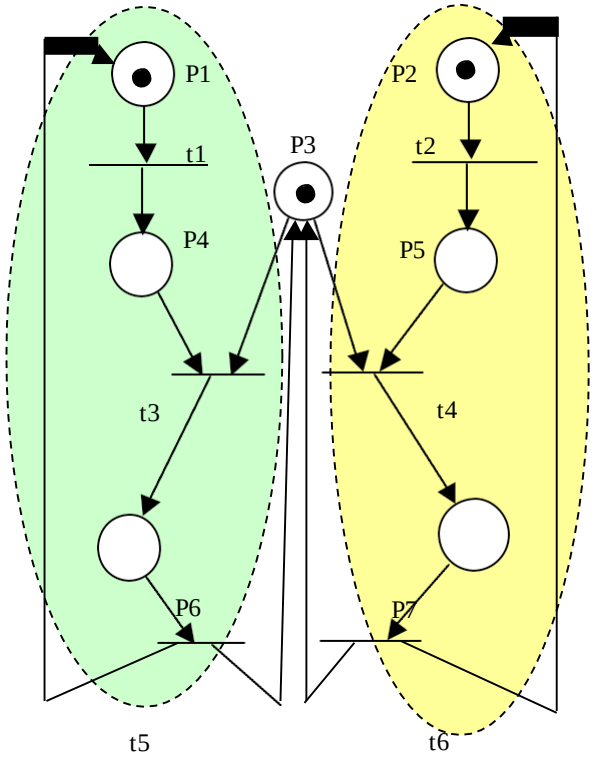
3.5. Les réseaux de Petri (RdP)

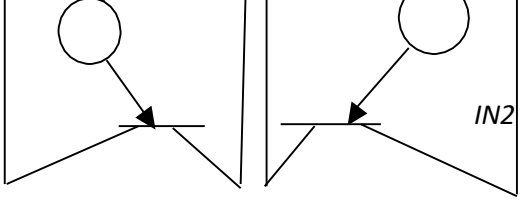
Cette technique formelle et opérationnelle est particulièrement bien adapté pour décrire le *comportement des systèmes asynchrones* avec des évolutions parallèles.
Un RdP est constitué :

- d'un ensemble fini de places (graphiquement des cercles),
- d'un ensemble fini de transitions (graphiquement des barres),
- d'un ensemble fini de flèches, connectant soit des places à des transitions, soit des transitions à des places.

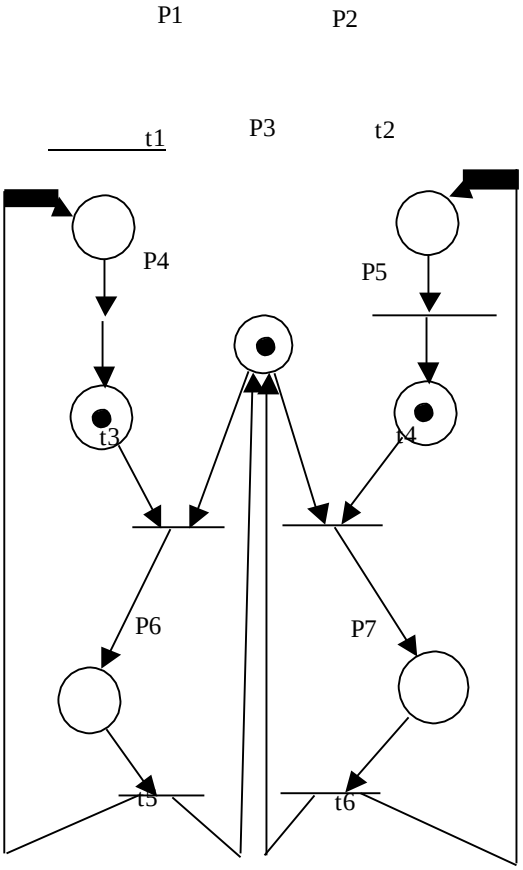
Chaque place peut contenir un ou des *jetons*. L'état du RdP est défini par le *marquage* de ses places. L'évolution du marquage obéit à la règle suivante : chaque transition a des places d'entrée et des places de sortie ; si toutes les places d'entrée contiennent au moins un jeton, la transition est franchissable ; elle peut alors être franchie ('tirée'), ce qui retire un jeton de chaque place d'entrée et ajoute un jeton dans chaque place de sortie. Si plusieurs transitions sont franchissables, le choix de celle qui est tirée est indéterministe.
Les transitions modélisent en général des actions et la présence des jetons la satisfaction des conditions nécessaires à leur réalisation.

Exemple : un RdP et son évolution (cf. Fig.6).
On peut interpréter ce réseau comme deux activités asynchrones qui se partagent une ressource (modélisée par la place P3). Les 2 jetons dans P1 et P2 modélisent l'état des 2 activités. Le jeton dans P3 modélise la ressource prise par une des 2 activités (transitions t3 et t4) et rendue après utilisation (transitions t5 et t6).

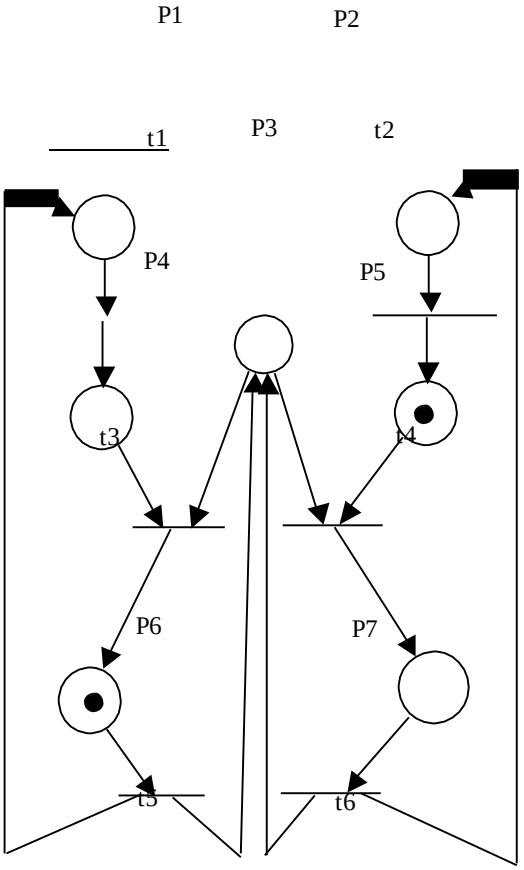




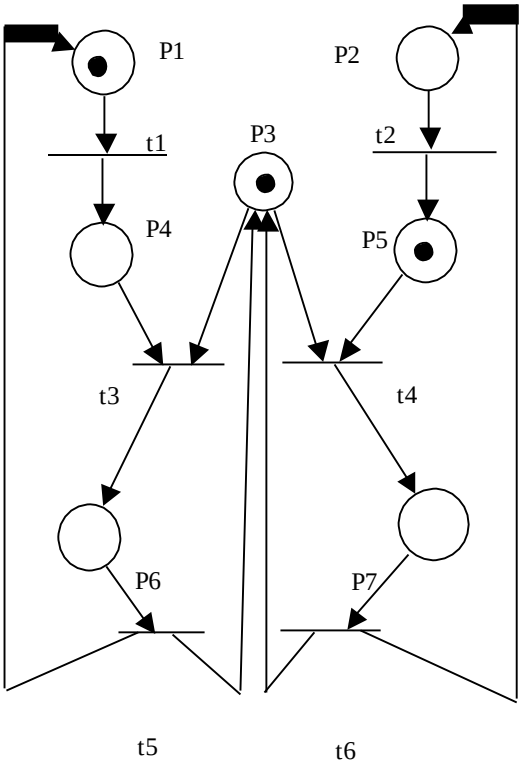
état initial (t1 et t2 franchissables) t1 franchie (t2 et t3 franchissables)



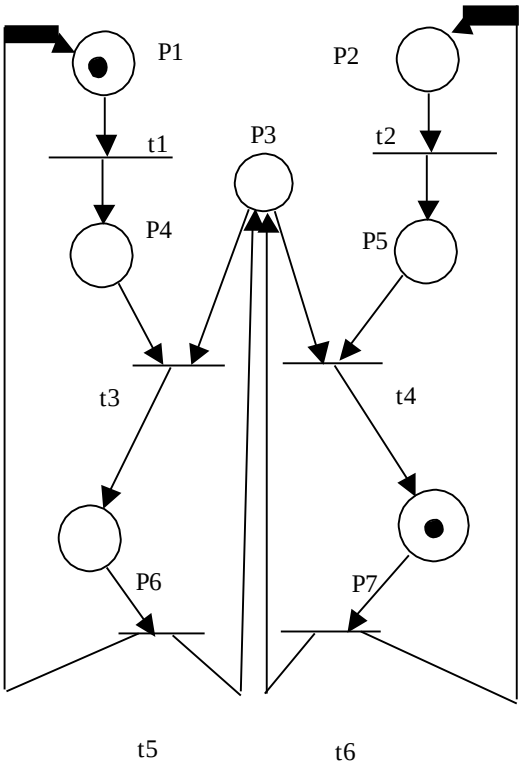
t2 franchie (t3 ou t4 franchissables)



t3 franchie (t5 franchissable)
la ressource est prise par l'activité de gauche



t5 franchie (t1 et t4 franchissables)



t4 franchie ((t1 et t6 franchissables)

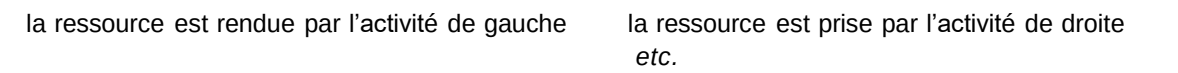


Fig.6. Un RdP et son évolution

Si on suppose que les 2 activités se partagent 2 ressources, on peut avoir par exemple le RdP de la Fig.7. Il souffre d'un danger d'*interblocage*. En effet si un jeton de R est consommé par t2 et l'autre immédiatement après par t6, t3 et t7 ne peuvent plus être franchies faute de jeton restant dans R. Rien ne pleut plus évoluer.

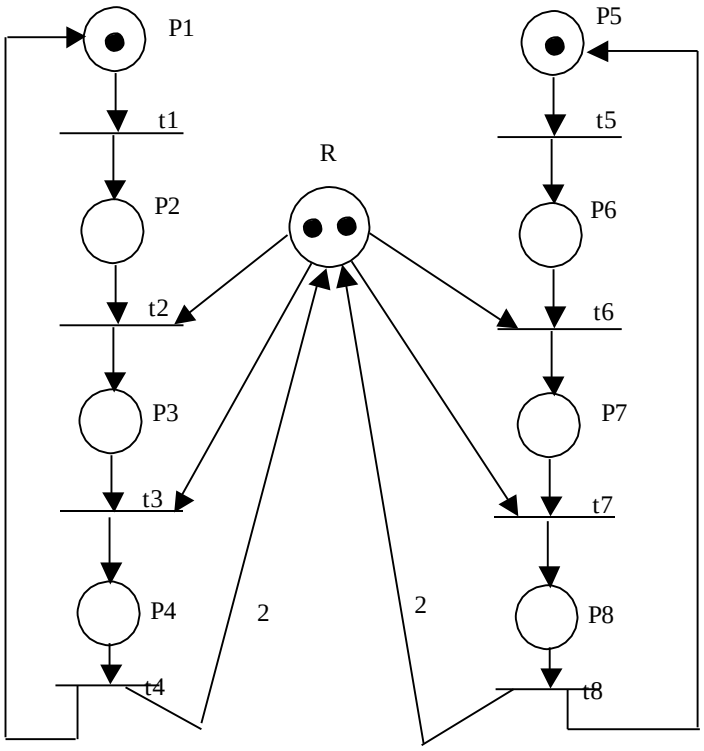


Fig.7. Un RdP risquant l'interblocage.

Le chiffre 2 sur certains arcs signifie que 2 jetons sont produits par la transition sur cet arc.
Au contraire, celui de la Fig.8. est *vivant*, c'est à dire sans possibilité d'interblocage, car les 2 jetons de R sont consommés et rendus simultanément.

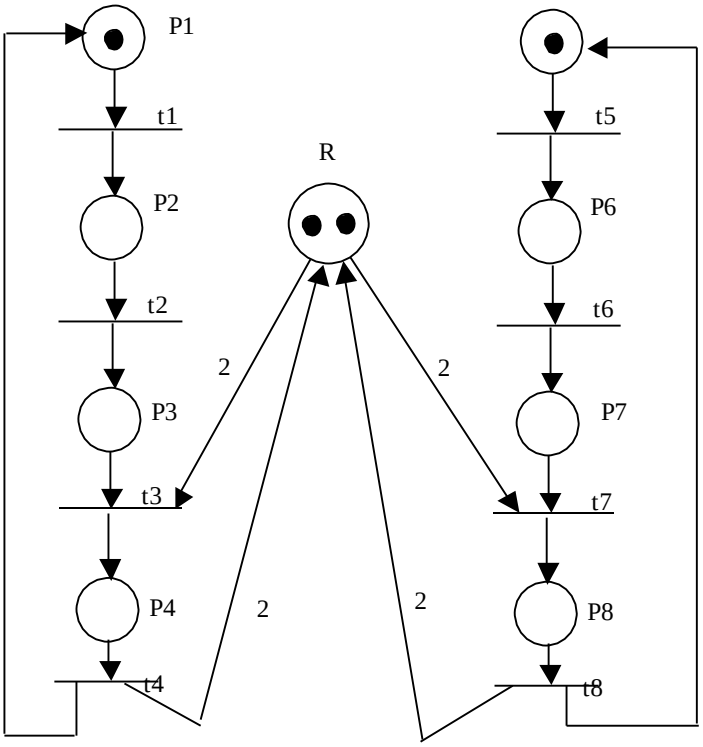


Fig.8 . Un RdP vivant.

Il existe des techniques mathématiques permettant de prouver des propriétés des RdP (par exemple, existence d'interblocage ou caractère vivant du réseau). Une technique de base est la construction de *l'arbre de tous les marquages accessibles*.
Les RdP permettent de modéliser très facilement le producteur consommateur introduit dans le paragraphe sur les machines à états finis (cf. Fig.9). On n'observe pas la même explosion combinatoire du nombre de places car chaque sous système conserve son état (jeton).

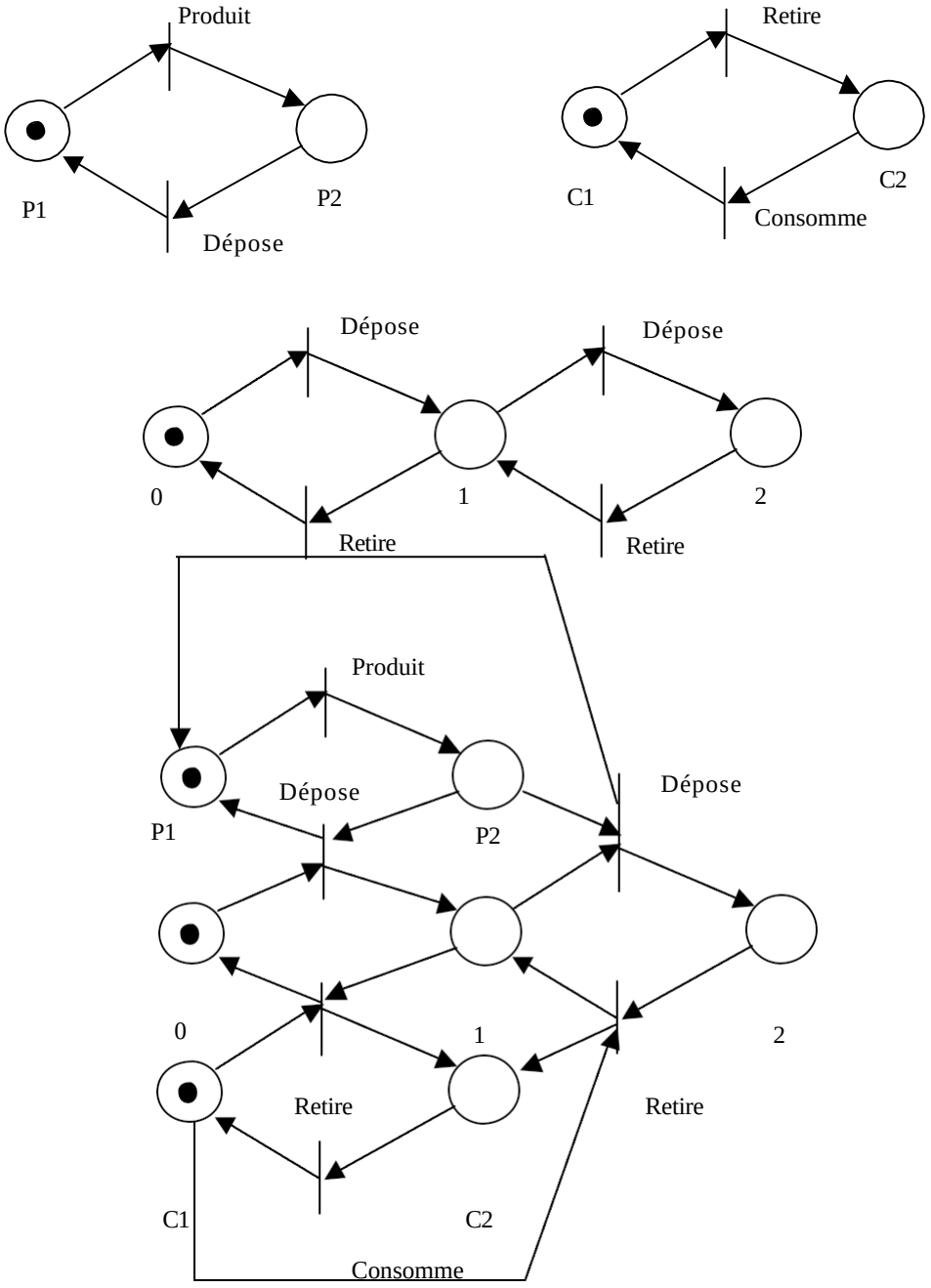


Fig.9. La modélisation du producteur / consommateur avec des RdP.

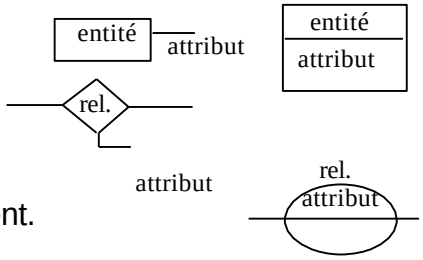
Les RdP souffrent eux aussi de plusieurs limitations que diverses extensions tentent de contourner : typage des jetons (RdP colorés), ajout de prédicats, ajout de caractéristiques temporelles au franchissement des transitions, etc. Il s'agit du modèle de comportement le plus fréquemment utilisé.

3.6. Les schémas entités associations EA (ou entités relations ER)

Il s'agit d'une technique semi formelle et déclarative (Peter Chen, 1976). Ces schémas permettent de spécifier *la structure des données et de leurs relations*, ce qui n'est fait ni dans les DFD, ni dans les modèles orientés contrôle. C'est indispensable pour les systèmes organisés autour de larges ensembles de données interconnectées. Les concepts du modèle de base sont :

les *entités*, qui sont des collections d'items partageant des propriétés communes (occurrences d'entités),

les *associations* (ou *relations*), qui traduisent l'existence de liens entre entités (occurrences d' associations entre occurrences d'entités),

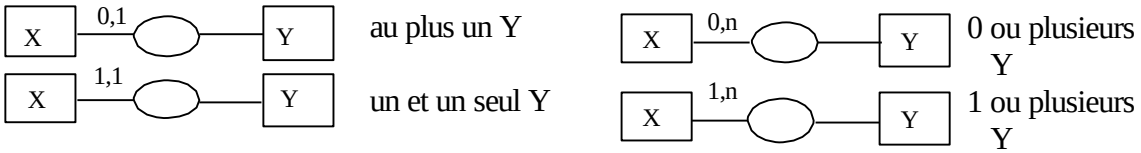


les *attributs* (ou *propriétés*), attachés aux entités et aux associations et qui les caractérisent.

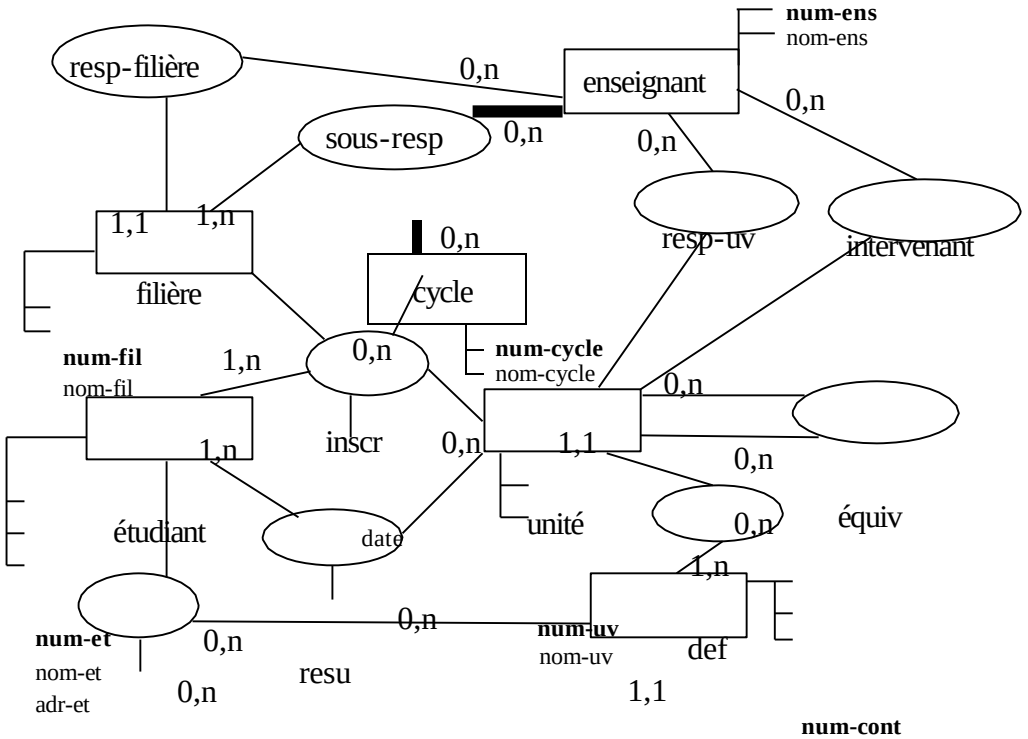
Une entité existe indépendamment de ce qui l'entoure. Une association n'existe que si les entités extrémités existent. Chaque entité à un attribut *identifiant* qui distingue univoquement chaque occurrence d'entité.

Certains modèles, dits modèles binaires, n'autorisent que des associations entre 2 entités. Les associations peuvent être partielles. Les associations sont souvent caractérisées par leur cardinalités, qui peuvent être notées de diverses manières.
Avec les notations de Merise, à tout X correspond :

cardinalité mini et maxi



La figure 10 donne un exemple de schéma entités associations.



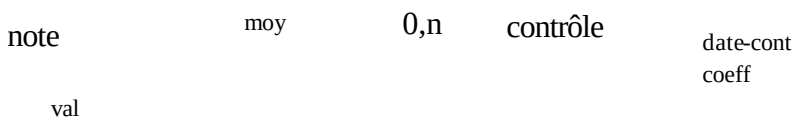


Fig.10. Un exemple de diagramme entités relations (identifiants en gras).

Le modèle EA (ou ER) n'est pas normalisé et de nombreuses notations et représentations coexistent. Certains ajoutent des relations d'agrégation et/ou d'héritage entre entités (nous les étudierons dans le cadre des modèles à objets), ainsi que des contraintes supplémentaires sur les données (ou contraintes d'intégrité).

3.7. Les spécifications formelles

Il existe beaucoup de techniques formelles et déclaratives, avec des fondements mathématiques divers (ensembles, logique classique, logiques non classiques, comme les logiques temporelles, etc.).

Elles ont été souvent utilisées à l'origine pour spécifier des *types de données abstraits*, puis ont été généralisées pour spécifier des systèmes complets. Nous nous limiterons à la seule *notation Z*, à base ensembliste. Elle est décrite dans un fascicule séparé.

4. Conclusion

Souvent les techniques de spécifications se complètent, en décrivant des vues complémentaires d'un système. Par exemple, un système peut être spécifié à travers un diagramme de flot de données (sources d'informations, types d'informations stockées et échangées, décomposition en fonctions), un schéma EA (structuration des informations) et des machines à états finis (comportement de certains composants). Les *méthodes tentent de proposer des assemblages efficaces de telles techniques avec des guides pour les construire et les valider*.

Parler des techniques de spécification est comme parler des langages de programmation. *Il n'y a ni langage ni technique idéale, ni langage ni technique permettant de tout faire.* L'informaticien doit avoir une culture assez étendue des diverses techniques comme des divers langages.

EXERCICES

Exercice 2.1 *Diagramme de contexte et diagramme de flots de données (DfD).* On considère la gestion d'un bureau de location pour plusieurs stations touristiques.
Différents prestataires offrent des locations. Ils peuvent les retirer tant qu'aucune réservation définitive n'est réalisée. Ils touchent 90% du prix de la location.
Les clients adressent des demandes de renseignement. Le bureau y répond par des propositions de location et d'assurance annulation ou une mise en attente. Le client peut alors refuser ou demander une réservation en envoyant des arrhes et en souscrivant éventuellement l'assurance annulation. Si la location choisie est encore libre, elle est réservée, sinon la demande est mise en attente. Après un délai de 8 jours, la réservation est confirmée de manière définitive. En cas d'annulation après ces 8 jours, un pourcentage est dû par le client sauf s'il a souscrit l'assurance annulation. L'annulation en cas de mise en liste d'attente est toujours possible sans frais.
La facture est envoyée avant le début du séjour. Un rappel suit en cas de non paiement. En cas de nouveau défaut de paiement le dossier est transmis au contentieux.
Dessiner le diagramme de contexte et décomposez le en un premier niveau de DFD. Raffiner la gestion des réservations avec des DFD plus détaillés.

Exercice 2.2 *Machine à état finis.*
Une montre digitale comporte un écran d'affichage et 2 boutons A et B. Le bouton A permet de changer de mode : affichage de l'heure (mode initial), à modification des heures, à modification des minutes, à affichage de l'heure. Le bouton B permet d'incrémenter les heures ou minutes dans les modes modification. Dessiner le diagramme d'états décrivant le comportement de cette montre digitale.

Exercice 2.3 *Réseau de Petri.*
On veut modéliser la gestion des cabines et des paniers dans une piscine. A l'entrée, une personne qui a trouvé une cabine libre se change en posant ses vêtements dans la cabine puis demande un panier qu'elle remplit pour libérer la cabine. Après la baignade, la personne rentre dans une cabine avec son panier, le vide, libère le panier et se rhabille pour libérer la cabine.
Modéliser cette organisation, avec une place représentant le stock des cabines (par exemple 3) et une place modélisant le stock des paniers (par exemple 5).
Quel est le maximum de baigneurs simultanés ? Montrer que cette organisation risque de conduire à un blocage.
Proposer une organisation qui évite la possibilité d'un blocage et modéliser la en RdP.

Exercice 2.4 *Diagrammes entités associations.*
On reprend l'énoncé de l'exercice 3. Après inventaire, les principaux concepts à représenter sont : les clients, les prestataires, les hébergements (offerts par les prestataires), les stations (où se situent les hébergements), les propositions, les réservations, les demandes en attente, les types de prestataires (souhaités dans les demandes en attente), les types d'hébergement (souhaités dans les demandes en attente), les activités (possibles dans les stations), les services (offerts par les prestataires).

Proposez un diagramme entités relations avec des cardinalités réalistes. Donnez les principaux attributs.

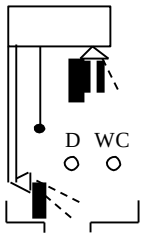
Exercice 2.5 *Modélisation de la dynamique (automate et réseau de Petri).*

Le système est composé de deux tâches informatiques cycliques T1 et T2 qui se partagent un processeur unique. Les tâches peuvent être soit en attente de processeur, soit en cours d'exécution sur le processeur. L'allocation et la désallocation des tâches au processeur se fait selon une certaine politique que l'on ne cherche pas à décrire.
a) Modéliser les états et les transitions d'état de ce système avec un diagramme d'état (automate à états finis). Expliquer la signification des états et des transitions.
b) Modéliser le même système avec un réseau de Petri. Expliquer la signification des places et des transitions. Le processeur sera modélisé par une place.

Pendant son exécution T1 peut se mettre en attente d'un événement. Le processeur peut alors être alloué à T2 jusqu'à ce que l'événement arrive. Modifier en conséquence la modélisation.
c) Nouveau diagramme d'état.
d) Nouveau réseau de Petri. L'arrivée de l'événement sera modélisé par une transition.

Exercice 2.6 *La 'douchiotte' (réseau de Petri)*

La 'douchiotte' est une douche-WC, pour installations à petit budget ... Elle est décrite par le schéma ci-dessous. Elle peut fonctionner comme une douche (arrosage par le haut) ou comme un WC (chasse d'eau vers le bas).



Initialement la douchiotte est dans l'état douche. On sélectionne l'état voulu en appuyant sur un des 2 boutons (D ou WC). Un seul état est possible à chaque instant. Quand on tire sur la ficelle et que l'état WC est choisi, la vanne de la chasse d'eau est ouverte. Quand l'état douche est choisi et que l'on tire sur la ficelle, c'est la vanne de la douche qui est ouverte. On ferme la vanne qui est ouverte en tirant de nouveau sur la ficelle. L'état ne peut être modifié quand une vanne est ouverte.

Décrivez par un réseau de Petri le comportement de ce système. Les transitions correspondront aux actions de l'utilisateur (sur les boutons ou la ficelle).

Exercice 2.7 : *Location de cassettes (DfD)*
On utilise les diagrammes de flots de données pour spécifier les fonctionnalités d'une application de location de cassettes vidéo.
Le cahier des charges précise que :

- le client peut
    - louer et rapporter des cassettes; les locations sont enregistrées ainsi que les clients
- le gérant peut

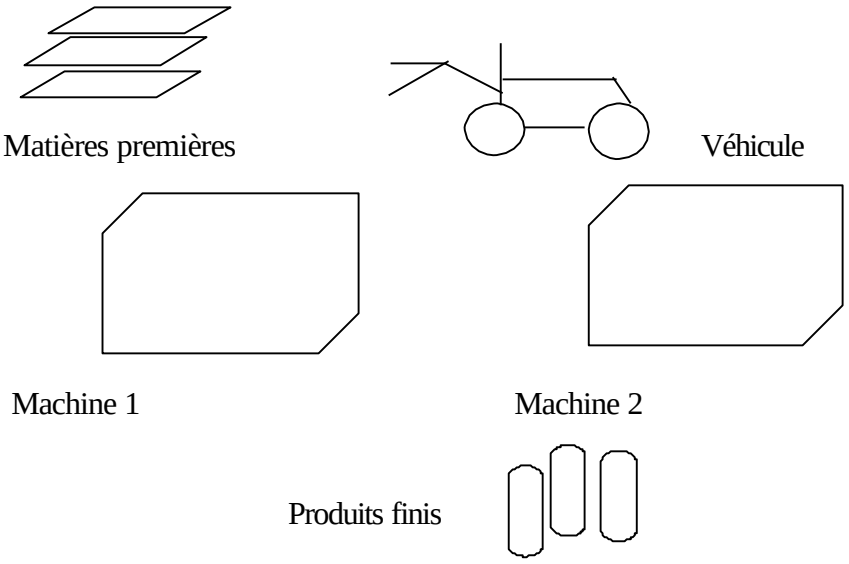
  - ajouter/supprimer des cassettes au catalogue
  - changer les prix des cassettes au tarif
- l'application
  - calcule le prix d'une location selon le tarif des cassettes empruntées et la durée de l'emprunt
  - génère des états de caisse en fin de journée.

a. Dessiner le diagramme de contexte.
b. Raffiner ce diagramme en faisant apparaître une fonction de gestion des locations de cassettes, entourée d'autres fonctions, flots et stockages.
c. Raffiner à un deuxième niveau la fonction de gestion des locations de cassettes.

Exercice 2.8 *Modélisations diverses par Réseaux de Petri*

Modélisez chacun des systèmes décrits ci-dessous par un réseau de Petri.

a) Pour sélectionner la bande de réception d'une radio, on doit tourner un bouton à gauche ou à droite, mais on ne peut pas faire le tour. Les trois positions sont de gauche à droite OL (ondes longues), OM (ondes moyennes), MF (modulation de fréquence). Ondes de départ : OL.
b) Vous modélisez un distributeur de boissons. Dans son état initial, il attend qu'on appuie sur le bouton 'café' ou 'sirop'. Après cette sélection il faut 'confirmer' ou bien 'annuler' si on s'est trompé de boisson. Si on confirme, le distributeur verse la boisson choisie et pendant cette opération tous les boutons sont inactifs. Quand la boisson est prête ou si l'on a annulé on se retrouve à l'état initial.
c) Le dernier système à modéliser est une petite usine avec 2 machines et un véhicule.



Le véhicule est utilisé pour charger une machine libre avec une matière première et pour décharger une machine qui a terminé son travail vers le stock de produits finis.
Votre modèle doit rendre compte du fait que le véhicule ne peut transporter qu'une chose à la fois et qu'il ne peut amener une matière première que s'il y a une machine libre.

Exercice 2.9 *Appels d'offres (modèle EA, DfD et Réseau de Petri)*
Pour ses divers projets de développement d'applications informatisées, une entreprise réalise des appels d'offre à des sociétés de service. A réception des propositions, l'entreprise les évalue pour en sélectionner une.

a) Esquisser une modélisation des données utiles pour réaliser la sélection par un modèle entités/associations. Vous préciserez les cardinalités des associations et donnerez quelques attributs. Vous vous limiterez aux 3 types d'entités et aux 2 types d'association les plus importants.

b) Pour évaluer les propositions, le responsable de chaque projet donne des critères de sélection. L'évaluation débouche sur une note établie à partir des critères. Des lettres de refus ou d'acceptation sont éditées. Esquissez une modélisation fonctionnelle par diagrammes de flots de données, à 2 niveaux de raffinement :

un diagramme de contexte, avec l'application d'évaluation, les acteurs externes impliqués et les flots d'entrée et de sortie de l'application,

un diagramme de flot de données qui décompose l'application d'évaluation, en décrivant la saisie des critères, le calcul de la note et les éditions. Vous ferez figurer les stockages de données (qui correspondent probablement aux entités de la question a).

c) Représentez par un réseau de Petri, la dynamique de la procédure de sélection suivante :

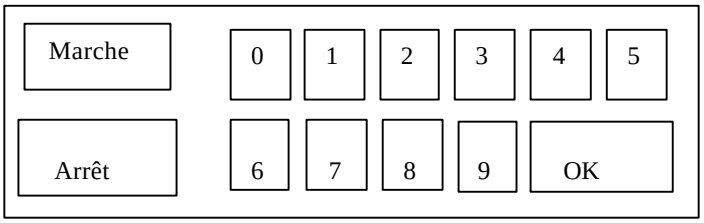
si l'appel d'offres est en cours,

dès qu'une proposition arrive, elle est évaluée,

selon la note obtenue, la proposition est acceptée ou refusée. Si elle est acceptée, l'appel d'offres est terminé (on ne peut plus évaluer d'autres propositions).

Conseils : une place avec un jeton modélise le fait que l'appel d'offres est en cours ; une autre place reçoit un jeton dès qu'une proposition arrive ; une place indique que la sélection est en cours. 2 places indiquent que la proposition est acceptée ou bien refusée.

Exercice 2.10 *Automate à états finis*
Modélisez sous forme d'un diagramme d'états (automate à états finis) le comportement du contrôleur de chauffage suivant. Vous indiquerez les événements en entrée et les ordres de démarrage/arrêt du chauffage en sortie.



« Le bouton Marche, met le contrôleur en attente d'une consigne de température. Après saisie du nombre, le bouton OK valide cette température souhaitée. Le chauffage est prêt à démarrer. Il démarre dès que la température (mesurée toutes les minutes) est inférieure de 1 degré à la consigne. Il s'arrête dès que la température dépasse de 1 degré la consigne. Le bouton Arrêt remet le système à

son état initial quel que soit l'état en cours (il faut alors réutiliser le bouton Marche). Une consigne non comprise entre 0 et 30 est ignorée ; il faut la ressaisir. Un changement de consigne est possible quand une consigne valide est enregistrée ; la saisie entraîne l'arrêt du chauffage s'il est en marche. »

Exercice 2.11 *Diagramme de flots de données*

L'ordinateur de bord d'une automobile doit réaliser les fonctions suivantes :

convertir les signaux de rotation des roues en valeurs numériques (en signaux par seconde),

afficher à l'aide de diodes électro-luminescentes la vitesse instantanée (conversion en tours/mn puis en km/h),

avertir par un signal sonore si la vitesse maximum (stockée) est dépassée,

signaler par une flèche lumineuse dessinée à l'aide des diodes le sens de l'accélération (différence entre deux mesures de signaux par seconde).

Dessiner le DfD correspondant.