Interface Homme Machine

1. Introduction
2. Définitions
3. Critères ergonomiques
4. Les étapes du processus de développement et IHM
5. Modèles de tâches
6. Architecture logicielle des systèmes interactifs
7. **Les formalismes de validation des DHMs** (Dialogues Homme Machine)

7. Les formalismes de validation de Dialogue Homme Machine

Les formalismes de validation ou de vérification de la dynamique permettent de:

- ✓ Décrire et vérifier le comportement de l'application ou parfois
- ✓ Uniquement celui de l'interface (Dialogue ou interaction),
- ✓ Représenter la dynamique du dialogue homme-machine.

Au cœur des sciences de l'interaction, le dialogue a été décrit par un nombre important de formalismes qui peuvent être classés en trois groupes :

1. **Les formalismes basés sur les états** tels que les machines à états et les statecharts*. Les dialogues exprimés sous forme de machines à états peuvent être facilement implémentés à l'aide de la bibliothèque JAVA/SwingStates.
2. **Les formalismes à base d'événements** qui sont utilisés dans la plupart des boîtes à outils comme Java/Swing,
3. **Les formalismes hybrides** tels les réseaux de Pétri de haut niveau.

*Où un état peut comporter une machine à états.

7. Les formalismes de validation de Dialogue Homme Machine

1. Les formalismes à base d'états:

- ✓ Les machines à états appelées aussi automates à états ou systèmes de transition étiquetés (STE) sont exprimables sous forme graphique.
- ✓ De plus, elles sont basées sur des modèles mathématiques;
- ✓ Elles permettent le raisonnement, la validation et la vérification de nombreuses propriétés du dialogue dans les IHM telles que les propriétés de sûreté ou de vivacité.
- ✓ Dans ce formalisme une application peut être vue comme un ensemble d'états qu'elle est susceptible de prendre au cours des sessions de travail.

1. Les formalismes à base d'états (Cont.):

- ✓ Les formalismes basés sur les états permettent de décrire les états et la façon de passer d'un état à un autre. Ce passage peut être décrit par un langage, une grammaire ou bien encore un graphe.
- ✓ L'avantage principal des formalismes à états est leur capacité à être validé formellement. On peut vérifier par exemple qu'un automate est vivant ou qu'il ne possède pas de puits.

a. Les états finis (Automates)

Les états finis constituent un des formalismes les mieux maîtrisés pour les IHM étant donné leur capacité à être traduits sous forme de grammaires (équivalence automate/grammaire).

1. Les formalismes à base d'états (Cont. Les états finis/Automates):

Un système basé sur ce formalisme par exemple *Rapid/Use* ou les *Statecharts* qui autorisent également le parallélisme manipule cinq éléments :

- ✓ Un ensemble fini d'états,
- ✓ Un ensemble fini d'entrées,
- ✓ Un ensemble fini de sorties,
- ✓ Une fonction $f_e(état_i, e_j) = état_{i+k}$ qui permet de passer à l'état suivant en fonction de l'état précédent et de l'entrée,
- ✓ Et une fonction de sortie $f_s(état_i, e_j) = s_j$ qui renvoie le résultat en fonction de ces mêmes paramètres,

1. Les formalismes à base d'états (Cont. Les états finis/Automates):

Pour représenter plus clairement ces deux dernières fonctions, les états finis et leurs relations sont représentés sous forme de diagrammes (***Figure ci-dessous***).

Il existe toujours un état initial (représenté sur la figure par l'état 0 doublement cerclé), mais il peut exister plusieurs états finals (ici le 1 et le 4). On remarquera également qu'un état peut s'appeler lui-même une ou plusieurs fois en fonction de certaines entrées.

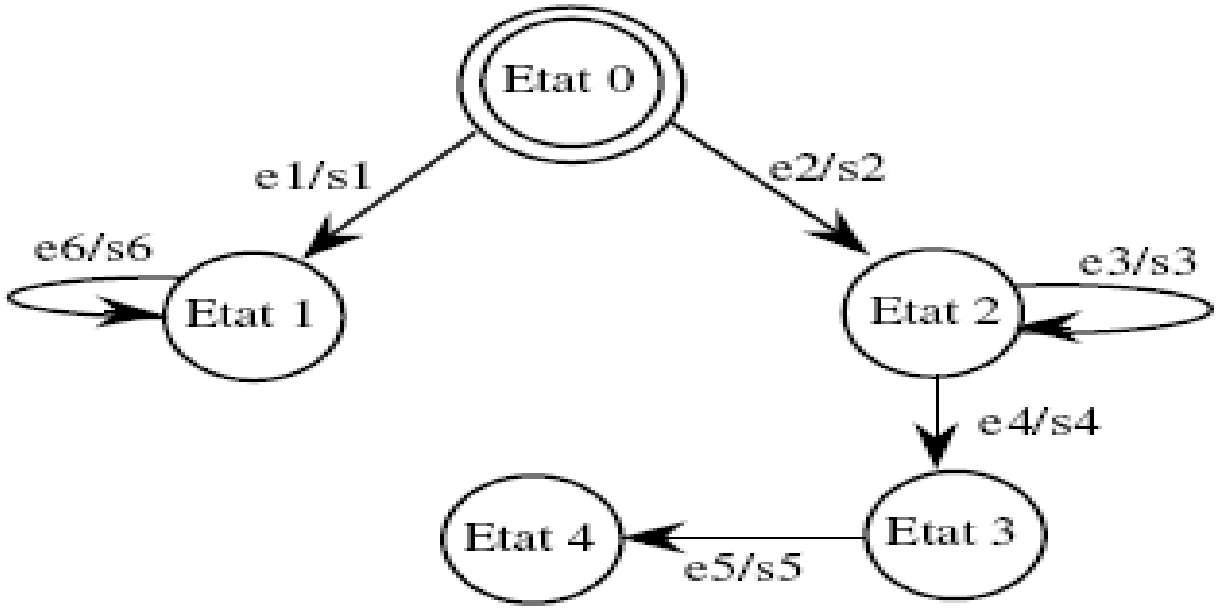
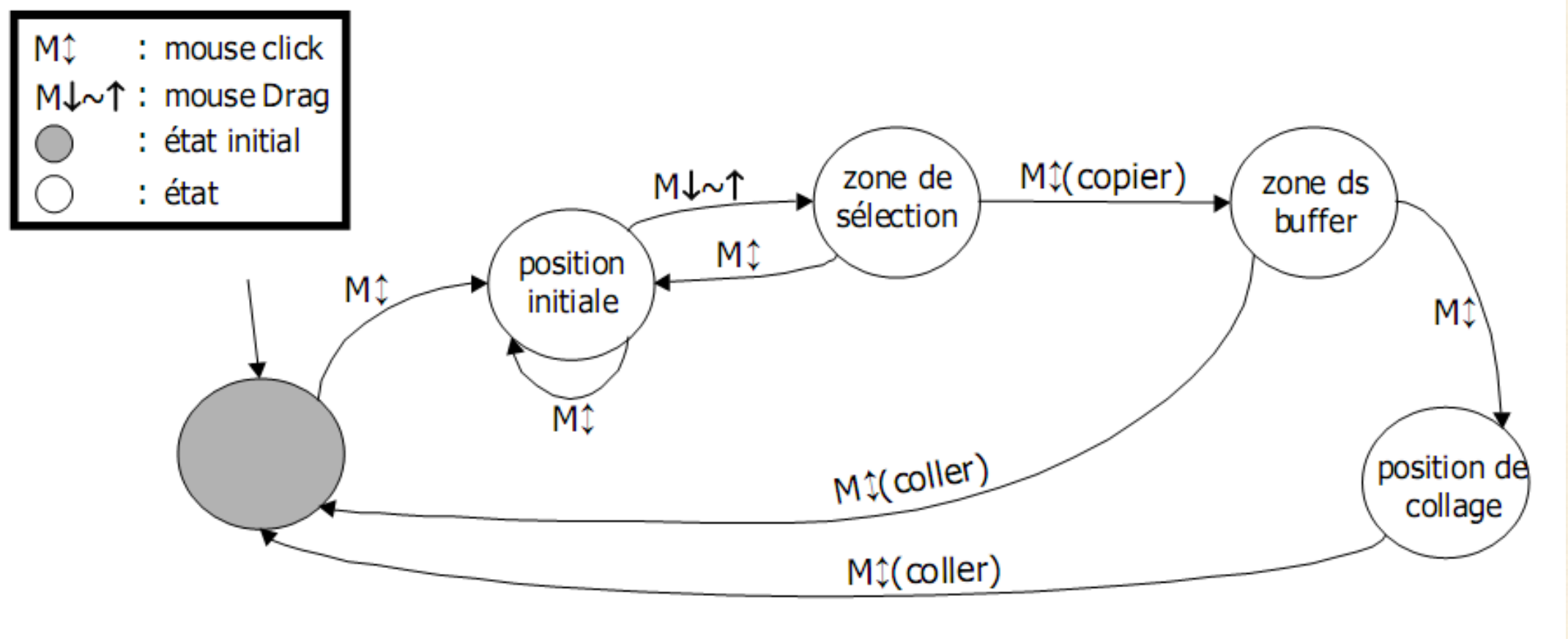


Figure : Un exemple de diagramme d'états finis.
Les e_i sont les entrées et les s_i sont les sorties.

Copier/Coller une zone de texte modélisée à l'aide d'un STE:



Cet exemple modélise les actions de l'utilisateur de choisir, dans un premier temps une zone de texte, d'effectuer la copie puis de terminer par le collage du texte soit à la suite du texte sélectionné ou soit dans une nouvelle position.

1. Les formalismes à base d'états (Cont.):

b. Les grammaires

Une grammaire est un ensemble de quatre éléments :

- un alphabet de symboles terminaux,
- un alphabet de symboles non-terminaux,
- un symbole non-terminal particulier nommé axiome, qui est le symbole par lequel l'analyse commence
- un ensemble fini de règles qui permettent de vérifier qu'une séquence d'actions (entrés par l'utilisateur) est correcte et d'appeler les actions correspondantes.

Par exemple, les règles présentées en slide suivant sont un extrait des règles de la grammaire exprimant le dialogue d'un Mailer. Cet extrait est particulièrement centré sur le dialogue relatif à l'envoi d'un email.

1. Les formalismes à base d'états (Cont. Les grammaires):

```
MAILER := DIALOG_ENVOI | DIALOG_CONSULTER | QUITTER
DIALOG_ENVOI := IDENTIFIER_EMAIL envoyer {send();}
IDENTIFIER_EMAIL :=  NOUVEL_EMAIL | EMAIL_BROUILLON
EMAIL_BROUILLON := select_brouillon {afficher();} MODIFIER_BROUILLON
MODIFIER_BROUILLON := modifier_dest{mettreDest();}MODIFIER_BROUILLON
      | modifier_message{mettreMessage();}MODIFIER_BROUILLON
      | ∅
```

Les non-terminaux sont en majuscule alors que les terminaux sont en minuscule. Le symbole « | » permet d'exprimer le choix et les actions à réaliser sont entre accolades.

1. Les formalismes à base d'états (Cont. Les grammaires):

Le dialogue d'une application interactive est spécifié comme une séquence de terminaux. Les non-terminaux ont pour rôle de factoriser des sous-dialogues. Dans les grammaires, l'itération est réalisée à l'aide de la récursivité, c'est le cas de **MODIFIER_BROUILLON** à la fin des deux premières règles de **MODIFIER_BROUILLON**.

La grammaire présentée précédemment indique que l'utilisation du **MAILER** se fait à travers **DIALOG_ENVOI** ou **DIALOG_CONSULTER**. **DIALOG_ENVOI** n'a qu'une seule production: **IDENTIFIER_EMAIL** *envoyer {send() ;},* cela signifie que le seul terminal accepté à la fin d'une séquence est *envoyer* et que l'action qui lui correspond est *send()*.

Pour cette production Le premier terminal possible est celui issu de l'application de la règle **IDENTIFIER_EMAIL.** Celle-ci est constituée de deux non-terminaux **NOUVEL_EMAIL** et **EMAIL_BROUILLON.** Ce sont les règles de ces deux non-terminaux qui s'appliquent alors. Seule la règle de **EMAIL_BROUILLON** est décrite dans cet exemple.

1. Les formalismes à base d'états (Cont. Les grammaires):

La règle de **EMAIL_BROUILLON** est composée d'un terminal *select_brouillon{afficher() ;},* qui est donc le seul terminal par lequel une séquence de dialogue peut commencer et du non-terminal **MODIFIER_BROUlLLON**.

Après le terminal *select_brouillon,* les terminaux possibles sont donc *modifier_dest* et *modifier_message.* Les actions correspondantes sont alors réalisées (*mettreDest()* et *mettreMessage()*).

La prochaine règle applicable est celle correspondant au non-terminal **MODIFIER_BROUILLON** (pour exprimer le caractère itératif).

Enfin, le non-terminal **MODIFIER_BROUILLON** peut être remplacé par le terminal **« Ø »** pour mettre fin à l'itération.

2. Les formalismes à événements:

- ✓ Le concept d'événement a complètement révolutionné le monde des IHM. Il a permis à celles-ci d'être pilotées de plus en plus par l'utilisateur et non plus uniquement par le système.
- ✓ Les événements permettent, à l'inverse des états, de décrire à quoi l'interface (ou l'application) est susceptible de réagir.
- ✓ Les états permettent de dire quelles sont les étapes à suivre alors que les événements permettent de dire comment y arriver.
- ✓ Lorsqu'un événement se produit, il est traité. Un gestionnaire d'événement le dirige alors vers une procédure de traitement qui, en fonction de ses paramètres, exécute un traitement particulier.

2. Les formalismes à événements (Cont.):

- ✓ Ce modèle est utilisé par de nombreux langages de programmation, comme Java/Swing car il possède un important pouvoir d'expression et d'exécutabilité.
- ✓ Cependant, les modèles à base d'événements ne permettent pas la vérification de propriétés ni la représentation explicite des états du dialogue.

a. Les scénarios:

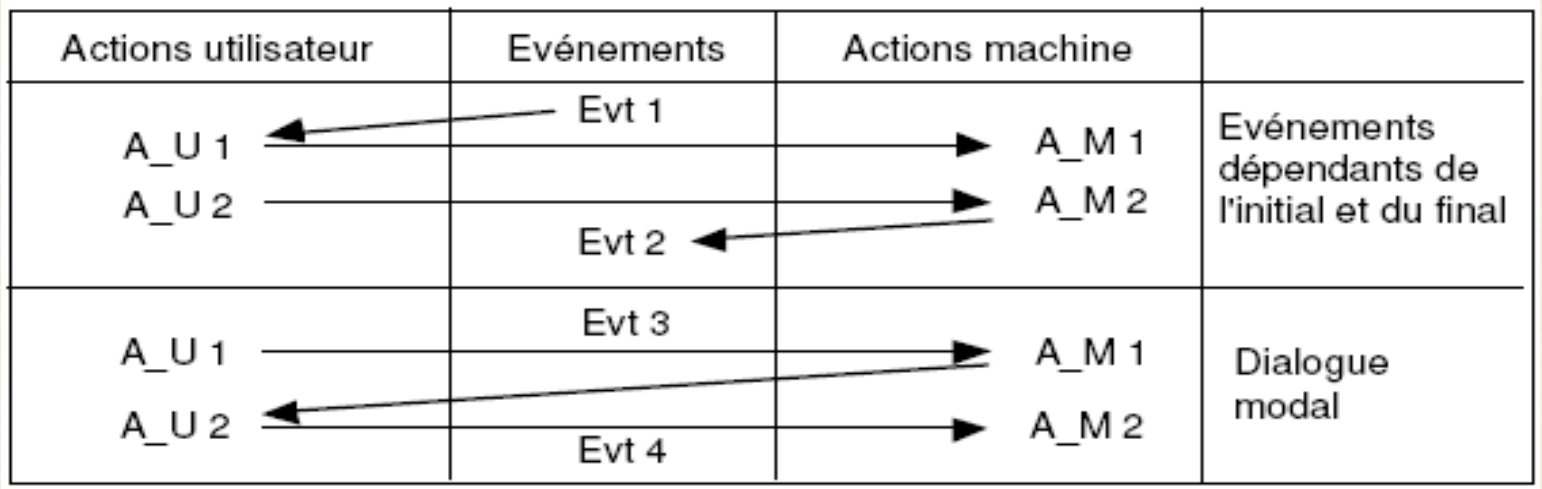
Les scénarios que nous présentons ici sont utilisés dans la méthode Grai. Ils permettent de représenter non seulement les actions de l'utilisateur et celles de la machine, mais également de dire qui pilote qui et à quelle occasion. Les scénarios se rapprochent en cela de la méthode Diane+.

2. Les formalismes à événements (Cont. Les scénarios):

La figure ci-dessous montre deux exemples de scenarii. Dans le premier cas, l'utilisateur agit sur le système à la suite d'un événement (Evt 1) produit par ce dernier. L'action (A_U 1) de l'utilisateur entraîne une réaction du système (A_M 1).

Dans le même temps, l'utilisateur a eu la possibilité d'agir de nouveau sur le système (A_U 2) ce qui a provoqué une autre réaction du système (A_M 2) ainsi que la production d'un événement (Evt 2).

Dans le second cas, l'utilisateur agit sur le système (A_U 1), ce qui produit un événement (Evt 3) qui déclenche une action machine (A_M 1). Celle-ci achevée, l'utilisateur peut de nouveau agir sur le système afin de produire de nouveaux événements, etc.



Exemples de scénarios avec la méthode Grai.

3. Les formalismes hybrides

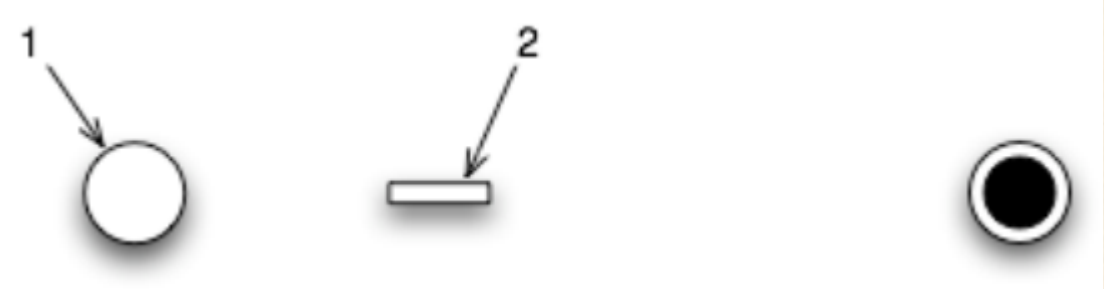
Les formalismes hybrides sont des formalismes qui utilisent à la fois les états et les événements.

a. Les Réseaux de Petri

Les réseaux de Petri (RdP) ont été créés pour modéliser des processus communicants. Tout comme les machines à états, ils permettent la vérification de propriétés et une représentation graphique du dialogue, cependant, ils ont une plus grande puissance d'expression car ils permettent de compter les répétitions.

Les réseaux de Petri sont constitués de trois types d'éléments,

- ✓ les places (voir figure ci-dessous: repère **1**) ,
- ✓ les transitions (repère **2**) et les arcs.
- ✓ Pour représenter l'état du réseau, des marques ou jetons sont utilisés sur les différentes places (repère **3**).



les différents éléments d'un réseau de Petri

3. Les formalismes hybrides (Cont. Les réseaux de Petri)

- ✓ Dans un Rdp les places représentent les états, les transitions, les événements et les arcs, les conditions et effets.
- ✓ Ils sont utilisés principalement pour représenter des traitements séquentiels parfaitement connus à l'avance (voir **le support Rdp en ligne**).
- ✓ Il existe à l'heure actuelle des systèmes qui utilisent les Réseaux de Petri en conjugaison avec une approche objet ou événementielle tels l'ICO. Ce formalisme représente le dialogue homme-machine par un ensemble de graphes d'événements **Rdps**.
- ✓ Chaque élément composite de l'IHM (par exemple une **fenêtre**, mais pas un bouton) est représenté par un **Rdp**.

3. Les formalismes hybrides (Cont. Les réseaux de Petri)

Le côté haut de La figure suivante représente un **exemple** de fenêtre de saisie de mot de passe associée à une fenêtre d'aide, le côté bas représente le réseau **Rdp** associé.

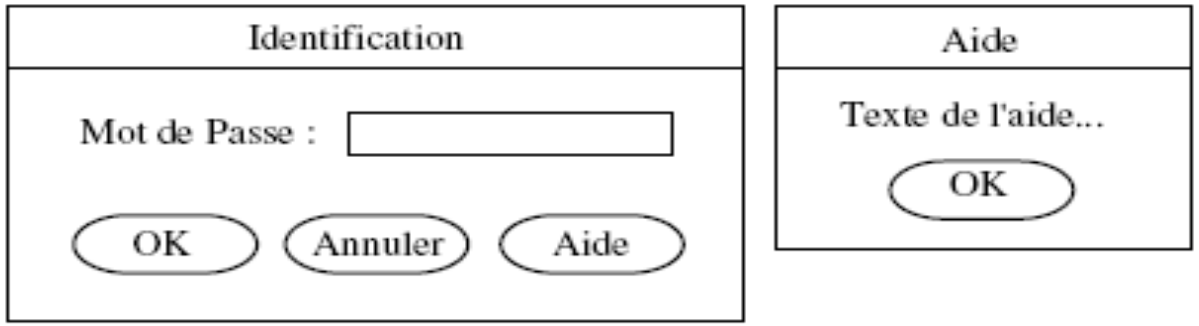
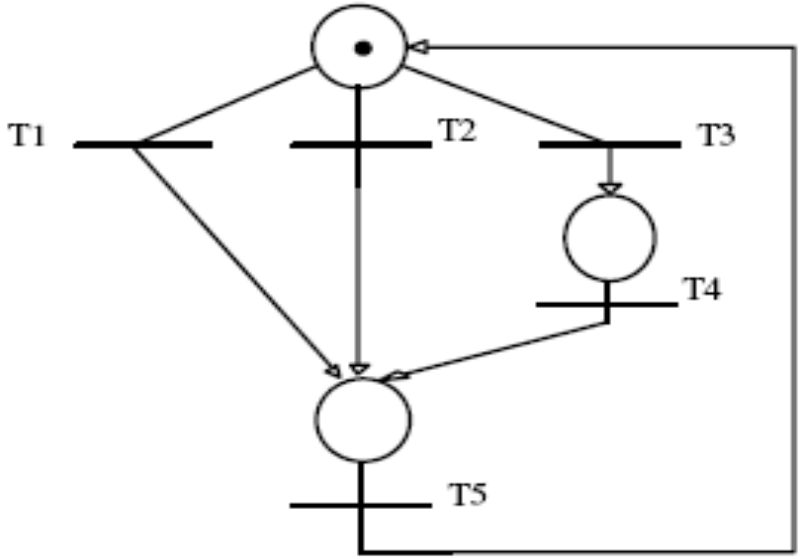


Figure a : Une fenêtre de saisie de mot de passe et sa fenêtre d'aide associée



T1 correspond au clic sur OK et déclenche la vérification du mot de passe
T2 correspond au clic sur Annuler et ferme la fenêtre
T3 correspond au clic sur Aide et affiche la fenêtre d'aide
T4 correspond au clic sur OK dans l'aide et ferme la fenêtre d'aide
T5 ne sert qu'à terminer proprement le réseau et ne déclenche rien.

Figure b : Le graphe d'événement associé à la fenêtre de saisie de mot de passe.

3. Les formalismes hybrides (Cont.)

b. Les Objets Coopératifs Interactifs (ICO)

Le formalisme des Objets Coopératifs Interactifs (ICO) a spécifiquement été développé pour exprimer la dynamique des applications interactives .

Ce formalisme combine les Objets Coopératifs et des réseaux de Petri.

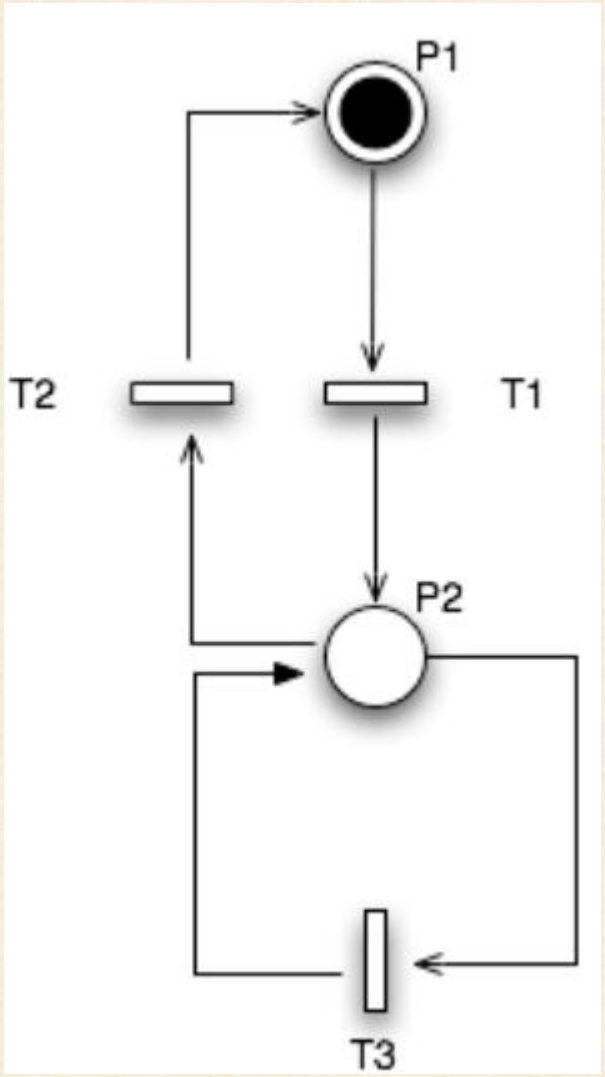
- ✓ L'outil **PetShop** a été développé afin d'animer le dialogue d'applications interactives exprimé avec ce formalisme.

3. Les formalismes hybrides (Cont. Les Objets Coopératifs Interactifs)

La Figure à côté représente un réseau de Petri pour l'envoi d'un email préalablement enregistré dans les brouillons d'un mailer.

Nous avons fait l'hypothèse qu'un email enregistré dans les brouillons dispose obligatoirement d'au moins un destinataire (le champ « @ » est rempli).
Les noms des places et transitions du réseau de Petri sont:

P1 : place initiale
P2 : place de présentation de l'email en cours
T1 : select Email
T2 : envoyer
T3 : modifier Email



Utilisation d'un réseau de Petri pour l'envoi d'un email

3. Les formalismes hybrides (Cont. Les Objets Coopératifs Interactifs)

Initialement, la place marquée avec un jeton est **P1**, le système est dans l'état correspondant. À partir de cette place, la seule transition exécutable ou franchissable est **T1** (activée par l'événement click).
Une fois cette transition réalisée, le jeton se trouve dans la place **P2**.
Deux transitions sont alors possibles : **T3** qui ramène le jeton à la place **P2** et **T2** qui met le jeton à la place **P1**.

Les **Objets Coopératifs Interactifs** (ICO) permettent la description formelle du comportement dynamique des systèmes interactifs.
Ils sont une extension des formalismes des Objets Coopératifs (OC) et des réseaux de Petri. Les réseaux de Petri décrivent le comportement des objets et leurs protocoles de communication.

Un objet est composé de deux éléments :

- ✓ un objet coopératif avec des services utilisateurs
- ✓ une partie présentation

3. Les formalismes hybrides (Cont. Les Objets Coopératifs Interactifs)

La liaison entre ces deux éléments est assurée par deux fonctions :

- ✓ **Une fonction d'activation** qui fait correspondre à une action de l'utilisateur sur un widget au service utilisateur à déclencher.
- ✓ **Une fonction de rendu** qui permet de maintenir la cohérence entre l'état interne du système et son apparence externe.

Un objet coopératif (CO) établit comment l'objet réagit à des stimuli extérieurs en fonction de son état.
Ils proposent deux types de services, ceux offerts aux autres objets de l'environnement et ceux offerts aux utilisateurs.
La partie présentation est l'apparence externe de l'objet.

3. Les formalismes hybrides (Cont. ICO)

Exemple du Modèle ICO de l'activité supprimer un fichier par **Drag & Drop**: