MASTER EN SCIENCES INFORMATIQUES

Modélisation et simulation d'interactions multimodales

Ladry, Jean-François

Award date:
2007

Awarding institution:
Universite de Namur

Résumé :

Ce mémoire se situe dans le domaine de l'ingénierie des systèmes interactifs et de l'interaction homme-machine. Il a pour but de définir des méthodes, des outils et des techniques permettant de concevoir des systèmes interactifs à la fois fiables et utilisables. Dans ce cadre, nous avons défini un processus basé sur une description formelle dans le but de modéliser des interactions multimodales. Cette modélisation a pour objectif d'augmenter l'utilisabilité des interactions, d'être réutilisable et facilement modifiable.

Ce mémoire est divisé en trois parties.

- La première partie, l'état de l'art, exprime les principes de multimodalité et explique les différents outils que nous utilisons pour notre modélisation.
- La deuxième partie, la contribution, explique comment construire nos modèles et l'architecture multimodale à partir des modèles de chaque périphérique.
- Enfin, la troisième partie présente une application dans laquelle nous avons intégré nos modèles multimodaux.

Mots clés: Réseau de Petri à objets, Multimodalité, Interaction Homme-Machine, Modélisation

Abstract :

This study lies in the field of the engineering of the interactive systems and computer human interaction. Its purpose is to define methods, tools and techniques making it possible to conceive interactive systems both reliable and (easily) usable. Within this framework, we have defined a process of modeling based on a formal description aiming at modeling multimode interactions. This modeling aims at increasing the utilisability of the interactions, it is also meant to be reusable and easily modifiable.

This study is divided into three parts.

- The first one, the state of the art, expresses the principles of multimodality and explains the various tools which we use for our modeling.
- The second one, the contribution, explains how to build our models and multimodal architecture starting from the models of each peripheral.
- Finally, the third part presents an application in which we have to integrate our multimodal models.

Keywords: Objects Petri Net, Multimodality, Computer Human Interaction, Modeling

Remerciements

Mes remerciements s'adressent, tout d'abord, à ma promotrice, Madame M. Noirhomme-Fraiture, pour son aide et ses conseils avisés.

Je tiens à remercier Monsieur P. Palanque pour m'avoir reçu à l'IRIT et me faire confiance pour ses prochaines années. Mes remerciements vont aussi à son équipe et plus particulièrement à D. Navarre et E. Barboni pour leur accueil.

Je remercie également les différentes personnes qui ont participés à la finalisation de ce mémoire par leurs corrections ou leurs remarques : Vincent, Manu, Thierry et François.

Je voudrais aussi remercier Sandra pour m'avoir soutenu et motivé pendant ces trois années de maîtrise.

Je terminerai en remerciant ma famille pour m'avoir permis de réaliser ces études et ce stage à Toulouse ainsi que pour leurs nombreux encouragements.

Merci à vous tous.

Table des matières

Table des figures

Table des tableaux

Glossaire

API (Application programming interface) : Ensemble de commandes externes publiées par un éditeur et permettant de recourir aux fonctions d'un logiciel depuis un autre logiciel. [01NET] http://www.01net.com/editorial/191701/api/

BNF (Backus-Naur Form) : Notation conçue par John Backus et Peter Naur permettant de décrire les règles syntaxiques d'une grammaire ou d'un langage.

Booléen (ou variable booléenne) : Variable ayant comme valeurs possibles les valeurs binaires Vrai ou True (correspondant à 1) et Faux ou False (correspondant à 0)

CO (Cooperative Object) : Objet coopératif (voir page 21)

CNES : Centre National d'Etudes Spatiales

CS : Communication et Systèmes

Diagramme de composants : Diagramme décrivant le comportement d'un système par les liens et dépendances entre ses modules.

Dragging : Technique d'interaction consistant à déplacer un objet en déplacant la souris ou un autre périphérique d'entrée avec un bouton enfoncé de la position de cet objet vers la nouvelle position désirée. Par extension : déplacement du périphérique avec un bouton enfoncé

ICO (Interactive Coopérative Object) : Objet coopératif interactif (voir page 25)

IRIT : Institut de Recherche en Informatique de Toulouse

LIIHS : Logiciels Interactifs et Interaction Homme-Système, équipe de l'IRIT

ObCS (Object Control Structure) : Structure de contrôle de l'objet (voir page 23)

OPN (Object Petri Net) : Réseau de Petri à objets (voir page 20)

PetShop : Outil permettant la modélisation et l'exécution de réseau de Petri (voir page 29)

Introduction

La démocratisation de l'informatique a transformé l'utilisateur standard de l'état de "spécialiste du système" (généralement celui qui a construit le système) à celui de "non-spécialiste". Le grand nombre d'utilisateurs ainsi potentiellement disponibles et les perspectives économiques qui en découlaient ont été à la base de la création du métier d'informaticien.

Les systèmes informatiques produits par ces informaticiens ne présentaient pas d'autres difficultés de conception que celles liées à la réalisation des fonctions minimales devant être remplies par le système. Mais les problèmes de visualisation et d'interaction n'étaient absolument pas une considération jugée pertinente. Le processus de démocratisation a donc fait tomber la dictature des informaticiens dont l'optique de développement est passée de l'état de "ça marche donc j'ai fini " à celui de "est-ce suffisamment *bien* fait pour que ça se vende".

Cette optique regroupe à elle seule l'ensemble des préoccupations du domaine de l'interaction homme-machine. On peut immédiatement en identifier deux aspects :

- l'aspect externe du système qui regroupe les problèmes d'esthétique, de facilité d'utilisation, de facilité d'apprentissage, etc. ;
- l'aspect interne qui, comme pour les systèmes précédents, se préoccupe des problèmes de "correction" (le système fait bien ce qu'il est supposé faire et tout ce que l'on voulait qu'il fasse) et de fiabilité (le système ne se bloque pas, toutes ses commandes sont potentiellement accessibles, etc.).

La prise en compte de l'aspect externe dans la construction des systèmes a généralement pour conséquence l'ajout de fonctionnalités au système qui sont liées non seulement à la présentation et à la manipulation des informations, mais aussi aux fonctions que doit offrir le système. Le domaine de l'interaction homme-machine a identifié depuis très longtemps l'ampleur de l'impact de la prise en compte de ces aspects sur la construction des systèmes interactifs. Les problèmes qui en résultent peuvent être classés dans deux grandes catégories : les problèmes de génie logiciel et les problèmes de prise en compte de l'utilisateur.

La fiabilité d'un système en conditionne son utilisabilité et il est donc indispensable de se préoccuper de la fiabilité d'un système avant de se préoccuper de son utilisabilité.
Néanmoins la fiabilité n'est pas une fin en soi dans la mesure où il est inutile de faire un système fiable si son inutilisabilité est telle qu'aucun utilisateur ne voudra s'en servir.

Dans le domaine de l'interaction homme-machine, l'équipe du LIIHS de l'Institut de Recherche en Informatique de Toulouse (IRIT) s'est spécialisée dans la gestion de périphériques d'entrée pour les systèmes interactifs et plus particulièrement la modélisation des interactions sur base de réseaux de Petri. La plupart de leurs projets sont liés à l'industrie spatiale et aéronautique où la fiabilité et l'utilisabilité sont essentielles. C'est dans cette équipe que nous avons réalisé notre mémoire.

Dans ces systèmes interactifs, il est possible de gérer plusieurs périphériques d'entrée pour que l'utilisateur puisse interagir plus efficacement avec le système tout en ayant moins

d'erreur d'utilisation. Cette interaction au moyen de plusieurs périphériques est appelée multimodalité.

Pour réaliser ces interactions multimodales de manière disciplinée, systématique et quantifiable, nous avons besoin de les abstraire.

Nous avons donc pour ce travail, défini un processus de modélisation de techniques d'interactions multimodales pour les systèmes interactifs basé sur une notation et une technique formelle à base de réseaux de Petri. Ces modèles sont dynamiques afin d'en voir l'exécution et peuvent être modifiés pendant leur exécution afin de changer certains paramètres.

Nos modèles sont réutilisables afin de pouvoir les tester avec différents paramètres pour en trouver l'optimum. Ceci permet d'en accroître l'utilisabilité. L'objectif est d'inhiber les problèmes de génie logiciel et de permettre la prise en compte du type d'utilisateur.

Ces modèles peuvent également être comparés afin de faciliter la création de modèles de nouveaux périphériques ou de modéliser de nouveaux comportements.

Pour réaliser ces modèles, nous avons utilisé l'outil PetShop créé par l'équipe du LIIHS. Cet outil permet de créer des réseaux de Petri à objets et d'exécuter ces réseaux.

Ce mémoire est structuré en trois parties.

Dans la première partie, l'état de l'art, nous expliquons les différentes notions qui nous seront utiles pour modéliser des interactions multimodales.

Nous présentons tout d'abord des modèles d'architecture des systèmes interactifs : Seeheim et son extension, Arch afin de situer notre travail, ainsi que d'en accroître la modularité et la réutilisabilité.

Puis, nous parlons des modèles de description formelle de systèmes : les automates à états, les statecharts et le réseaux de Petri. Nous nous consacrons plus particulièrement aux réseaux de Petri à Objets (OPN). A la fin de cette partie, nous introduisons l'outil PetShop que nous avons utilisé pour modéliser nos réseaux de Petri.

Nous énonçons, ensuite, les différents types de périphériques d'entrée à l'aide de la taxonomie de Buxton en nous focalisant sur ceux que nous utiliserons dans la contribution à savoir la souris, l'écran tactile et la reconnaissance vocale.

Enfin, nous terminons par une définition et quelques exemples de multimodalité. Ceci nous permet de voir ce qui existe dans ce domaine.

Dans la deuxième partie de notre travail, nous modélisons tout d'abord trois périphériques d'entrée : la souris, la reconnaissance vocale et l'écran tactile dans le but ensuite de modéliser des techniques d'interaction multimodale.
Nous détaillons, pour chaque périphérique, l'architecture de nos modèles en Arch et le diagramme de composants exprimant les échanges entre les modèles. Ensuite pour chaque modèle, nous présentons le processus qui a conduit à cette modélisation et nous expliquons le modèle en détail.
Enfin, nous fusionnons les architectures pour réaliser des modèles d'interactions multimodales.

Enfin, dans la troisième partie, nous implémentons ces modèles dans Image, une application de gestion de satellite. Ce projet nous permet de tester nos différents modèles d'interactions multimodales dans une application réelle.

Partie 1 Etat de l’art

1 Introduction

Notre objectif est de modéliser des interactions multimodales. Nous allons donc expliquer, dans cet état de l'art, les différentes notions qui seront utiles pour la seconde partie de ce mémoire.

Nous présentons tout d'abord les modèles d'architecture des systèmes interactifs : Seeheim et son extension, Arch. Ceux-ci nous permettront de situer notre travail, ainsi que d'en accroître la modularité et la réutilisabilité.

Puis, nous abordons des modèles de description formelle de systèmes. Nous commençons par les automates à états et les statecharts. Ensuite, nous nous consacrons aux réseaux de Petri et plus particulièrement aux réseaux de Petri à Objets (OPN). Dans cette partie, nous expliquons par des exemples les différents composants d'un réseau de Petri, les particularités des réseaux de Petri à objets ainsi que les différentes structures remarquables qui seront utilisées dans notre contribution comme les classes émetteurs-récepteurs et les fonctions d'activation et de rendu.

Nous détaillons ensuite les différents types de périphériques d'entrée à l'aide de la taxonomie de Buxton en expliquant plus particulièrement ceux que nous utiliserons dans la contribution à savoir la souris, l'écran tactile et la reconnaissance vocale. Ceci permettra de situer les différents périphériques sélectionnés pour nos modèles d'interaction et de discuter des évolutions possibles de ceux-ci vers d'autres périphériques.

Enfin, nous terminons par l'état de l'art dans le domaine de la multimodalité. Nous commençons par définir les notions de modalité et de multimodalité, les différents types de multimodalité pour finir par deux exemples de multimodalité.

2 Architecture des systèmes interactifs

2.1 Introduction

Les modèles d'architecture de systèmes interactifs sont des guides de développement pour favoriser la modularité, la modifiabilité et la réutilisabilité. Ils ont pour objectif de structurer le code. Cette décomposition des applications facilite leur conception ainsi que l'interaction par le biais de couches simples et distinctes.
Nous présentons de manière sommaire deux grands modèles pour les systèmes interactifs, le modèle de Seeheim et le modèle Arch.

2.2 Seeheim

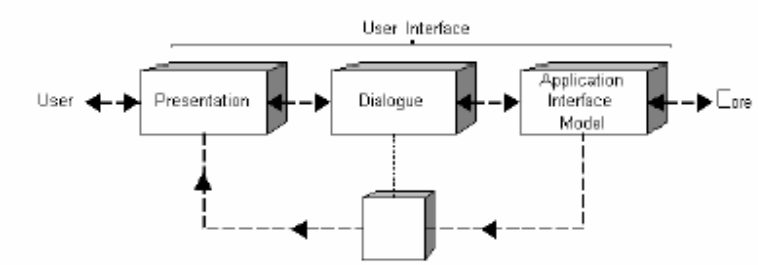


Figure 1 Le modèle Seeheim

Le modèle de Seeheim (Figure 1) fut le premier modèle unificateur d'architecture de systèmes interactifs [Pfaff85]. Il décrit l'interface utilisateur en trois parties ce qui facilite la conception et la décomposition. Pfaff définit ces composants de la manière suivante:

- **Le composant de présentation :** *«Le composant de présentation gère les aspects lexicaux de l'interaction et décrit les objets d'interaction (boutons, zone de saisie, etc.). Il traduit aussi les unités lexicales d'entrée en expressions compréhensibles pour le contrôleur de dialogue et inversement.»*
- **Le contrôleur de dialogue :** *«Le contrôleur de dialogue décrit les aspects syntaxiques de l'interaction, l'enchaînement du dialogue et les réponses dynamiques des objets d'interaction. Il se charge également d'assembler les unités reçues du composant de présentation et de les transmettre à l'interface de l'application (et inversément).»*
- **L'interface de l'application :** *«L'interface de l'application exprime le propos du dialogue dans un langage directement compréhensible pour l'application non interactive. »*

De plus Mark Green [Green85] y ajoute un composant particulier :

- **Le composant Feedback :** *«Le composant Feedback est présent pour décrire le retour des informations directement entre le noyau fonctionnel et la présentation (sans passer par le dialogue). L'exemple courant est celui de la modification de l'affichage de l'icône cible lors d'un « glisser - déposer » afin d'indiquer si celui-ci est valide ou pas.»*

2.3 Le modèle Arch

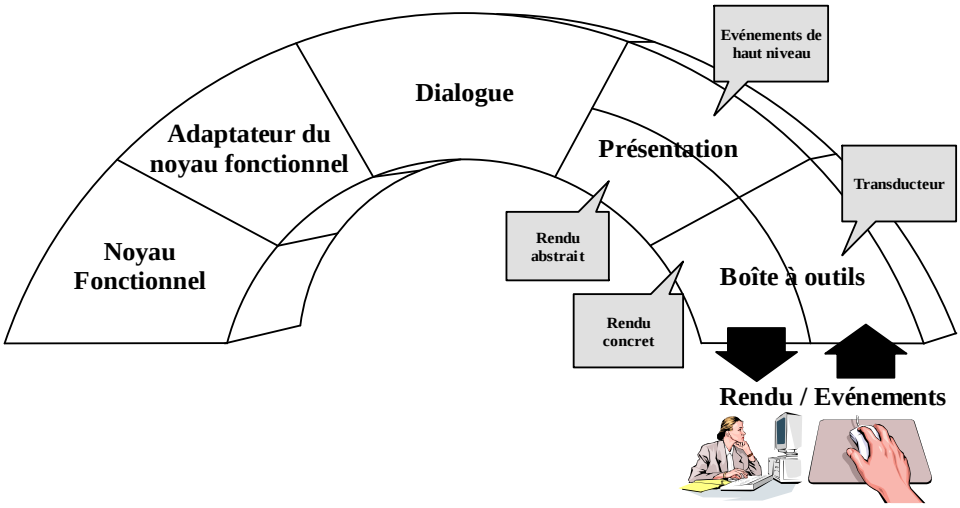


Figure 2 Le modèle Arch (version étendue)

Le modèle Arch [Bass91] (Figure 2) permet d'apporter des solutions à certains des problèmes soulevés par le modèle de Seeheim. Ces problèmes viennent de la complexification de création d'interfaces. De plus, le modèle Arch permet de prendre en compte l'utilisation de boites à outils qui, la plupart du temps, ne sont pas modifiables. Enfin, il raffine les trois composants de Seeheim, et en introduit deux nouveaux. Ces composants sont définis par Bass comme suit :

- **Le composant d'interaction physique (Boîte à outils) :** *« Le composant d'interaction physique permet de gérer l'interaction au niveau lexical. Il transmet les entrées de l'utilisateur sur les objets de l'interaction et gère la visualisation des informations transmises par le composant de présentation. »*
- **Le composant d'interaction logique (Présentation) :** *« Le composant d'interaction logique transforme les informations fournies par le composant de présentation physique en objets indépendant de l'interface et les transmet au contrôleur de dialogue (et inversement). Il correspond au niveau syntaxique de l'interaction. »*
- **Le contrôleur de dialogue :** *« Le contrôleur de dialogue est la clé de voûte du système. Il gère l'enchaînement des tâches au niveau du système, assure la transformation des formalismes, garantit la consistance entre les différents rendus d'un même objet et fait appel, si nécessaire, à l'adaptateur du noyau fonctionnel (et inversement). »*
- **L'adaptateur du noyau fonctionnel :** *« L'adaptateur du noyau fonctionnel traduit les appels du contrôleur de dialogue en instructions compréhensibles par le noyau fonctionnel et les retours du noyau fonctionnel dans un langage compréhensible par le contrôleur de dialogue. »*

- **Le noyau fonctionnel :** *« Le noyau fonctionnel contrôle, manipule et exécute les actions sur les objets du domaine sans savoir comment ces derniers sont rendus perceptibles à l'utilisateur. »*

Parmi ces composants, la boîte à outil et la présentation sont séparées en deux canaux. Ces canaux représentent le sens du flux d'information.
Le flux des événements de l'utilisateur vers le système interactif montre la remontée de l'information entre une action sur l'interface qui est traduite par la boîte à outils en événement puis est transformée en événement de haut niveau par la partie présentation.
Le flux de rendu montre comment les informations de changements d'états du système sont traduites en rendu abstrait dans la couche présentation, en rendu concret dans la boîte à outils et enfin en changement sur les périphériques de sortie.

2.4 Conclusion

Notre objectif est de créer des modèles pour des périphériques qui soient réutilisables et modulables. Il nous sera donc très pratique d'utiliser un modèle d'architecture. Nous avons choisi le modèle Arch qui est plus détaillé sur la partie présentation et interaction (boîte à outils). Il nous permettra de situer notre travail.

3 Description formelle de systèmes

3.1 Introduction

Les modèles de description de systèmes permettent de modéliser l'état d'un système, les différentes exécutions possibles à partir de cet état et l'état résultant de cette exécution.

Nous allons créer des modèles pour représenter la multimodalité. Pour la réalisation de ces modèles, il est donc important d'utiliser un langage de description du système qui soit le plus complet, le plus compréhensible et qui permette le plus simplement d'avoir un nombre important de paramètres sans que la taille du modèle n'explose. Nous allons donc introduire différents types de description formelle de systèmes afin de sélectionner la plus intéressante.

Nous allons tout d'abord présenter brièvement les automates à états finis et les statecharts. Nous avons choisi ces deux descriptions formelles de systèmes car ils sont les plus répandus. De plus, les statechart font partie d'UML. Ensuite, nous détaillerons le fonctionnement et l'utilisation des réseaux de Pétri et plus particulièrement des réseaux de Petri à objets (OPN) pour comprendre le fonctionnement de nos différents modèles dans la seconde partie de ce mémoire.

3.2 Automates à états finis

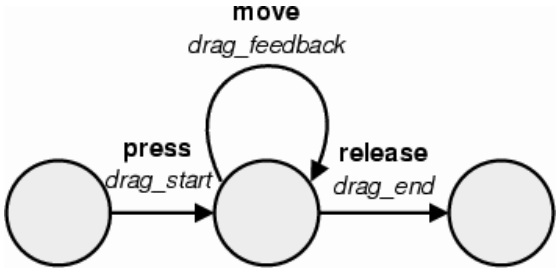


Figure 3 Automate à états finis représentant le Drag

« Les automates à états finis [Parnas69] *(Figure 3) sont utilisés pour modéliser le comportement dynamique de systèmes et notamment pour décrire certaines parties des interfaces utilisateur. Ils sont représentés graphiquement par des diagrammes de transitions d'états, graphes orientés dont les nœuds sont des états et les arcs des transitions. Un état est un ensemble de valeurs qui caractérise le système à un moment donné dans le temps. Une transition d'état est une relation entre deux états indiquant un changement d'état possible et qui peut être annoté pour indiquer les conditions et les sources de déclenchement et les opérations qui en résultent. »* [Dragicevic04]

Néanmoins, les automates à états finis posent certains problèmes. Il est par exemple impossible de modéliser les systèmes à états infinis. De plus, la description par automate n'est pas modulable pour la concurrence ou le parallélisme. Il n'est pas non plus possible

d'échanger des informations par événements ou d'envoyer des requêtes aux objets qui composent les automates à états finis. Enfin, ils mènent rapidement à une explosion du nombre d'états et de transitions.

3.3 Les stateCharts

« Le stateChart [Harel88] ou diagramme d'états-transitions décrit le comportement interne d'un objet à l'aide d'un automate à états finis. Il offre une vision complète et non ambiguë de l'ensemble des comportements d'une instance de classe. Le modèle dynamique du système comprend plusieurs diagrammes d'états-transitions : un pour chaque classe possédant un comportement dynamique important. Tous les automates à états finis s'exécutent concurremment (peuvent changer d'état de façon indépendante). Un diagramme d'états-transitions n'offre pas une vision globale car il ne s'intéresse qu'à un seul élément du système modélisé. »[Audibert06]

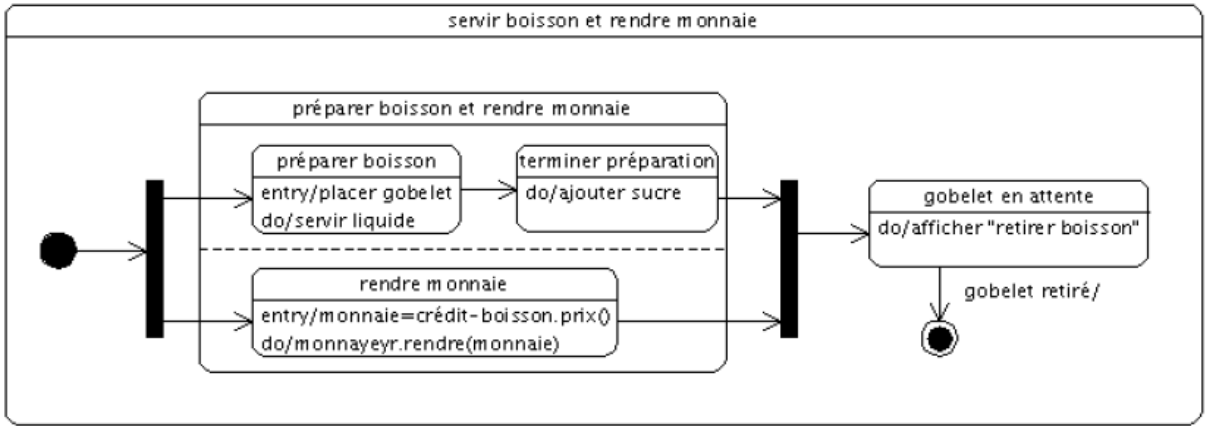


Figure 4 Exemple de Statechart

Comme nous pouvons le voir sur la Figure 4, les statecharts permettent de représenter la concurrence. Malheureusement, comme pour les automates à états finis, le nombre d'états explose lorsque l'on doit modéliser le fait que l'état du système dépend de la valeur d'un grand nombre d'objets.

3.4 Les réseaux de Petri[1]

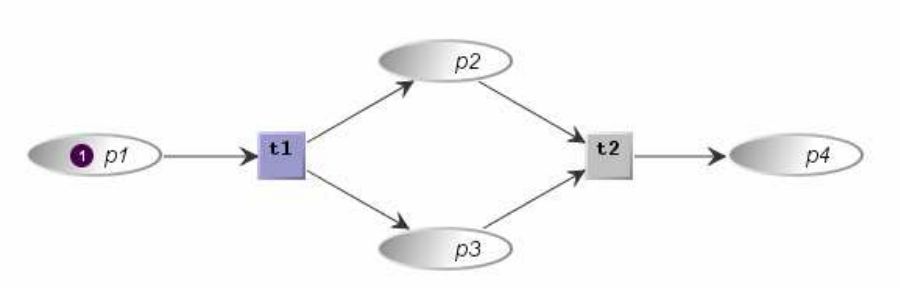


Figure 5 Exemple de réseau de Petri

3.4.1 Introduction

« Les réseaux de Petri constituent une technique de description formelle de la dynamique des systèmes. Ils permettent de modéliser les états d'un système et ses changements d'états.

La technique de réseaux de Petri constitue actuellement l'outil le plus avancé et le plus complet pour la spécification et la description des systèmes de processus fonctionnant en parallèle et avec des contraintes de synchronisation.

Il s'attache en particulier à décrire deux aspects de ces systèmes : des événements (ou actions) et des conditions, ainsi que les relations entre événements et conditions. » [Fichefet]

3.4.2 Place et transition

Les réseaux de Petri sont constitués de quatre types d'éléments :

- Les places, représentées par des ellipses, permettent de modéliser les variables d'état du système.
- Les jetons présents dans cette place donnent la valeur de cette variable.
- Les transitions, représentées par un rectangle, correspondent à un opérateur de changement d'état.
- Les arcs permettent de relier les places et les transitions. Un arc allant d'une place à une transition indique une condition nécessaire au changement d'état du système et un arc allant d'une transition à une place précise l'impact du changement d'état du système. L'état du système est donné par une distribution des jetons dans les places du réseau.

[1] Les différents exemples suivants sont inspirés des travaux de l'équipe du Liihs ([Palanque92], [Navarre01], [Schyn05] et [Barboni06]) [Fichefet], [Diaz01], ainsi que [David92]. Ils sont ici pour permettre une meilleure compréhension de la suite de l'état de l'art ainsi que de la partie contribution. Les figures ont été réalisées avec l'outil PetShop.

3.4.3 Franchissabilité et Franchissement

« On appelle place d'entrée de la transition t toute place p1 reliée à t par un arc entrant. Pour qu'une transition t soit franchissable (c'est-à-dire pour qu'un opérateur de changement d'état soit disponible), il faut qu'il y ait au moins un jeton dans toutes les places d'entrées de t. La transition t pourra alors être franchie (c'est-à-dire que l'opérateur de changement d'état pourra s'exécuter). Le franchissement d'une transition dans un réseau de Petri est une opération locale qui ne met en jeu que les jetons contenus dans les places d'entrées et de sorties de la transition franchie. » [Schyn05]

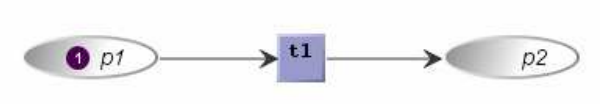


Figure 6 Franchissabilité

La Figure 6 donne un exemple de réseau de Petri. Au départ, la place *p1* contient un jeton et la place *p2* n'en contient pas. Cette distribution de jeton dans le réseau se note : m = {M(p1)=1 ; M(p2)=0} = [1, 0] où M(p) représente la quantité de jetons contenue dans la place p. La transition *t1* est franchissable s'il y a au moins un jeton dans la place *p1*. La transition *t1* est donc franchissable (la place *p1* contient un jeton).

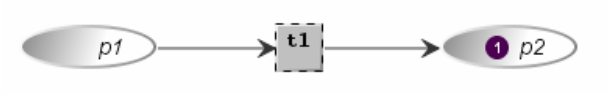


Figure 7 Franchissement

Si elle est franchie, un jeton est retiré de la place *p1* et un jeton est déposé dans la place *p2* (Figure 7). Cette distribution de jetons dans le réseau se note : m = [0, 1].

3.4.4 Le Conflit structurel

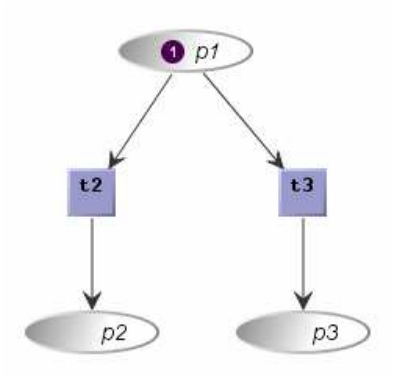


Figure 8 Conflit structurel

La Figure 8 présente un cas de conflit structurel. En effet, les transitions *t2* et *t3* peuvent être franchies l'une comme l'autre s'il y a un jeton dans la place *p1*. Dans cette situation, on parle d'indéterminisme dans le réseau de Petri, c'est-à-dire que la transition qui va être franchie est choisie au hasard.

3.4.5 Arc inhibiteur et arc de test

Outre les arcs classiques, il existe deux types d'arcs permettant de modéliser les tests d'existence ou non de jetons dans une place.

3.4.5.1 Arc inhibiteur

Figure 9 Arc inhibiteur

L'arc inhibiteur (entre la place *p1* et la transition *t1* dans la Figure 9) indique que la transition *t1* ne peut pas être franchie si la place *p1* contient un jeton.

3.4.5.2 Arc de test

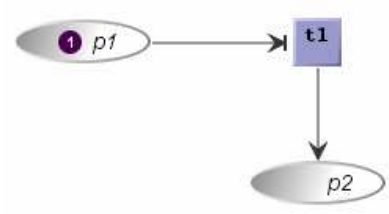


Figure 10 Arc de test (1)

L'arc de test (entre *p1* et *t1* dans la Figure 10) indique que *t1* n'est franchissable que si *p1* contient au moins un jeton.

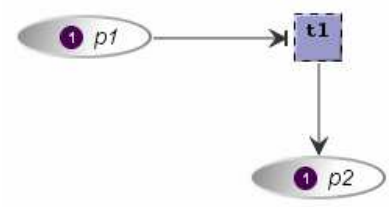


Figure 11 Arc de test (2)

Comme le montre la Figure 11, le franchissement de *t1* ne consomme pas de jeton dans *p1*, par contre un jeton est déposé dans la place *p2*.

3.4.6 Transition temporisée

Outre les transitions classiques, il existe une transition permettant de modéliser le temps que l'on appelle la transition temporisée.

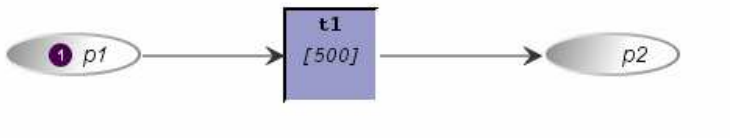


Figure 12 Transition temporisée

Ces transitions sont franchies après un délai exprimé en millisecondes que l'on peut lire dans la transition entre parenthèses (500 dans la Figure 12). Après ce délai, le jeton qui se trouvait dans la place *p1* est déposé dans la place *p2*.

3.4.7 Poids sur les arcs

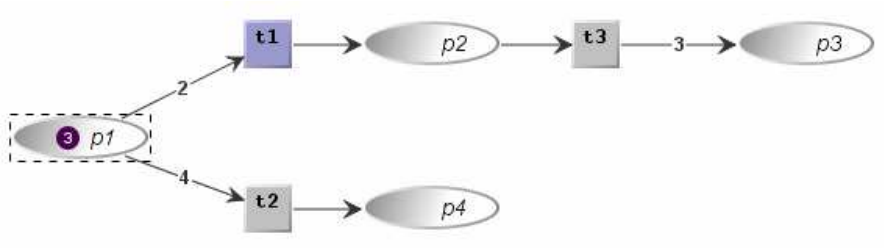


Figure 13 Poids sur les arcs (1)

Il est également possible de mettre un poids sur un arc. Dans l'exemple de la Figure 13, l'arc qui relie la place *p1* à la transition *t1* porte le poids 2. Cela signifie que la place *p1* doit contenir au moins deux jetons pour que *t1* soit franchissable.

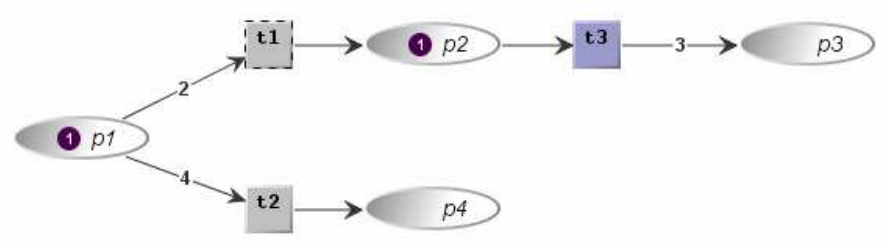


Figure 14 Poids sur les arcs (2)

Comme le montre la Figure 14, si *t1* est franchie, deux jetons sont retirés de la place *p1* et un nouveau jeton est déposé dans p2. L'arc reliant la transition *t3* à la place *p3* porte lui le poids trois. Cela signifie que lorsque la transition *t3* est franchie, trois jetons sont retirés de la place *p2* et déposés dans la place *p3*.

3.4.8 Représentation des structures de contrôle

Il est possible de représenter en réseaux de Petri les différentes structures de contrôle telles que la séquence, l'itération, le choix, le parallélisme et le sémaphore. Chacun de ces réseaux sera expliqué et exprimé ensuite en pseudo code.

3.4.8.1 La séquence

Figure 15 La séquence

La Figure 15 présente la séquence. Dans cette situation, le tir de la transition *Processus_1* déposera un jeton dans la place *p2* qui permettra le franchissement de la transition *Processus_2* et ainsi de suite jusqu à ce que le jeton soit déposé dans la place *p4*.

Ce réseau est équivalent au pseudo code suivant :

```
{
      Processus_1
      Processus_2
      Processus_3
}
```

Tableau 1 Pseudo code de la séquence

3.4.8.2 L'itération

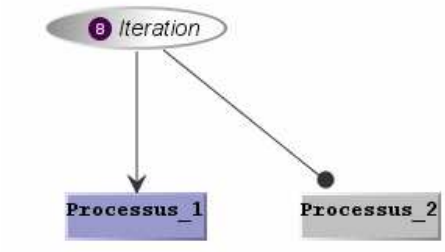


Figure 16 L'itération (1)

La Figure 16 présente l'itération. Dans cette situation, seul le tir du *Processus_1* est possible tant qu'il y a un ou des jetons dans la place *Itération*. La transition *Processus_1* sera donc tirée le nombre de fois qu'il y a de jetons dans cette place (Ici 8) et le code à l'intérieur de cette transition sera exécuté autant de fois.

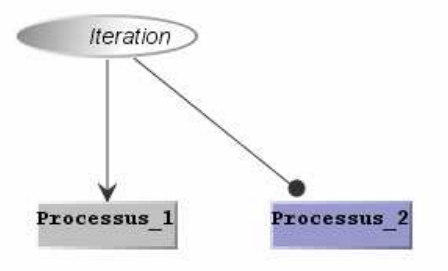


Figure 17 L'itération (2)

Lorsque la place *Itération* sera vide (Figure 17), l'arc inhibiteur activera le franchissement de la transition *Processus_2* et l'exécution du code s'y trouvant.
Ce réseau est équivalent au pseudo code suivant :

```
{
      for (int i =8 ;i > 0 ;i--)
      {
            Processus_1
      }
      Processus_2
}
```

Tableau 2 Pseudo code de l'itération

3.4.8.3 Le choix

La Figure 18 présente le choix. Dans ce réseau, nous allons voir le principe de transition ayant une condition. Lors du tir de la transition *Processus_1*, le code dans cette transition placera dans la variable booléenne (*a)* une valeur (*true* ou *false*) et déposera un jeton dans la place *Choix* avec cette valeur booléenne. Si cette valeur est *true*, la transition ayant comme condition (`a==true`) sera franchissable et un jeton sera déposé en *Début_Processus_True* sinon la transition (`a==false`) sera franchissable et le jeton sera déposé dans la place *Début_Processus_False*.
Selon que le jeton se trouve dans la place *Début_Processus_True* ou *Début_Processus_False* la transition *Processus_true* ou la transition *Processus_False* sera franchissable. Le tir d'une de cette transition dépose le jeton dans la place *Fin*.

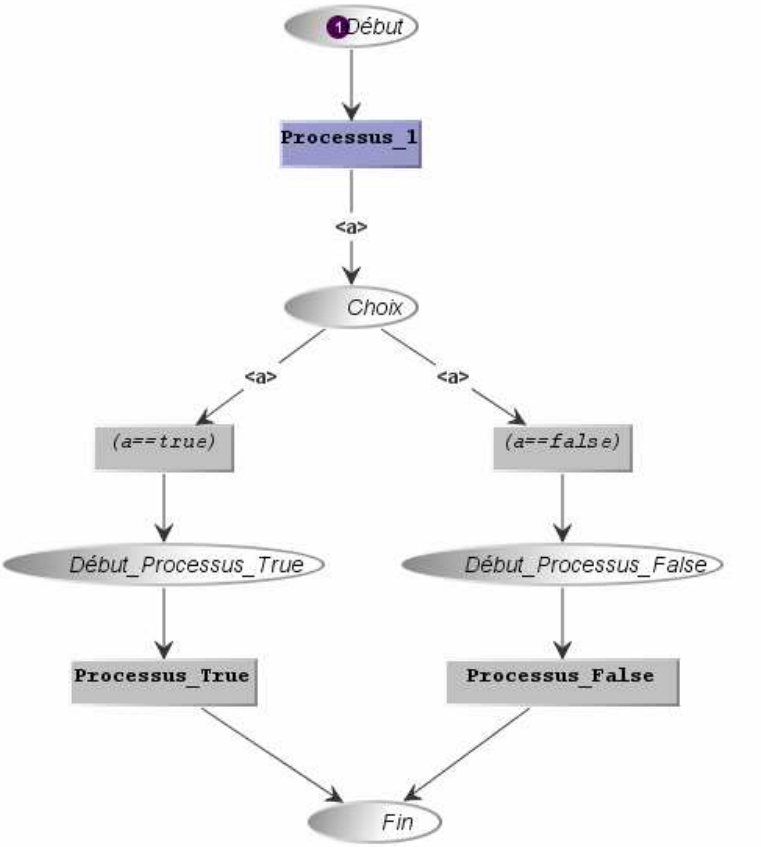


Figure 18 Le choix

Ce réseau est équivalent au pseudo code suivant :

```
{
      Processus_1
      If (a==true)
      {
            Processus_True
      }
      Else
      {
      Processus_False
      }
}
```

Tableau 3 Pseudo code du choix

3.4.8.4 Le parallélisme

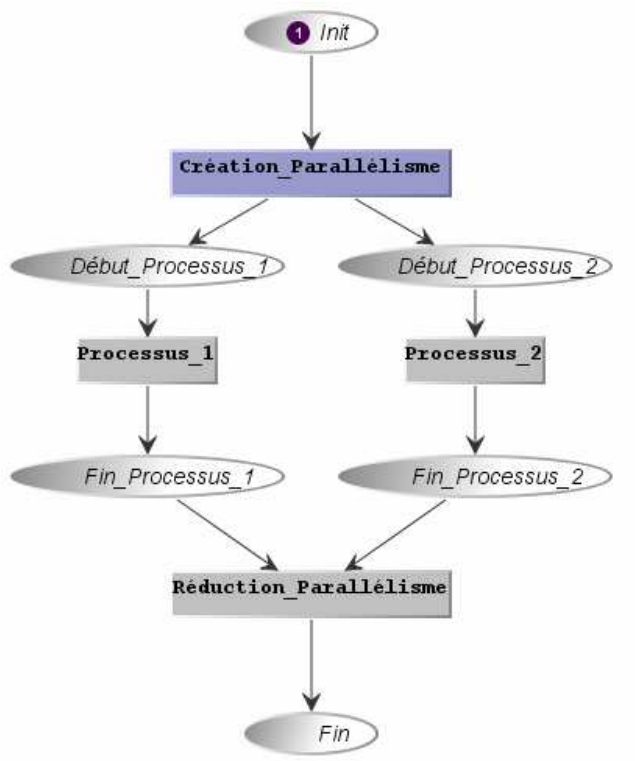


Figure 19 Le parallélisme

Les réseaux de Petri sont souvent utilisés pour modéliser la concurrence et la synchronisation. La Figure 19 nous montre un exemple de parallélisme. Dans ce réseau, lors du tir de la transition *Création_parallélisme*, un jeton sera déposé dans la place *Début_Processus_1* et un autre dans la place *Début_Processus_2*. Les transitions *Processus_1* et *Processus_2* sont alors franchissables. Celles-ci peuvent être tirées dans n'importe quel ordre. Le tir de *Processus_1* dépose un jeton dans *Fin_ Processus_1*, le tir de *Processus_2* dépose un jeton dans *Fin_Processus_2*. Lorsque ces deux places auront chacune un jeton, la transition *Réduction_Parallélisme* sera franchissable et son tir placera un jeton dans la place *Fin*. Cette réduction de parallélisme montre bien l'attente des deux processus avant de pouvoir continuer l'exécution.

3.4.8.5 Le sémaphore

Les réseaux de Petri permettent de modéliser la concurrence. Un cas classique de concurrence est l'accès à une ressource partagée (sémaphore) par deux processus concurrents.

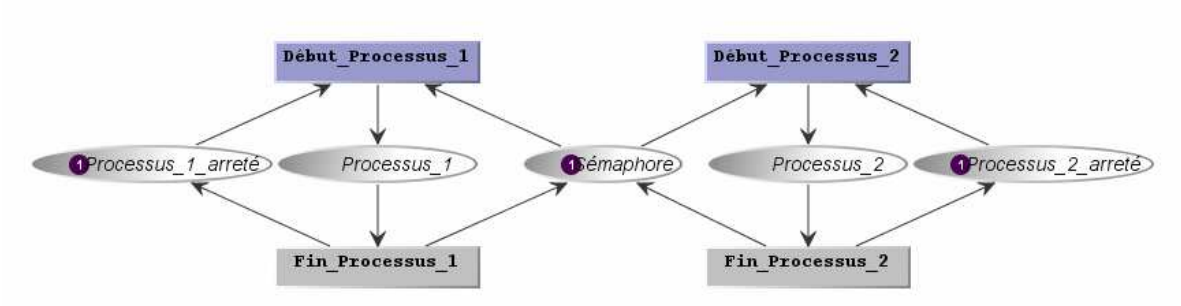


Figure 20 Le sémaphore (1)

La Figure 20 présente le sémaphore. Une ressource partagée est disputée par deux processus *Processus_1* et *Processus_2*. Cette ressource est représentée par le jeton qui se trouve dans la place *Sémaphore*. A l'état initial, les deux processus peuvent être exécutés. Ceci est représenté par la franchissabilité des transitions *Début_Processus_1* et *Début_Processus_2*. Les jetons qui se trouvent dans les places *Processus_1_arreté* et *Processus_2_arreté* expriment l'état du système à cet instant.

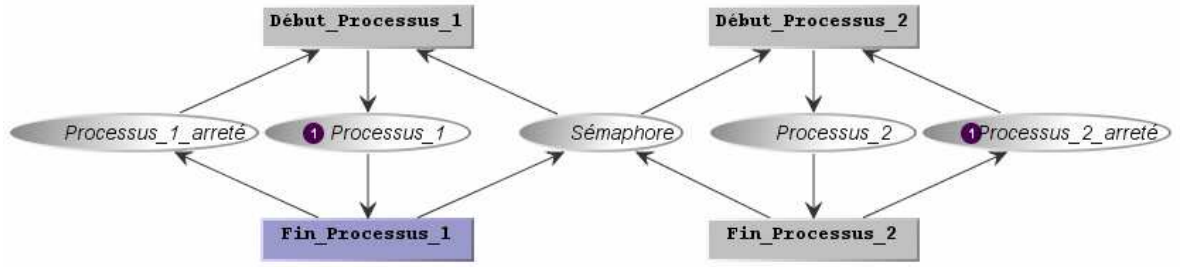


Figure 21 Le sémaphore (2)

Lors du tir de la transition *Début_Processus_1* (Figure 21), les jetons se trouvant dans les places *Sémaphore* et *Processus_1_arreté* sont enlevés et un jeton est déposé dans la place *Processus_1* exprimant l'exécution de ce processus. Nous pouvons voir qu'à cet instant il n'est plus possible de tirer *Début_Processus_2* car cette transition nécessite un jeton dans la place *Sémaphore* pour être franchissable. Ceci représente bien le principe de concurrence de deux programmes pour une ressource partagée. La seule action possible à cet instant est le tir de la transition *Fin_Processus_1* qui dépose un jeton dans les places Processus_1_arreté et *Sémaphore* replaçant ainsi le réseau dans son état initial.

3.5 Réseaux de Petri haut niveau - Extensions

Il existe différents types de réseaux de Petri qui présentent chacun leurs spécificités, liées aux domaines auxquels ils s'appliquent.
Parmi ceux-ci, nous avons les réseaux colorés [Jensen96], les réseaux stochastiques [Ajmone95] et les réseaux à objets [Valk98].
Nous utiliserons ces derniers, appelés OPN (Object Petri Nets), pour la description formelle que nous présentons ensuite.

« Les réseaux de Petri à objets reposent sur l'introduction des nouveautés suivantes :
les jetons circulant dans un OPN sont des références à d'autres objets du système, ajoutant aux réseaux de Petri la notion de dynamicité qui permet d'utiliser des instances puisque les objets font référence à des classes d'objets et non plus à un objet ;

- *les arcs sont étiquetés par des noms de variables ;*
- *les transitions peuvent contenir des actions ;*
- *la franchissabilité d'une transition est conditionnée non seulement par la présence de jetons dans ses places d'entrée, mais aussi par une précondition portant sur la valeur de ces jetons. »* [Navarre01]

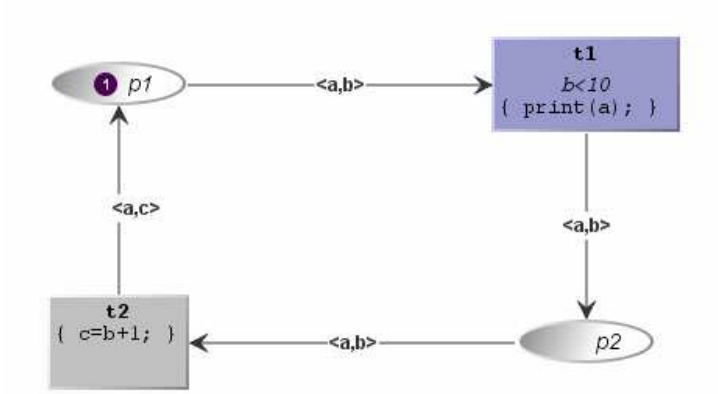


Figure 22 Réseau de Petri à Objets (OPN)

Sur la Figure 22, par exemple, les jetons circulant dans le réseau sont des couples constitués de :

- un objet *a* de type chaîne de caractères ;
- un entier *b*.

Supposons, par exemple, que dans l'état initial, le jeton de la place *p1* contienne un objet a du bon type et de l'entier 0.
La transition *t1* possède :

- une précondition `b < 10`, portant sur l'entier contenu par le jeton (ici, *b* vaut 0, la transition t1 est donc franchissable) ;
- une action `print(a)` qui affiche le contenu de l'objet *a* sur l'écran.

La transition *t2* ne possède pas de précondition, mais possède une action `c = b +1`.
Lorsqu'elle est franchie, elle prend le jeton de la place *p2* (ici, <a, 0>) et dépose un jeton dans la place *p1* (ici,<a,1>).
La transition *t1* est de nouveau franchissable, tant que la valeur de *b* est inférieure à 10 (c'est-à-dire jusqu'à ce que la transition *t2* ait été franchie dix fois).

Contrairement aux réseaux de Petri classiques, les jetons consommés lors du franchissement d'une transition transportent des valeurs qui peuvent être utilisées dans l'exécution de cette transition.
Les réseaux de Petri à objets définissent la notion de substitution comme l'ensemble des couples (nom de variables de l'arc, valeur correspondante du jeton). Dans l'exemple de la Figure 21, la substitution pour laquelle t1 est franchissable est {{a => x},{b => 0}}.

3.5.1 Objets Coopératifs (CO)

« Le formalisme des Objets Coopératifs permet la description de l'adaptateur du noyau fonctionnel et d'une partie de ce noyau fonctionnel (la partie manipulation de données, mais pas les données) d'un système interactif. Une description dans ce formalisme est constituée d'un ensemble de classes d'objets coopératifs (les CO-classes) qui coopèrent au moyen d'un protocole de type client-serveur. Une CO-classe spécifie une classe d'objets qui se conforme à une interface logicielle (qui décrit des services et leurs signatures) et utilise les réseaux de Petri à objets pour décrire le comportement des objets. L'interface logicielle est décrite à l'aide du langage Java. Le réseau de Petri décrivant le comportement d'un objet sera appelé ObCS (pour Structure de Contrôle de l'Objet, Object Control Structure). Ceci nous donne la définition suivante : CO-classe = interface Java + ObCS (exprimé par un réseau de Petri) » [Barboni06]

Le formalisme des Objets Coopératifs se distingue par trois caractéristiques :

- des objets dans des réseaux de Petri. Comme tout réseau de Petri à objets (OPN), l'ObCS d'une CO-classe contient des jetons qui font référence à des objets. Cette caractéristique permet d'utiliser les techniques de composition et de décomposition de l'approche à objets afin de construire le modèle d'un système ;
- des réseaux de Petri dans des objets. Le comportement d'une classe d'objets peut être représenté par un OPN, permettant ainsi de décrire précisément des comportements pouvant être concurrents ;
- un protocole client-serveur. Le formalisme des CO définit une communication entre objets en terme de réseau de Petri. Ainsi, un système est modélisé comme une collection d'objets qui interagissent en appelant mutuellement leurs services.

3.5.1.1 Notion de services dans les Objets Coopératifs

L'interface logicielle associée à chaque CO-classe permet de mettre en avant l'ensemble des services offerts par cette classe en en donnant la signature.

```
public interface MouseListener extends EventListener {
    public void mouseClicked(MouseEvent e);
    public void mousePressed(MouseEvent e);
    public void mouseReleased(MouseEvent e);
    public void mouseEntered(MouseEvent e);
    public void mouseExited(MouseEvent e);
}
```

Tableau 4 Signature de l'interface java java.awt.event.MouseListener

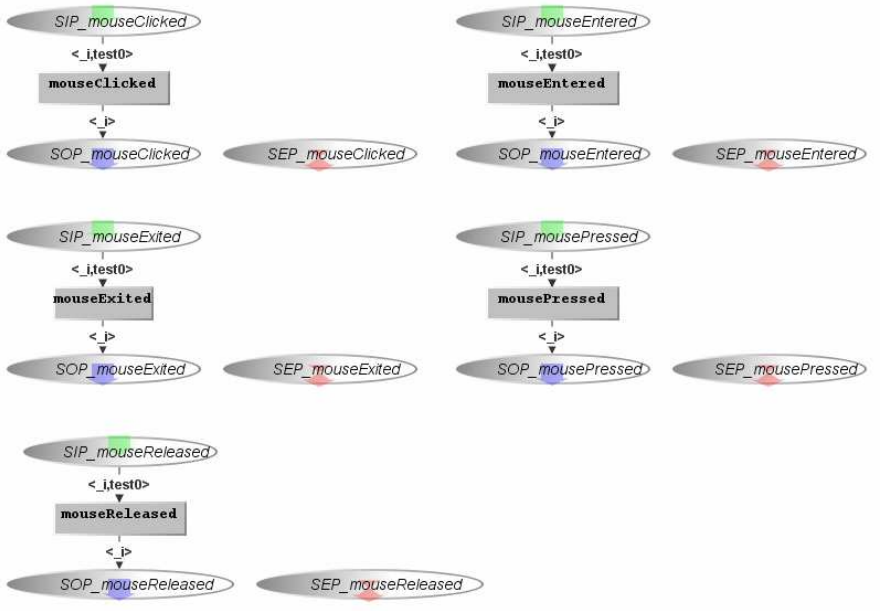


Figure 23 Représentation de l'interface MouseListener

Pour chacune des fonctions de l'interface, trois places sont créées. Une place d'entrée, une place de sortie et une place exception.

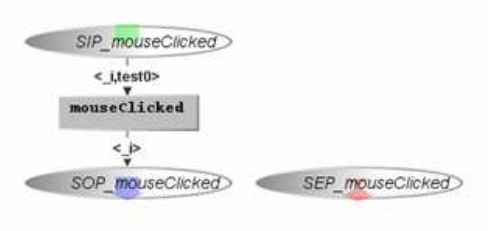


Figure 24 Détail du réseau MouseClicked

Dans le Tableau 4, nous voyons l'interface java *java.awt.event.MouseListener* contenant entre autres, la méthode *MouseClicked*. Pour chacune de ces méthodes, trois places sont créées dans PetShop.
Dans l'exemple de la Figure 24, nous trouvons donc la place d'entrée *SIP_mouseClicked*, la place de sortie *SOP_mouseClicked* et la place exception *SEP_mouseClicked*. Lors d'un appel de méthode *mouseClicked*, un jeton est déposé dans la place d'entrée avec comme valeurs, les différents paramètres de la fonction. Nous pouvons ensuite utiliser ces différents paramètres pour, par exemple, ajouter des nouveaux objets dans le réseau ou modifier les paramètres de ces objets. Pour rendre la main à l'application, un jeton est placé dans la place de sortie avec comme valeurs les paramètres de sorties de la fonction. Si nous désirons envoyer une exception, un jeton est déposé dans la place exception. L'exécution de ces fonctions est synchrone.

3.5.1.2 Structure de contrôle de l'objet (ObCS)

« L'ObCS permet de décrire le comportement d'un objet, c'est-à-dire sa façon de réagir aux stimuli extérieurs en fonction de son état interne. Cet ObCS est décrit par un réseau de Petri de haut niveau dérivant du dialecte des réseaux de Petri à objets. » [Navarre01]

3.5.2 Notions d'événements dans les CO - La programmation par événements

L'interaction homme-machine est dirigée par des événements. Ceux-ci proviennent des actions des utilisateurs sur des objets d'interaction et sont vus comme des transitions particulières. Ces transitions particulières, appelées transitions synchronisées, pour pouvoir être franchies, doivent non seulement être franchissables comme toute autre transition mais également recevoir un signal qui sert de déclenchement au tir de la transition.
Ces signaux sont produits à la suite d'actions de l'utilisateur sur les objets d'interactions. Pour utiliser ces événements, trois classes sont nécessaires : la classe émetteur, la classe récepteur et la classe abonnement.

3.5.2.1 Classe émetteur

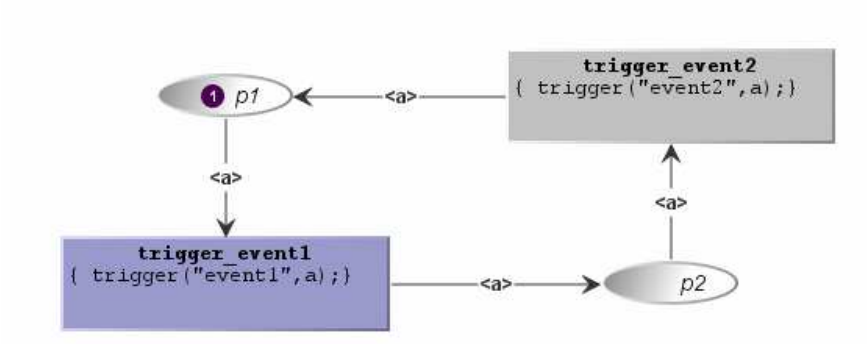


Figure 25 Classe Emetteur

La Figure 25 présente une classe émettant deux événements : *event1* et *event2*. L'émission d'événement est déclenchée par la commande `trigger (nom, paramètre)` qui envoie l'événement *nom* ayant comme valeur *paramètre*. Une classe capable d'émettre un événement implémente une interface *EventSource* permettant l'abonnement par d'autres classes.
Le comportement de cette classe est le suivant : Au départ, la place *p1* contient un jeton. La transition *trigger_event1* est donc franchissable. Un jeton est retiré de la place *p1*. Lors du franchissement de *t1*, un événement nommé *event1* est levé contenant a en paramètre. Un jeton est ensuite déposé dans *p2* ; La transition *trigger_event2* est alors franchissable. Un jeton est retiré de la place *p2*. Lors du franchissement de *trigger_event2*, un événement nommé *event2* est levé contenant *a* en paramètre. Un jeton est ensuite déposé dans *p1*.

3.5.2.2 Classe récepteur

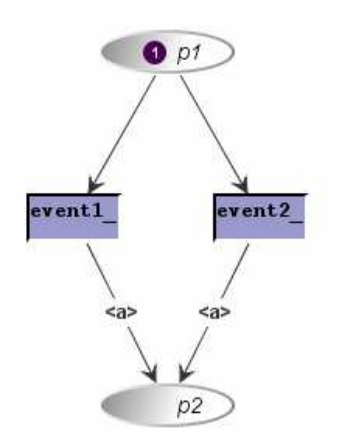


Figure 26 Classe Récepteur

La Figure 26 présente une classe écoutant deux événements *event1* et *event2*. Une classe capable d'émettre un événement implémente une interface *EventSink* permettant de démarquer ce type de classe.

Le comportement de cette classe est le suivant :

- Au départ, la place *p1* contient un jeton.
- Les transitions *event1_* et *event2_* sont franchissables. Il s'agit de transitions synchronisées. Ces transitions attendent l'arrivée d'un événement pour pouvoir être franchies.

Si un événement *event1* est reçu alors un jeton contenant le paramètre est déposé dans la place *p2*. Il en va de même pour *event2*.

3.5.2.3 Gestion des abonnements

Chaque instance de la classe *récepteur* a besoin de recevoir des événements pour pouvoir fonctionner. Les classes capables d'émettre des événements possèdent une méthode `addEventListener(nom,récepteur)` où le premier paramètre représente le nom de l'événement et le deuxième représente une instance de classe de type *EventSink*. La Figure 27 représente une classe tierce responsable de l'abonnement.

Le comportement de cette classe est le suivant :

- la place *Init_Emetteur* contient un jeton et la place *Init_Receveu*r contient un jeton ;
- la transition *creationEmetteur* va prendre un jeton de la place *Init_Emetteur*, exécuter l'action `emetteur = create Emetteur()` qui permettra la création d'une instance de la classe Emetteur. Un jeton contenant la référence de l'émetteur sera déposé dans la place *EmetteurCréé* ;
- la transition *créationReceveur* va prendre un jeton de la place *Init_Receveur*, exécuter l'action `receveur = create Recepteur()` qui permettra la création d'une instance de la classe Récepteur. Un jeton contenant la référence du récepteur sera déposé dans la place *ReceveurCréé*.

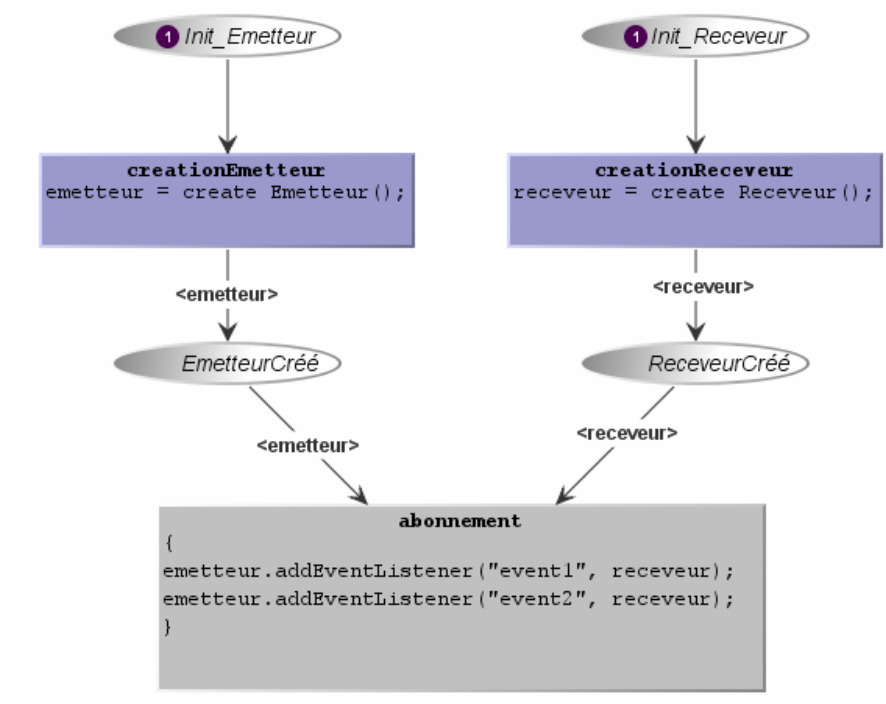


Figure 27 Classe Abonnement

Lorsqu'un jeton est dans la place *EmetteurCréé* et un autre dans la place *ReceveurCréé*, la transition abonnement devient franchissable. Lorsqu'elle est franchie, un jeton est retiré de la place *EmetteurCréé* et un de la place *ReceveurCréé*.
L'action de la transition abonnement est double :

- la première ligne `emetteur.addEventListener("event1", receveur)` indique que l'instance receveur va écouter l'événement *event1* émis par l'instance *émetteur* de la classe Emetteur.
- la seconde ligne `emetteur.addEventListener("event2", receveur)` abonne de la même manière le couple émetteur récepteur à l'événement *event2*.

3.5.3 Les Objets Coopératifs Interactifs - ICO

Les ICO [Navarre03] sont une extension des CO (objets coopératifs) permettant de prendre en compte les systèmes interactifs de type WIMP (Windows, Icon, Menus, Pointing). Les ICO permettent de décrire :

- le comportement du système grâce à un ensemble de CO-Classe.
- l'apparence graphique du système interactif grâce à la partie présentation.
- Le lien entre l'aspect comportement du système et l'aspect graphique se fait grâce à:
  - la fonction de rendu qui maintient la consistance entre l'état de la CO-Classe et son apparence.
  - la fonction d'activation, qui indique comment le système réagit aux entrées de l'utilisateur.

3.5.3.1 La représentation graphique

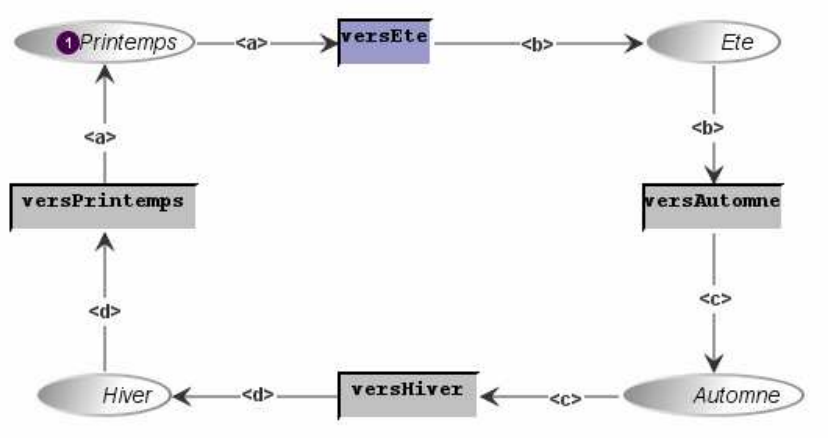


Figure 28 Réseau de Petri représentant le cycle des saisons

PetShop possède une API ICO en Java qui permet de réaliser les fonctions d'activation et de rendu permettant de connecter une spécification et une présentation. Pour chaque système interactif modélisé dans le formalisme ICO, nous devons donner l'ensemble des objets d'interaction qui la constitue et des méthodes de rendu qui peuvent être appliquées sur cet ensemble.

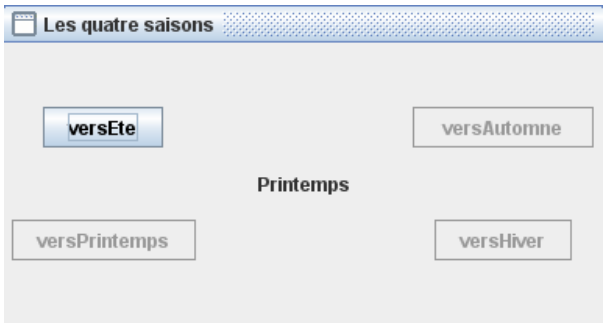


Figure 29 Interface liée au réseau de Petri des Quatre Saisons

La partie présentation nécessite une partie implémentée permettant la connexion avec les deux types de fonctions (activations et rendus). L'API ICO convient d'une dénomination particulière permettant la recherche par introspection sur des classes java de ces fonctions. Cette convention consiste à préfixer le nom des fonctions java destinées à l'activation par *ICOActivationRendering_* et le nom des fonctions java destinées au rendu par *ICORendering_.*

3.5.3.2 Fonction de rendu

Dans un réseau de Petri, les places sont les variables d'état du système (l'état du système étant modélisé par une distribution de jetons dans les différentes places) et les transitions sont les opérateurs de changement d'état. Les arcs, quant à eux, ne modélisent que la structure de causalité du système, représentant les conditions pré-requises aux changements d'état et leurs effets sur les états du système. La spécification du rendu doit donc être liée aux places des réseaux de Petri. Un changement d'état ne peut être représenté que par l'entrée ou la sortie d'un jeton d'une place. Si nous ajoutons la description de l'état initial, à chaque place d'un ObCS peuvent être associées trois méthodes de rendu pour trois cas bien distincts :

- une méthode associée à l'événement *jeton_entré* qui sera appelée à chaque fois qu'un jeton entre dans la place (par le franchissement d'une transition) ;
- une méthode associée à l'événement *jeton_retiré* qui sera appelée à chaque fois qu'un jeton est retiré de la place ;
- une méthode associée à l'événement réinitialisation qui sera appelée à chaque fois que l'on remet cette place dans son état initial.

La présentation d'informations auprès de l'utilisateur se décrit à travers la fonction de rendu qui met en relation les places et les transitions des ObCS du dialogue et un ensemble d'objets d'interaction, qui peuvent être utilisés pour présenter (rendre) de l'information à l'utilisateur. Le formalisme des ICO se base sur des réseaux de Petri de haut niveau pour spécifier le dialogue et l'interaction.

Nous pouvons représenter un lien entre le couple place et événement à une méthode de rendu dans ce tableau :

Place	Evénement	Méthode de rendu
Printemps	Jeton_entré	setSeasonLabel
Eté	Jeton_entré	setSeasonLabel
Automne	Jeton_entré	setSeasonLabel
Hiver	Jeton_entré	setSeasonLabel

Tableau 5 Liens entre les couples Place-Evénement et la méthode de rendu

Le rendu graphique est ensuite obtenu par des méthodes sur les objets de l'interface.
Ci-dessus *setSeasonLabel* modifie le texte de *seasonLabel* :

```
public void ICORendering_setSeasonLabel(MarkingEvent event) {
  seasonLabel.setText(event.getToken().getSlot(0).toString());
}
```

Tableau 6 Méthode de rendu

3.5.3.3 Fonction d'activation

L'activation des objets de l'interaction représente en partie l'espace d'interaction de l'utilisateur, en permettant de déterminer l'ensemble des actions utilisateur (ou événements) qui auront un impact sur le système interactif. Plus précisément, la fonction d'activation décrit le lien qui existe entre la disponibilité de services utilisateur et la possibilité d'accomplir une action sur un objet de l'espace d'interaction. Cet espace d'interaction est réduit aux états possibles du dialogue et à l'ensemble des événements jugés pertinents pour l'interaction avec le système modélisé. Nous pouvons représenter cette interaction par un tableau liant les actions sur les objets aux événements qui seront envoyés au réseau (Figure 28).

Action	Méthode de Rendu	Evénement Envoyé
Click BoutonPrintemps	setClassicEnabled	versEte
Click BoutonEté	setClassicEnabled	versAutomne
Click BoutonAutomne	setClassicEnabled	versHiver
Click BoutonHiver	setClassicEnabled	versPrintemps

Tableau 7 Liens entre les actions sur les objets et les événements envoyés au réseau

Dans une spécification par ICO, l'ensemble des interactions possibles peut être déduit du marquage courant de l'ObCS des différentes CO-classes qui modélisent le dialogue du système interactif.
La fonction d'activation se fait dans les deux sens :

- Les actions sur les objets de l'interface graphique envoient des événements vers le réseau de Petri

```
        versEte.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                support.fireWidgetEvent(FourSeasonsEventList.VERSETE_PERFORMED);
            }
```

Tableau 8 Envoi d'événements vers le réseau

- Le lien entre le réseau et la présentation active les comportements possibles des objets graphiques. Dans notre exemple (Figure 29), la fonction d'activation permet (*setEnabled*) le fonctionnement du bouton correspondant à la transition suivante dans le réseau (*versEte*)

```
public void ICOActivationRendering_setClassicEnabled(int theEvent,
            java.util.List<ISubstitution> theSubstitutions) {

 switch (theEvent) {
             case QuatreSaisonsEventList.VERSETE_PERFORMED:
                    versHiver.setEnabled(true);
                    versEte.setEnabled(False) ;
             break;
```

Tableau 9 Activation / désactivation des objets graphiques de l'interface

3.6 PetShop

Pour améliorer l'utilisabilité de cette approche, l'équipe du LIIHS a développé un environnement d'édition, de vérification et d'exécution de réseaux de Petri de haut niveau : PetShop.

L'accent a été mis sur son architecture logicielle afin de maximiser les possibilités d'extension de cet outil, notamment à la modélisation de systèmes interactifs.

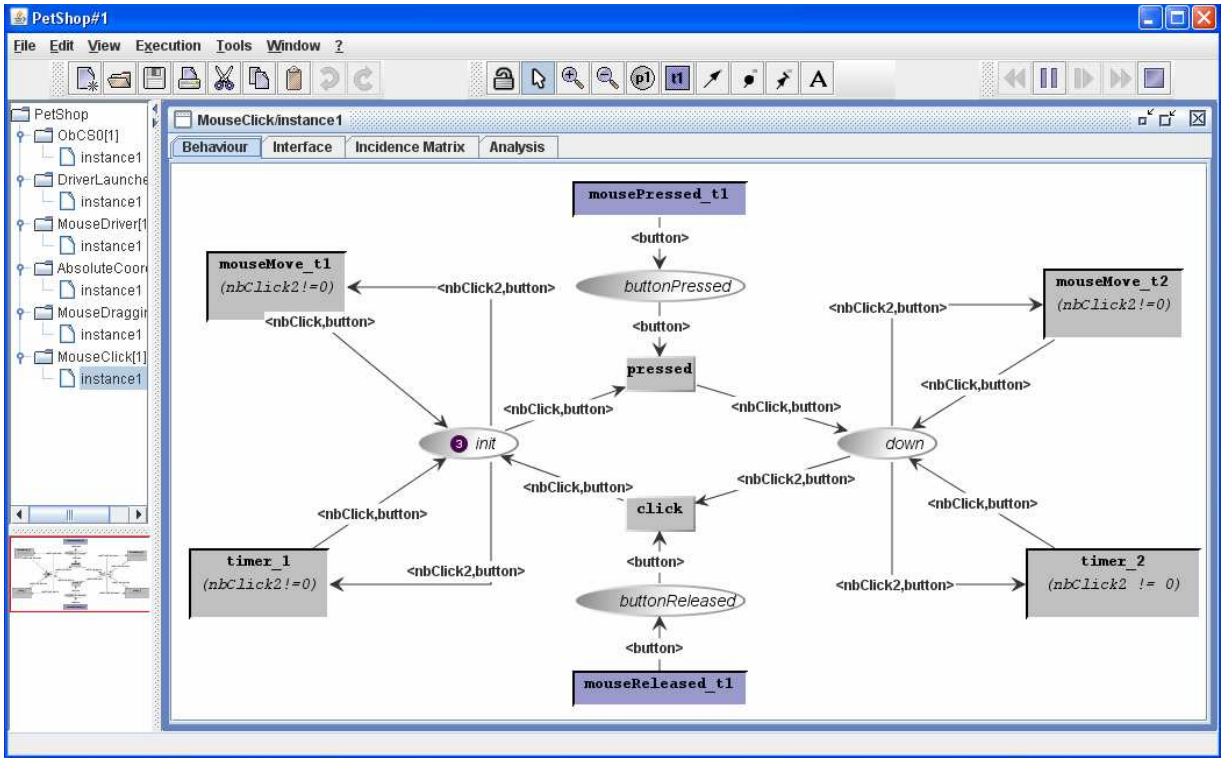


Figure 30 L'application PetShop

PetShop permet de réaliser certaines fonctions pour la conception de réseaux de Petri :

- édition de la spécification comportementale : PetShop comprend un éditeur de réseau de Petri;
- exécution, interprétation et débogage des réseaux de Petri : un avantage reconnu des réseaux de Petri est leur caractère exécutable. Cette fonctionnalité est exploitée dans PetShop qui fournit un environnement interprété avec certains avantages en terme de flexibilité, d'interactivité, de facilité d'utilisation et de productivité ;
- analyse mathématique des réseaux de Petri : un module d'analyse permet de vérifier des propriétés sur les réseaux, aussi bien structurelles (les invariants, les conflits…) que comportementales (la vivacité, le caractère borné des places, la présence d'un état d'accueil…). PetShop inclut ainsi des algorithmes d'analyses classiques qui vérifient des propriétés sur le réseau de Petri Place/Transition ;
- génération de la spécification comportementale du système à partir d'une interface Java (fournie par les concepteurs du système).

L’édition des réseaux de Petri à objets se fait graphiquement en utilisant la palette au centre de la barre d’outils. La partie gauche de cette barre d’outils est utilisée pour les fonctions classiques telles que l’ouverture ou la sauvegarde de fichiers ou les fonctions copier, couper, coller. La partie droite contrôle l’exécution de la spécification.

L’utilisation d’un environnement d’édition pour une notation formelle permet de faire passer celle-ci du stade de notation utile au stade de notation utilisable. Le caractère exécutable des réseaux de Petri, qui en forment la base, en augmente l’utilisabilité, en permettant le prototypage haute-fidélité.

L'exécution est fortement couplée à l'édition afin de pouvoir tester de façon interactive le comportement du modèle en cours de conception (sans besoin de phase de compilation préalable ou autre changement de mode).

PetShop offre la possibilité d’exécuter une spécification à la manière d’un débuggeur c'est-à-dire qu’il est possible d’exécuter la spécification courante en mode automatique ou en mode manuel. Le mode automatique permet de laisser à l’interprète le libre choix dans les substitutions à employer pour le franchissement. Ce mode automatique peut être réglé pour le pas à pas ou pour une exécution illimitée. Il permet également de revenir au marquage initial et de mettre en pause. Le mode manuel permet au concepteur de choisir la transition à franchir et éventuellement la substitution à effectuer.

C’est à l’aide de cet outil que nous modéliserons et exécuterons les réseaux de Petri pour notre contribution.

3.7 Avantages des réseaux de Petri sur les autres modèles

Dans les réseaux de Petri, les événements sont explicités dans le modèle par des transitions appelées transitions événements. Contrairement à l’approche orientée états, les événements ne sont pas dupliqués dans le modèle.
Les actions sont représentées par des transitions. Elles apparaissent explicitement et ne sont pas non plus dupliquées.
L’ensemble des états n’est pas directement visible, ce qui évite une explosion combinatoire en cas de multiples objets dans le graphe. Néanmoins, les différents états du système peuvent être obtenus par le graphe d’accessibilité du réseau de Petri.
La fonction de condition (association des événements aux actions) est clairement représentée par les arcs des places vers les transitions. Cette fonction n’est pas explicite dans les modèles à états finis.
Enfin, la fonction de l’impact est décrite par les arcs entre les transitions et les places.
On remarque également que les réseaux de Petri permettent de modéliser des comportements qui sont non déterministes (et fournissent des outils permettant de déceler s’ils le sont ou non) ce qui n’est pas le cas pour les automates à états finis ou les statecharts. Ces différents avantages sont détaillés dans [Caise93].

3.8 Conclusion

Notre objectif est de modéliser des systèmes interactifs complexes. Nous pouvons voir dans ce chapitre que les automates à états finis et les stateCharts ne sont pas adaptés à cette modélisation étant donné le nombre d'objets potentiellement infinis qui sont appelés à interagir avec notre système.

Cette partie nous a permis de montrer les avantages des réseaux de Petri sur les autres modèles de description formelle de systèmes et d'en expliquer les bases pour une meilleure compréhension de la suite de ce travail.

Nous avons tout d'abord présenté les différents modèles de description formelle de systèmes en commençant par les automates à états finis et les statecharts. Nous nous sommes ensuite principalement axés sur les réseaux de Petri. Dans ce chapitre, nous avons premièrement expliqué le fonctionnement des réseaux de Petri. Ensuite, nous avons exprimé les possibilités offertes par ce type de modélisation, plus particulièrement en ce qui concerne les réseaux de Petri à objets (OPN) et les objets coopératifs. Dans cette partie, nous avons également expliqué le fonctionnement des différents réseaux liés à la programmation par événements ainsi que les réseaux utilisés pour l'activation et le rendu d'objets graphiques. Nous avons introduit l'outil PetShop qui permet l'interprétation de nos modèles. Enfin, nous avons énuméré les avantages de la modélisation en réseaux de Petri par rapport aux automates à états finis et aux stateCharts.

4 Les périphériques d'entrée

4.1 Introduction

Dans cette partie, nous commencerons par expliquer la taxonomie de Buxton qui nous permettra de passer en revue les différents périphériques d'entrée et de lier ces périphériques par certaines propriétés comme le type de calcul de coordonnées ou le mouvement à réaliser pour l'utilisation du périphérique. Nous détaillerons ensuite plus particulièrement les périphériques qui seront utilisés dans la deuxième partie de ce mémoire (l'écran tactile, la parole, la souris et le TrackBall)

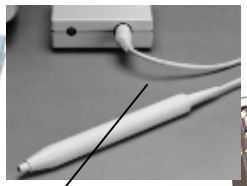
4.2 La taxonomie des dispositifs physiques selon Buxton

Buxton [Buxton90] classe les différents dispositifs d'entrée suivant deux critères principaux (Figure 31). Le premier correspond aux propriétés que le dispositif est capable d'interpréter : la position, la pression et le mouvement. Ce critère est lui même découpé en deux sous-critères qui nous permettent de savoir s'il y a un intermédiaire mécanique (M) ou non (T). Le second critère nous donne le nombre de dimensions dans lesquelles évolue le dispositif. Par exemple pour une tablette, la dimension est deux (on prend une valeur sur l'axe x et une autre sur l'axe y). Ce critère est lui aussi décomposé en sous-parties qui définissent la façon dont on se sert du périphérique.

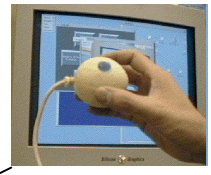
Parmi ces différents critères, on trouve le type de position envoyée par le périphérique pour se situer dans l'espace : la position absolue ou la position relative. Lorsqu'il travaille en position absolue (comme un écran tactile), le périphérique envoie au système sa position exacte (absolue) en x et en y (éventuellement en z). Lorsqu'un périphérique travaille en position relative, il envoie au système les valeurs dx et dx (éventuellement dz) correspondant au déplacement depuis la dernière position connue. La position de ce périphérique est à l'origine initialisé au centre ou dans un coin de l'écran.

La taxonomie est tournée vers l'aspect technique du dispositif (comment il fonctionne, comment il est utilisé).






Property sensed		Number of dimensions							
		1		2				3	
Position		Rotary pot.	Sliding pot.	Tablet & puck	Tablet & stylus PenCat	Light pen	Isotonic Joystick GamePad	Joystick Stylet (Phantom)	M
					Touch tablet	Touch screen			T
Motion		Continuous rotary pot.	Treadmill Wheel	Mouse Trackpoint mouse		TouchPad	Trackball Trackpoint	Mouse Aircraft	M
			Ferinstat				X / Y Pad	GamePad Glove	T
Pressure		Torque sensing	Pressure pad				Isometric Joystick		T
		Rotary	Linear	Puck	Stylus finger vert.	Stylus finger hor.	Small fixed location	Small fixed with twist	



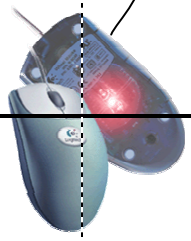






Figure 31 Tableau adapté de la taxonomie de Buxton.

4.3 Tour d'horizon des dispositifs existants

4.3.1 Dispositifs à une dimension

- Le potentiomètre rotatif et coulissant :

Le potentiomètre rotatif et coulissant est un dispositif minimaliste borné. Il est utilisable pour des tâches simples et précises (exemple : contrôle du volume).

- Le volant :

Le volant s'utilise comme un volant de voiture normal. Il permet de se déplacer sur un axe horizontal (position absolue). Certains volants bénéficient d'un retour de force, ce type de dispositif est surtout utilisé pour des simulations de jeux de voitures.

4.3.2 Dispositifs à deux dimensions

- Tablette et palet :

La tablette est accompagnée d'un palet. Le pointeur se déplace à l'écran lorsqu'on déplace le palet sur la tablette (position absolue).

- Tablette et stylet :

Le stylet est tenu dans la main (comme un stylo classique) que l'on déplace sur la tablette et cela fait déplacer le pointeur à l'écran (position relative).

- Le Pantograph :

Le Pantograph se comporte comme un dispositif d'entrée classique, auquel est ajouté un retour d'effort et de sensation. Il travaille en position absolue.

- PenCat Pro :

Le PenCat Pro s'utilise comme un stylo normal. Son envergure couvre la totalité de l'écran (position absolue). Ce dispositif comprend un retour de force qui permet à l'utilisateur de sentir des formes (fossé, bosse, …).

- TouchTablet :

La TouchTablet est une tablette posée (généralement) à l'horizontale capable de situer la position du doigt lorsqu'il est en contact (position absolue). Un dispositif permettant de capter le degré de pression du doigt sur la tablette peut être ajouté mais aussi le rendre capable de percevoir plusieurs contacts en même temps.

- Lightpen :

Le Lightpen est utilisé comme un stylo normal pour pointer sur l'écran (position absolue).

- Touchscreen ou écran tactile :

L'écran tactile repère la position du doigt lorsqu'il touche l'écran (position absolue). Ce dispositif ne prend pas de place (intégré à l'écran), et permet de garder le regard de l'utilisateur sur l'écran (pas de discontinuité du regard entre l'écran et le dispositif).

- Manche à balai ou joystick :

Les manches à balai utilisent la technique de position relative ou absolue.
Il faut empoigner le manche à balai, que l'on déplace avec des mouvements de poignets vers la direction voulue.

- Souris :

La souris est, à l'heure actuelle, le dispositif le plus robuste et le plus usité. Ce dispositif utilise la technique de position relative.

- GamePad ou manette de jeu :

La manette de jeu monopolise les deux mains. Le pouce gauche sert au déplacement grâce à un pavé multidirectionnel, l'autre pouce permet d'activer des boutons prédéfinis. Le GamePad est surtout utilisé pour les jeux. Le retour de force est disponible sur ce dispositif.

- TouchPad :

Le TouchPad s'utilise en faisant glisser son index sur la surface du dispositif. Il utilise la position relative. Il y a deux façons pour réaliser l'équivalent du clic souris : lever et taper (lift and tap) ou utiliser des boutons physiques. Un des avantages du TouchPad est son encombrement réduit ce qui fait de lui un dispositif de choix pour les ordinateurs portables.

- TrackBall :

Le TrackBall utilise une boule roulante sur le dessus que l'on fait tourner à l'aide des doigts (position relative). Tout comme pour le TouchPad il y a des interférences entre les muscles car le click est réalisé avec le pouce et le déplacement de la boule est réalisé avec les doigts.

- Trackpoint :

Le TrackPoint s'utilise en pressant doucement sur le bouton avec l'index dans la direction voulue (position relative). Ce dispositif est souvent rencontré sur les ordinateurs portables en raison de son encombrement très faible (il se place entre deux touches d'un clavier classique).

4.3.3 Dispositifs à trois dimensions

Il existe deux catégories de dispositifs possédant six degrés de liberté. La première baptisée « souris volantes » correspond à tous les types de dispositif s'inspirant du succès de la souris classique (bidimensionnelle). Leur avantage est l'apprentissage facile et plus rapide que les autres types de dispositif. Par contre, il y a des problèmes que l'on ne rencontre pas avec la souris classique : lorsqu'on lâche le dispositif, le pointeur ne reste pas à sa place ; de plus, à l'usage, ce type de dispositifs fatigue l'utilisateur (le bras est toujours levé) ; enfin l'acquisition est difficile. La seconde catégorie regroupe les dispositifs « bureau ». Ces derniers sont des dispositifs posés sur une surface stable. Les avantages de cette catégorie sont de réduire la fatigue (le bras repose sur le bureau) et de faciliter l'acquisition par une stabilité accrue. Par contre, un manque de retour d'information est noté ainsi qu'un entraînement conséquent pour parvenir à la rapidité des « souris volantes ».

Souris volantes :

- La FingerBall est un des seuls dispositifs avec 6 degrés de liberté manipulable avec les doigts. Ce dispositif utilise la technologie AscensionBird™ monté au centre de la sphère de 6cm de diamètre (position relative). Ce dispositif est considéré comme l'un des meilleurs dispositifs 3D car il est le seul qui utilise les doigts.

- Le Cricket permet de se déplacer librement dans l'environnement tridimensionnel, un retour de type vibration est envoyé lorsque l'utilisateur entre en contact avec un objet. De plus, le bouton est accessible avec le pouce dans toutes les positions possibles.

- La CubicMouse s'utilise avec les deux mains à la fois. Il se repère grâce à un système de potentiomètre (position relative). La sélection se fait par des boutons de contrôles situés sur les faces du cube.

- Le GamePad s'utilise comme un GamePad normal, mais celui-ci détecte et traduit les mouvements du dispositif.

- Le gant se repère par rapport à un capteur magnétique situé sur la paume de la main. Il peut aussi être accompagné d'un retour de force ou de sensation de toucher.

Dispositifs de bureau :

- Le Phantom s'utilise comme un stylo que l'on fait évoluer dans un monde 3D. Ce dispositif donne le sens du touché à l'utilisateur. Ce dernier peut aussi manipuler des objets en trois dimensions. Il fait partie de la catégorie dispositif de bureau.

- Le joystick 3D se présente comme le joystick 2D, et permet d'évoluer dans un environnement en trois dimensions. Certains joystick bénéficient du retour de force et/ou de la sensation du touché.

- La SpaceMouse, la SpaceBall et le PuckMan (Figure 32) sont des dispositifs assez semblables dans leurs fonctionnements. C'est l'équivalent du TrackBall avec 6 degrés de liberté. En exerçant des pressions sur la balle, on déplace le pointeur.

Figure 32 La SpaceBall, la SpaceMouse et le PuckMan.

4.3.4 Les dispositifs haptiques - Le retour d'effort et sensitif :

Les systèmes à retour d'effort permettent de restituer une force en réponse aux manipulations de l'utilisateur. Les dispositifs d'entrée possédant un retour d'effort et sensitif, sont à la fois des dispositifs d'entrée et des dispositifs de sortie comme nous le montre la Figure 33 . Cette caractéristique leur donne un statut particulier dans la grande famille des dispositifs d'entrée. Avec cette technologie, on exploite ainsi un sens pas ou peu exploité jusqu'à maintenant : le sens du toucher.

Le nombre de travaux exploitant ces propriétés augmente de façon considérable. Une partie concerne une aide aux personnes non ou mal voyantes.

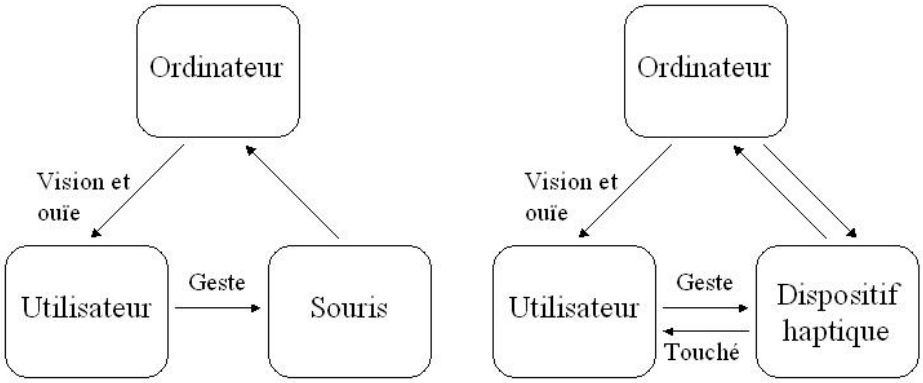


Figure 33 A gauche, boucle conventionnelle d'interaction. A droite, interaction avec un dispositif haptique.

Le retour de force (force feedback) et le retour sensitif (sensitive feedback) ont fait l'objet de plusieurs études afin de connaître l'utilisabilité de cette technologie et son apport.
La technologie haptique doit être utilisée à bon escient. La technologie peut fournir un énorme avantage aux personnes ayant un handicap (visuel ou auditif) et compléter leur manque de perception.

4.3.5 Les exclus de la classification de Buxton

Dans la classification de Buxton, on ne retrouve que les dispositifs (continus) actionnables par le geste. Les systèmes d'entrée tels qu'un microphone ou une caméra numérique sont omis. Ces dispositifs aboutissent à de la reconnaissance vocale, à un suivi du regard ou encore à une reconnaissance de geste. Pour le suivi du regard, il existe deux façons de procéder, soit le dispositif (caméra(s)) est placé en face de l'utilisateur (généralement sur l'écran), soit le dispositif est placé sur l'utilisateur à l'aide d'un casque (plus contraignant et fatiguant).

Ces dispositifs, qui ne sont pas manipulables directement par l'utilisateur, permettent de réduire la fatigue physique de ce dernier, mais aussi de garder les mains de l'utilisateur libres pour exécuter d'autres actions. Néanmoins, l'utilisateur doit être concentré sur ce qu'il fait (suivi du regard) et sur ce qu'il dit (reconnaissance vocale), ce qui peut provoquer une fatigue nerveuse prématurée. Enfin, ces systèmes d'entrée ne peuvent pas bénéficier de la technologie haptique.

4.4 Présentation détaillée de quelques dispositifs

Dans la partie contribution, nous donnerons une spécification complète des techniques d'interactions multimodales dans un système interactif pour différents périphériques. Il ne nous semble pas utile de détailler la souris et le trackball qui sont bien connus. Par contre, nous détaillons ci-dessous la reconnaissance vocale et l'écran tactile en donnant un exemple d'utilisation.

4.4.1 L'écran tactile

Présentation

Un écran tactile est un dispositif d'affichage et de désignation directe : l'utilisateur pointe un élément affiché sur un écran avec son doigt ou un stylet. Ce périphérique travaille avec une position relative.
Les écrans tactiles sont sortis des laboratoires de recherche depuis plusieurs années. Ils sont utilisés dans les musées, les kiosques multimédia, les bornes de vente de tickets, …

Figure 34 Ecran tactile - M150 FPD Touch Monitor

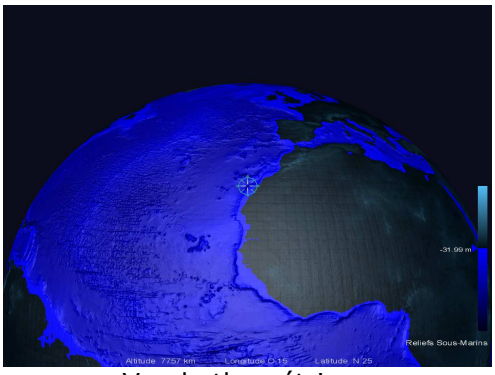
Les deux premiers modèles s'intègrent dans un poste de travail. Le dernier est un moniteur au sens classique complété par une dalle tactile. Le mode d'interaction des écrans tactiles (désignation directe) ne demande pas d'apprentissage. C'est pourquoi il est utilisé dans les lieux grand public. De plus, il n'impose pas de préférence de manipulation (main droite ou main gauche). Enfin, une application peut être plus facilement pilotée par plusieurs utilisateurs.

La précision du pointage dépend du media utilisé (doigt, stylet, …).Toutefois, il est difficile d'atteindre la précision de la souris.

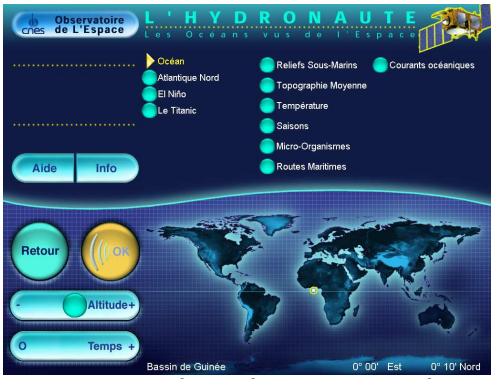
Exemple

CS (Communication et Systèmes) a développé pour le CNES (Centre National d'Etudes Spatiales) l'application HYDRONAUTE. Il s'agit d'une application permettant la visualisation et la découverte en temps réel du fond des océans, des grands courants océaniques, du déroulement des phénomènes climatiques (« El Niño »), d'une épave célèbre (le Titanic), etc.

Borne HYDRONAUTE



Vue bathymétrique



Interface présentée sur un écran tactile

Figure 35 La borne Hydronaute à la Cité de l'Espace

(© P.Dumas/Observatoire de l'Espace – CNES – CS/SI)

4.4.2 La reconnaissance Vocale

Présentation

La modalité la plus courante pour communiquer entre êtres humains est la parole. Celle-ci peut également avoir de nombreux avantages pour une interaction homme-machine :

- Elle peut facilement être utilisée en addition d'un autre dispositif : en effet, elle n'utilise ni les mains, ni les yeux.
- Il est possible de désigner des objets non visibles à l'écran en les désignant par exemple par leur nom ou par leur description. Ce qui n'est évidement pas possible avec un dispositif de type pointeur. On peut donc par exemple accéder à des commandes telles que « écrire un email à Sandra» sans devoir passer par un menu de choix d'application, le menu de cette application, le choix du destinataire, etc.

En contrepartie, la parole souffre de quelques désavantages :

- La parole ne se dirige pas que vers la machine. Elle occasionne donc une gêne pour les utilisateurs voisins et n'assure pas la fiabilité des informations.
- Etant donné qu'elle est utilisée comme mode de communication entre humains, il faut prévoir un dispositif de basculement pour que la machine sache si l'utilisateur lui parle ou parle à une autre personne.
- La dépense physique est importante surtout en cas de répétition de mêmes séquences de commandes.

Applications

Nous pouvons citer comme applications utilisant la reconnaissance vocale :

- La numérotation téléphonique automatique sur les Gsm.
- L'entrée de données et de textes dans les applications bureautiques (Office) ou dans des formulaires.
- Le pilotage d'un système d'exploitation (Windows Vista, Apple)
- Les outils de traductions simultanées et l'enseignement des langues avec correction des erreurs de l'étudiant.
- L'identification de personnes au moyen d'empreinte vocale et reconnaissance d'un code oral.
- La commande d'appareillage de navigation Gps en voiture.
- Les applications grand public (jeux, matériel hi-fi)
- L'aide aux personnes handicapées (aveugles, paralysées)

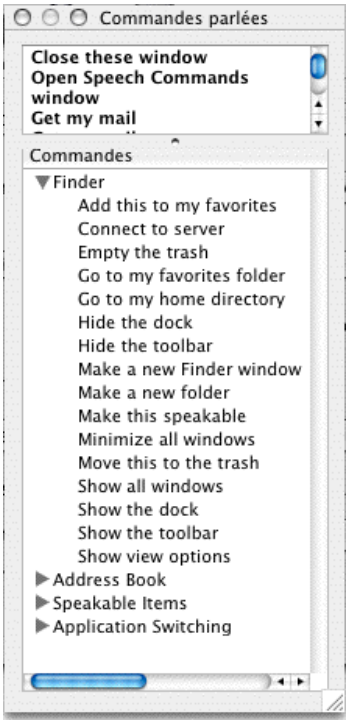


Figure 36 Pilotage d'un système d'exploitation (MacOS)

4.5 Conclusion

L'objectif de cette partie était de situer les différents périphériques que nous allons utiliser dans ce mémoire au moyen de la taxonomie de Buxton.

En replaçant les périphériques choisis dans la taxonomie de Buxton, nous remarquons qu'ils sont de types fort différents. Nous avons un périphérique à position absolue (l'écran tactile), un périphérique à position relative (la souris) et un périphérique exotique qui ne fait pas partie de la taxonomie de Buxton (la reconnaissance vocale). Ils représentent donc un sous-ensemble complémentaire pour la réalisation de nos modèles dans la seconde partie de ce mémoire.

5 Multimodalité

5.1 Introduction

Etant donné que nous allons réaliser des modèles d'interactions multimodales, il nous paraît important de définir la multimodalité et d'en donner quelques exemples.

Nous commencerons par définir les termes de modalité et de multimodalité. Nous parlerons ensuite de la classification des types de modalité de Caelen. Enfin, nous présenterons un bref aperçu de deux exemples de multimodalités se rapprochant de nos architectures multimodales.

5.2 Définition

5.2.1 Modalité

« La modalité est définie comme étant un couple <d, l> où d désigne un dispositif physique et l un langage d'interaction. Le langage d'interaction fait référence au langage permettant à l'utilisateur de communiquer avec la machine au travers d'un dispositif physique particulier. Un langage d'interaction se définit donc par un vocabulaire d'éléments terminaux et une grammaire. Les éléments terminaux sont produits ou captés par les dispositifs d'entrée/sortie. » [Nigay94]

Turk et Robertson [Turk00] distinguent deux catégories de modalités d'entrée qu'ils définissent comme suit :

- **Modalité active :** *« Une modalité d'entrée est dite active si l'utilisateur doit l'utiliser de façon consciente. Par exemple, la métaphore du glisser-déposer avec une souris correspond à une modalité active, dans la mesure où l'utilisateur doit réaliser une action précise et explicite pour utiliser cette modalité. »*

- **Modalité passive :** *« À l'opposé, une modalité est passive si elle n'est pas utilisée de façon consciente par l'utilisateur. C'est le cas par exemple de la plupart des modalités faisant intervenir le suivi du regard de l'utilisateur. Ces modalités fournissent des informations sur l'utilisateur au système sans que celui-ci intervienne explicitement. »*

5.2.2 Multimodalité

« Un système est multimodal s'il dispose d'au moins deux modalités pour un sens donné. Ainsi, le système peut être qualifié de multimodal en entrée si au moins deux modalités d'entrée sont disponibles et de multimodal en sortie si au moins deux modalités de sortie existent. Il est donc possible qu'un système soit multimodal même s'il ne fait intervenir qu'un seul dispositif physique. Il suffit pour cela que plusieurs langages d'interaction soient associés à un même dispositif physique ». [Bellick95]

5.3 Les types de multimodalité

Les types de multimodalité désignent la manière dont l'utilisateur en entrée ou la machine en sortie interagit. Une première classification des types de multimodalité a été introduite par Caelen [Caelen91]. Celle-ci repose sur deux critères permettant de définir les différents types de multimodalité :
• L'usage des modalités (en parallèle ou en séquentielle),
• Le lien entre les informations provenant de deux modalités (combinées ou indépendantes).

		Usage des modalités	
		Séquentiel	Parallèle
Fusion / Fission	Combiné	Alterné	Synergique
	Indépendant	Exclusif	Concurrent

Tableau 10 Les différents types de multimodalité

On obtient ainsi quatre types d'interaction multimodale, valables en entrée comme en sortie:

- Exclusif : les modalités ne peuvent pas être fusionnées et elles doivent être utilisées en séquence.
- Concurrent : les modalités ne se fusionnent pas pour former une commande, mais on peut les utiliser en parallèle à des tâches différentes.
- Alterné : les modalités peuvent être fusionnées entre elles, mais elles doivent être utilisées en séquence.
- Synergique : les modalités peuvent être utilisées librement : la fusion est possible ainsi que le parallélisme.

5.4 Quelques exemples de multimodalité

Cette section a pour but de présenter les systèmes de référence dans le cadre de la multimodalité en entrée. Nous présentons ici le premier système multimodal réellement implémenté, le système de Bolt ainsi qu'un des premiers systèmes utilisant réellement la multimodalité à deux mains, les Toolglass™.

Cette section n'a pas pour but d'être exhaustive mais de montrer des exemples de multimodalité dont nous nous sommes inspirés pour ce mémoire.

5.4.1 « Put That Here » ou« Mets ça là » [Bolt80]

Bolt [Bolt80] a été un des premiers à travailler sur l'utilisation conjointe de la parole et du geste et c'est lui qui a développé le premier prototype réel. Cet exemple reste une référence dans le domaine de la multimodalité. On parle du paradigme du « mets ça là ». L'utilisateur interagit grâce à la parole et au geste de désignation sur des objets graphiques affichés sur un mur devant lui. La parole est le mode dominant, au moyen duquel sont exprimées les différentes commandes. Le geste permet de compléter ces commandes en désignant les objets et les positions concernés. Il existe cinq commandes prédéfinies : Créer, Déplacer, Transformer, Effacer et Nommer. Chaque objet a une forme (triangle, carré ou losange), une couleur (jaune, orange, rouge, violet, bleu, vert, …) et une taille (petite, moyenne ou

grande). Par défaut, la taille d'un objet à la création est « moyenne ». Par contre, il n'y a ni couleur ni forme par défaut. L'utilisateur pourra par exemple dire : « Crée un carré bleu ici » tout en indiquant une zone de l'écran, ou bien « Déplace ça à la droite du carré vert. » (Le pronom « ça » allié au pointage permet d'indiquer un objet) ou encore « Mets ça là. ». L'usage des modalités est donc qualifié d'alterné et de synergique.

5.4.2 Toolglass and Magic Lenses [Bier93]

Bier et son équipe [Bier93] ont développé une technique d'interaction multimodale originale. En effet, cette technique utilise les deux mains plutôt que la combinaison parole et geste. De plus, il s'agit simplement de l'augmentation d'une interface GUI traditionnelle.
Une Toolglass™ est une fenêtre semi-transparente qui contient différentes Magic Lenses™ (lentilles magiques). Ces lentilles sont des zones de l'écran associées à une fonction. Par exemple, certaines lentilles permettent de zoomer, c'est-à-dire que la zone qui est sous la lentille apparaît plus grosse (ou plus petite). D'autres lentilles permettent de colorier un dessin, de dessiner une forme particulière, etc. L'utilisateur peut positionner la Toolglass ™ au-dessus des objets d'interaction avec la main dominante et interagir avec la main non dominante. Un exemple d'interaction est donné par la Figure 37.

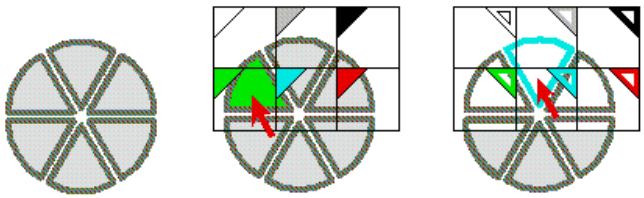


Figure 37 Exemple d'interaction avec une Magic Lense

Plusieurs lentilles ayant des fonctions différentes peuvent être placées sur une ToolglassTM. L'utilisateur peut donc passer d'une commande à une autre (ou d'une visualisation à l'autre) simplement en repositionnant la fenêtre.
Si l'on dispose de plusieurs lentilles, on peut les superposer et leurs actions seront alors combinées. La Figure 38 montre ce qu'il se passe si on superpose une lentille permettant de zoomer et une lentille permettant de colorier une zone.

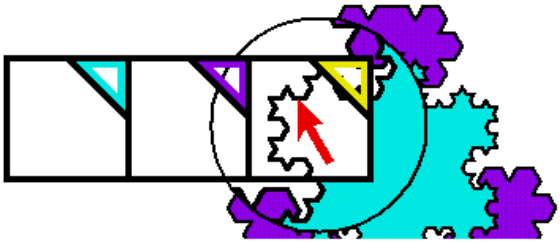


Figure 38 Combinaison de deux lentilles

L'usage des modalités peut donc être qualifié d'alterné et de synergique.

5.5 Conclusion

Dans cette partie, nous avons tout d'abord défini les termes de modalité et de multimodalité. Nous avons ensuite présenté la classification des multimodalités définie par Caelen. Cette classification nous permettra de situer le type de multimodalité de nos exemples dans la seconde partie de ce mémoire. Enfin, nous avons présenté deux exemples de multimodalité le « Put That Here » et les « Magic Lenses ».

6 Conclusion

Nous avons défini dans cet état de l'art différents concepts qui nous seront utiles pour positionner notre travail.

Nous avons tout d'abord présenté les modèles d'architecture des systèmes interactifs. Parmi Seeheim et Arch, nous avons choisi ce dernier car la partie présentation est plus détaillée.

Nous avons ensuite explicité des modèles de description formelle de systèmes. Nous avons expliqué pourquoi les automates à états et les statecharts ne convenaient pas aux domaines d'application ciblés, c'est-à-dire, les techniques d'interactions multimodales.

Ensuite, nous avons expliqué par différents exemples les possibilités de modélisations des réseaux de Petri ainsi que celles des réseaux de Petri à Objets (OPN). Dans cette partie, nous avons également énoncé les différentes notions utilisées dans la contribution telles que la programmation par événements et les méthodes d'activation et de rendu. De plus nous avons introduit l'environnement PetShop que nous utiliserons pour la création de nos modèles en réseau de Petri et pour leur exécution.

Nous avons ensuite défini les périphériques d'entrée que nous avons sélectionnés (la parole, l'écran tactile et la souris). A l'aide de la taxonomie de Buxton, nous avons pu remarquer qu'ils appartenaient à des types de périphériques bien différents et exprimaient un sous-ensemble intéressant pour cette étude.

Enfin, pour terminer, nous avons défini les termes de modalité, de multimodalité et les différents types de multimodalité selon Caelen.
Dans cette dernière partie, nous avons également vu deux exemples de multimodalité : le « Put That Here » de Bolt et les « Magic Lenses » de Bier.

Partie 2 Contribution : Modélisation et simulation d’interactions multimodales

1 Introduction

Dans ce deuxième chapitre, la contribution à proprement parler de ce mémoire, nous modélisons tout d'abord trois périphériques d'entrée : la souris, la reconnaissance vocale et l'écran tactile dans le but ensuite de modéliser des techniques d'interaction multimodale.

Dans cette contribution, nous avons défini un processus de modélisation de techniques d'interactions multimodales pour les systèmes interactifs, basé sur une notation et une technique formelle à base de réseaux de Petri.

L'originalité de cette contribution est de rendre visible tous les états et les changements d'états et de minimiser le code caché dans les transitions. L'objectif est de gérer les états dans les modèles et donc hors du code.

Dans cette partie, nous détaillons tout d'abord, pour chaque périphérique, l'architecture de nos modèles en Arch et le diagramme de composants exprimant les échanges entre les modèles. Ensuite, pour chaque modèle, nous présentons le processus qui a conduit à cette modélisation et nous expliquons le modèle en détail.

Le processus de création des architectures et principalement des modèles a été fortement itératif. Nous ne présentons ici que les versions finales de cette recherche. Néanmoins, nous trouvons utile de vous présenter dans certains cas, les réflexions qui ont mené aux modèles finaux.

Note Importante

Pour des raisons de places, l'accent dans cette contribution est mis sur la modélisation.
De plus, la présentation de l'outil nécessite une plate-forme d'exécution qu'il est difficile de décrire dans un mémoire.

Néanmoins, la partie présentation du mémoire mettra l'accent sur l'outil, la simulation, l'exécution et la modifiabilité des modèles.

2 La souris

2.1 Introduction

La souris est le périphérique d'entrée le plus utilisé. Il est donc normal de commencer par la modélisation de celui-ci.
Dans ce chapitre, nous verrons la modélisation complète de la gestion des événements souris par un modèle en couche. Nous commencerons par situer ces couches à l'aide du modèle Arch et exprimer leurs liens à l'aide d'un diagramme de composants. Ensuite, nous modéliserons les différentes couches en réseaux de Petri et nous expliquerons leur exécution.

2.2 Architecture

La gestion des événements souris est réalisée par un modèle en couche. Ces couches affinent les événements bas niveau tel que le déplacement de la souris avec un bouton enfoncé vers des événements haut niveau liés à des objets graphiques (déplacement d'un objet graphique). L'objectif de chaque couche est de fournir le niveau d'information suffisant pour la couche suivante. Selon les besoins des plus hautes couches, certaines couches inférieures peuvent donc être dans des ordres différents. Il nous revient alors de choisir l'ordre des couches en fonction de nos besoins. Ce type de choix dépend de ce que nous voulons privilégier dans notre gestion des événements. Nous verrons plus en détail un exemple de ce type de choix pour le calcul des coordonnées absolues.

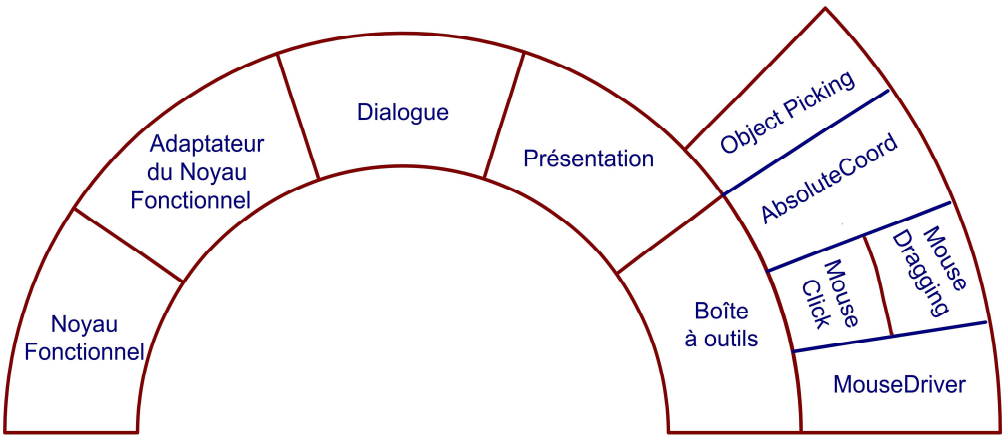


Figure 39 Modèle Arch de la souris

Nous avons représenté ci-dessus (Figure 39) l'architecture de notre système de gestion d'événements souris dans un modèle Arch. Cette architecture est divisée en quatre couches.

- Une première couche se charge de récupérer les événements bas niveau. Celle-ci est appelée MouseDriver. Cette couche est liée à un objet Java qui fournit des méthodes permettant de connaître l'état de la souris. La couche MouseDriver se charge alors de récupérer cet état et de le transmettre en événements aux couches supérieures. Ces événements sont de deux types :

    - Soit l'état d'un bouton de la souris a changé et cette couche envoie un événement boutonPressé ou boutonRelaché.
    - Soit un déplacement a eu lieu, et cette couche envoie un événement mouseMove contenant le déplacement relatif de la souris.

- La couche suivante est divisée en deux blocs qui fonctionnent en parallèle.
    - Nous avons tout d'abord, une gestion du click (MouseClick) qui se charge de comptabiliser le nombre de click en fonction des événements boutonPressé ou boutonRelaché. Un déplacement de la souris ou un temps d'attente trop long réinitialise le comptage.
    - Le deuxième bloc de cette couche s'occupe de gérer le type de déplacement (MouseDragging). Ce bloc reçoit les événements déplacement et les événements boutonPressé ou boutonRelaché. Il s'occupe ensuite d'analyser si le déplacement est réalisé avec un bouton enfoncé, plusieurs ou aucun. Ceci permet de séparer les déplacements simples et les déplacements glissés. Il envoie alors aux couches supérieures le type de déplacement accompagné des valeurs dx dy du déplacement.

- Au troisième niveau, se trouve le calcul des coordonnées absolues (AbsoluteCoord). Cette couche se charge de récupérer tous les événements de la couche précédente provenant des deux blocs MouseClick et MouseDragging. Lors de la réception des événements de déplacements, une variable est mise à jour contenant la position absolue. Ces événements sont ensuite renvoyés au niveau supérieur avec les coordonnées absolues. Lors de la réception des événements bouton, ceux-ci sont également envoyés à la couche supérieure avec comme paramètres supplémentaires, les coordonnées absolues stockées dans la variable.

- Enfin, la dernière couche dont nous allons nous occuper dans cette contribution se charge de la gestion des objets graphiques (Object Picking). Celle-ci reçoit les différents événements de la couche précédente et les compare à une liste d'objets graphiques pour voir si ces événements sont liés à un objet et donc si un événement a lieu sur un objet ou non.

Nous pouvons exprimer les relations et les messages échangés entre ces couches à l'aide d'un diagramme de composants (Figure 40).

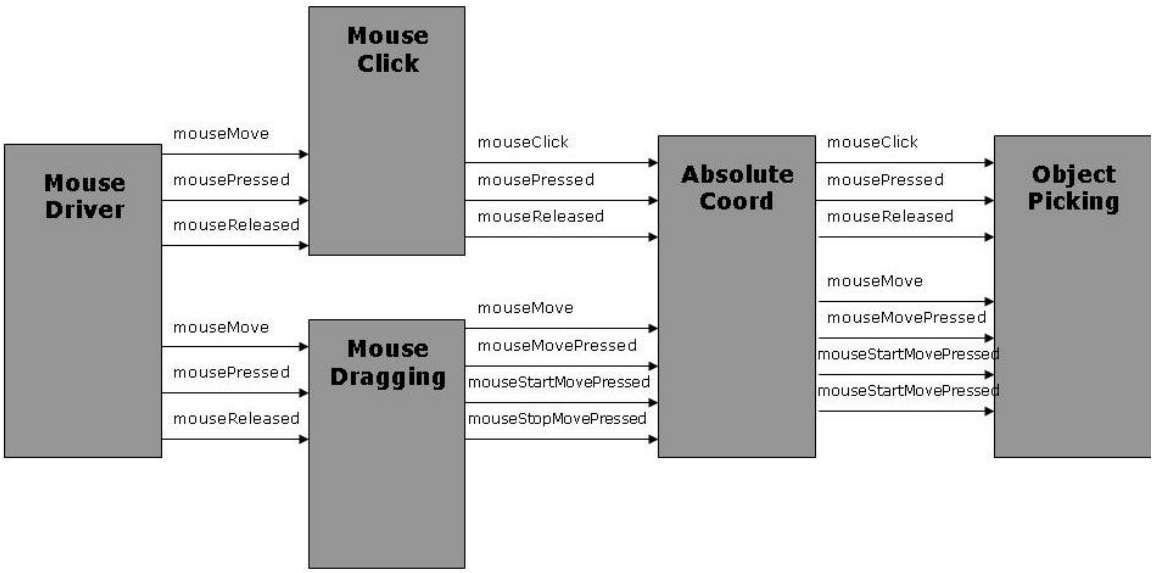


Figure 40 Diagramme de composants

Remarque :

Comme nous en avons parlé plus haut, certains choix sont effectués dans cette architecture sur l'ordre des couches et leurs fonctions. Il aurait pu être raisonnable de placer le calcul des coordonnées absolues plus tôt dans ce modèle et donc d'inverser les couches MouseClick-MouseDragging et AbsoluteCoord de ce modèle. Notre choix a été de placer cette couche à un niveau où il est possible de différencier les deux types de déplacement (déplacement simple et déplacement glissé). Nous pouvons alors calculer le déplacement de façon différente selon son type (en accélérant par exemple le déplacement glissé).

2.3 Modèles

2.3.1 Récupération des événements bas niveau (MouseDriver)

Processus de modélisation

Le processus de modélisation a commencé par l'analyse des événements que l'on pouvait recevoir à partir de l'objet Java *CPNMouse*. Cet objet permet d'utiliser plusieurs souris en parallèle dans Java et d'en récupérer les événements. Il a été codé par [Beaudouin-Lafon 2000].

Cet objet nous fournit les services suivants :

- *hasChanged*(numéro de la souris) qui renvoie un booléen vrai si l'état de la souris a changé, faux sinon.
- *getXDelta*(numéro de la souris) et *getYDelta*(numéro de la souris) qui renvoie le déplacement relatif en x (ou y) depuis le dernier appel de cette méthode.
- *buttonStat*e(numéro de la souris, numéro du bouton) qui renvoie un booléen donnant l'état du bouton (vrai pour enfoncé, faux pour relâché)

Nous avons donc commencé ce modèle en créant une boucle temporisée chargée de vérifier l'état de l'objet. Notre première solution était ensuite de transmettre cet état à une couche supérieure qui se chargerait de gérer les différents types de changement et les répercuter. Nous avons finalement choisi de gérer les différents états de cet objet dans le même modèle ce qui permet d'éviter l'envoi d'un trop grand nombre d'événements.

Vue générale sur le modèle

Ce modèle (Figure 41) se divise essentiellement en trois parties :

- La partie de gauche se charge de vérifier l'état de l'objet *CPNMouse* et de voir si son état a changé ou non.
- La partie supérieure droite se charge d'envoyer les événements de changement d'état des boutons (pressé relâché) pour chaque bouton (ici 3) selon l'état antérieur (si l'état était pressé, le changement d'état implique un événement relâché)
- La partie inférieure droite vérifie si un déplacement a été effectué et envoie alors un événement déplacement.

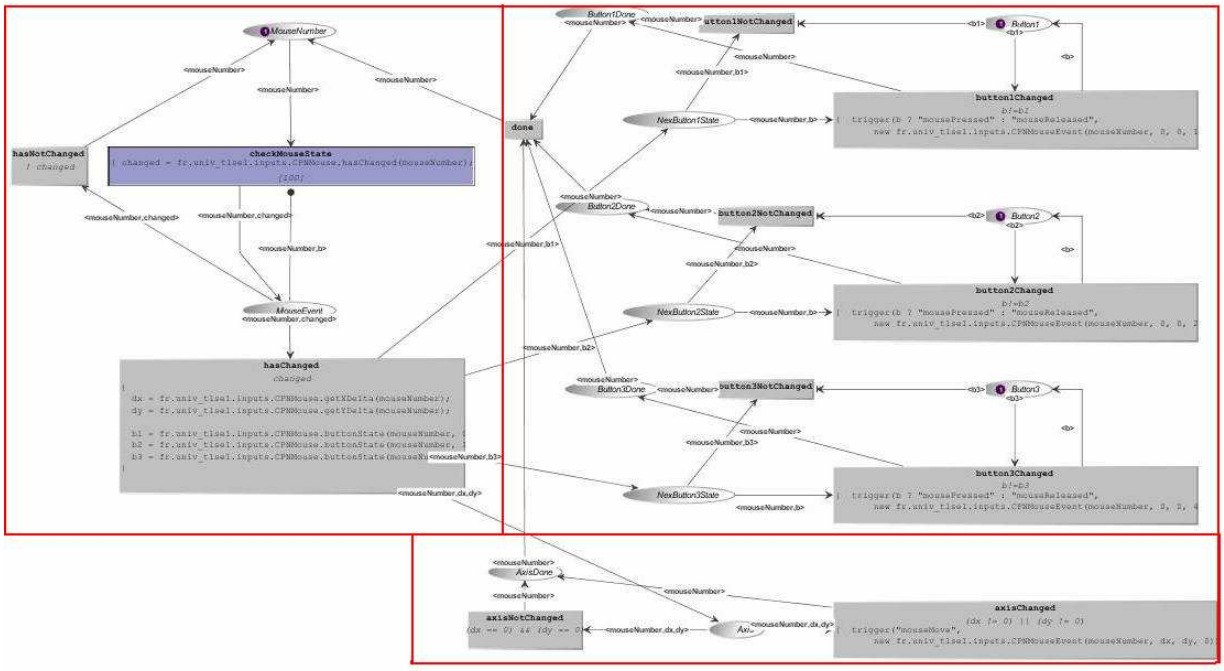


Figure 41 Récupération des événements bas niveau

Description détaillée du modèle

A l'initialisation de ce réseau, nous lui donnons en paramètre le numéro de la souris.
Un seul jeton circule dans ce réseau. Il est initialement à l'emplacement *MouseNumber* avec la valeur du paramètre. Néanmoins, trois autres places sont initialisées avec un jeton *button1, button2 et button3* ayant comme valeur le booléen *false*.

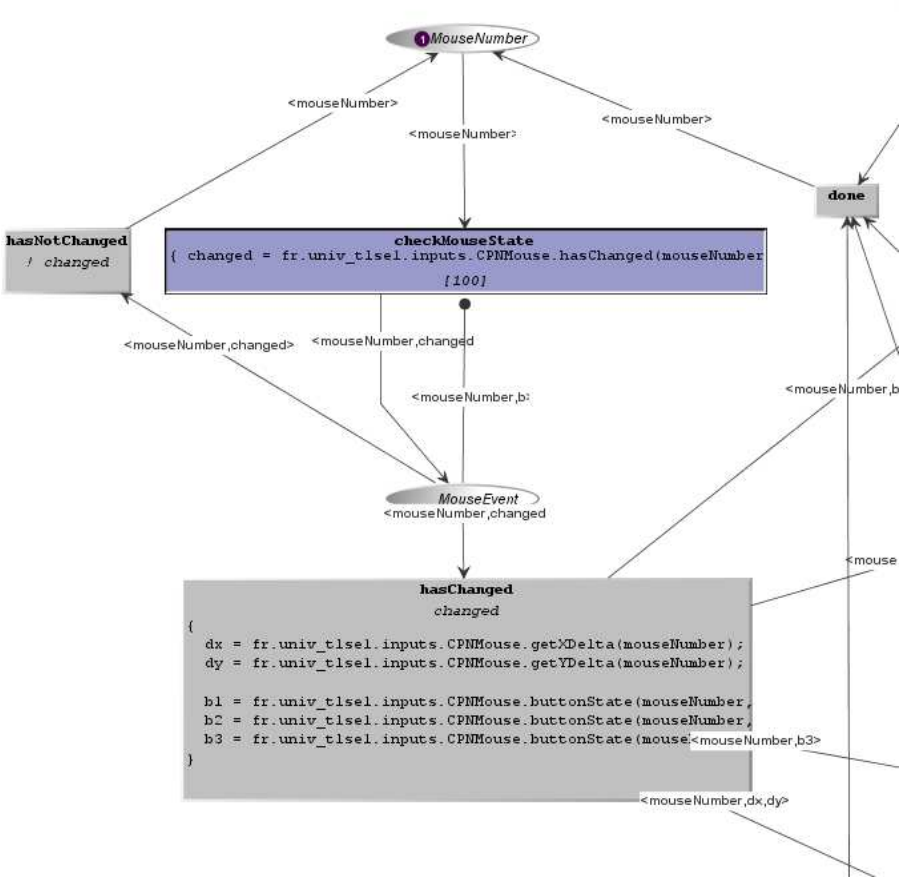


Figure 42 Récupération des événements bas niveau (détail 1)

Un Timer exécute la transition *checkMouseState* toutes les 100 millisecondes.
Dans cette transition, une action qui renvoie un booléen est exécutée :

```
changed = fr.univ_tlse1.inputs.CPNMouse.hasChanged(mouseNumber);
```

Selon le résultat envoyé dans *changed*, la transition *hasChanged* ou *hasnotChanged* est tirée.

- Si *hasnotChanged* est tirée, on retourne à l'état initial.
- Sinon, la transition *hasChanged* est tirée et le code à l'intérieur de cette transition est exécuté. Celui-ci récupère l'état des différents boutons ainsi que les déplacements horizontaux et verticaux (*dx, dy*).

```
dx = fr.univ_tlse1.inputs.CPNMouse.getXDelta(mouseNumber);
```

Cette transition envoie un jeton sur les places *axis* contenant les valeurs *dx, dy*, *nextbuton1state* contenant l'état du bouton 1, *nextbuton2state* contenant l'état du bouton 2, *nextbuton3state* contenant l'état du bouton 3 (voir plus si l'on désire modéliser une souris ayant plus de 3 boutons)

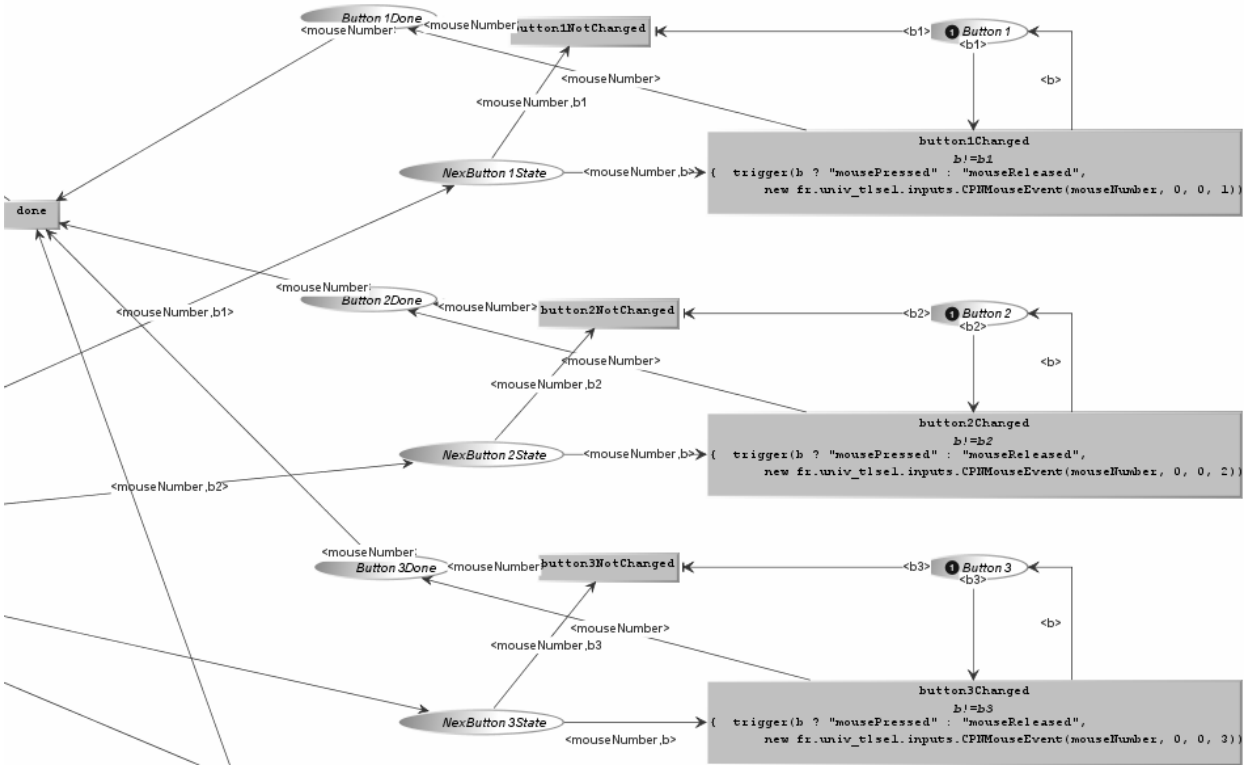


Figure 43 Récupération des événements bas niveau (détail 2)

Lorsqu'un jeton se trouve en *nextbutonstate1* (respectivement 2 et 3) *button1changed* teste si *b1* est différent de *b* et donc si l'état du bouton 1 a changé.

- Si c'est le cas un *mousePressed* ou un *mouseReleased* est envoyé (selon le nouvel état de celui-ci).
- Sinon, la transition *button1notchanged* est tirée.

Dans les deux cas, *button1changed* et *button1notchanged* dépose un jeton dans la place *button1done*. Il en va de même pour les autres boutons.

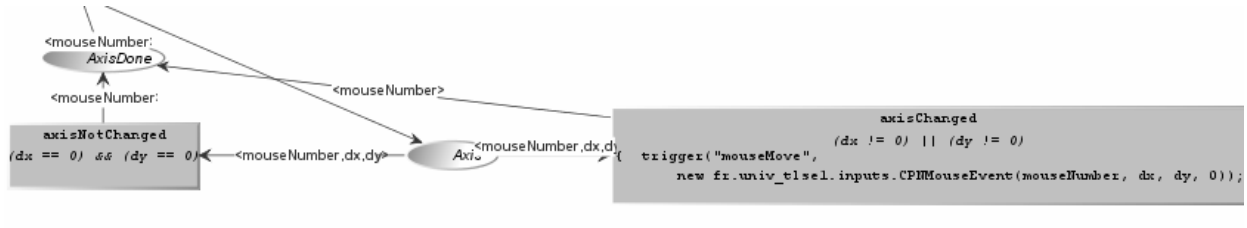


Figure 44 Récupération des événements bas niveau (détail 3)

Pour la place *Axis*, si *dx* et *dy* sont nuls, la transition *axisnotchanged* est franchie et va placer un jeton dans la place *axisdone*.

Dans les autres cas, la transition *axischanged* est franchie. Le code de celle-ci envoie *mouseMove* et place un jeton dans la place *axisdone*.

Lorsque les places *button1done, button2done, button3done* et *axisdone* contiennent chacune un jeton, la transition done est franchie et on retourne a l'état initial. A partir de là, la transition temporisée *checkmouseState* est réexécutée.

La désactivation de *checkmouseState* pendant l'analyse des états des différents boutons et de l'axe permet d'éviter une surcharge du réseau. De plus, lors de l'appel des différentes méthodes dans *haschanged*, l'objet *CPNMouse* fusionne les différents événements (comme l'addition des déplacements horizontaux).

2.3.2 Gestion du click bouton (Mouse Click)

Processus de modélisation

Pour la deuxième couche de notre architecture, nous avons décidé de la séparer en deux blocs : le modèle de gestion de click et celui de la gestion du type de déplacement. Ces deux blocs étant indépendants, il nous a paru plus intéressant de les dissocier pour permettre une meilleure clarté de ceux-ci.

Nous avons ensuite créé ce modèle pour un bouton.
L'utilisation des réseaux à objets nous a permis de simplifier le modèle en lui donnant plusieurs objets bouton sur un même réseau de Petri.

Nous pouvons voir, ici, l'intérêt évident d'utiliser un modèle à objets. En effet, sans celui-ci, le modèle se serait multiplié pour chaque nouveau bouton. Or, ce modèle-ci est valable quel que soit le nombre de boutons.

Vue générale sur le modèle

Le réseau *MouseClick* est abonné aux événements *mousePressed, mouseReleased* et *mouseMove*. Celui-ci s'occupe de compter le nombre de click de chaque bouton de la souris indépendamment.
Le principe du comptage du nombre de click est assez simple. Un clic*k* est comptabilisé à la suite d'un autre, s'il n'y a pas un trop long délai entre un *released* et un *pressed* ou entre un *pressed* et un *released*. Lorsque le délai est dépassé, le nombre de click est remis à zéro. Il en va de même en cas de déplacement de la souris entre un *pressed* et un *released* ou un *released* et un *pressed*. Ce déplacement remet instantanément le compteur de click à zéro. Néanmoins, cette technique serait trop sensible à des tremblements de l'utilisateur. Grâce à ce réseau, nous pouvons choisir les seuils pour *dx* et *dy* en dessous desquels le nombre de clicks ne sera pas réinitialisé. L'envoi du nombre de clicks se fait lors du *released* du bouton. Ce nombre est obtenu en stockant le nombre de clicks courants dans une variable (*nbClick*).

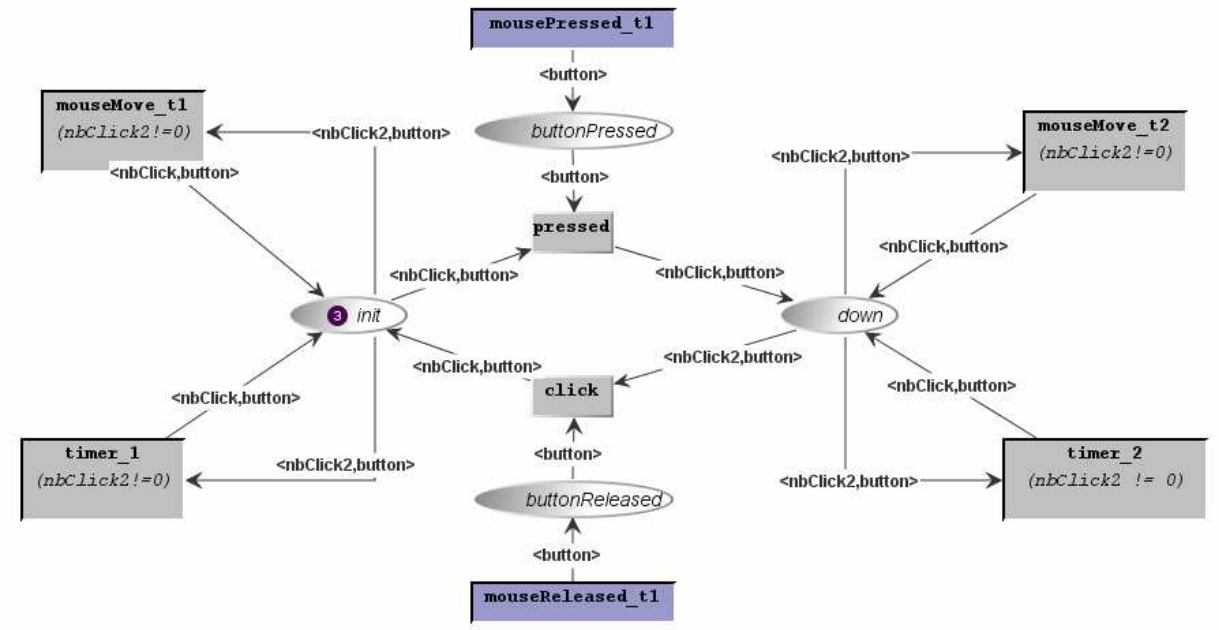


Figure 45 Gestion du click bouton

Description détaillée du modèle

A l'initialisation, un jeton par bouton se trouve dans la place Init avec comme valeurs :

- l'entier 0 pour *nbClick*. Cette variable comptabilise le nombre de click.
- le numéro du bouton dans la variable *button*.

Lors de la réception d'un événement *mousePressed* (respectivement *mouseReleased*), un jeton est déposé dans la place *buttonPressed* (respectivement *buttonReleased*). La transition *pressed* (respectivement *click*) peut être franchie si un jeton se trouvant dans la place *init* (respectivement *down*) a la même valeur *button* que celui se trouvant dans la place *buttonPressed* (respectivement *butonReleased)*. Cette valeur est obtenue lors de la réception de l'événement *mousePressed* (respectivement *mouseReleased*) dans le code de la transition.

```
{button = ((fr.univ_tlse1.inputs.CPNMouseEvent)_event).getButton();}
```

Lorsque des jetons se trouvent dans les places *init* et *down*, d'autres transitions peuvent être tirées :

- Si un événement *mouseMove* est reçu, la transition *mouseMove* est franchie.
- S'il y a un délai de 500 millisecondes, la transition timer est franchie.

Ces deux transitions réinitialisent *nbClick* à zéro après avoir envoyé sa valeur à la couche supérieure.

2.3.3 Gestion du type de déplacement (Mouse Dragging)

Processus de modélisation

Nous n'avons gardé de ce modèle que la partie s'occupant de la gestion du type de déplacement. La difficulté essentielle de création de ce modèle est de trouver une technique permettant de transformer des objets bouton en un seul objet pour exprimer les moments où plusieurs boutons sont enfoncés. Notre choix a été de placer l'état de ces boutons dans une variable entière ayant la valeur binaire de l'état des boutons (O pour relâche, 1 pour enfoncé), chaque bouton ayant un cœfficient 2 expo n permettant de les différencier (1 pour le premier bouton, 2 pour le deuxième, 4 pour le troisième et ainsi de suite)
Par exemple, lorsque le bouton 1 et 3 de la souris sont enfoncés, la valeur de cette variable est de 5 (4*1 + 2*0 +1*1 ou en binaire 101)

Vue générale sur le modèle

Ce réseau s'occupe de gérer les différents types de déplacement. En cas de déplacement simple, sans bouton pressé, ceux-ci sont envoyés directement à la couche supérieure. Par contre si un bouton est enfoncé lors d'un déplacement, ce réseau va tout d'abord envoyer un événement signifiant le début d'un déplacement glissé. A cet instant, le réseau se retrouve dans l'état de déplacement glissé (Dragging), nous pouvons alors avoir deux comportements :

- soit tous les boutons sont relâchés et on retrouve l'état initial (tout en envoyant un événement de fin de déplacement glissé)
- soit on continue le déplacement avec un bouton enfoncé et des événements déplacement glissé sont envoyés à la couche supérieure.

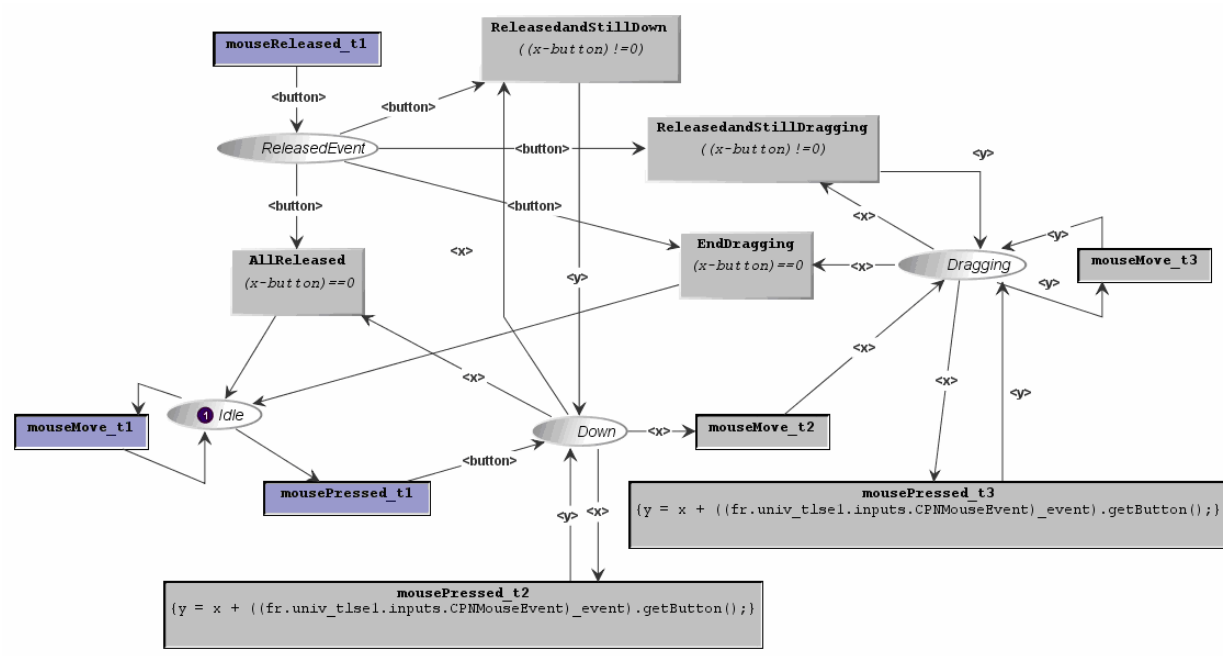


Figure 46 Gestion du type de déplacement

Description détaillée du modèle

A l'état initial, un jeton se trouve dans la place Idle.

Deux actions sont possibles selon les événements reçus (le *mouseReleased* est également actif mais il est assez évident qu'une souris ne pourra pas émettre de *mouseReleased* avant d'avoir émis un *mousePressed*).

- Si un *mouseMove* est reçu, nous aurons le cas d'un déplacement dit simple.
- Si un *mousePressed* est reçu, ceci entraînera le passage du jeton jusqu'à la place *down* avec comme valeur, l'entier correspondant au numéro de bouton.

Lorsque le jeton se trouve sur la place *down*, plusieurs transitions peuvent être tirées :

Soit le réseau reçoit un autre événement *mousePressed* et la valeur du bouton de ce *mousePressed* viendra s'additionner dans le jeton *down*.

```
{y = x + ((fr.univ_tlse1.inputs.CPNMouseEvent)_event).getButton();}
```

Soit un *mouseReleased* est reçu et envoie un jeton sur la place *ReleasedEvent*.

- Si le *mouseReleased* correspond au dernier bouton qui était enfoncé le jeton est supprimé de down et se retrouve à l'état initial *Idle* avec une souris sans aucun bouton enfoncé.
- Si, lors de la réception de l'événement *mouseReleased,* plusieurs boutons sont enfoncés, la transition *ReleaseandStillDown* est franchie et celle-ci déduit la valeur du bouton de l'entier dans la place *down*.

Enfin, avec un jeton dans la place *down*, le réseau peut recevoir un événement *mouseMove* qui va envoyer un événement déplacement glissé (*movePressed*). Lorsque la transition *mouseMove_t2* est franchie, celle-ci envoie l'événement *Startmousemovepressed* et envoie le jeton dans la place *Dragging*. De cette place, on peut recevoir :

- Des événements *mouseMove* qui seront renvoyés aux couches supérieures en *mouseMovePressed.*
- Des événements *mousePressed* qui ajouteront la valeur du bouton dans la place *Dragging* ou des événements *mouseReleased* qui les déduiront de cette place.

Enfin, lorsqu'un événement *mouseReleased* est reçu :

- S'il correspond au dernier bouton enfoncé, la transition *EndDragging* sera franchie et celle-ci renverra le jeton dans la place initiale *Idle*. De plus, il enverra l'événement *mouseStopMovePressed* au réseau supérieur.
- Si cet événement *mousereleased* correspond à un des boutons enfoncés mais qu'il existe encore au moins un autre bouton enfoncé, la transition *releaseandStillDragging* est tirée et déduira la valeur du bouton relâché de la variable se trouvant dans le jeton de la place *Dragging*.

2.3.4 Calcul des coordonnées absolues (AbsoluteCoord)

Processus de modélisation

L'essentiel du processus de modélisation a été de choisir à quel niveau se situerait cette couche dans le modèle Arch. Le positionnement de cette couche à un niveau plus bas aurait permis une simplification de celui-ci.
Notre choix a été de placer cette couche à un niveau permettant de dissocier les différents types de mouvement. Grâce à cela, nous pouvons modéliser des comportements différents selon l'action de l'utilisateur. Nous pouvons par exemple imaginer une accélération des déplacements du curseur lors d'un drag, accélération qui serait absente lors d'un déplacement simple.
Comme notre choix s'est porté sur la différentiation des calculs des deux types de déplacements, il apparaît une sorte de redondance. En effet, à première vue ce réseau parait symétrique. La principale différence entre ces deux modèles se situe au niveau du paramètre des coefficients de déplacement entre le mouvement simple et le mouvement glissé. Si ces cœfficients sont égaux, une version repliée est équivalente. (Voir le calcul des coordonnées absolues de la reconnaissance vocale page 76)

Vue générale sur le modèle

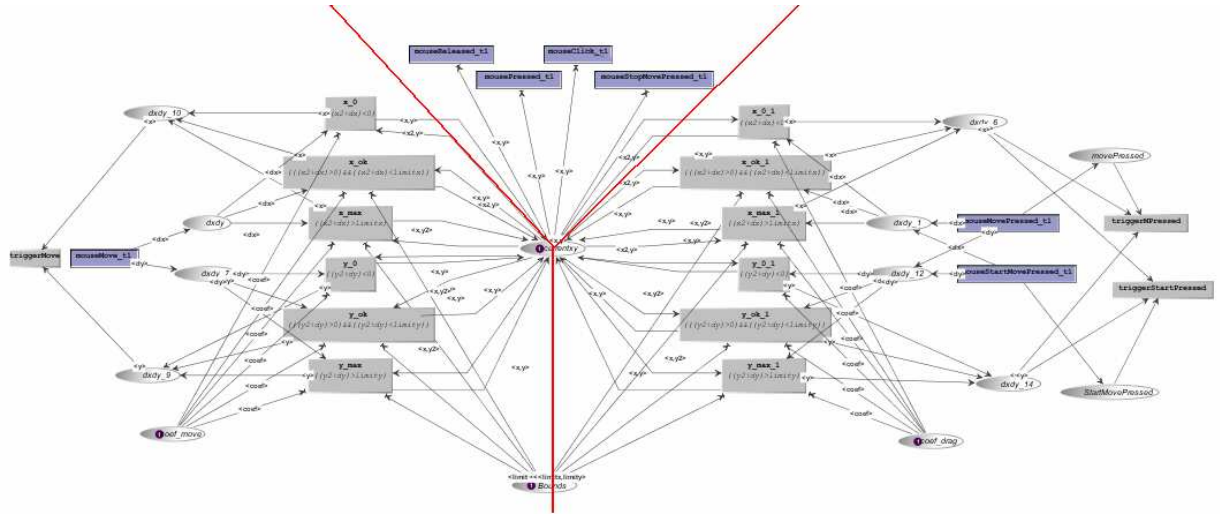


Figure 47 Calcul des coordonnées absolues

Cette couche s'occupe de calculer le x et le y absolus à partir des déplacements dx, dy de la souris. Ces déplacements peuvent provenir soit d'un déplacement simple ou glissé.
Ce réseau est divisé en 3 parties :

- La partie de gauche gère les déplacements simples, met à jour la position absolue et envoie au niveau supérieur l'événement mouvement accompagné de la position absolue.
- La partie de droite gère les déplacements glissés de la même manière.
- La partie du milieu haut reçoit des événements bouton et les renvoie au niveau supérieur avec la position absolue contenue dans le jeton de la place *currentxy*.

Description détaillée du modèle

A l'initialisation de ce réseau, la place *currentxy* contient un jeton ayant comme variables deux entiers *x* et *y* initialisés à 0 et la place *limitxy* reçoit en paramètres la taille de l'écran et donc la limite x et y *(limitx* et *limity*).

De plus, pour obtenir des comportements différents selon le type de déplacement, deux places *coef_move* et *coef_drag* reçoivent en paramètre le cœfficient de déplacement.
Ce réseau reçoit tous les événements émis par les couches *mouseDragging* et *mouseClick*.

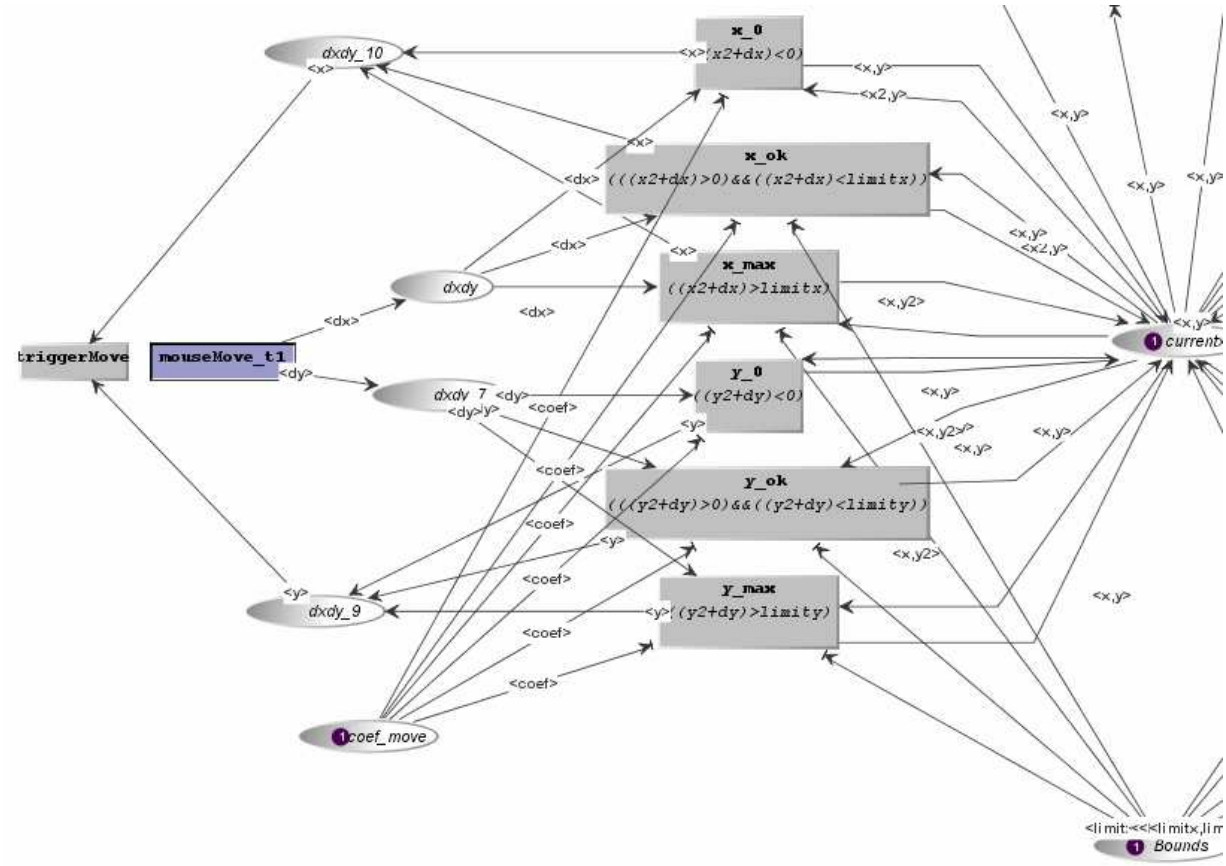


Figure 48 Calcul des coordonnées absolues (détail 1)

Lorsque ce réseau reçoit un *mouseMove*, le code à l'intérieur de cette transition récupère le *dx* et le *dy* qui sont envoyés respectivement dans les places *dxmove* et *dymove*.
Ces jetons sont ensuite gérés de la même façon par un triplet de transitions (*x_0, x_Ok, X_Max*) (respectivement *y_0, y_ok, y_max*)

- Si le calcul de `currentx + dx` est inférieur à 0, la transition *x_0* est franchie et la valeur O est placée en *currentx* et en *xMove*
- Si le calcul de `currentx + dx` est supérieur à *limitx,* la transition *x_max* est franchie et la valeur *limit*x est placée en *currentx* et en *xMove*.

- Dans les autres cas, la transition *x_ok* est franchie et la valeur *currentx* + *dx* est déposée dans *currentx* et dans *moveX*.

Il en va de même pour les *dy*.
A ces déplacements, est ajouté ou multiplié le cœfficient *coef_move* pour les déplacements simples.

Lorsque les places *xmove* et *ymove* contiennent chacune un jeton (soit après le passage par le triplet (*x_0, x_Ok, X_Max*), la transition *triggerMove* est tirée et envoie un événement *mouseMove* avec comme paramètre le x, y coordonnées absolues.

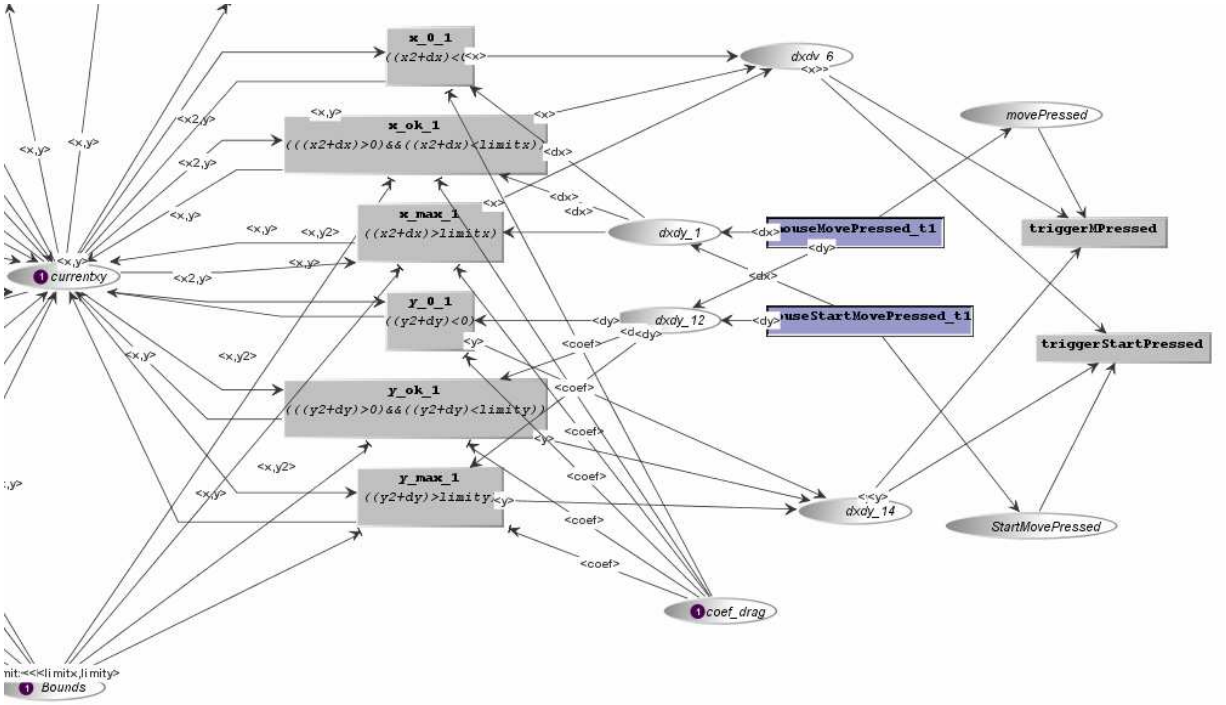


Figure 49 Calcul des coordonnées absolues (détail 2)

Le même comportement se trouve de l'autre coté de ce réseau lorsque celui-ci reçoit les événements *mouseStartMovePressed* et *mouseMovePressed*. Ceux-ci placent les valeurs *dx* et *dy* en *dxdrag* et *dydrag*, valeurs qui sont ensuite analysées par le triplet *0, Ok, Max*.
De plus, un jeton est placé dans la place *movePressed* ou *StartMovePressed* selon que la transition *mouseMovePressed* ou *mouseStartMovePressed* a été franchie. Ceci permet lors de l'arrivée des jetons en *dragx* et *dragY* de savoir si la transition *triggerStartMovePressed* ou *triggerMovePressed* est tirée.

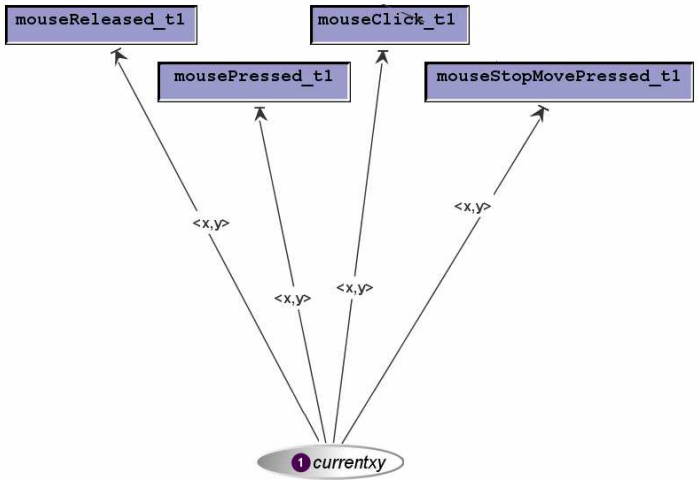


Figure 50 Calcul des coordonnées absolues (détail 3)

Enfin, si ce réseau reçoit les événements *mouseReleased, mousePressed, mouseClick* et *mouseStopMovePressed*, il les renvoie au niveau supérieur en y ajoutant les coordonnées absolues x et y provenant de la place *currentxy*.

2.3.5 Gestion des objets graphiques (ObjectPicking)

Processus de modélisation

L'essentiel de la sémantique de ce modèle est de lier des événements de déplacement à une liste d'objets. Pour chaque objet pris indépendamment, nous pouvons avoir quatre comportements :

- le pointeur entre dans un objet.
- le pointeur reste sur l'objet.
- le pointeur sort de l'objet.
- le pointeur n'est pas en contact avec un objet du début à la fin du mouvement.

Ensuite, lorsque le pointeur est sur un objet, les autres événements sont répercutés sur cet objet (un click, un début de déplacement glissé). Si ces événements arrivent sans que le pointeur soit sur un objet, ces événements sont ignorés.

Vue générale sur le modèle

Ce modèle reçoit les événements provenant de la couche AbsoluteCoord.
Il est essentiellement constitué de deux parties :

- la partie de gauche vérifie lors d'un *mouseMove* si le pointeur entre ou non sur un objet et gère également les actions hors d'un objet comme par exemple la sélection.
- la partie de droite, avec comme place centrale *MouseOnObject*, gère les différentes actions de l'utilisateur lorsque le pointeur est sur un objet comme le click ou le drag.

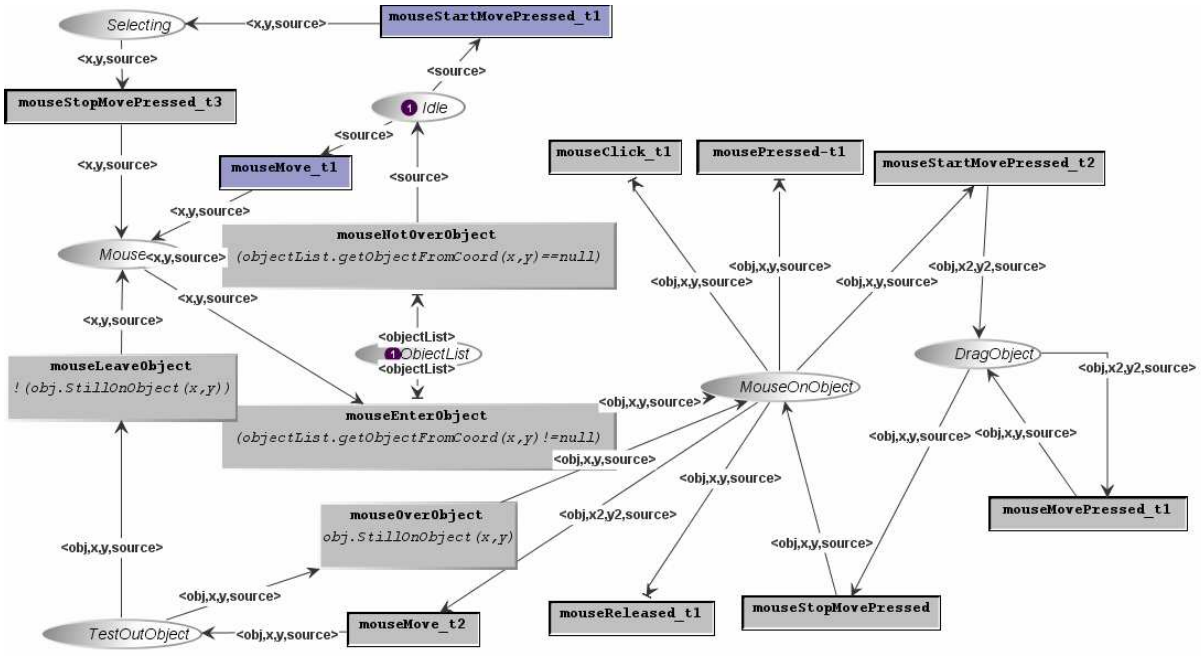


Figure 51 Gestion des objets graphiques

Description détaillée du modèle

A l'initialisation, nous plaçons une liste d'objets graphiques dans la place *ObjectList* et un jeton contenant la référence de la souris et ses coordonnées absolues dans la place *Mouse*.

Lorsqu'un jeton se trouve dans la place *Mouse*, deux transitions sont possibles. Ces transitions sont exclusives et contiennent dans leur condition, la fonction *getObjetctFromCoord* sur la liste d'objet avec comme paramètres les coordonées du pointeur.

```
    public Object getObjectFromCoord(int coox, int cooy)
    {
        GraphicObject obj;
        int size = getSize();
        for (int i=0;(i<size);i++)
        {
            obj = (GraphicObject) this.get(i);
            int x = obj.getLocationOnScreen().x;
            int y = obj.getLocationOnScreen().y;
            int h = obj.getHeight();
            int w = obj.getWidth();

            if ((coox >= x) && (cooy >= y) &&
                        (coox <= (x+ w)) && (cooy <= (y+ h)))
            {
                return obj;
            }
        }
        return null;
    }
```

Tableau 11 Code de la fonction getObjectFromCoord

- Si la fonction renvoie un objet, les coordonnées correspondent à un objet (le pointeur se trouve sur un objet). La transition *mouseEnterObject* est tirée et un jeton est placé dans la place *MouseOnObject* avec comme valeur, le couple (*Souris,GraphicObject)*. De plus, la fonction *setOnIt* est appelée sur l'objet graphique. Cette fonction peut avoir des comportements différents selon le type d'objet.

```
public void setOnIt(boolean onIt)
{
            if (onIt == true){
                jButton.setForeground(Color.RED);
            }
            else{
                jButton.setForeground(Color.BLACK);
            }

}
```

Tableau 12 Code de la fonction setOnIt

- Si la fonction renvoie *null*, les coordonnées ne correspondent pas à un objet. Le jeton souris est déposé dans la place *Idle*.

Lorsqu'un jeton se trouve dans la place *MouseOnObject*, cinq transitions sont franchissables.

A la réception d'un événement *mouseClick, mousePressed* ou *mouseReleased*, une méthode est appelée sur l'objet correspondant à l'événement (un click sur un bouton par exemple).
La transition *MouseMove_t2* est également franchissable à partir de cette place. Lors du franchissement de cette transition, un jeton est déposé dans la place *TestOutObject* qui exécute la méthode *stillOnObject* sur le *GraphicObject* avec les nouvelles coordonnées absolues. Cette méthode renvoie un booléen :

- vrai si le pointeur est encore sur l'objet. La transition *mouseOverObject* est alors franchie et le jeton est redéposé dans la place *MouseOnObject*.
- faux, le jeton est alors déposé dans la place *mouse*. De cette place initiale, on teste si le pointeur est allé se placer sur un nouvel objet ou non.

A partir de la position *MouseOnObject*, une dernière transition est franchissable *mouseStartMovePressed.* Celle-ci dépose le jeton dans la place *DragObject* et lance le déplacement glissé de l'objet en exécutant la méthode *startDragObject* sur le *GraphicObject*. Ceci a comme résultat le déplacement de l'objet et le calcul du différentiel entre la position du coin gauche de l'objet et la position du pointeur, soit la position du pointeur sur l'objet.

```
public void startDragObject(int pointX, int pointY)
{
    dragX = pointX-getLocation().x;
    dragY = pointY-getLocation().y;
}
```

Tableau 13 Code de la fonction startDragObject

A partir de cette place, deux transitions sont franchissables :

- *mouseMovePressed* exécute la méthode *dragObject* sur l'objet graphique.

```
public void dragObject(int pointX, int pointY)
      {
      int newX = 0;
      int newY = 0;

      if ((pointX - dragX) != 0)
       {
           newX = (pointX - dragX);
       }
      if ((pointY - dragY) != 0)
       {
            newY = (pointY - dragY);
       }

      int h = this.getHeight();
      int w = this.getWidth();
      this.setBounds(newX,newY,w,h);
}
```

Tableau 14 Code de la fonction dragObject

- *mouseStopMovePressed* replace le jeton dans la position *MouseOnObject* et exécute la méthode *stopDrag* qui scelle l'objet graphique aux coordonnées du relâchement du bouton (de la fin du drag) moyennant le différentiel entre la position de l'objet et la position du pointeur sur celui-ci.

Enfin, une dernière action est réalisable : lorsqu'un jeton se trouve dans la place *Idle*, lors de la réception d'un événement *mouseStartMovePressed*, la transition est franchie et un jeton est déposé dans la place *selecting*. Cette position représente la sélection d'objet.

2.4 Rendu graphique

Le rendu graphique se fait à deux niveaux :

- l'affichage du pointeur se fait grâce aux fonctions de rendu sur l'état d'une place du réseau de calcul des coordonnées absolues. Lorsqu'un jeton entre dans la place centrale de ce modèle, la fonction de rendu est appelée (Tableau 15).
- le rendu des objets graphiques se fait dans le modèle *ObjectPicking* (Figure 51) par appel de méthodes sur ces objets.

```
    public void ICORendering_renderMouse(MarkingEvent event) {

draw((Integer) event.getToken().getSlot(0),
                 (Integer) event.getToken().getSlot(1),
                  event.toString().toString());
}
```

Tableau 15 Code de la fonction ICORendering_renderMouse

La fonction *ICORendering* est appelée lorsqu'il y a un changement de marquage dans le réseau *AbsoluteCoord* sur la place *Currentxy*. Cette méthode appelle alors la fonction *draw*, avec comme paramètre la nouvelle position de la souris, ainsi que la référence de cette souris.

```
public void draw(int x, int y, String ref) {

  if (mouseTable.containsKey(ref))
  {
      MouseObj mouse = (MouseObj)mouseTable.get(ref);
      mouse.x = x;
      mouse.y = y;
  }
  else
  {
      MouseObj mouse = new MouseObj(ref,getColor());
      mouse.x = x;
      mouse.y = y;
      mouseTable.put(ref, mouse);
  }
  this.repaint();

}
```

Tableau 16 Code de la fonction draw

La fonction *draw* (Tableau 16) se charge alors de vérifier si la souris existe déjà et elle met à jour sa position. Si la souris n'existe pas, elle est créée et placée dans la Hashtable *mouseTable*.

2.5 Conclusion

Nous avons créé dans ce chapitre, un modèle d'interaction de la souris.
Nous avons commencé par définir l'architecture des différentes couches au moyen d'un modèle Arch et les événements échangés entre cette couche par un diagramme de composants. Nous avons expliqué chaque couche en commençant par le processus de modélisation pour ensuite détailler l'exécution de ce modèle.

Enfin, nous avons terminé en expliquant comment était réalisé le rendu des pointeurs et les interactions avec les objets graphiques.

3 La reconnaissance Vocale

3.1 Introduction

Après la souris, nous avons ensuite modélisé la reconnaissance vocale.
Nous avions essentiellement deux possibilités. Nous pouvions tout d'abord utiliser la parole comme un système de pointage simple, comme nous allons le faire dans cette contribution. Ce système de pointage exécute les mêmes types d'actions qu'une souris : déplacements et clicks.
L'autre possibilité que nous avons implémentée dans l'application Image, est d'utiliser la parole pour accomplir des actions spécifiques de plus haut niveau comme pour le « Put That Here » de Bolt.

3.2 Architecture

Nous avons tout d'abord créé une classe Java qui se connecte à un API de reconnaissance vocale et qui transmet les événements qu'elle reçoit vers nos modèles dans PetShop. Les événements transmis sont :

- les événements Ok : lorsque l'API reconnaît un mot dans notre grammaire
- les événement Ko lorsque l'API a enregistré un son au micro mais n'a pas trouvé l'équivalent dans la grammaire. Ce dernier événement n'est pas nécessaire mais il nous permettra de calculer le taux d'échec et éventuellement d'effectuer certaines actions sur cet événement selon les choix du concepteur.

La gestion de ces événements se fait dans PetShop par un modèle en couche affinant les événements bas niveau vers des commandes pour piloter une souris. Nous avons donc réalisé un driver transformant des événements sons en provenance du micro vers des événements permettant de déplacer une souris ou d'effectuer des actions (click).

Nous avons choisi de mettre l'intelligence du driver dans les réseaux. De ce fait, la grammaire de reconnaissance vocale se fait dans une des couches du driver. L'API envoie simplement un couple (le mot reconnu et la règle correspondant à ce mot). L'avantage d'utiliser ces différentes couches est que nous pouvons assez facilement modifier le comportement du driver en changeant une ou plusieurs couches. De plus, ce type de modèle permet une vérification du comportement du driver. De ce fait, la grammaire utilisée par l'API ne comprend que des directions et des actions simples. L'analyse des suites de mots possibles se fait dans la grammaire créée par un réseau PetShop.

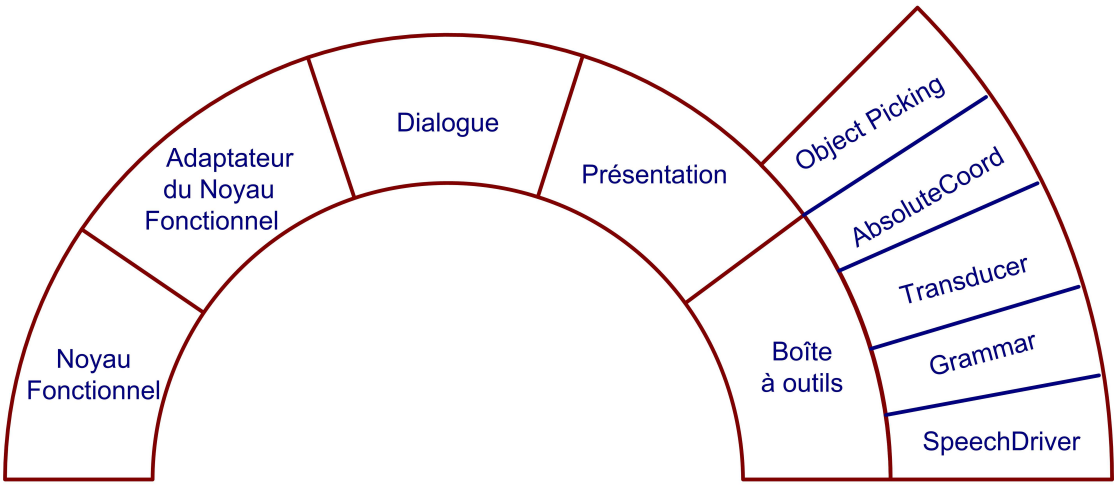


Figure 52 Modèle Arch de la reconnaissance vocale

L'architecture de ce modèle est composée de cinq couches.

- La première couche (Speech Driver) se charge de récupérer les événements provenant de notre classe Java. Celle-ci comptabilise les événements ko et ok et envoie ces derniers à la couche supérieure.

- La deuxième couche (Grammar) se charge de la grammaire. Selon le choix du concepteur, plusieurs grammaires peuvent être utilisées à ce niveau. Elles doivent néanmoins correspondre au fichier donné en paramètre à notre classe Java. Dans cet exemple, notre choix s'est porté sur une grammaire qui gère le déplacement et les actions d'une souris. Dans d'autres applications comme Image, cette couche sera remplacée par une grammaire faite de différents types d'actions comme Modifier, Effacer, etc.

- La troisième couche (Transducer) se charge, à partir de la règle reconnue par la grammaire de la couche inférieure, de transformer le déplacement gauche droite haut et bas en un dx dy.

,

- La quatrième couche (Absolute Coord) a la même fonction que notre calcul de coordonnées absolues pour la souris. Celle-ci se charge de transformer les déplacements relatifs en déplacements absolus et d'ajouter la position absolue aux actions comme le click.

- Enfin, la cinquième couche (Object Picking) est chargée de gérer les interactions avec les objets graphiques, tout comme la couche ObjectPicking du modèle de la souris.

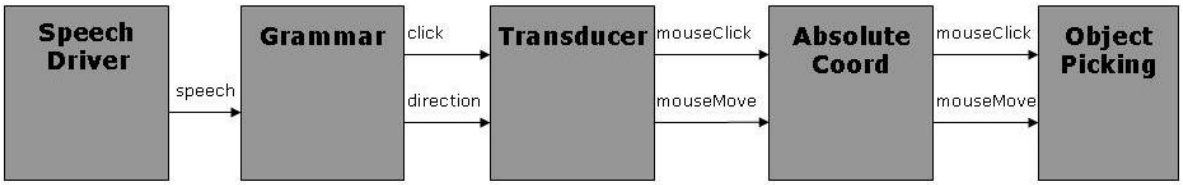


Figure 53 Diagramme de composants : Reconnaissance vocale

3.3 Modèles

3.3.1 Code Java du driver

Processus de modélisation

Quelle que soit la méthode choisie, nous devons tout d'abord lier une API de reconnaissance vocale à PetShop pour pouvoir gérer les événements reçus via PetShop. Parmi les différentes API testées, nous avons choisi celui de Cloudgarden pour cette contribution

La première étape a été de créer un programme de test pour récupérer une série de mots suivant les grammaires exemples de l'Api. Ensuite, nous avons créé une grammaire personnalisée, et, adapté le programme pour lui permettre d'envoyer des événements lors de la réception de mots reconnus par la grammaire.
Ce programme de test a ensuite été utilisé pour réaliser le driver transformant des événements sons en provenance du micro vers des événements utilisables par nos modèles.

Description détaillée du code

Nous avons tout d'abord créé une grammaire contenant les actions possibles d'un pointeur graphique auxquelles nous avons ajouté des couleurs permettant de distinguer les différents pointeurs.

```
#JSGF V1.0;

grammar grammars.mouse;

public <direction> = left | right | up | down ;
public <button> = click ;
public <color> =  red | blue ;
```

Tableau 17 Exemple de grammaire utilisée par l'API de CloudGarden

Il était tout à fait possible de créer la grammaire intégralement dans ce type de fichier. Mais notre but était de pouvoir connaître à tout instant l'état du système et donc entre autres l'endroit où se trouve l'analyseur sémantique.

De ce fait, le fichier grammaire est plutôt utilisé comme dictionnaire recensant tous les mots que l'API doit reconnaître. Ici, un choix doit être fait entre les performances de l'API et le nombre de mots du dictionnaire. Plus il y a de mots, plus il y a de risques d'erreurs de la part de l'API. Ceci est principalement vrai si deux mots du dictionnaire sont très ressemblants phonétiquement.

La partie code Java se charge donc de récupérer les mots reconnus (*token*) par l'API ainsi que la règle à laquelle ils correspondent dans la grammaire (*ruleName*).

```
public class RecoObject extends EventObject{
    public String token;
    public String ruleName;

    public RecoObject(Object source, String tok, String ruleN) {
        super(source);
        token = tok;
        ruleName = ruleN;
    }
}
```

Tableau 18 Code de RecoObject

Le Tableau 18 nous montre le code de l'objet Java *RecoObject* contenant le mot (*token*) et la règle correspondante (*ruleName*).

```
rec.addResultListener(new ResultListener() {

public void resultAccepted(ResultEvent e) {

    javax.speech.recognition.FinalRuleResult r =(FinalRuleResult)(e.getSource());
    ResultToken tok = r.getBestToken(0);

    System.out.println("Result Accepted: "+tok.getSpokenText());
    System.out.println("Rule name = "+((FinalRuleResult)r).getRuleName(0));

    Speech.this.obcs.acceptEvent("ok",new RecoObject
(Speech.this.obcs,tok.getSpokenText(),((FinalRuleResult)r).getRuleName(0)));
}

public void resultRejected(ResultEvent e) {

    Speech.this.obcs.acceptEvent("ko", new EventObject("Echec"));
}
```

Tableau 19 Partie du code chargée d'envoyer des événements Ok et Ko et l'objet RecoObject.

Enfin, notre programme se charge d'envoyer les événements contenant un objet *RecoObject* vers nos modèles PetShop lorsque l'API lui a envoyé les événements *resultAccepted* et *resultRejected* (Tableau 19)

3.3.2 Modèle de liaison avec le code du driver

Processus de modélisation

Le processus de création de ce modèle est relativement simple. Le seul choix que nous avons fait a été d'ajouter deux états, activé et désactivé. Ces deux états permettent de désactiver la reconnaissance vocale lorsque l'utilisateur parle à une autre personne et non à la machine.

Vue générale sur le modèle

Ce modèle se charge essentiellement de faire l'interface entre le code Java et les couches supérieures.
Il permet également d'activer ou non le transfert des événements vers les couches supérieures. De plus, il compte les événements correctement reconnus (événements ok) et les mots non reconnus par l'API (événements ko).

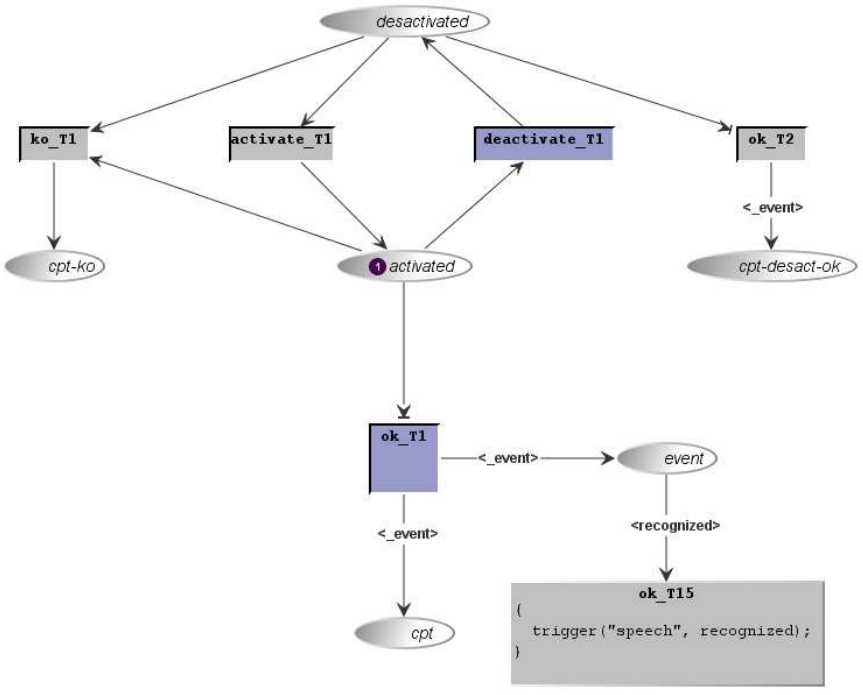


Figure 54 Calcul des événements reconnus ou non reconnus

Description détaillée du modèle

A l'initialisation du modèle, un jeton se trouve dans la place *activated*. Trois transitions sont exécutables :

- soit le réseau reçoit un événement ok, ce qui correspond à un mot reconnu par la reconnaissance vocale. L'événement est envoyé dans la place *event* et dans cpt qui compte le nombre d'événements ok reçus. Lorsqu'un jeton se trouve dans la place *event*, la transition *send_event* peut être tirée. Elle envoie à la couche supérieure un message speech accompagné d'un *RecoObject*.
- soit le réseau reçoit un événement *desactivate* et le jeton de la place *activated* est déposé dans la place *desactivated.* Si le réseau reçoit un événement ok, il sera envoyé sur la place *cpt-desact-ok* qui comptabilise les réceptions de mots acceptés par la reconnaissance vocale alors que le réseau ne transmettait pas aux couches supérieures.
- Enfin, le réseau peut également recevoir des événements ko que celui-ci soit en mode activé ou désactivé. Ces événements sont alors envoyés dans la place *cpt-ko* qui comptabilise le nombre d'événements non acceptés par la reconnaissance vocale.

3.3.3 Modèle de la grammaire

Processus de modélisation

L'objectif de ce modèle était de représenter la grammaire suivante :

```
S = COLOR (COLOR*(DIRECTION | BUTTON))*
DIRECTION = left | right | down | up
BUTTON = click
COLOR = blue | red
```

Tableau 20 Grammaire BNF de la reconnaissance vocale

Il est indispensable de commencer par une couleur. Ensuite, différentes actions peuvent être appelées telles que des déplacements ou des clicks précédés ou non d'une couleur.

Notre choix s'est porté sur une grammaire assez simple. Mais, étant donné les possibilités offertes par notre architecture, nous pouvons changer facilement une de ces couches. Cette grammaire pourrait donc être facilement changée sans trop modifier le comportement de toutes les couches.

Vue générale sur le modèle

Ce modèle représente la grammaire ci-dessus auquel sont ajoutées les transitions d'événements attendus.

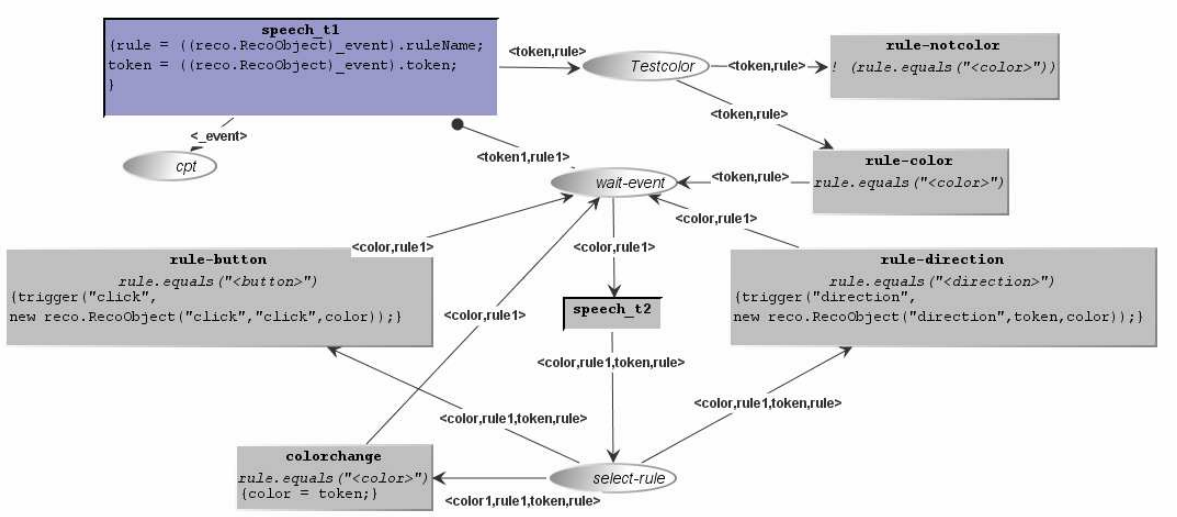


Figure 55 Grammaire de la reconnaissance vocale

Description détaillée du modèle

A l'origine, seule la transition *speech_t1* est franchissable. Lorsque le réseau reçoit un événement *speech* provenant de la couche précédente, cette transition est franchie et dépose un jeton dans la place *TestColor*. Etant donné qu'il faut absolument une couleur pour commencer cette grammaire, on teste la règle de l'objet *RecoObject*.

- Si la règle est bien une couleur, la transition *rule-color* est tirée et un jeton est déposé dans la place *wait-event*
- sinon la transition *rule-notcolor* est franchie et le jeton est évacué du réseau. La transition *speech_t1* reste alors franchissable et attend un événement *speech* contenant une couleur.

Par contre, lorsqu'un jeton se trouve dans la place *wait-event*, un arc inhibiteur désactive la transition *speech_t1*. Seule, la transition *speech_t2* est activée. A la réception de cet événement, un jeton est déposé dans *select-rule* avec comme paramètre color correspondant à la couleur obtenue précédemment et *rule* et *token* correspondant à la règle et à la valeur du nouvel événement reçu.
A ce moment, nous avons trois possibilités :

- la règle est une autre couleur. La transition *colorchange* est franchie et la nouvelle couleur est placée dans le jeton de la place wait-event.
- La règle est une direction. La transition *rule-direction* est franchie, le code contenu dans cette transition envoie un événement contenant la direction et la couleur actuelle. Le jeton est ensuite déposé dans la place *wait-event*.

- La règle est de type button. La transition rule-button est franchie, le code contenu dans celle-ci envoie également un événement contenant la couleur et le click. Le jeton est déposé dans la place *wait-event*.

3.3.4 Modèle du traducteur en événements souris

Processus de modélisation

Ce réseau permet de transformer les différents événements reçus de types *RecoObject* en événements de plus haut niveau appelés *RecoMouseEvent*. Les recherches principales pour ce modèle ont été de créer un objet permettant de s'approcher du niveau de détails des objets événements de la souris.

Vue générale sur le modèle

Le réseau est essentiellement divisé en deux parties :

- la partie de gauche se charge de récupérer les événements reçus du modèle précédent
- la partie de droite envoie les événements à la couche supérieure.

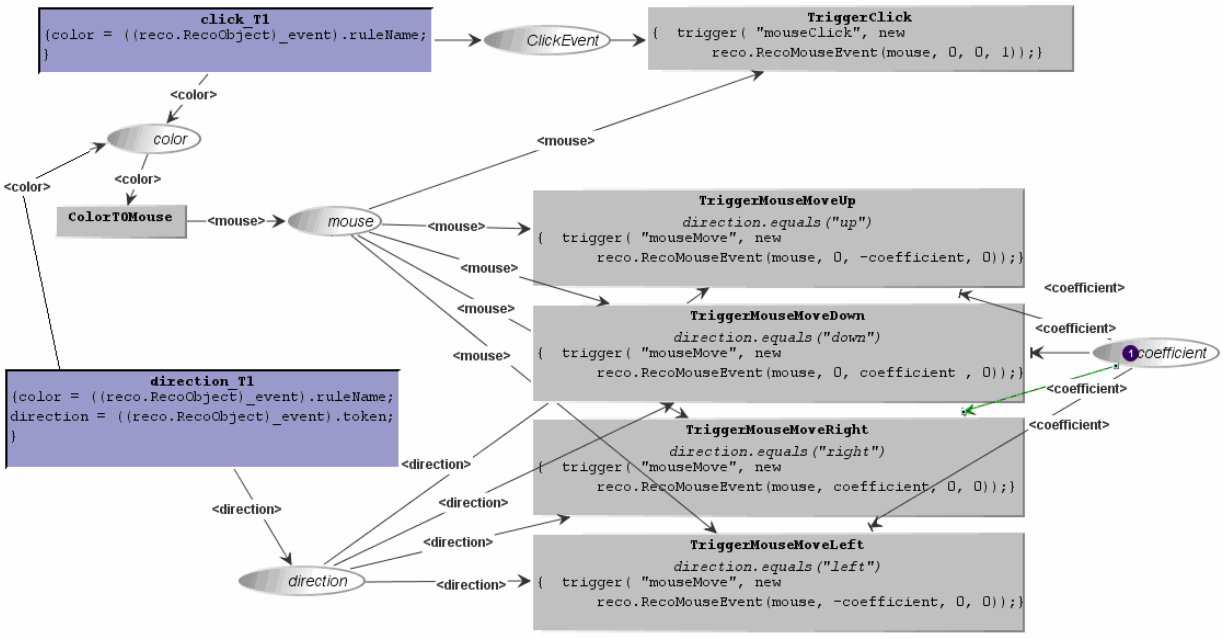


Figure 56 Traducteur en événements souris

Description détaillée du modèle

A l'initialisation du réseau, nous avons un jeton *cœfficient* correspondant au nombre de pixels de déplacement sur l'écran lors de la réception d'un événement direction.
Ce réseau peut recevoir deux types d'événements.

- Lors de la réception d'un événement *click*, la transition *click_t1* est franchie et dépose un jeton contenant la couleur dans la place *color* et un dans la place *ClickEven*t. La transition *ColorTOMouse* est alors franchie et transforme la couleur en un entier qu'elle dépose dans la place *mouse*. Lorsque la place *ClickEvent* et la place *mouse* contiennent un jeton, la transition *TriggerClick* est franchie et un événement mouseClick est envoyé à la couche supérieure contenant le numéro de la souris.

- Lors de la réception d'un événement *direction*, la transition *direction_t1* est franchie et dépose un jeton contenant la direction dans la place *direction* ainsi qu'un jeton dans la place *color*. La transition *ColorTOMouse* est de nouveau tirée et le jeton se retrouve dans la place *mouse*. A cet instant, selon la valeur *direction* du jeton de cette place, une des quatre transitions de type *TriggerMouseMoveRight* sera franchie. Elle enverra un événement de type *mouseMove* à la couche supérieure contenant comme valeur pour *dx, dy* le déplacement selon la direction, multiplié par le cœfficient et comme valeur de souris l'entier se trouvant dans la place *mouse*.

3.3.5 Modèle de calcul des coordonnées absolues

Processus de modélisation

Ce modèle a pour objectif de calculer les coordonnées absolues du pointeur déplacé par la reconnaissance vocale pour permettre d'utiliser notre périphérique vocal dans notre programme de rendu avec une liste d'objets graphiques

Vue générale sur le modèle

Ce modèle est la version simplifiée du calcul de coordonnées absolues pour la souris. Il ne reçoit en effet que les événements *mouseMove* et *mouseClick.* Néanmoins, comme le modèle *AbsoluteCoord* de la souris contient celui-ci, il pourrait également être utilisé.

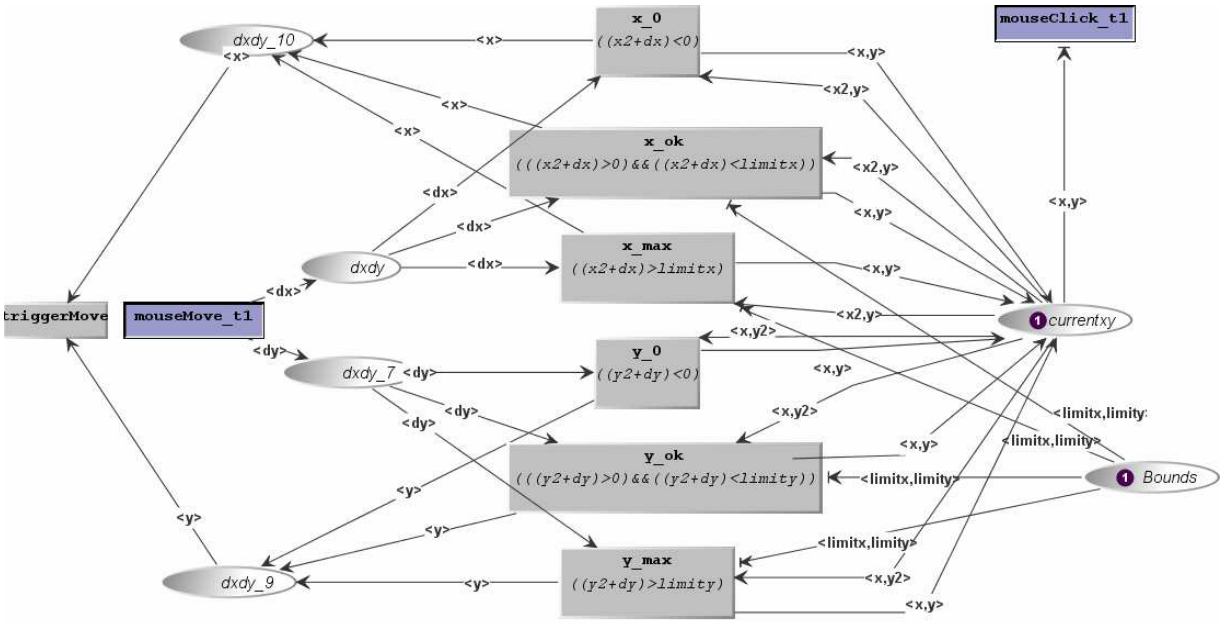


Figure 57 Calcul des coordonnées absolues

Description détaillée du modèle

La version complète de ce modèle est détaillée plus haut (page 60).

3.3.6 Modèles supérieurs et rendu graphique

Le modèle de gestion des objets graphiques a exactement le même comportement que celui de la souris mais en version plus simple. Les seuls événements possibles sont un *mouseMove* et un *mouseClick*. Le fonctionnement de ce modèle est identique au modèle de la souris. Tout comme pour la couche précédente, cette couche pourrait être remplacée par la couche *ObjectPicking* de la souris (Figure 51).

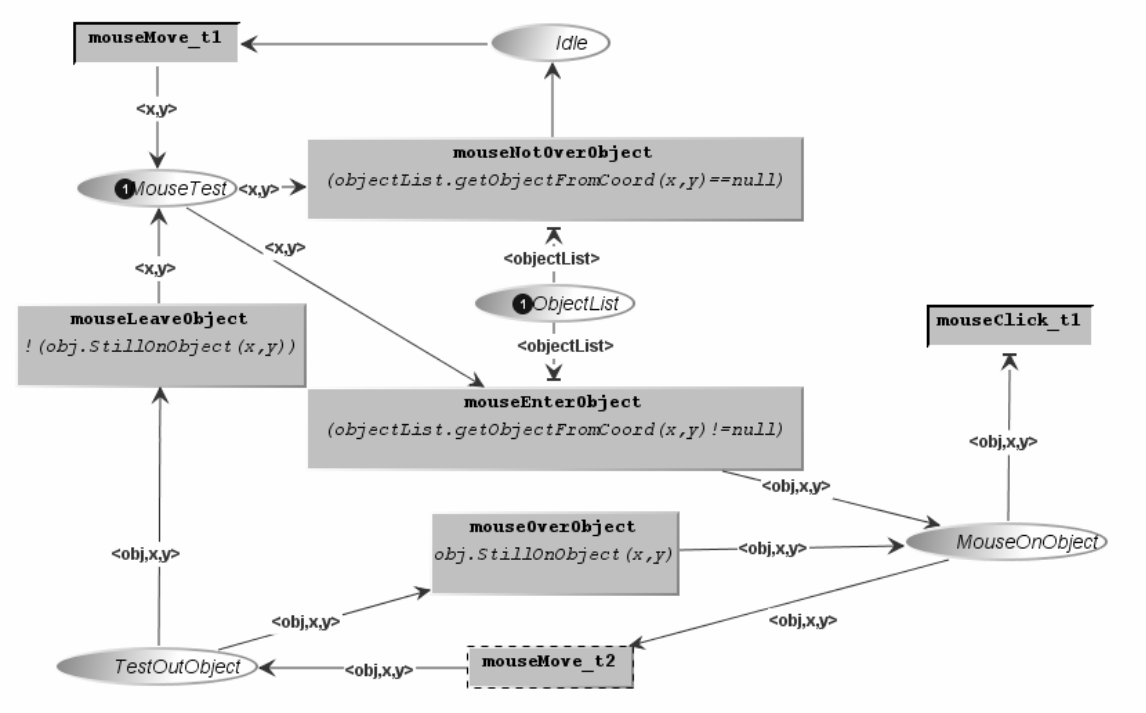


Figure 58 Object Picking

Le rendu graphique se fait de la même manière que pour la souris. Le pointeur est affiché grâce aux entrées de jeton dans la place *currentxy* du réseau *AbsoluteCoord.* Le rendu des objets graphiques se fait par appel de méthodes dans les transitions du réseau *ObjectPicking* (Figure 58)

3.4 Conclusion

Dans cette partie, nous avons adapté à la modélisation de la reconnaissance vocale le même processus que pour la souris. Nous avons tout d'abord créé l'architecture des couches au moyen d'un modèle Arch et les événements entre ces couches grâce à un diagramme de composants. Nous avons ensuite expliqué les nouvelles couches créées en commençant par le processus de modélisation pour finir par l'exécution des modèles. Pour les deux dernières couches qui ne sont que des versions allégées de celle de la souris, nous avons simplement exprimé leurs différences avec l'autre version. Les méthodes de rendus n'ont pas non plus été détaillées dans cette partie car elles utilisent les mêmes méthodes que celles de la souris.

Nous avons également montré que les deux dernières couches pouvaient être remplacées par celle de la souris. Nous pouvons déjà voir ici une possibilité de liaison de différents périphériques lors d'une interaction multimodale.

4 L'écran tactile

4.1 Introduction

La création des modèles de l'écran tactile devait au départ être sensiblement la même que pour la reconnaissance vocale : récupérer les événements de l'écran via une librairie ou un API pour ensuite les traiter afin de les utiliser au final dans une interface Java. Les librairies n'étant pas disponibles, il a fallu développer un driver à partir des signaux venant du port série. L'absence de documentation a posé différents problèmes et une recherche analytique a été nécessaire afin de trouver le « protocole » utilisé par l'écran tactile pour envoyer les informations via le port série.

Différentes librairies de communications ont été testées et notre choix s'est porté sur la librairie gnu.io. A partie de cette librairie, les données reçues par le port série ont été analysées pour trouver les événements correspondant à une action sur l'écran et la position de cette action.

Nous avons ensuite traité ces données afin d'envoyer vers PetShop des événements de type *pressed*, *moved* et *released* avec les x, y absolus correspondant.

4.2 Architecture

Lors de l'utilisation d'un écran tactile, le doigt joue à la fois le rôle de pointeur et de bouton pour le click mais seul le click gauche (du bouton gauche sur une souris) est possible. La gestion des blocs mouseClick s'en trouve simplifiée. Il n'existe pas de couche de gestion du type de déplacement étant donné qu'un déplacement sur l'écran équivaut directement à un déplacement glissé. De plus, le driver créé nous donne directement la position absolue. Il ne faut donc pas passer par une couche de calcul des coordonnées.

Néanmoins, il nous faut une place pour la fonction de rendu du pointeur. Nous avons donc créé un modèle simulant le calcul des coordonnées. La fonction de rendu du pointeur pourrait être appelée directement dans le driver Java mais nous avons choisi de le mettre dans une couche pour une meilleure visibilité.

Notre modèle est donc composé de quatre couches :

- La première couche (Touch Driver) se charge de récupérer les événements provenant du driver. Cette couche sert d'interface. Elle n'accepte que les suites d'événements licites.
- La deuxième couche (Touch Click) se charge de la gestion du click.
- La troisième couche (Touch Coord) simule un calcul de coordonnées pour obtenir une place contenant la position absolue du pointeur.
- La dernière couche (Touch Picking) se charge de gérer les objets graphiques.

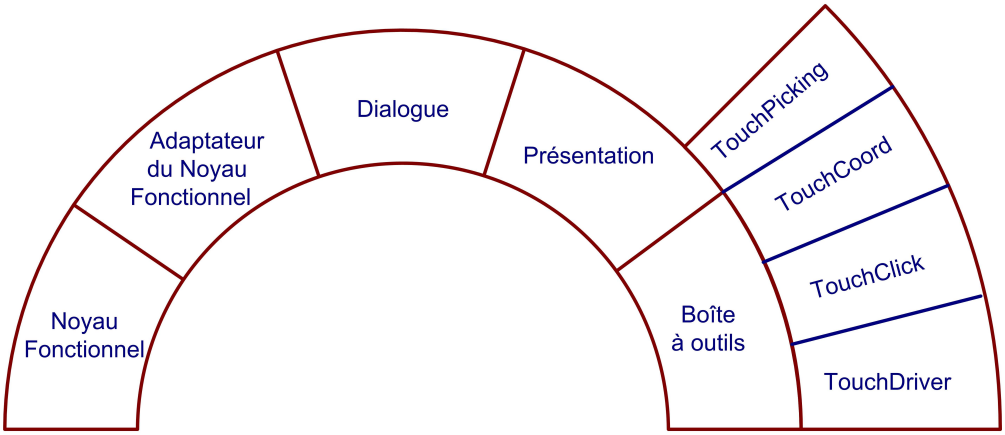


Figure 59 Modèle Arch de l'écran tactile

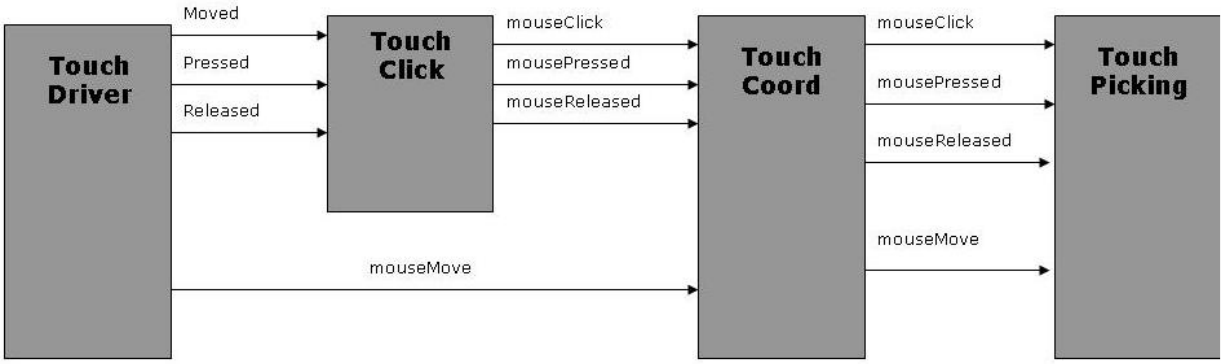


Figure 60 Diagramme de composants de l'écran tactile

4.3 Modéles

4.3.1 Code Java du driver

```
case SerialPortEvent.DATA_AVAILABLE :
     int x = 0, oldx =-1, y = 0, oldy =-1, h =0,preci=2;
     boolean pressed = false;
     try {//lecture du buffer et affichage
          h = (int) fluxLecture.read();
          while (!(h==732))
          {
          if (h == 216) {
               y = (int) fluxLecture.read();
               x = (int) fluxLecture.read();

               if (pressed==false)
                      {
                       System.out.println("Pressed");
                       GnuComRs.this.obcs.acceptEvent("Touch", new
                             TouchObject(GnuComRs.this.obcs,"Pressed",x,y));
                       pressed = true;
                         }
               else
                      {
                      if ((x > oldx+preci)||(x < oldx-preci)||(y > oldy+preci)||(y <
oldy-preci))
                         {
                           System.out.println("Moved");
                           GnuComRs.this.obcs.acceptEvent("Touch", new TouchObject
               (GnuComRs.this.obcs,"Moved",((124-x)*(1280/128)),((127-y)*(1024/128))));
                          }
                          h = (int) fluxLecture.read();
                     }
                     oldx = x;
                     oldy = y;
                 }
                 else
                 {
                    h = (int) fluxLecture.read();
                 }
                 System.out.flush();
               }
            System.out.println("Released");
            GnuComRs.this.obcs.acceptEvent("Touch", new
       TouchObject(GnuComRs.this.obcs,"Released",oldx,oldy));
              } catch (IOException e) {}
      break;
```

Tableau 21 Code du driver de l'écran tactile

Le Tableau 21 présente le code de notre driver Java pour l'écran tactile. Lors de la réception de données sur le port série (`SerialPortEvent.DATA_AVAILABLE`), le driver analyse les données reçues et les envoie sous forme d'événements vers les modèles PetShop. Ces événements sont accompagnés d'un objet *TouchObject* qui contient le type d'action (*Moved*, *Pressed*, *Released*) et la position où a eu lieu cette action sur l'écran. De plus, pour éviter de surcharger PetShop en événements, un tri est effectué. Si la nouvelle position reçue ne diffère pas d'un certain seuil par rapport à l'ancienne, elle est ignorée. Ce tri peut également être géré au niveau des modèles mais il induit une surcharge d'événements envoyés.

4.3.2 Récupération des événements de l'écran tactile.

Processus de modélisation

Le but essentiel de cette couche est de servir d'interface entre le code java et les couches supérieures. Cette couche limite les événements reçus aux événements licites.

```
Pressed Moved* Released
```

Vue générale sur le modèle

Ce modèle est composé d'une transition synchronisée et de trois transitions mutuellement exclusives en fonction de la valeur de l'événement reçu.

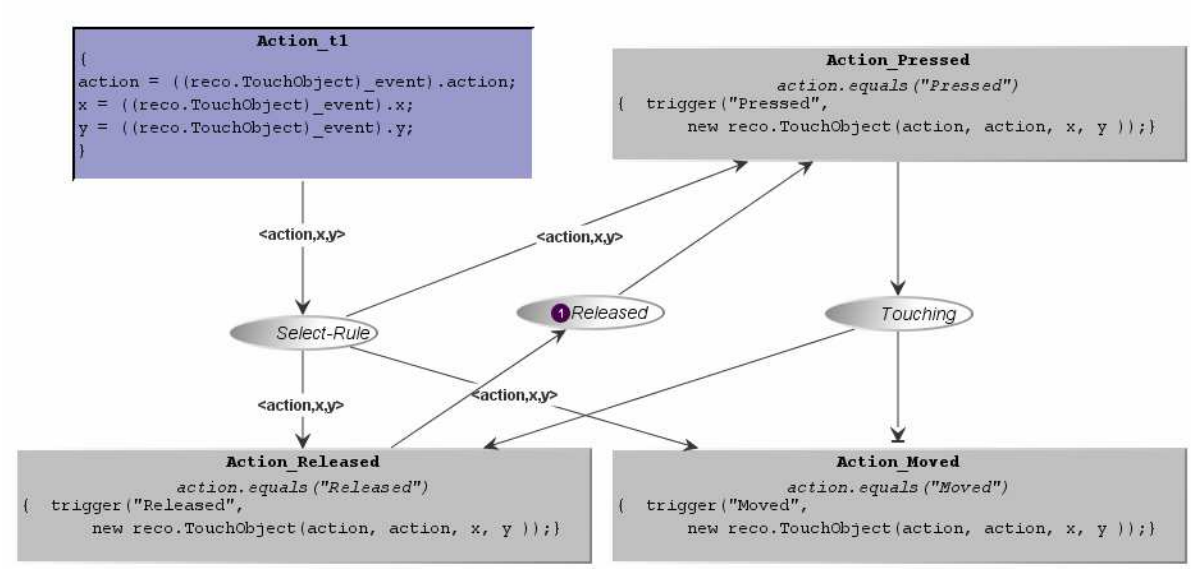


Figure 61 Interface de l'écran tactile

Description détaillée du modèle

A l'initialisation de ce modèle, seule la transition *Action_t1* est franchissable.
Lors de la réception de cet événement, un jeton est déposé dans la place *select-Rule*. Ce jeton contient le type d'action reçue ainsi que la position absolue de l'endroit où a eu lieu cette action. Selon la valeur de cette action, les transitions *Action_Pressed*, *Action_Moved* ou *Action_Released* sont tirées et envoient respectivement un événement Pressed, Moved ou Released.
Pour éviter d'obtenir des séquences non conformes, les places *Released* et *Touching* sont ajoutées. Lorsqu'un événement *Pressed* est reçu, il n'est accepté que si *Released* contient un jeton. De même, la transition Action_Released n'est franchie que si *Touching* contient un jeton. Enfin, un arc de test entre *Touching* et *Action_Moved* permet de n'accepter les événements Moved que lorsque l'on a reçu un *Pressed* (lorsque le doigt de l'utilisateur est encore sur l'écran).

4.3.3 Gestion du click (TouchClick)

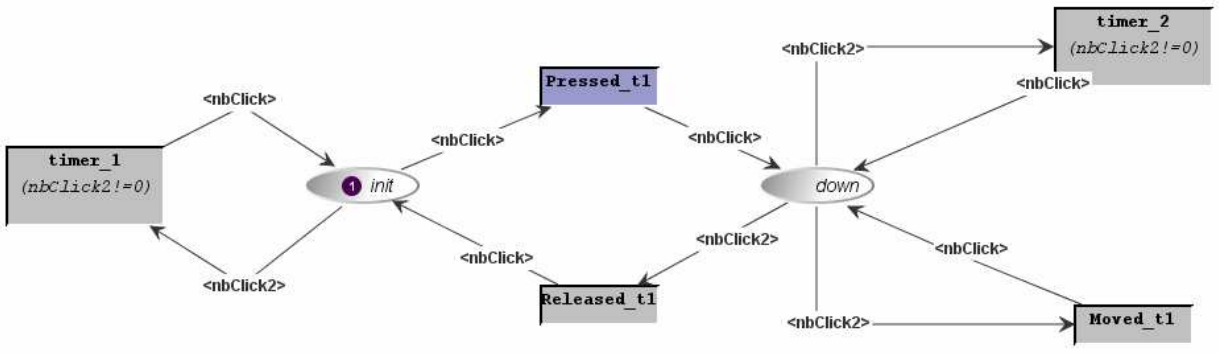


Figure 62 Gestion du click de l'écran tactile

Pour ce modèle, nous reprenons une version simplifiée de celui de la souris.
Nous lui avons retiré les couples places transitions servant à différencier les boutons. Une explication de ce modèle se trouve plus haut (page 56).

4.3.4 Calcul des coordonnées absolues

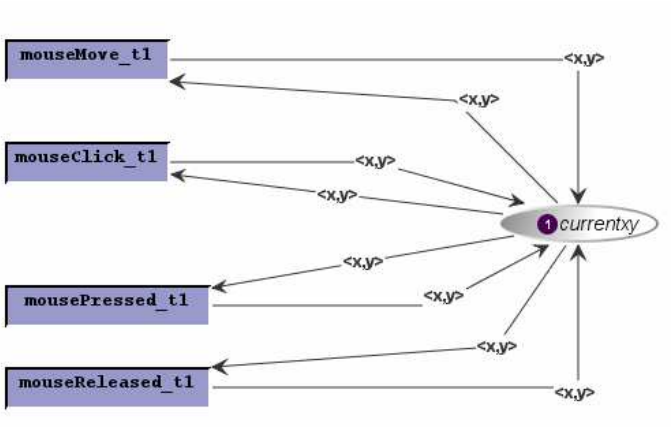


Figure 63 Calcul des coordonnées absolues de l'écran tactile

Cette couche n'a pas réellement pour but de calculer les coordonnées absolues mais simplement de créer une place permettant à la fonction de rendu de connaître la position du pointeur. Ce réseau reçoit donc les différents événements, récupère leurs positions absolues et transfère les événements à la couche supérieure.

4.3.5 Modèles supérieurs et rendu graphique

Comme nous venons de l'expliquer, le rendu du pointeur est réalisé de la même façon que pour la souris grâce à la place *currentxy* du réseau *AbsoluteCoord* ci-dessus.

A la différence d'une souris, le pointeur d'un écran tactile peut entrer en contact avec un objet seulement lors de la pression du doigt sur l'écran. Il n'est donc pas possible de réutiliser le modèle *ObjectPicking* de la souris. En effet, dans le modèle de la souris, la méthode *getObjectFromCoord* est appelée lors de la réception d'un événement *mouseMov*e. Cet événement est l'événement essentiel pour la gestion des objets graphiques.
Par contre, pour l'écran tactile, le pointeur peut se retrouver sur l'objet directement sans qu'il y ait eut un déplacement. Il suffit simplement que l'utilisateur touche l'emplacement de l'objet sur l'écran. L'événement essentiel pour l'écran tactile est donc le mousePressed

Si l'utilisateur n'est pas sur l'objet et qu'il déplace son doigt, le système reconnaîtra cette action comme une sélection.

Lorsque l'utilisateur a son doigt sur un objet, il peut :

- déplacer son doigt ce qui déplacera l'objet,
- retirer son doigt de l'écran ce qui enverra un événement *Released* et éventuellement un événement *click* si le délai entre le *Pressed* et le *Released* n'est pas trop grand (ceci est réalisé par la couche *touchClick*)

La gestion des objets graphiques est réalisée par le modèle ci-dessous.

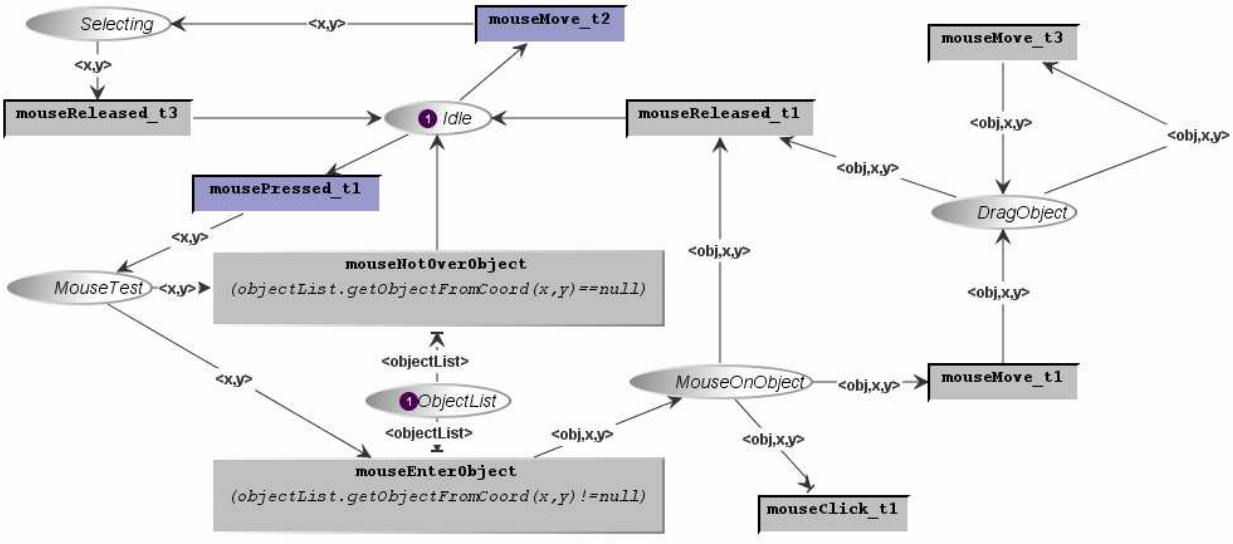


Figure 64 Gestion des objets graphiques pour l'écran tactile

A l'origine, le modèle peut recevoir deux types d'événements *mousePresse*d et *MouseMove* (même si le premier événement sera toujours un *mousePressed*).

A la réception de cet événement, tout comme pour le modèle *ObjectPicking* de la souris, deux places peuvent être tirées selon la position du doigt sur un objet ou non grâce à la fonction *getObjectFromCoord*.

- Si aucun objet graphique n'est sous le doigt, la transition *mouseNotOverObject* est tirée et le jeton retourne dans la place *Idle*. A cet instant, si un événement mouseMove est reçu, un jeton est déposé dans *selecting* jusqu'à la réception d'un *mouseRelased* qui terminera la sélection. Le jeton est alors redéposé dans la place *Idle.*

- Si un objet graphique est sous le doigt, la transition *mouseEnterObjec*t est tirée et un jeton est déposé dans la place *MouseOnObject*.

    - Si un événement mouseClick est reçu, la méthode pour le click est exécutée sur l'objet.
    - Si un événement *mouseMove* est reçu, le jeton est déposé dans DragObject et la fonction *startDragObjet* est appelée sur l'objet. Les événements *mouseMove* suivants appellent la méthode *dragObjet*. Lors de la réception d'un *mouseReleased*, le déplacement de l'objet est terminé et le jeton est replacé dans la place *Idle*.
    - Si un événement *mouseReleased* est reçu, la méthode correspondante est appelée sur l'objet et le jeton est également replacé dans la place *Idle*.

Remarque

Il serait néanmoins possible de créer un modèle pour l'écran tactile qui transformerait les événements reçus en événements de même type que ceux d'une souris. Nous pourrions ajouter une couche qui transforme les événements déplacements en positions relatives et une couche qui simule des déplacements simples. Lors de la réception d'un *Pressed*, un événement *mouseMove* serait envoyé simulant le trajet du doigt lorsque celui-ci n'est pas en contact avec l'écran tactile. Ceci permettrait d'utiliser les couches supérieures de la souris, comme la gestion des objets graphiques.

4.4 Conclusion

Dans ce chapitre, nous avons exprimé les fortes différences qui existent entre les modèles pour l'écran tactile et les deux périphériques précédents. En effet, le pointeur de l'écran tactile ne se déplace que lors d'un déplacement glissé et un objet ne peut être actionné que par une pression à sa position sur l'écran. De plus, le déplacement simple est réalisé lorsque le doigt n'est pas en contact avec l'écran et ne peut donc être modélisé.

Nous avons donc créé une nouvelle architecture pour ce périphérique grâce à un modèle Arch et exprimé les événements transmis entre ces couches par un diagramme de composants. Ensuite, nous avons expliqué chacune des couches et, plus particulièrement, les différences par rapport aux modèles de la souris. Enfin, nous avons exprimé une solution pour pouvoir utiliser les couches haut niveau de la souris.

5 Multimodalité

5.1 Introduction

Une des possibilités offertes par la multimodalité est par exemple de déplacer un objet graphique à l'aide de deux souris. La première souris se place sur l'objet, clique sur l'objet et le déplace. Ensuite, la seconde souris vient se placer à la nouvelle position de cet objet, clique et se déplace. A cet instant, plusieurs possibilités s'offrent à nous :

- la première souris garde la main sur l'objet et l'objet reste à sa position.
- la seconde souris prend la main et déplace cet objet.
- bien d'autres comportements modélisables (la sélection du pointeur en fonction de celui qui tire le plus fort, l'objet est déposé et perdu pour les deux souris, etc)

Afin de pouvoir modéliser ces comportements, l'événement de haut niveau « déplacer un objet graphique » n'est pas suffisant. Il nous faut connaître le détail des actions effectuées sur chaque périphérique. Nous pouvons donc voir l'utilité de notre modèle en couche qui permet de décrire l'événement déplacer un objet graphique en actions et conditions : une pression d'un bouton, suivi d'un déplacement avec le bouton enfoncé, dont le début de ce déplacement a été effectué sur un objet graphique.

C'est pourquoi nous avons eu pour objectif dans ce mémoire, tout d'abord, de détailler dans l'état de l'art comment arriver à modéliser des comportements de périphériques et de définir la multimodalité, et, ensuite de modéliser les différents périphériques indépendamment.

Dans ce chapitre, nous allons intégrer ces différents périphériques dans une même architecture pour obtenir la multimodalité.

Nous commencerons par exprimer ces architectures à l'aide de modèles Arch, ensuite nous détaillerons les changements effectués sur la couche de gestion des objets graphiques afin de permettre la multimodalité.

5.2 Architecture

L'architecture de notre système multimodal est une fusion des architectures de chaque périphérique. Lorsque nous créons un système multimodal, la montée des informations dans le modèle Arch se fait par plusieurs canaux en parallèle. Le nombre de ces canaux dépend du nombre de périphériques utilisés. La jonction entre les différents périphériques peut être réalisée à différents niveaux selon leurs types et l'architecture de leurs modèles. De nombreuses combinaisons sont possibles mais nous nous limiterons à trois exemples :

- deux souris
- une souris et la reconnaissance vocale
- l'écran tactile et la reconnaissance vocale

5.2.1 Deux souris

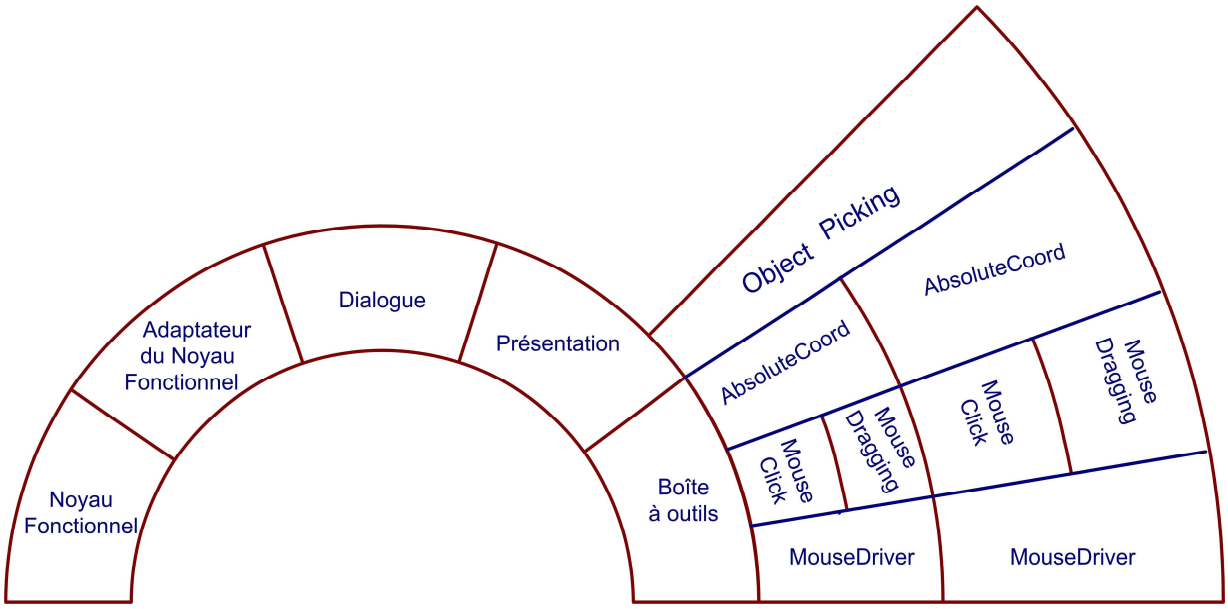


Figure 65 Modèle Arch de deux souris

Lorsque deux souris sont utilisées en parallèle, il nous faut utiliser un modèle de la première couche pour chacun des périphériques. Il en va de même pour la couche suivante : les couches mouseDragging et mouseClick et la couche de calcul des coordonnées (AbsoluteCoord) apparaissent deux fois.

Par contre, pour la couche chargée de la gestion des objets graphiques (ObjectPicking), il est possible d'envoyer les événements des deux périphériques dans un même modèle, moyennant quelques modifications permettant de reconnaître la source de cet événement, ce que nous verrons dans le point modélisation. Pour cette dernière couche, le nombre de périphériques reliés importe peu (si ce n'est en terme de performances), il est donc possible de connecter à cette couche un nombre illimité de souris. Le rendu des pointeurs est réalisé à partir des deux couches de calcul des coordonnées.

5.2.2 Une souris et la reconnaissance vocale

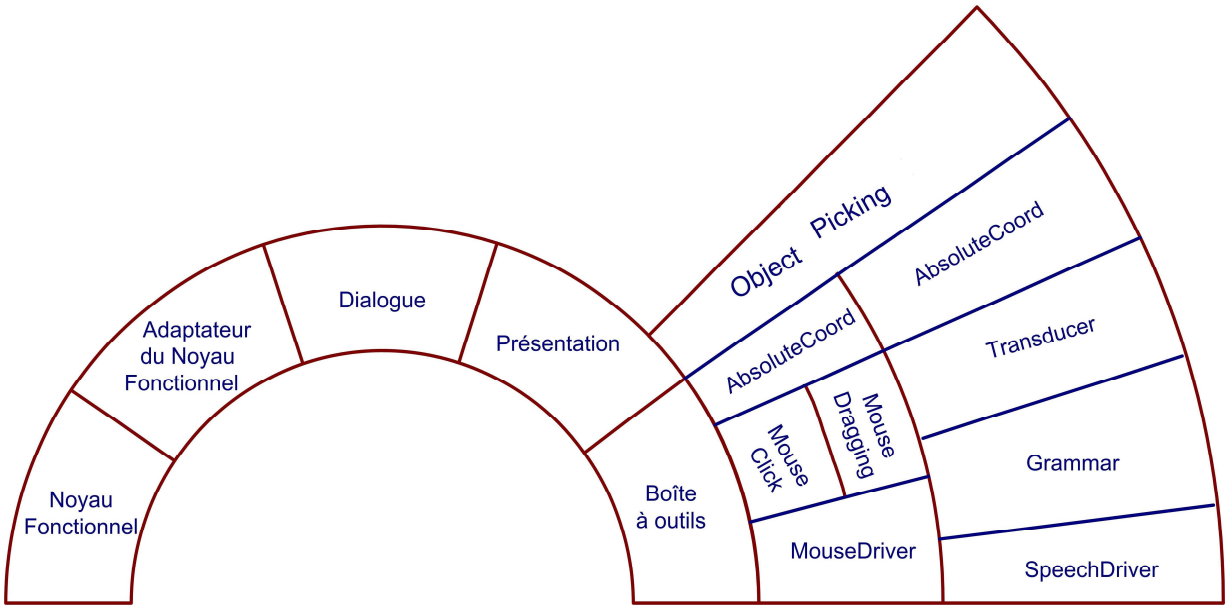


Figure 66 Modèle Arch d'une souris et de la reconnaissance vocale

Lorsqu'une souris et la reconnaissance vocale sont utilisées en parallèle, nous pouvons également utiliser la couche supérieure de gestion des objets graphiques. La seule différence avec l'architecture de plusieurs souris est le canal destiné à la reconnaissance vocale composé des couches *SpeechDriver*, *Grammar* et *Transducer* qui se substitue au canal d'une souris. La version modifiée de la couche de gestion des objets pour plusieurs souris est également valable pour la reconnaissance vocale étant donné que nous avons adapté le comportement de la reconnaissance vocale à celui d'une souris. De même, le rendu des pointeurs est réalisé à partir des deux couches de calcul des coordonnées.

5.2.3 L'écran tactile et la reconnaissance vocale

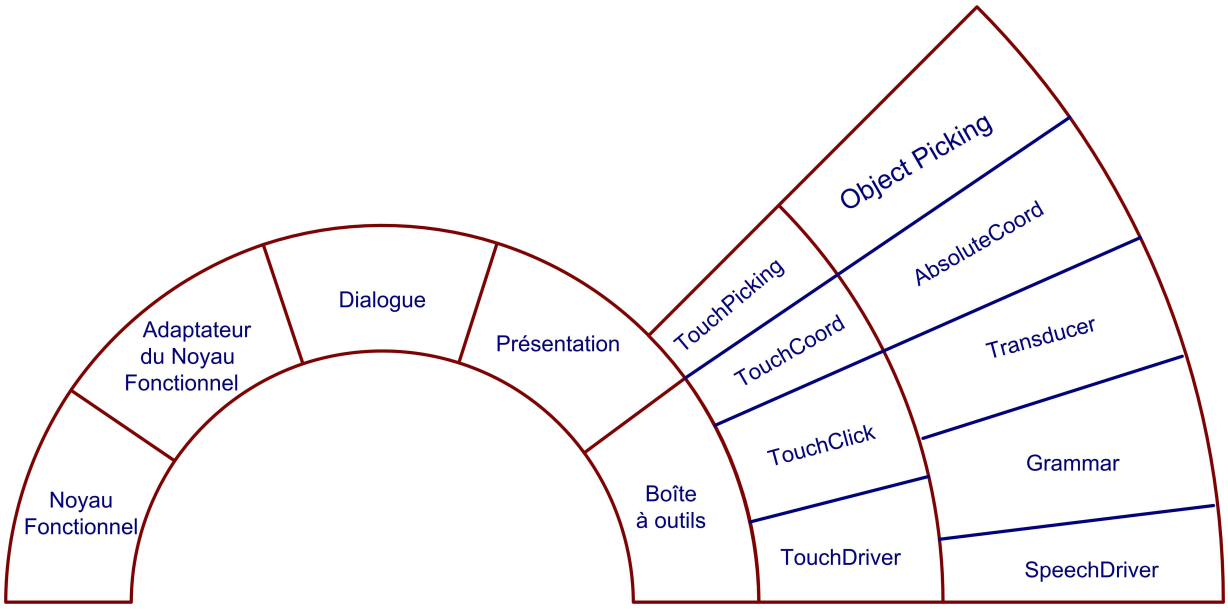


Figure 67 Modèle Arch de l'écran tactile et de la reconnaissance vocale

Par contre, lors de la mise en parallèle de l'écran tactile et de la reconnaissance vocale, les deux canaux restent séparés. En effet, les modèles de l'écran tactile n'ont pas la même gestion des objets graphiques (l'événement qui compare la position avec la liste d'objets pour l'écran tactile est le mousePressed alors que pour les autres périphériques, c'est le *mouseMove*). Ces deux canaux ne paraissent donc pas liés. Néanmoins, la liste des objets graphiques étant partagée entre les deux modèles de gestion des objets, il est possible d'utiliser ces périphériques en parallèle. Il en est de même si nous remplaçons la reconnaissance vocale par une ou plusieurs souris.

Comme nous l'avons expliqué lors de la modélisation de l'écran tactile, il aurait été possible de créer une architecture permettant d'utiliser les mêmes couches haut niveau que celle de la souris.

5.3 Modèles

Lors du processus de création des différents modèles de chaque périphérique, nous avons toujours eu comme objectif d'avoir des modèles permettant de gérer la multimodalité. De ce fait, les modèles peuvent, à quelques modifications près, gérer plusieurs périphériques d'entrée. Ceci est réalisé par l'ajout dans les modèles d'un couple composé d'une place et d'une transition qui donne aux transitions événements la faculté de pouvoir comparer le périphérique source de l'événement avec les périphériques du modèle (le modèle possède une référence de cette source représentée par un jeton dans une place contenant les références des différents périphériques).

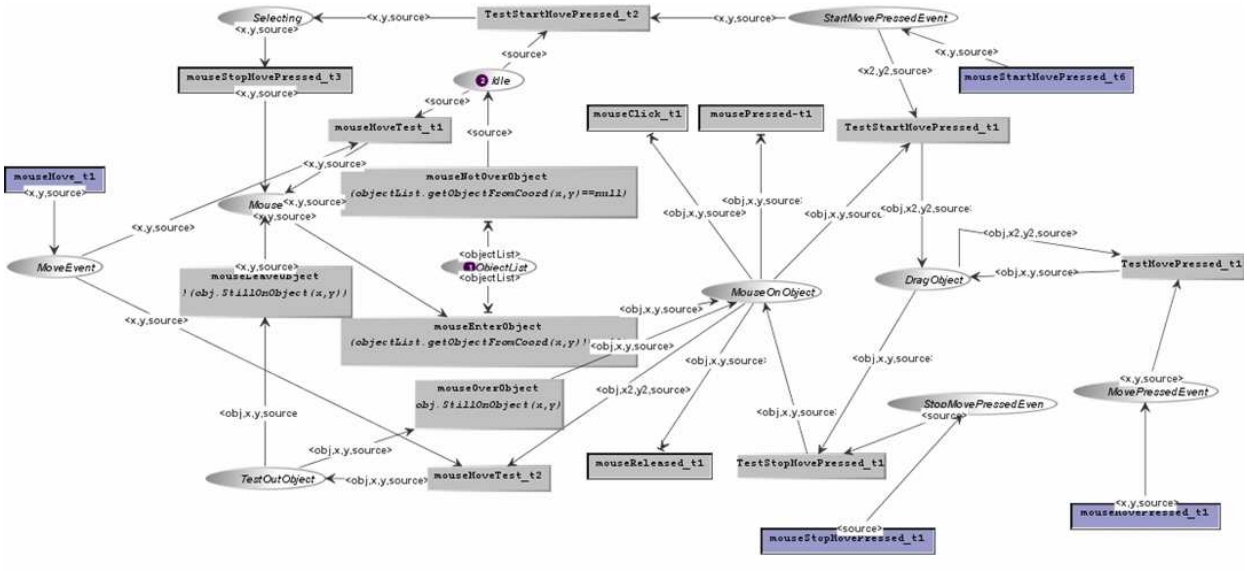


Figure 68 Gestion des objets graphiques pour la multimodalité

Comme nous pouvons le voir par rapport au réseau *ObjectPicking* simple (Figure 68), les transitions *mouseMove*, *mouseStartMovePressed*, *mouseMovePressed* et *mouseStopMovePressed* ont été remplacées par des transitions *mouseMoveTest*, *TestStartMovePressed, TestMovePressed et TestStopMovePressed.*

Les transitions *mouseMove, mouseStartMovePressed, mouseMovePressed* et *mouseStopMovePressed* existent toujours mais ne sont plus directement liées au réseau. En effet, si nous laissions par exemple le modèle *ObjectPicking* simple (Figure 68), lorsque qu'un jeton se trouve dans la place *MouseOnObject* et un autre dans la place *Idle*, si le réseau reçoit un événement mouseMove, les deux transitions *mouseMove* auraient été franchies alors que ce déplacement ne provenait que d'un seul périphérique. Il nous faut donc stocker cet événement (*MoveEvent* pour le mouseMove) et en comparer la source avec celle qui se trouve dans le jeton de la place *MouseOnObject* et dans celui de la place *Idle*.

Il en va de même pour les transitions citées plus haut. Nous ajoutons donc pour chaque jeton, une valeur source correspondant au numéro de la souris.
A l'origine, ce réseau contient un jeton par périphérique avec comme valeur les entiers 0 pour les variables x et y et un entier correspondant au numéro du périphérique pour la variable source.

5.4 Conclusion

Dans ce chapitre, nous avons exprimé trois exemples de multimodalité : deux souris, une souris et la reconnaissance vocale, et l'écran tactile et la reconnaissance vocale.

Nous avons fusionné les architectures réalisées pour chaque périphérique pour obtenir une architecture multimodale. Cette architecture a été présentée à l'aide d'un modèle Arch.
Ensuite, nous avons expliqué les changements effectués sur la couche de gestion des objets graphiques pour parvenir à cette multimodalité.

6 Conclusion

Dans cette partie, nous avons tout d'abord créé une architecture en couches pour la souris. Cette architecture est composée de quatre couches :

- La gestion du driver de la souris,
- La gestion du click et du type de déplacement (couche décomposée en 2 modèles),
- Le calcul des coordonnées absolues,
- La gestion des objets graphiques.

Nous avons détaillé ces différentes couches en expliquant la conception, le principe et le fonctionnement détaillé.

Nous avons également créé l'architecture pour la reconnaissance vocale utilisée à la manière d'une souris. Nous avons donc pu récupérer les couches de gestion des objets graphiques et de calcul des coordonnées. Trois couches bas niveau ont été ajoutées :

- La gestion du driver de la reconnaissance vocale,
- La grammaire de la reconnaissance vocale,
- La traduction en évènements souris des mots reconnus par cette grammaire.

Ensuite, nous avons créé l'architecture pour l'écran tactile. Malheureusement, il n'a pas été possible de récupérer les couches des autres modèles étant donné le fonctionnement de l'écran tactile. Nous avons donc créé quatre nouvelles couches :

- La gestion du driver de l'écran tactile,
- La gestion du click,
- Le calcul des coordonnées absolues,
- La gestion des objets graphiques.

Nous avons néanmoins énoncé une solution permettant d'utiliser les couches de la souris pour l'écran tactile.

Notre objectif était de pouvoir lier ces différentes architectures afin de pouvoir utiliser plusieurs périphériques de manière multimodale.

Enfin, à partir de ces modèles, nous avons créé une architecture multimodale pour chacun de nos trois exemples :

- deux souris,
- une souris et la reconnaissance vocale,
- l'écran tactile et la reconnaissance vocale.

Dans ces deux premières architectures, nous avons modifié la couche de gestion des objets graphiques pour que celle-ci puisse recevoir des événements de plusieurs périphériques en pouvant en distinguer la source.

Pour l'exemple de la reconnaissance vocale et de l'écran tactile, un modèle de ce type n'a pu être réalisé mais notre liste d'objets utilisée dans les modèles de gestion des objets graphiques peut être partagée entre ces deux modèles.

Cette partie nous a montré les avantages d'une modélisation complète à partir du plus bas niveau possible pour chaque périphérique afin de pouvoir modéliser des comportements multimodaux complexes.

Il peut apparaître étrange que le cœur de notre travail, la multimodalité, ne soit pas plus long. Mais, étant donné notre choix de commencer par le plus bas niveau possible, nous avons développé des modèles de couches basses permettant d'être facilement fusionnés entre eux. Les architectures de nos modèles multimodaux sont donc principalement une fusion des modèles des périphériques.

Partie 3 Application : Image

1 Introduction

L'équipe du LIIHS devait réaliser une application pour le CNES (Centre National d'études spatiales). Le projet avait comme cadre d'étude les différents modes d'interactions pour la gestion de satellites. Nous avons pris part à ce projet, ce qui nous a permis d'intégrer nos modèles dans une application réelle et d'en tester l'exécution.

L'application Image est un agenda qui permet de définir les différentes procédures qui devront être exécutées sur les satellites. Cette application permet de définir l'ordre et les dépendances entre ces différentes procédures.

Le but de ce projet est de tester l'opportunité d'une interface multimodale pour la gestion de satellites et de démontrer l'utilité des techniques de description formelle de systèmes interactifs pour la spécification des systèmes interactifs multimodaux.

Image permet de tester de nouveaux modes d'interaction en vue de réduire les coûts, d'améliorer l'utilisabilité et de tester l'exploitation des techniques de description formelle pour des applications spatiales multimodales.

L'utilisation de la multimodalité dans cette application permet de maîtriser la complexité croissante des systèmes en améliorant l'interaction homme-machine en augmentant l'interaction entre utilisateur et systèmes interactifs et en gérant plus efficacement le nombre de commandes.

De plus, de nouveaux styles d'interactions sont implémentés afin également d'augmenter l'interaction entre l'utilisateur et le système.

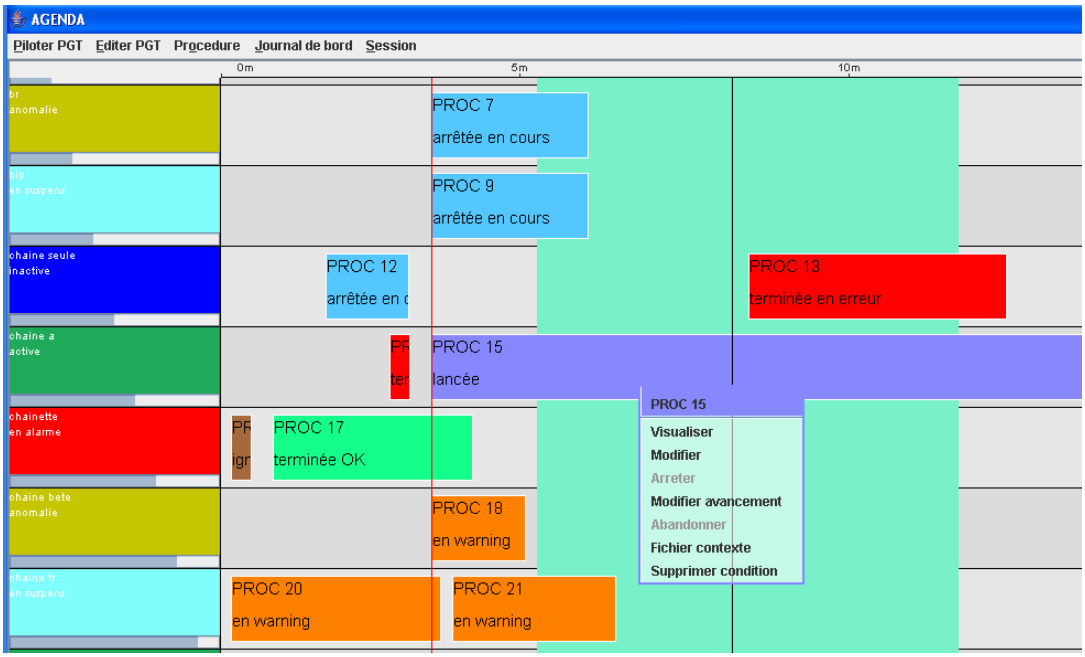


Figure 69 Ecran principal de l'application Image

2 Présentation de l'application

De nombreuses techniques d'interactions ont été implémentées dans cette application.

La Figure 69 présente l'écran principal de l'application ainsi qu'une technique d'interaction. Lorsque l'on réalise un click droit sur une procédure, le menu déroulant est semi-transparent.

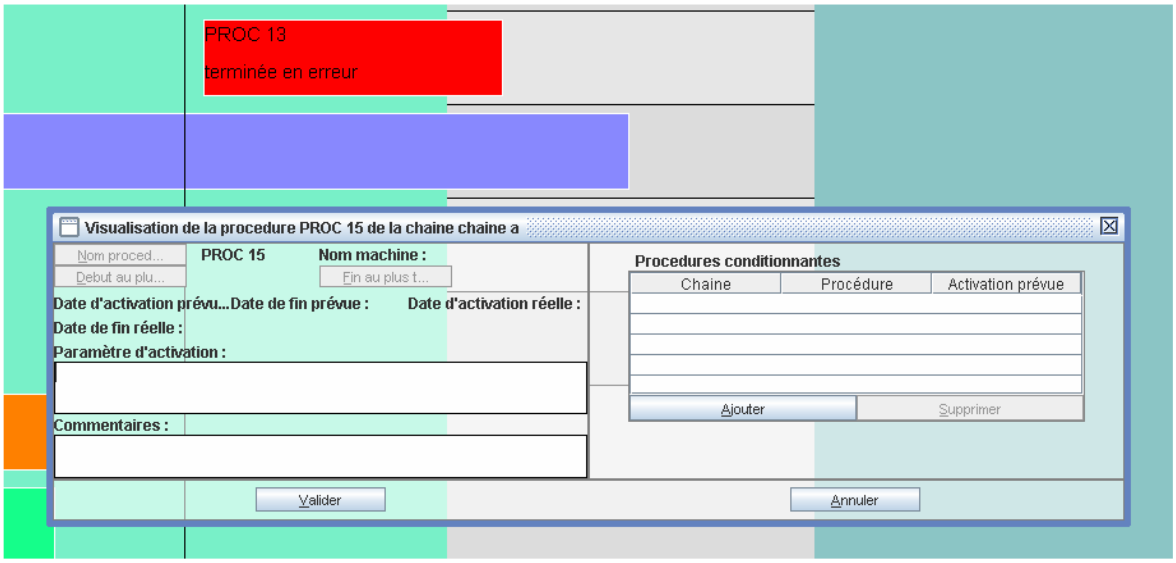


Figure 70 Visualisation d'une procédure

La Figure 70 présente la fenêtre de modification d'une procédure. Nous pouvons par exemple lui ajouter des procédures conditionnantes[2]. Cette fenêtre est également semi-transparente.

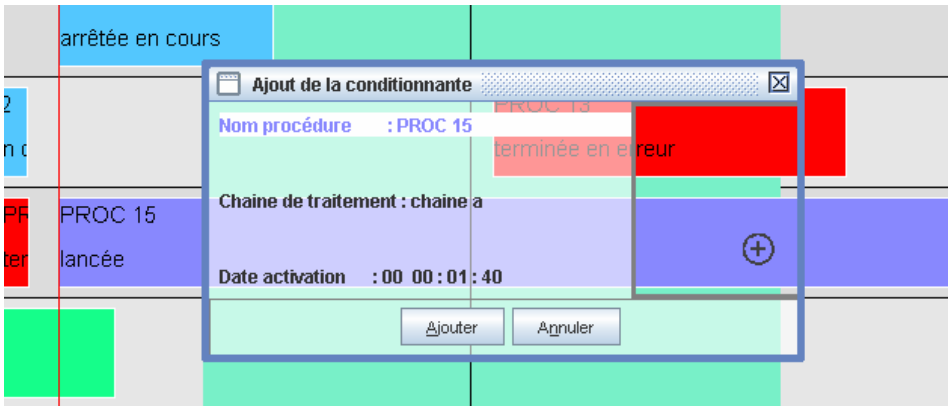


Figure 71 Ajout d'une procédure conditionnante

[2] Les procédures peuvent être liées entre elles. On appelle procédure conditionnante, une procédure qui est nécessaire à l'exécution d'une autre.

Enfin, la Figure 71 représente l'ajout d'une procédure conditionnante. Cet ajout peut être réalisé à l'aide de deux souris et la technique d'interaction appelée « click-trough ».
La fenêtre d'ajout de la procédure contient un cadre transparent permettant de sélectionner une procédure à travers la fenêtre à l'aide d'une souris, tout en déplaçant cette fenêtre avec l'autre souris.

3 Architecture

Pour cette application, nous avons utilisé différents types d'interaction multimodale.

L'application se pilote essentiellement avec deux souris en parallèle. Etant donné les délais de livraison de l'application, les modèles de la souris vus dans le chapitre précédent n'ont pas pu être implémentés.

Néanmoins, nous lui avons ajouté une partie des modèles de l'écran tactile et de la reconnaissance vocale.

La partie reconnaissance vocale a été modélisée d'une autre manière que celle vue dans la partie contribution (page 68). Au lieu de piloter une souris par la voix, l'utilisateur fait appel à des fonctions de plus haut niveau (de la même manière que pour le Put That Here de Bolt). Après avoir sélectionné une procédure avec son doigt sur l'écran tactile, les fonctions modifier et effacer peuvent être appelées. De plus, l'utilisateur peut accéder à une procédure particulière en l'appelant par son numéro ou en énonçant « procédure » suivi du numéro. Enfin, il peut également appeler les procédures qui ont produit une erreur en appelant la première par la commande « erreur » et en les faisant défiler à l'aide des commandes « précédente » et « suivante »

Nous avons donc créé une grammaire dictionnaire pour ces différents mots. Ensuite, nous avons repris nos différentes couches de driver de l'écran tactile et de la parole.

Pour la partie reconnaissance vocale, il n'est bien entendu pas utile de conserver les couches les plus hautes que nous avons expliquées dans le chapitre précédent. Il nous suffit de garder la couche gérant le transfert des événements du driver Java vers les réseaux PetShop (page 72) et de créer une couche permettant de transmettre les différentes commandes vers le programme Java.
Pour la partie écran tactile, nous gardons la couche qui se charge de récupérer les trois types d'événements et nous transmettons les événements vers ce même programme. Une explication de cette couche est disponible à la page 82.

4 Modélisation

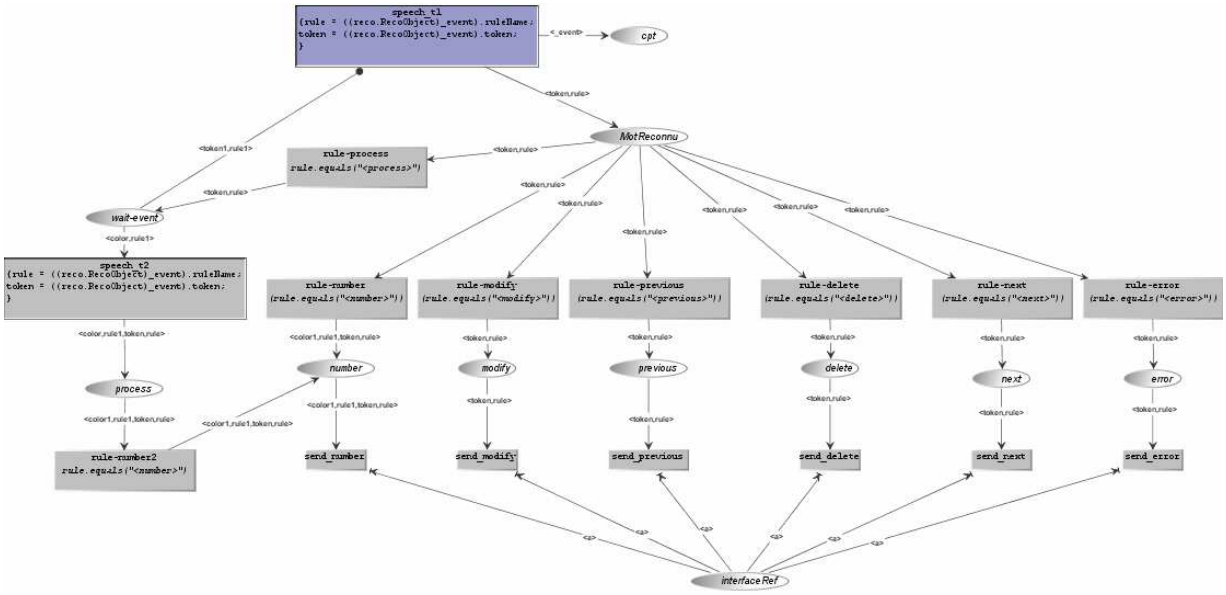


Figure 72 Grammaire de la reconnaissance vocale pour l'application Image

Ce modèle (Figure 72) exprime la grammaire de l'écran tactile pour l'application Image. Lors de la réception d'un mot reconnu, un jeton est déposé dans la place MotReconnu. Selon sa valeur *rule*, il passe par la transition *rule-number, rule-modify, rule-previous, rule-delete, rule-next ou rule-error*. Il est alors déposé dans la place correspondante *number, modify, previous, delete, next ou error*. Si le mot reconnu est « *process* », un jeton est déposé dans la place *wait-event*. Le réseau attend alors un second mot reconnu qui correspond au numéro de cette procédure. Lorsque ce mot est reçu et correspond bien à un numéro, un jeton est déposé dans la place *number* de la même façon que si l'utilisateur avait directement énoncé ce numéro.

Lorsqu'un jeton se retrouve dans la place *previous* (Figure 73), la transition *send_previous* envoie un événement vers l'interface de l'application Image que l'on trouve en paramètre dans la place *interfaceRef*.

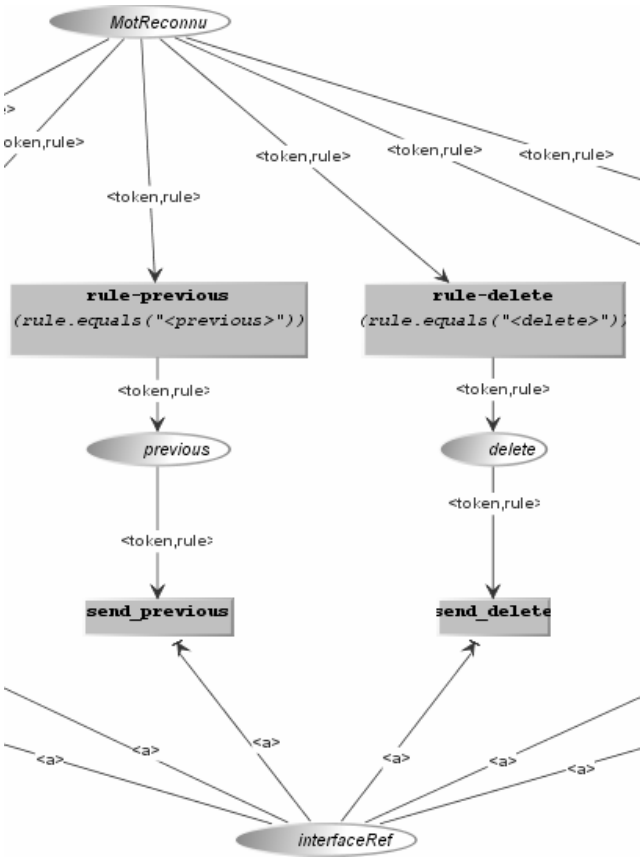


Figure 73 Gestion de la reconnaissance vocale pour l'application Image (Détail)

5 Conclusion

Dans cette partie, nous avons montré brièvement la possibilité d'intégrer nos modèles dans une application de taille réelle.

Le projet IMAGE nous a permis de tester nos modèles dans une application. Néanmoins, comme elle devait être livrée pendant la durée du stage, tous nos modèles n'ont pas pu être implémentés. Nous avons malgré tout déjà pu voir l'intérêt de nos modèles en terme de réutilisabilité et de modifiabilité.

Enfin, différents tests d'utilisabilité de l'application Image ont été réalisés et ont montré un gain de temps et de confort pour l'utilisateur grâce aux nouvelles techniques d'interactions et à l'usage de la multimodalité.

Conclusion générale

Pouvoir intégrer des périphériques d'entrée-sortie "exotiques" avec des composants logiciels conventionnels, permettre des interactions concurrentes et parallèles, des comportements temporels et pouvoir fusionner les événements d'entrée pour construire une commande complexe, telles sont les spécificités de notre approche.

Les avantages de cet outil et de cette notation sont essentiellement la validation et la vérification de modèles formels. Ces modèles permettent d'être analysés par des outils mathématiques d'analyse de réseaux de Petri. De plus, la validation permet par des jeux de test de vérifier si les modèles correspondent à ce que nous voulions modéliser. Ces modèles seront particulièrement utiles pour la conception de systèmes interactifs critiques dans la partie spécification de l'interaction. Dans tout système, la fiabilité globale est la fiabilité du composant le plus faible. Il apparaît donc important d'avoir une spécification complète du système, des techniques d'interactions au noyau fonctionnel. L'objectif de ces modèles est aussi de permettre une évaluation des performances d'un système lié aux techniques d'interaction. Ceci peut également être réalisé au moyen de jeux de test.

Les autres avantages sont l'adaptabilité et la réutilisabilité. Nos modèles étant entièrement paramétrables, ils permettent une adaptabilité à la situation, à l'utilisateur ou à de nouveaux contextes. De plus, comme nous avons pu le voir dans les derniers chapitres de notre partie contribution, ces modèles sont facilement réutilisables dans d'autres applications ou pour d'autres périphériques d'entrée à quelques modifications près.

Il est évident qu'implémenter ces modèles dans l'architecture d'un logiciel serait lourd et cher mais la fiabilité est primordiale pour des systèmes critiques de types stations de contrôle, satellites, cockpit d'avions, médecine chirurgicale.

Bibliographie

[Accot96] Accot J., Chatty S., Palanque P., A formal description of low level interaction and its application to multimodal interactive systems. in: proceedings of the Eurographics workshop on Design, Specification and Verification of Interactive Systems'96, Springer Verlag; 1996.

[Ajmone95] Ajmone M., Balbo G., Conte C., Donatelli S., and Franceschinis G., Modelling With Generalisez Stochastic Petri Nets. 1995.

[Audibert06] Audibert L., UML 2.0 Diagramme d'états transitions, Octobre 2006, http://www-lipn.univ-paris13.fr/~audibert/

[Barboni06] Barboni E., Méthodes formelles pour les composants logiciels appliqués aux systèmes interactifs critiques, Université Toulouse III, Toulouse, 2006

[Bass91] Bass, L., Pellegrino R., Reed S., Seacord R., Sheppard R., Szezur M. R., The Arch model: Seeheim revisited. proceeding of the User Interface Developpers' workshop. 1991.

[Bastide90] Bastide R., Palanque P., Petri net objects for the design, validation and prototyping of user-driver, interfaces. 3rd IFIP conference on Human-Computer Interaction, Interact'90, North-Holland; 1990; 625-631.

[Bastide92] Bastide R., Objets Coopératifs : un formalisme pour la modélisation des systèmes concurrents, Toulouse: Université Toulouse III; 1992.

[Bastide96] Bastide R., Palanque P., Implementation techniques for Petri net based specifications of human computer dialogues. in: Vanderdonckt, Jean, Editor. 2nd workshop on Computer Aided Design of User Interfaces, CADUI'96. Presses Universitaires de Namur; 1996; 285-302.

[Beaudouin00] Beaudouin-Lafon M., Lassen H., The architecture and implementation of CPN2000, a post-WIMP graphical application. Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00), San Diego, CA, USA, 5-8 Novembre 2000. ACM Press, New York, NY, USA; 181-190.

[Bellik95] Bellik Y., Interfaces multimodales : concepts, modèles et architectures, Université de Paris-XI: 1995.

[Bier 93] Bier E. A., Stone M. C., Pier K., Buxton W., Derose T., Toolglass and Magic Lenses: The see-through interface, Computer Graphics, T. Kajiya ed.1993;27:73-80.

[Bolt80] Bolt R., Put That There : Voice and Gesture at the Graphics Interface, SIGGRAPH'80 Proceeding. Vol 14, Number 3: ACM Press; 1980: 262-270.

[Buxton90] Buxton W., A three-state model of graphical input. Proceedings of INTERACT'90, 1990.

[Caelen91] Caelen J., Coutaz J. Interaction Homme-Machine Multimodale : Problèmes Généraux, IHM'91, Dourdan, 1991.

[Coutaz96] Coutaz J., Nigay, L., Propriétés en interaction homme-machine. 1996.

[David92] David R., Hassane A., Du Grafcet aux réseaux de Petri, Editions Hermès Paris, 1992.

[Diaz01] Diaz M., Les réseaux de Petri : Modèles fondamentaux. Hermès Science Europe Ltd, Paris, 2001.

[Dragicevic04] Dragicevic P., Un modèle de configurations d'entrée pour des systèmes interactifs multi-dispositifs hautement configurables,. Thèse de doctorat, 2004.

[Dragicevic04b] Dragicevic P., Navarre D., Palanque P., Schyn A., Bastide R., Very-High-Fidelity Prototyping for both Presentation and Dialogue Parts of Multimodal Interactive Systems, EHCI-DSVIS'2004 , The 9th IFIP Working Conference on Engineering for Human-Computer Interaction Jointly with The 11th International Workshop on Design, Specification and Verification of Interactive Systems. Rémi Bastide, Philippe Palanque, Jörg Roth (Eds.), Springer; 2004; LNCS 3425:185-206.

[Fichefet] Fichefet J., Introduction aux réseaux de Petri, Support de Cours, Université de Namur, 1995

[Green85] Green M., Design Notations and User Interface Management Systems. In Günther E.Pfaff, editor, User Interface Management Systems – Proceedings of the Workshop on User Interface Management Systems held in Seeheim,FRG, November 1-3, 1983. Spriger-Verlag, 1985.

[Jensen96] Jensen K., Coloured Petri Nets. Basic Concepts, Analysis Methods and Practical Use. Springer-Verlag; 1996.

[Harel88] Harel D., On visual formalisms, Communications of the ACM, 31(I5), 1988, 514-530.

[Navarre03] Navarre D., Palanque P., Bastide R., A Tool-Supported Design Framework for Safety Critical Interactive Systems in Interacting with computers, Elsevier, Vol. 15/3, 2003, pp 309-32

[Navarre01] Navarre D., Contribution à l'ingénierie en Interaction Homme-Machine: Une technique de description formelle et un environnement pour une modélisation et une exploitation synergique des tâches et du système. Toulouse: 2001.

[Navarre00] Navarre D., Palanque P., Bastide R. et Ousmane S., Structuring interactive systems specifications for executability and prototypability. In DSV-IS, 2000, 97–119.

[Navarre05] Navarre D., Palanque P., Schyn A., Nedel L., Freitas C., Winckler M., A Formal Description of Multimodal Interaction Techniques for Immersive Virtual Reality Applications. Interact 2005. IFIP; 2005.

[Nigay94] Nigay L., Conception et Modélisation Logicielle des Systèmes Interactifs : Application aux Interfaces Multimodales. Université Joseph Fournier, Grenoble I: 1994.

[Palanque92] Palanque P., Modélisation par Objets Coopératifs Interactifs d'interfaces homme-machine dirigées par l'utilisateur. Université Toulouse I ed. Toulouse: 1992.

[Parnas69] Parnas D. L., On the Use of Transition Diagram in the Design of a User Interface for Interactive Computer System, Proceedings of the 24th ACM Conference, ACM Press, 1969.

[Petri62] Petri, C. Kommunikation mit Automaten". Technical University Darmstadt; 1962.

[Pfaff85] Pfaff G. E., User Interface Management Systems. Proceedings of the Seeheim Workshop. ed. Berlin : Springer Verlag; 1985.

[Schyn05] Schyn A,. Une approche fondée sur les modèles pour l'ingénierie des systèmes interactifs multimodaux. Université Toulouse III, Toulouse, 2005

[Sy01] Sy O., Spécification comportementale de composants CORBA. Université Toulouse I; 2001.

[Turk00] Turk M., Robertson G,. Perceptual user Interfaces. 2000;43: 3.

[Uims92] UIMS 92 The UIMS Workshop Tool Developers : A Metamodel for the Runtime Architecture of an Interactive System, SIGCHI Bulletin, 24, 1, p. 32-37, Janvier 1992

[Valk98] Valk, Rüdiger. "Petri nets as token objects: an introduction to elementary object nets". 19th international conference on application and theory of Petri nets, ICATPN'98, Springer, 1998.