UNIVERSITE CATHOLIQUE DE LOUVAIN

FACULTE DES SCIENCES APPLIQUEES

DÉPARTEMENT D'INGÉNIERIE INFORMATIQUE

Dégradation harmonieuse d'interfaces utilisateur

Promoteur : Pr. Jean Vanderdonckt

Mémoire présenté en vue de l'obtention
du grade de ***licencié*** **en** ***informatique***
par Benoît Collignon

Louvain-la-Neuve
Année académique 2003-2004

REMERCIEMENTS

Je tiens à remercier mon promoteur, le Professeur Jean Vanderdonckt, et Madame Murielle Florins, pour leur sympathie, l'aide et l'encadrement qu'ils m'ont apportés tout au long de la réalisation de ce mémoire.

Je remercie également Donatien Grolaux pour son aide précieuse dans la résolution des problèmes rencontrés dans le développement de l'application de dégradation harmonieuse.

Enfin, je tiens également à remercier toute ma famille, et plus spécialement mes parents, pour leur soutien moral durant mes études et leur encouragement dans la réalisation de ce travail.

TABLE DES MATIERES

Chapitre 1 : Introduction

1.1 Constatation générale : les nombreux contextes d'utilisation d'IHM

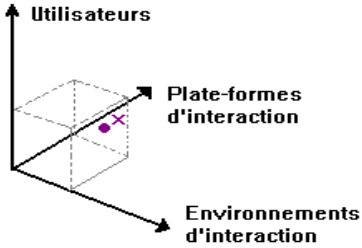
Le domaine de l'Interaction Homme-Machine (IHM) s'est toujours concentré sur la conception de systèmes interactifs utiles et utilisables en identifiant et en donnant la possibilité aux utilisateurs de réaliser des tâches interactives adaptées à leurs caractéristiques propres (expérience de la tâche/du système/des moyens d'interactions, motivation) et au contexte d'utilisation (plate-forme, environnement) dans lequel elles sont déclenchées.

Aujourd'hui plus particulièrement, avec l'expansion du phénomène multimédia et l'intégration d'applications Internet au sein de milieux socioculturels ou socioprofessionnels de plus en plus variés, l'importance de l'adaptation des systèmes interactifs aux utilisateurs s'est largement renforcée. Ainsi, si ces systèmes restent conçus de prime abord pour un certain profil d'utilisateurs, le raffinement opéré est devenu de plus en plus pointilleux et on assiste également à une augmentation des systèmes multi-profil. On peut citer, par exemple, la gestion différenciée des sites Internet faite par les agences bancaires en fonction du type d'utilisateur.

De même, les dispositifs physiques mis à disposition se diversifient de plus en plus. En plus de l'ordinateur personnel apparaissent d'autres appareils, tel la borne interactive, le PDA ou encore les téléphones portables, chacun avec une utilisation spécifique ou générale. Ainsi, l'espace des dispositifs d'interactions permet d'envisager la création d'application pour plusieures plate-formes physiques différentes, tout en profitant des avantages de chacune d'entre-elles (taille, performance, mobilité, …) et laissant donc le choix à l'utilisateur en fonction de ses attentes.

Par ailleurs, la multitude de dispositifs d'interaction engendre également une diversification des environnements d'interaction et, *in fine*, les frontières entre milieux socioculturels ou socioprofessionnels s'estompent car l'accès à l'information devient possible en tout lieu et à toute heure. Nous pourrions imaginer, à titre d'exemple, le cas du PDA, utilisable à titre professionnel (saisie d'informations pertinentes, aide-mémoire, …) ou à titre privé (système de récolte d'information automatisée en fonction du lieu dans lequel l'utilisateur se trouve (ex :sites touristiques)), à l'extérieur comme à domicile (commande de produits de grande consommation, consultation d'offres d'agences de voyage, …).

En résumé, les contextes d'utilisation pour un même système évoluent et changent en même temps que la diversité des utilisateurs s'accroît et que le nombre de dispositifs possible grandit. On assiste donc progressivement à la création d'une trame d'informations extrêmement flexible, ce qui demandera sans aucun doute la création de nouveaux outils de développement et une approche par processus plus spécialisée.



Trois dimensions susceptibles de faire varier le contexte :
*plate-formes : supports matériels/logiciels à l'interaction (ex : PC, PDA, GSM).
*environnements : milieux entourant l'utilisateur effectuant sa tâche (ex : l'atmosphère, le lieu, le jour, l'heure).
*utilisateurs : stéréotypes d'utilisateurs regroupés par tranche d'âge, connaissance du système/de la tâche, en fonction de la motivation, …
le triplet x correspond donc à un contexte d'utilisation spécifique.

Figure 1 - Les trois dimensions à l'origine des différents contexte d'interaction homme-machine

1.2 Le problème spécifique de l'IHM graphique multiplate-forme.

Le développement d'interfaces utilisateur multiplate-formes est en soi une tâche plutôt difficile car coûteuse et consommatrice de temps. Le temps passé pour la conception de l'interface utilisateur (en moyenne, 48% du code d'une application interactive correspondent à l'IHM et le temps moyen passé dans les différentes phases de conception et réalisation est estimé entre 40% et 50% *[Myers92-1])* et représente une grande partie dans le cycle de développement d'une application, notamment en terme de développement pur et de maintenance, même lorsqu'il s'agit d'applications de type « traditionnel » pour une seule plate-forme cible.

De plus, les concepteurs d'interfaces n'ont pas toujours la possibilité de prévoir sur quelles plate-formes les applications devront être utilisées dans le futur : les dispositifs physiques peuvent varier sensiblement, notamment en terme de taille d'écran, et on comprend dès lors mieux que programmer une interface par plate-forme ne soit pas un problème simple en soi, surtout si les changements (maintenance) deviennent sensiblement importants.

Les méthodes traditionnelles prônent le développement d'interfaces séparément par plate-forme, avec le risque encouru de ne plus respecter la cohérence et la consistance entre les différentes interfaces spécifiques. Aussi, ce mémoire nous permettra de mettre en pratique et d'évaluer une alternative au développement séparé d'interfaces spécifiques aux différentes plate-formes choisies : la « dégradation harmonieuse » d'interfaces, à savoir la conception d'une interface pour la plate-forme possédant la meilleure résolution d'écran et la réalisation d'adaptations automatiques de cette dernière pour les écrans plus petits par l'application de règles de transformation.

1.3 Motivations

L'avantage principal de cette approche réside sans aucun doute dans le principe d'adaptation, cela nous permettra de prévoir à l'avance différentes plate-formes physiques et d'anticiper des résolutions d'écran de futures plate-formes.

Enfin, et c'est ce qui rend probablement le travail plus original, le lecteur pourra se rendre compte dans l'état de l'art du nombre relativement limité de cas déjà consacré à la dégradation harmonieuse d'interface utilisateur. En général, les seules règles de transformation considérées dans les cas existants n'abordent qu'une seule facette du problème. Notre étude de cas tentera donc d'aborder plusieurs aspects différents de l'interface utilisateur en y intégrant des règles de transformation à différents niveaux d'abstraction.

1.4 Cas d'étude et Méthodologie suivie

Le présent travail comportera aussi bien une partie pratique qu'une partie théorique. Nous nous limiterons cependant à l'étude des interfaces graphiques à deux dimensions. Les interfaces tridimensionnelles, vocales, … ne seront pas abordées.

Dans la partie théorique, nous axerons notre recherche sur l'analyse d'un cas de dégradation harmonieuse *basé modèle* (cfr. infra) programmée en Oz *[Mozart]* et développée avec le toolkit QTk *[QTk]*. Nous verrons et appliquerons différentes techniques d'analyse et de modélisation de transitions entre toutes les fenêtres possibles de l'application choisie.

Pour ce faire, nous procéderons par étapes chronologiques :

- étude de notation suivie pour la transition entre fenêtres
- étude de modélisation de transitions de fenêtres : machines à états finis, …
- critique des modélisations retenues et sélection de celle qui offre le plus d'avantages

Une fois le choix de modélisation des transitions entre fenêtres déterminé, nous tenterons de fournir une ébauche d'un logiciel de génération automatique de modélisation de transitions de fenêtres (étude de règles de passage d'un type de modélisation à une autre, design graphique de l'application) pour des applications du même type que celle étudiée.

Préalablement à la partie pratique, nous entamerons une discussion sur la nature de l'application analysée dans la partie théorique. Celle-ci satisfait-elle à nos attentes du point de vue de la dégradation harmonieuse ? Dans quel cas, nous pourrions alors enrichir cette dernière de nouvelles règles de transformation et donc nous baser sur la même architecture logicielle pour notre application. Bien qu'intéressant sur de nombreux points, nous montrerons que l'approche suivie dans le cas d'étude analysé ne convient malheureusement pas pour atteindre nos objectifs, à savoir la transformation de dégradation harmonieuse appliquée *at runtime*.

L'objectif de la partie pratique sera de réaliser l'étude de certaines règles de dégradation d'une interface utilisateur et de vérifier leur degré d'efficacité dans le cadre d'une application en QTk. Pour ce faire, la démarche méthodologique suivie se divisera en plusieures étapes. En premier lieu, nous réaliserons une petite application de génération automatique d'interfaces utilisateur sur base d'une spécification d'IU donnée en amont.

Nous considérerons dans un premier temps dans notre définition de l'interface à construire les données manipulées (modèle du domaine (cfr. infra)). Ensuite, nous critiquerons l'approche suivie : les données suffisent-elles pour appliquer de futures transformations ?, la structure utilisée est-elle suffisamment riche et donc exploitable ?

Nous montrerons que non. Afin d'enrichir la spécification, nous considérerons plusieurs niveaux d'abstraction, ce qui nous permettra d'introduire le *framework* Cameleon : nous décrirons les quatre niveaux d'abstraction qu'il utilise et les différents modèles sous-jacents. Enfin, nous expliciterons les différentes règles de transformation susceptibles d'être appliquées à chaque niveau d'abstraction et celles que nous retiendrons dans notre application.

Ensuite, nous passerons à la pratique pure et dure, en décrivant l'architecture de notre application, les structures utilisées et nous exposerons enfin les résultats obtenus ainsi que les conclusions à retenir de ce travail.

Chapitre 2 : L'approche de dégradation harmonieuse d'IU

2.1 La dégradation harmonieuse et la propriété d'adaptation des IUs

La propriété d'adaptation des interfaces homme-machine est atteinte lorsque cette dernière possède la faculté de mimétisme comportemental vis-à-vis de son utilisateur (en fonction de ses caractéristiques ou de ses choix) ou vis-à-vis d'autres contraintes (physiques, environnementales).

Ainsi, une interface peut être qualifiée de :

- *Adaptable* : c'est l'utilisateur qui prend l'initiative de modifier l'interface (ex : l'utilisateur choisit une présentation alternative)
- *Adaptative* : c'est le système qui adapte l'interface en fonction des caractéristiques de l'utilisateur (ex : le système change la présentation pour un utilisateur souffrant de problèmes oculaires)
- *Flexible* : c'est le système qui adapte l'interface en fonction de faits contextuels autres qu'utilisateur, typiquement la plate-forme et l'environnement tout en préservant l'utilisabilité. *[Thevenin & Coutaz 99]*

Selon Dieterich et al. *[Dieterich et al. 1993]*, le processus d'adaptation peut être divisé en quatre étapes : l'*initiative* (c'est le système ou l'utilisateur qui suggère l'adaptation), la *proposition* (plusieurs alternatives sont émises), la *décision* (une alternative est choisie), l'*exécution* (l'alternative choisie est déclenchée et exécutée).

Cependant, même si l'adaptation est souvent *effective* lors de l'*exécution*, il n'y a pas qu'à ce moment-là qu'elle peut être réalisée. Les éventuels intervenants (concepteur, utilisateur, etc…) peuvent être mis en corrélation avec plusieurs moments clés de l'adaptation. Ainsi, on distingue trois grands moments d'adaptation: au développement, à l'exécution et à l'installation.

1. *Adaptation au développement* : L'adaptation est alors réalisée par les concepteurs et les codeurs qui ont prévu l'ensemble des cibles possibles. Par exemple les outils comme UIML *[Phanariou 2000]* ou AUIML permettent la production d'interfaces adaptées à différentes cibles, mais ces IHM ne peuvent pas s'adapter à l'exécution.

2. *Adaptation à l'exécution* : Comme son nom l'indique, l'IHM s'adapte ici à la volée.
   Ex : La boîte à outils MultimodalToolKit *[Crease et al. 2000]*

3. *Adaptation à l'installation du logiciel* : Le produit s'adapte et/ou est configuré pour correspondre à la cible donnée.

Ces trois instants d'adaptation nous conduisent à définir trois types d'IHM exécutables : les IHM précalculées, les IHM calculées dynamiquement et les IHM hybrides.

Une *IHM multicible précalculée* est le résultat d'une adaptation réalisée à la conception ou à l'installation. Les cibles possibles sont alors identifiées par avance et les IHM correspondantes sont créées et configurées en conséquence. Nous parlons donc d'*adaptation précalculée*.

Une IHM *multicible calculée dynamiquement* est élaborée en cours d'exécution dès qu'une condition d'adaptation se révèle vraie. On parle alors d'*adaptation dynamique*.

Une IHM *multicible hybride* relève d'IHM *précalculée* et d'IHM *calculée dynamiquement*. On parle d*'adaptation hybride*.

Dans notre cas, la dégradation harmonieuse se concentre sur les IHM *hybrides*.

Finalement, nous pouvons dire que la dégradation harmonieuse constitue une technique particulière où :

- la responsabilité de l'adaptation est assumée par le système ou le designer
- l'adaptation montre ses effets sur la présentation (indirectement sur le dialogue)
- l'adaptation peut se réaliser lors du design ou lors de l'exécution (redimensionnement des fenêtres)
- le développement commence avec la spécification d'une IU (flexible) pour la plate-forme qui a la meilleure résolution
- les propriétés d'utilisabilité du système sont atteintes en réalisant un compromis entre les différentes contraintes imposées par les plate-formes cibles (plus petite résolution, même résolution mais les objets manipulés sont différents (taille, …), widgets différents, en moindre quantité, ou obsolètes) et en minimisant les changements de présentation.

2.2 La Dégradation harmonieuse et le développement basé Modèle

L'approche basée modèle pour le développement d'interfaces utilisateurs a démarré dans les années 80. Celle-ci comporte essentiellement trois facettes :

1. les *modèles* qui reprennent les spécifications de l'interface utilisateur suivant des abstractions appropriées
2. les *méthodes* qui permettent de structurer la définition et l'utilisation des modèles sous-jacents par une approche divisée en différentes étapes
3. les *outils* qui serviront d'aide et de support à l'utilisation des *méthodes* pour la conception des différents *modèles*

Cette approche a été renforcée suite aux nombreux problèmes des outils de construction d'interface traditionnels, à savoir, les constructeurs d'interfaces (Interface Builders), les systèmes de gestion d'interface utilisateur (UIMS), les boîtes à outils (Toolkit libraries) et enfin, les environnements de développement d'interface (Design Exploration Tools) *[Myers92-2]*.

Pour évaluer ces outils, différents critères entrent en jeu : la facilité d'utilisation, les aspects de l'interface ciblés lors du développement, les parties du cycle de développement visées et enfin, la performance des applications construites. L'évaluation nous donne les résultats suivants :

	Ease of Use	**Classes of Designs**		**LifeCycle Support**	**Performance**
		Supported	**Not Supported**		
Interface Builders	Excellent for static facets Poor for dynamic ones	Menus, buttons, sliders, etc …	Dynamic aspects	Detailed design, implementation	Good
UIMSs	Fair	Dialogue control	Everything doable, but hard to do	Implementation	Moderate
Toolkit Libraries	Poor	Everything doable, but hard to do		Implementation	Excellent
Design Exploration Tools	Excellent	Presentation, limited behavior	Complex behaviors	Early and detailed design	Poor

Figure 2 – Résultats d'évaluation des outils de construction d'IU traditionnels

Au regard du tableau suivant, on constate une série d'inconvénients :

- les outils ne couvrent que des portions limitées de l'interface
- les phases du cycle du développement couvertes sont limitées (aucun outil ne propose de faire du design complet et de l'implémentation)
- les changements sont difficiles à propager
- les interfaces créées ne sont pas portables et très difficilement personnalisables

Ainsi, les buts de l'approche de développement d'interface basée modèle sont donc de proposer des environnements de développement étendus (s'attachant aussi bien aux phases de design que d'implémentation), d'améliorer la portabilité des interfaces, de s'appuyer de manière solide sur le cycle de développement, d'intégrer dans le développement des études d'*utilisabilité* (pour favoriser la facilité d'utilisation des interfaces par leurs utilisateurs) et enfin, spécifier l'UI à un haut niveau d'abstraction indépendamment de l'implémentation (approche déclarative plutôt que procédurale).

Les composants typiques d'un environnement nécessaire au développement d'interfaces fondées sur la génération orientée modèle (Model-Based Interface Developpment Environments ou MB-IDE) sont les suivants *[Szekely96]* :

1. Le composant Modèle

   Il représente l'information selon trois niveaux d'abstraction.
   Le plus haut niveau regroupe généralement le domaine et le modèle de tâche véhiculé par le système :

   • Modèle du Domaine: données et opérations réalisées dans le noyau fonctionnel.
   • Modèle de la Tâche: hiérarchie des tâches et sous-tâches ordonnancées.

Le niveau intermédiaire, appelé “abstract user interface spécification”, sert à spécifier :

- les tâches de bas niveau d’interaction, par exemple la navigation dans une liste déroulante
- les informations qui doivent être présentées, c’est-à-dire rendues observables (Presentation Units).

Le troisième niveau du modèle, appelé “concrete user interface specification”, spécifie le style d’affichage de l’unité de présentation. Les Modèles de ce niveau sont plus communément appelés modèle de la présentation (concernant les aspects statiques de l’IU) et modèle du dialogue (concernant les aspects dynamiques de l’IU).

Ces modèles ne sont cependant pas les seuls. D’autres modèles existent et pourraient être intégrés de manière à spécifier encore plus précisément l’IU : le modèle de l’utilisateur, de l’application, de la plate-forme, … Nous reviendrons plus en détails sur ces modèles par la suite.

2. Les utilitaires de spécification du modèle

Les utilitaires de spécification du modèle assistent le développeur dans la spécification des modèles. Leur intérêt principal est de cacher la partie langage de spécification en procurant une interface graphique pour la modélisation.

3. Les utilitaires d’aide et d’évaluation

En rassemblant une grande quantité de connaissances sur l’utilisateur, sur le noyau fonctionnel, le contexte d’exécution, … ces utilitaires d’aide et d’évaluation permettent d’évaluer la conception de manière efficace. Ainsi, ces utilitaires d’évaluation contiennent une base de connaissances sur les règles de conception (point de vue ergonomique, …).

4. Les utilitaires de construction automatique

Les utilitaires de construction automatique permettent, à partir des spécifications de modèles et de règles heuristiques de construire automatiquement l’IU au niveau du noyau fonctionnel de l’application interactive (la partie *restitution*). Selon les produits, on observe différents niveaux d’interaction avec le développeur : dans Mastermind *[Szekely95]*, par exemple, une partie de la présentation est faite par le développeur alors que dans TRIDENT *[Trident]* la génération est complètement automatisée.

5. Le Logiciel d’implémentation

Le logiciel d’implémentation traduit les spécifications concrètes de l’interface en langage directement exécutable ou utilisable par un constructeur d’interface. Par exemple, Mastermind génère du code source C++. D’autres, comme Genova *[Genova]*, permettent de générer du code source dans différents langages.

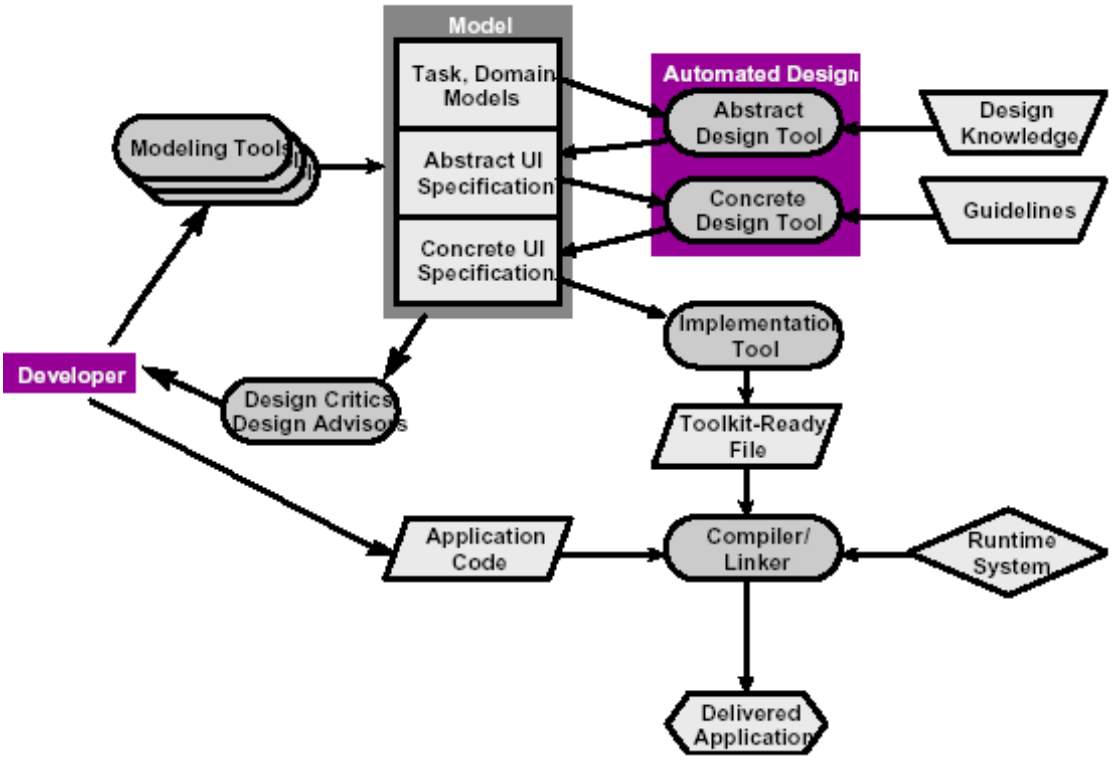


Figure 3 – Composants d'un MB-IDE

Dans ce cadre, la technique de dégradation harmonieuse permet de garder un certain compromis entre d'une part, la *spécification* et d'autres parts, la *construction automatique*. Elle permet, en effet, de *spécifier* l'interface de base (celle pour l'écran de résolution de départ) : les tâches, les interacteurs physiques (widgets), le placement de ces derniers, … De même, celle-ci permet de garder la consistance entre les différentes versions d'interface après une transformation *automatisée*, ce qui réduit donc la quantité de travail à fournir (une seule interface de base).

2.3 Du développement d'IHM multiplate-formes à la dég. harmonieuse : Etat de l'art

Plusieurs techniques pour le développement d'interfaces multiplate-formes peuvent être mises en évidence :

1. développer une interface spécifique pour chaque plate-forme
2. développer une seule interface exécutable sur plusieurs plate-formes
3. développer une description abstraite pour la partie commune à toutes les interfaces spécifiques aux plate-formes choisies et des descriptions supplémentaires pour les parties différentes (basée modèle ou non)

Comme nous l'avons vu dans l'introduction, la première technique se révèle particulièrement inefficace (vu le temps moyen pris pour une interface utilisateur dans le cadre du développement d'une application) et surtout, ne garantit aucune consistance entre les différentes IUs spécifiques.

La deuxième technique se base sur l'utilisation de clients génériques (browsers) et/ou de toolkits virtuels (ex : Java Swing, Tcl/Tk). Dans ce cas précis, l'adaptation n'est pas, en tant que tel, perceptible par l'utilisateur. A l'exception de quelques différences de présentation et de représentation de l'information entre les browsers et entre les toolkits, l'utilisateur ne remarque rien. Cependant, ce système n'offre pas de solution réellement satisfaisante en regard d'applications devant s'exécuter sur des dispositifs très différents en termes de capacités en entrée/sortie. Ex : le système XWeb *[Xweb]* produit des IUs pour différents dispositifs à partir d'une description *multi-modale* de l'interface utilisateur abstraite. Ce système s'utilise sur des serveurs XWEB spécifiques et via des browsers prédisposés pour les capacités des plate-formes spécifiques en communiquant suivant un protocole XTP approprié.

La troisième approche étend et complète le client générique de la deuxième technique (utilisation de documents XML transformés (WML, XHTML, …)). Les interfaces générées via cette approche deviennent alors consistantes du point de vue de l'information, même si la présentation peut changer sensiblement. Néanmoins, cette technique impose encore de réaliser un document XML par plate-forme cible (car le langage peut varier), ce qui est encore inefficace.

Les outils récents issus de la même veine utilisent dorénavant une approche basée modèle (cfr. supra). On peut citer par exemple ARTStudio, ou TERESA qui sont plus focalisés sur *génération automatisée*, ou encore LiquidUI, qui se concentre plus sur la *spécification* accrue des interfaces.

ARTStudio est un outil de génération utilisant différents modèles : la tâche (spécification d'un modèle de la tâche ConcurTaskTree (nous reviendrons plus en détails sur la modélisation CTT par la suite) avec certains paramètres additionnels), le domaine (spécification en UML), la plate-forme, les objets interactifs, l'interface abstraite (AUI) représentant le premier processus de réification des modèles de la tâche et du domaine à un haut niveau d'abstraction, et l'interface concrète (CUI).

TERESA, quant à lui, génère des interfaces multiplate-formes à partir d'un modèle de tâche CTT unique. Pour chaque tâche, les objets qui seront manipulés ainsi que la cible choisie seront mentionnés. Ensuite, ce modèle est passé sous différents filtres afin de produire un modèle de tâche spécifique à une plate-forme bien précise. Après quoi, ce modèle est transformé en AUI (un ensemble de présentations et leur comportement dynamique) suivant un algorithme particulier (Enable Task Sets algorithm), qui produit l'ensemble des tâches à accomplir en même temps par présentation. Ensuite, chaque présentation est définie en termes d'objets interactifs abstraits. Et pour terminer, chaque objet interactif abstrait est transposé en son homologue concret suivant la plate-forme désirée.

LiquidUI utilise le langage UIML. Une interface spécifiée en UIML inclut trois composants :

- le composant *logique* : décrivant la manière de communiquer avec l'application indépendamment des protocoles, noms de méthodes, …
- le composant de *présentation* : décrivant la manière de présenter l'IU indépendamment des widgets finaux et de leurs propriétés ainsi que la gestion d'évènements concrets.
- Le composant de l'*interface* : décrivant l'interaction entre l'utilisateur et l'application indépendamment de la plate-forme. Ce composant est subdivisé en quatre parties : la *structure*, le *style* (couleur, taille de la fonte utilisée, …), le *contenu* (texte, images,

vidéo, …) et finalement, le *comportement* (les règles qui le régissent). Ensuite, l'interface est spécifiée suivant le vocabulaire de la plate-forme spécifique. Ainsi, chaque spécification spécifique à la plate-forme peut être fournie par LiquidUI.

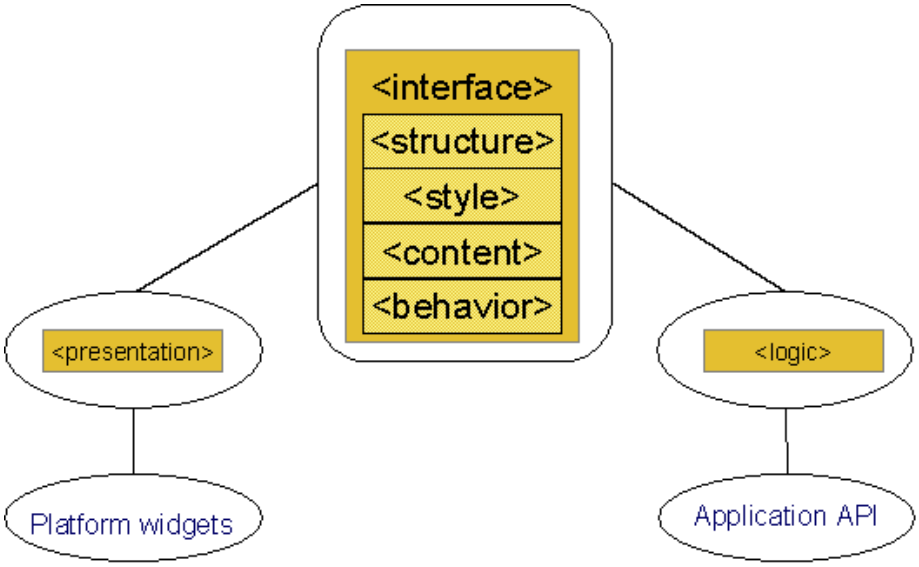


Figure 4 – Spécification d'une interface en UIML

Le gros désavantage de l'utilisation de UIML réside dans l'utilisation d'un vocabulaire spécifique à la plate-forme choisie, ce qui, en fin de compte, n'est pas très différent du développement d'une interface spécifique pour chaque plate-forme, même si le vocabulaire semble simple à apprendre.

D'autres techniques que l'approche orientée XML ont été étudiées. AUI *[Schneider 2002]* est un langage fonctionnel permettant, à l'instar des techniques précédentes, de décrire la fonctionnalité et l'apparence de l'interface. Bien que les langages fonctionnels ne constituent pas en soi un paradigme dominant, AUI propose une syntaxe claire et précise, en identifiant les opérations comme une transformation d'une structure vers une autre. Il se base sur un modèle comprenant : une interface utilisateur concrète, une interface utilisateur abstraite et un noyau fonctionnel. L'interface abstraite gère les événements et adapte la présentation de la CUI en fonction.

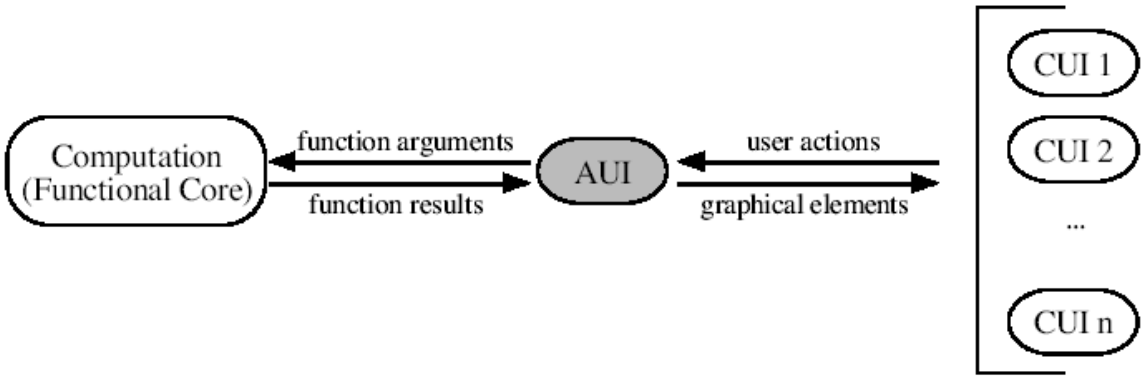


Figure 5 – Le modèle AUI

En réalité, le gros désavantage d'AUI réside dans la mauvaise séparation entre l'interface et le reste du système. En toute logique, toutes les opérations ne devraient pas être réalisées au sein d'AUI. Par contre, AUI permet de décrire les interfaces de manière abstraites sans qu'il n'y ait aucun lien avec la plate-forme choisie et vocabulaire spécifique à utiliser en fonction de cette dernière, ce qui permet d'avoir un plus large éventail d'interfaces générées que via UIML. Plusieurs extensions ont été réalisées : PAUI *[Watson 2003]*, une version procédurale d'AUI ; ou encore AUI3D *[Watson 2004]*, incluant la gestion des objets et scènes en trois dimensions.

Enfin, toujours dans le deuxième cas, les recherches en terme de spécification ont été étendues à l'utilisation de plusieurs niveaux d'abstraction plutôt qu'un seul. Ainsi, l'outil TIDE *[Ali2003]* utilise un modèle de tâche, une description de l'IU via UIML pour les parties communes aux familles de dispositifs physiques, une description de l'UI via UIML avec un vocabulaire spécifique à la plateforme et finalement, l'interface utilisateur finale. De plus, TIDE permet le *mapping* (mise en correspondance) entre modèles.

Une autre technique de transformation basée modèle a été approchée par *[Wong 2002]* : *The Scalable Web technique*. Cette technique permet de construire un modèle de présentation indépendant de la plate-forme lors du design. Ce modèle est réalisé pour la plate-forme ayant la résolution la plus élevée. Ensuite, la présentation peut être soumise à deux types d'adaptation : la pagination de grandes présentations vers de plus petites et plus simples et les transformations de contrôle. Des transformations liées à l'agencement des objets interactifs sont également réalisées. Ce type d'approche permet alors de garantir la continuité entre les différentes versions en même temps que l'adaptation d'interfaces pour différentes cibles physiques.

C'est cette dernière technique qui se rapproche le plus de la dégradation harmonieuse, car le calcul se fait suivant l'application de règles et de conditions de déclenchement. Cependant, l'ensemble de règles de transformation fourni dans la technique étudiée semble assez limité, il nous faudra donc l'étendre.

Chapitre 3 : Etude théorique : Description générale des moyens mis à disposition

Dans ce chapitre, nous allons aborder les outils et applications qui serviront à notre étude théorique. Ainsi, nous commencerons par une description du toolkit que nous utiliserons pour ce travail et ensuite, nous décrirons en détails le fonctionnement de l'application que nous avons choisi pour réaliser notre étude théorique.

3.1 QTk : Une approche intégrée basée modèle pour le développement d'IU

QTk est une boîte à outils développée au-dessus de Tcl/Tk permettant d'adopter une approche basée modèle pour le développement d'interfaces utilisateur. QTk se base sur une approche *descriptive* qui permet d'utiliser un style déclaratif pour les structures et objets interactifs concernant la *présentation* de l'interface et un style procédural pour les objets et *threads* concernant le partie *dialogue*.

Quatre grands avantages de QTk sont à mettre en évidence : l'unicité du langage, le faible coût de développement d'IU, l'intégration et étroite collaboration des outils (les outils de spécification, construction et d'exécution sont tous intégrés) et enfin, l'expressivité élevée qui en découle.

Ces quatre avantages sont rendus possibles grâce à l'intégration de QTk en tant que module au sein du système de programmation Mozart, sur lequel repose le langage Oz. Oz est un langage de programmation multiparadigme avec un appui de haut niveau sur les structures de données symboliques que sont les listes et les enregistrements (records).

Or, ces derniers conviennent très valablement à l'approche basée modèle pour le développement d'interfaces utilisateur : chacun des modèles choisis pourront, en effet, être représentés sur base de ces records.

Ainsi, grâce à son orientation modèle, QTk permet de palier les différents désavantages déjà exposés plus tôt dans ce travail par rapport aux méthodes et outils traditionnels de construction d'interfaces utilisateur. Voici un résumé des opportunités offertes par QTk :

- QTk peut utiliser un simple record pour représenter et construire une fenêtre de présentation
- Plusieurs types de fenêtres peuvent être construites avec différentes techniques de gestion de la géométrie et de l'espace, en incluant des structures avec grillage
- Tous les états initiaux des objets interactifs concrets (OIC) utilisés peuvent être définis
- Tous les OICs peuvent être contrôlés par l'application générée via des poignées (handles)
- Tous les types d'événements supportés par les OICs sont gérés
- Toute ou une partie des fenêtres construites peut être modifiée puis, affichée à l'écran durant l'exécution grâce à des objets interactifs appelés *pladeholders*.

Le dernier point est relativement intéressant dans notre cas, car nous voulons justement effectuer des modifications *at runtime* de nos présentations dans le cadre de la dégradation harmonieuse.

Voici l'exemple tiré du manuel d'utilisation de ce module pour le démontrer :

```
local
   P I1 I2 I3

   % initial place created on the fly
   Desc=placeholder(glue:nswe
                    td(handle:I1
                       label(text:"Hello"))
                    handle:P)
in
   {{QTk.build td(Desc)} show}

    % second place
   {P set(td(handle:I2 glue:nw
             label(text:"World")))}

    % third place
   {P set(td(handle:I3 glue:se
             button(text:"Close"
                    action:toplevel#close)))}

    % places back then hello widget
   {P set(I1)}
   {Delay 2000}
   {P set(empty)}
   {Delay 1000}
   {P set(I2)}
   {Delay 2000}
   {P set(I3)}
end
```

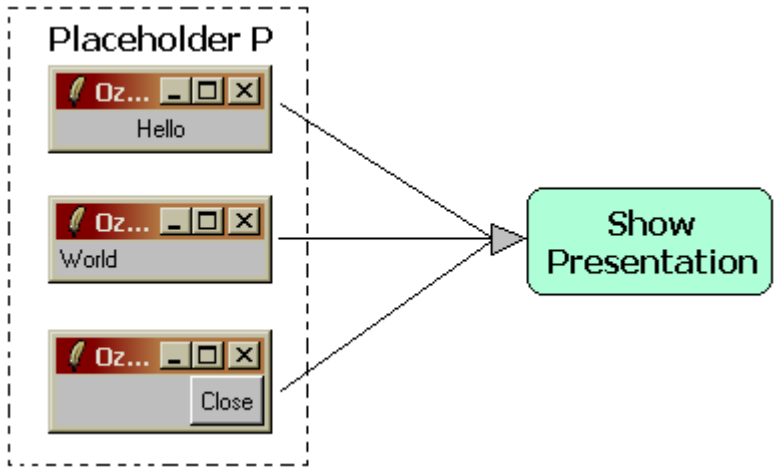


Figure 6 – Exemple d'utilisation d'un placeholder dans l'environnement QTk

3.2 FlexClock

FlexClock est l'application sur laquelle nous allons nous baser pour notre approche théorique. Celle-ci est particulièrement intéressante car elle introduit de manière simple la dégradation d'interfaces utilisateur en se basant sur le choix d'une présentation adéquate parmi un groupe prédéfini. Nous allons ainsi avoir la possibilité d'analyser les différentes transitions possibles entre les différentes présentations existantes. Voici la description de cette application.

L'application FlexClock *[Grolaux & al. 2002]* consiste en un programme de remplacement à l'utilitaire d'affichage de l'heure courante sur les systèmes Unix. Concrètement, FlexClock affiche l'heure et la date courante sous différents formats en fonction de la taille de la fenêtre.

Il existe 16 différentes vues possibles pour afficher l'heure, du simple écran type HH:MM à l'écran complexe composé d'une horloge analogique, d'une horloge digitale et d'un calendrier. Celles-ci existent simultanément à l'intérieur de l'application, et il n'y a donc aucune différence entre l'écran qui est visible pour l'utilisateur et les autres écrans possibles.

Chacune des vues (views) est caractérisée par trois points essentiels :

- le modèle de la présentation (la structure abstraite est définie via QTk)
- la manière de mettre à jour l'heure sur les écrans (chacun d'entre eux la définit via une procédure à un argument prenant le temps comme paramètre)
- la manière de choisir l'écran à afficher (chacune des 16 présentations est également définie avec leur taille (hauteur#largeur) respective en pixels. Ainsi, cette information permet de choisir les candidats potentiels pour affichage)

La *règle de sélection* utilise alors l'information sur la taille pour déterminer le meilleur candidat à afficher à l'écran. De manière plus formelle, si nous posons *wh* et *ww*, la hauteur et la largeur de la fenêtre courante (en pixels) et mh_i et mw_i , la hauteur et la largeur minimale de l'écran i (en pixels également), l'écran choisi sera alors celui :

- qui est affichable pour $mh_i > wh$ et $mw_i > ww$
- qui minimise $(wh - mh_i)^2 + (ww - mw_i)^2$ (si on trouve plusieurs candidats possibles, on en choisira un au hasard)

Remarquons que chaque écran définit en réalité une taille minimale mais pas maximale : chacune des vues peut être élargie (le replacement des objets s'effectuera de manière automatisée) jusqu'à arriver à une taille caractérisant une autre vue.

Le déclenchement de la procédure de sélection de l'écran se réalise de manière assez simple. En effet, QTk fournit un ensemble d'événements que nous pouvons affecter à la plupart des widgets. Parmi ceux-ci se trouve l'événement ***Configure*** qui se lance à chaque fois qu'une fenêtre est redimensionnée. Aussi, nous attribuons à cet événement la responsabilité de lancer la procédure *Place*, qui calculera la prochaine fenêtre à afficher (règle de sélection) et effectuera la mise à jour (réaction).

Voici ces éléments isolés dans le code de la procédure Place :

```
proc{Place}
     WW={QTk.wInfo width(P)}
     WH={QTk.wInfo height(P)}

     fun{Select Views Max#CH}
     case Views of W#H#Handle|R then This=(W-WW)*(W-WW)+(H-WH)*(H-WH) in
            if W<WW andthen H<WH andthen (Max==inf orelse This<Max) then
                   {Select R This#Handle}
            else
                   {Select R Max#CH}
            end
     else CH end
end

in

{P set ({Select Views inf#Views.1.3})}
```

Règle de Sélection

Réaction

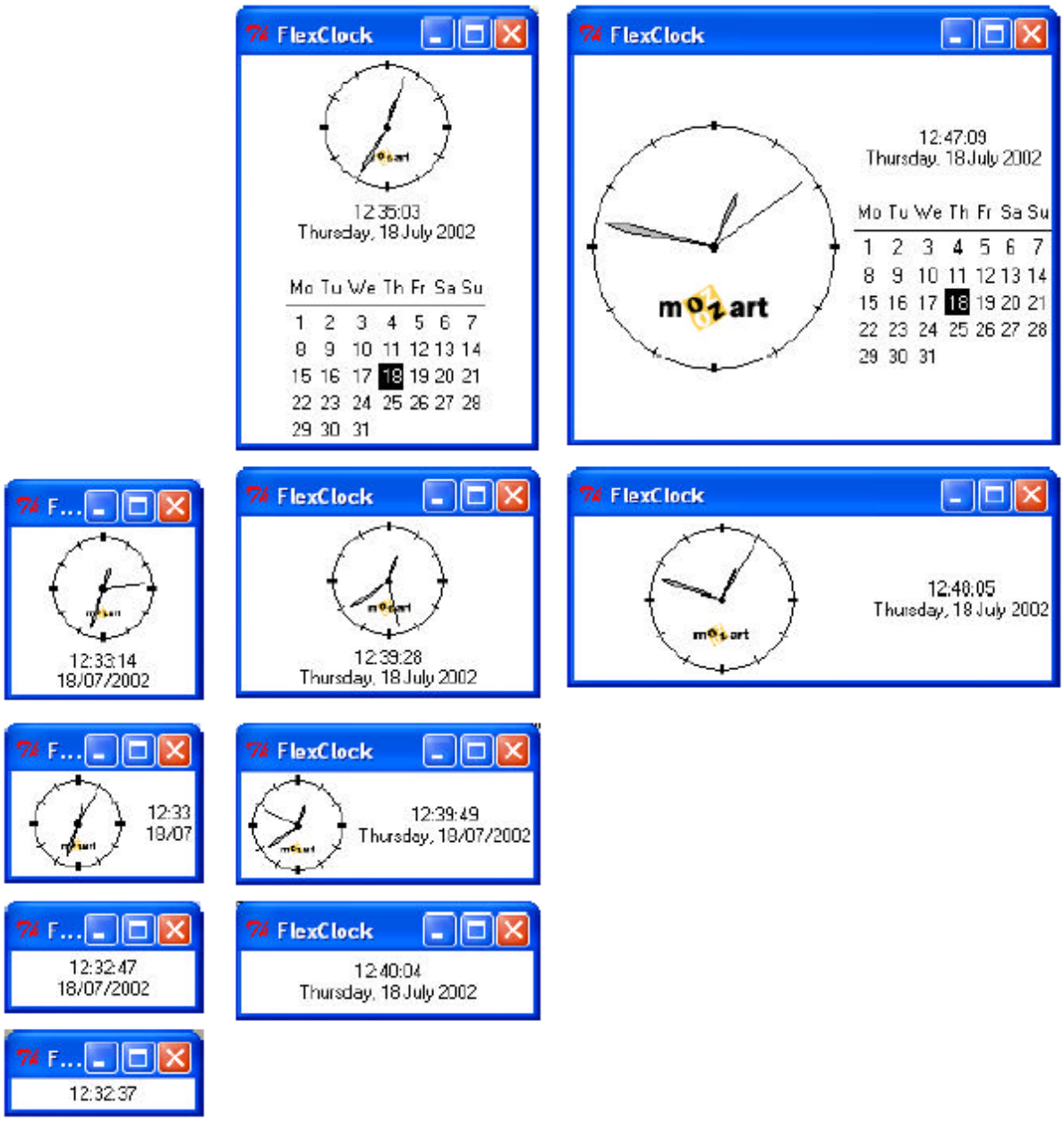
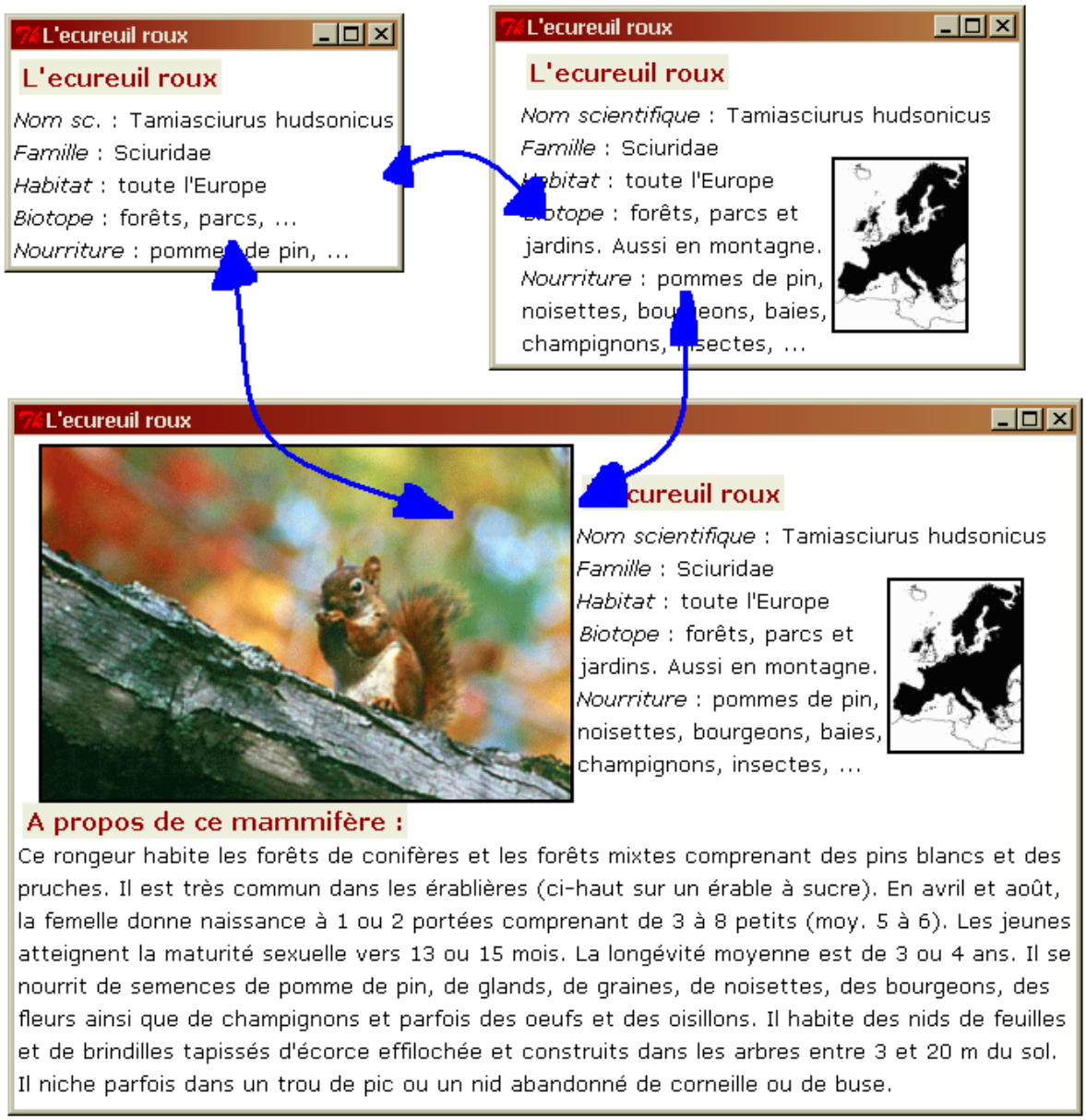


Figure 7 – Quelques-unes des vues possibles pour FlexClock

Le grand avantage de cette application réside dans le fait que l'activité liée à la sélection de la fenêtre à afficher et à la mise à jour de l'écran soit entièrement indépendante de la définition des différentes vues. Ainsi, on peut facilement concevoir une application du même type en reprenant le code lié au traitement et en recréant/spécifiant d'autres fenêtres en QTk.

En voici un exemple avec l'adaptation à une petite application éducative traitant des caractéristiques d'un petit animal de nos régions (le code se trouve dans les annexes):



Chapitre 4 : Modélisation de transitions de fenêtres

Dans cette partie, nous allons analyser les transitions possibles entre toutes les fenêtres affichables de FlexClock, trouver une notation adéquate pour modéliser ces même transitions, et enfin, réaliser l'ébauche d'une future application générant automatiquement le code et les modèles de transitions de fenêtres d'une application similaire à FlexClock.

4.1 Etude de notation choisie pour la transition de fenêtre.

Comme nous l'avons vu précédemment, le design d'une interface utilisateur basé modèle requiert et implique la création de modèles formels et déclaratifs décrivant l'apparence et la dynamique de l'IU. Ainsi, nous avions isolé des modèles **abstraits** et des modèles **concrets**, parmi lesquels notamment :

Partie abstraite

- le modèle des tâches, décrivant les buts que l'utilisateur espère accomplir et les actions à réaliser pour les atteindre
- le modèle du domaine, décrivant les objets et données que l'utilisateur manipulera
- le modèle de l'utilisateur, décrivant les propriétés ou caractéristiques du stéréotype d'utilisateurs visé, comme le niveau d'expérience, la motivation, …

Partie concrète

- le modèle de la présentation décrivant l'apparence visuelle de l'interface
- le modèle du dialogue, montrant les mécanismes et techniques d'interaction entre l'utilisateur et l'interface

Or, le design basé modèle se concentre aussi sur la mise en évidence de *mappings* (associations) entre différents modèles. Par exemple, lorsque nous élaborons des règles de sélection de widgets en fonction des données et caractéristiques du domaine d'application, nous réalisons en fait un *mapping domaine-présentation*.

Dans le cas qui nous intéresse, nous nous focaliserons donc sur un mapping tâche-présentation-dialogue puisque le but recherché est de trouver une notation adéquate pour modéliser les transitions entre présentations qui sont effectuées lorsque l'utilisateur redimensionne celle affichée.

Une *transition de fenêtre* peut se définir comme tout changement dans l'ensemble des fenêtres visibles (exemples : une fenêtre précédemment minimisée peut être ensuite maximisée ; une fenêtre disposée à l'avant de l'écran peut être mise en *background*).

Pour cela, nous allons nous baser sur la notation introduite par *[Vanderdonckt & al. 2003]*. Nous ne nous limiterons cependant qu'à ce qui nous intéresse ici, à savoir les transitions possibles suite à un redimensionnement.

Soit : WS, la fenêtre source (c'est-à-dire la fenêtre initiale avant la transition)
WT, la fenêtre destination (la fenêtre résultant de la transition)

Un événement généré par l'utilisateur ou le système sur une fenêtre source WS créant une transition de type A vers la fenêtre destination WT se note de la manière suivante :

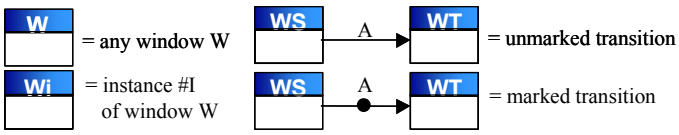


Figure 8 – Notation suivie pour les transitions de fenêtres

Chaque fenêtre est représentée par une icône et une transition de type A est indiquée par une flèche avec libellé. La transition *marquée* indique que l'utilisateur aura encore la possibilité de travailler avec WS une fois cette transition effectuée.

Plusieurs opérations peuvent être réalisées sur les fenêtres source et destination lorsqu'une transition a lieu. Toute opération réalisée sur la fenêtre source, respectivement la fenêtre destination, se note par une icône située du côté gauche, respectivement du côté droit de la flèche de transition.

Icon	Name	Operation definition
	Opening operations	open the window ={any state → max, title, min, tiling, normal overlap, system overlap, user overlap}
	Closing operations	close the window = { any state → close }
	Reducing operations	reduces the window = {max→ titling, min, tiling, normal/system/user overlap; titling → min; tiling → titling, min; normal/ system/user overlap → titling, min}
	Restoring operations	Restores the window = {min → titling, tiling, max, normal/ system/user overlap; titling → tiling, max, normal/system/user overlap; tiling → max, normal/ system/user overlap; normal/ system/user overlapping → max}

Figure 9 – Notation suivie pour les opérations génériques applicables aux fenêtres

Cette notation graphique nous permet d'étudier plusieures combinaisons d'opérations et donc de transitions. Voici quelques exemples :

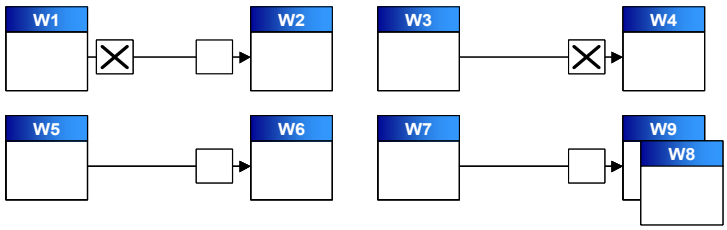


Figure 10 – Exemples de transitions de fenêtres

Dans le cas supérieur gauche de la figure 10, la transition ferme la fenêtre W1 et ouvre la fenêtre W2. Dans le cas supérieur droit, la transition ferme la fenêtre W4 mais laisse W5 dans son état initial. Le cas inférieur droit, la transition représente d'une double ouverture.

Lorsque aucune icône n'est spécifiée à la source ou à la destination, nous dirons que ces deux fenêtres gardent le même état initial, si ce n'est que le centre d'intérêt se portera sur la destination.

D'autres opérations, notamment sur la présentation d'une fenêtre (ces opérations déclenchant par après d'autres actions : lancement d'une fonction sémantique, affichage d'une autre fenêtre), sont possibles :

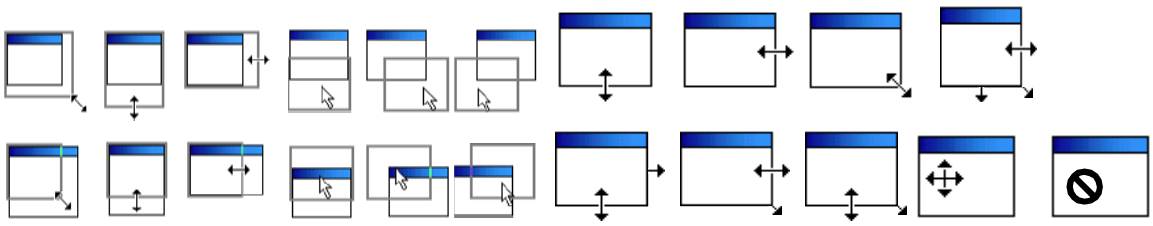


Figure 11 – Opérations permises ou non sur les fenêtres

Ce seront plus particulièrement sur celles-ci que nous nous baserons pour notre notation finale.

4.2 Modélisation des transitions de fenêtres par machine à états finis.

La notation graphique qui vient d'être introduite va s'avérer très utile par la suite. Cependant, il est nécessaire de suivre une logique avant de modéliser tout diagramme ou schéma représentant nos transitions entre fenêtres. Aussi, il peut être utile de se baser sur les définitions formelles déjà introduites sur les diagrammes d'état-transition, qui représentent des machines à états finis.

4.2.1 Notre notation finale

Quelques définitions

Un diagramme d'état-transition consiste en un ensemble de cercles représentant les états et de flèches dirigées représentant les transitions entre les états. Une ou plusieurs actions (outputs) peuvent être associées à chaque transition. Ce diagramme représente une machine à états finis.

Une machine à états finis représente une machine abstraite consistant en un ensemble d'états (incluant l'état initial), un ensemble d'événements en entrée et en sortie, et une fonction de transition d'états. La fonction prend l'état courant ainsi qu'un événement d'entrée et renvoie le nouvel ensemble d'événements possibles et le prochain état. Certains états sont qualifiés d'états *terminaux* (une fois ceux-ci atteints, il n'est plus possible d'appliquer la fonction de transition. (à moins que celle-ci ne mène à un état déjà visité mais dans ce cas, nous ne pouvons plus parler de machine à états finis)). La machine à états peut également être vue comme une fonction qui lie un séquence ordonnée d'événements en entrée avec une séquence correspondante d'(ensemble d') événements en sortie.

Une machine à états finis peut être *déterministe* ou *non-déterministe*. Dans le premier cas, le prochain état est toujours déterminé par un seul événement en entrée. Tandis que dans l'autre cas, le prochain état ne dépend pas uniquement que de l'événement courant reçu en entrée, mais d'un nombre arbitraire d'événements consécutifs en entrée. Dans ce cas, il ne sera pas possible de déterminer dans quel état la machine se trouve si aucun de des événements n'est atteint.

Les machines de Mealy *[Ullman 79]*, *[Salomaa 73]* sont des machines à états finis agissant comme des traducteurs, prenant une chaîne de caractère issue d'un alphabet d'entrée et produisant une chaîne de caractères de longueur égale en utilisant un alphabet de sortie.

Formellement, une machine de Mealy est un sextuple $M = (Q, \Sigma, \Gamma, \delta, \lambda, q_I)$ où :

- $Q = \{q_1, q_2, q_3, \ldots, q_{|Q|}\}$ est un ensemble fini d'*états* ;
- $\Sigma = \{\sigma_1, \sigma_2, \sigma_3, \ldots, \sigma_{|\Sigma|}\}$ est un *alphabet fini en entrée* ;
- $\Gamma = \{\gamma_1, \gamma_2, \gamma_3, \ldots, \gamma_{|\Gamma|}\}$ est un *alphabet fini en sortie* ;
- $\delta : Q \times \Sigma \rightarrow Q$ est la *fonction de transition*, telle qu'une machine dans un état q_j, transite vers l'état $\delta(q_j, \sigma_k) \in Q$ après avoir lu le symbole σ_k ;
- $\lambda : Q \times \Sigma \rightarrow \Gamma$ est la *fonction de sortie*, telle qu'une machine dans un état q_j, écrit le symbole $\lambda(q_j, \sigma_k) \in \Gamma$ après avoir lu le symbole σ_k ;
- q_I est l'*état initial* dans lequel la machine se trouve avant que le premier symbole de la chaîne de caractères soit traitée.

Exemple : Soit la machine $M = (Q, \Sigma, \Gamma, \delta, \lambda, q_I)$ telle que :

- $Q = \{q_1, q_2\}$, $q_I = q_1$
- $\Sigma = \{0,1\}$
- $\Gamma = \{E,0\}$
- $\delta(q_1,0) = q_1$; $\delta(q_1,1) = q_2$; $\delta(q_2,0) = q_2$; $\delta(q_2,1) = q_1$
- $\lambda(q_1,0) = E$; $\lambda(q_2,0) = 0$; $\lambda(q_2,1) = E$

Cette machine peut être représentée de manière graphique, générant en sortie E si le nombre de 1 lus est pair et 0 s'il est impair ; par exemple, la traduction de 11100101 donne 0E000EE0.

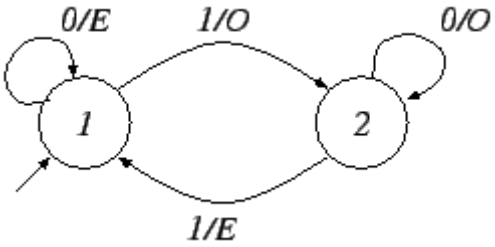
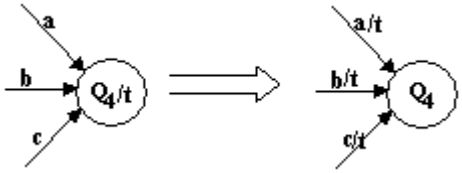


Figure 12 – Machine de Mealy : Exemple

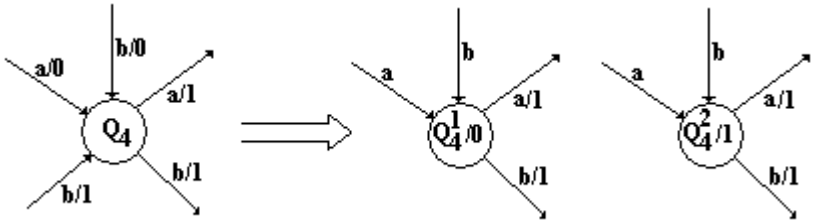
Les machines de Moore peuvent être définies de manière similaire (sextuple), avec la différence que les symboles sont générés en sortie après que la transition vers un nouvel état soit réalisée, et le symbole de sortie dépend uniquement de l'état qui vient d'être atteint par $\lambda : Q \rightarrow \Gamma$. Ainsi, l'ensemble des traductions qui peuvent être réalisées par machine de Mealy et l'ensemble des traductions qui peuvent être réalisées par machine de Moore sont identiques. En effet, pour toute machine de Mealy, il est possible de construire une machine de Moore équivalente et vice-versa.

Théorème d'équivalence des machines : Pour toute machine de Moore, il existe une machine de Mealy telle que les deux machines sont équivalentes. De même, pour toute machine de Mealy, il existe une machine de Moore telle que les deux machines sont équivalentes. La preuve est faite par construction en considérant chacune des parties séparément.

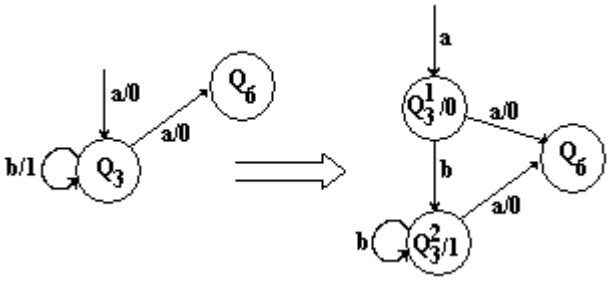
Moore ⇒ Mealy



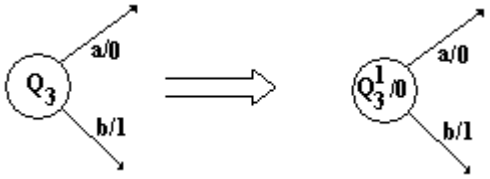
Mealy ⇒ Moore



On réalise une copie de l'état pour chaque lettre de Γ qui étiquette une flèche entrant.



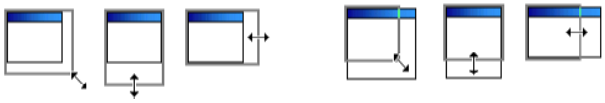
Les boucles et les flèches sortant doivent également être considérées.



S'il n'y a pas de flèche qui entre, on choisit n'importe quelle lettre de l'alphabet Γ.
Et enfin, on choisit n'importe quelle copie de l'état initial comme nouveau point de départ.

Adaptation au cas des transitions de fenêtres

En combinant les définitions qui viennent d'être données aux notations suivies dans le point 3.1, nous utiliserons donc les icônes de fenêtres pour représenter nos états, les opérations de redimensionnement pour représenter les différents événements, et les fonctions de transition et de sortie seront représentée par les fonctions de passage de fenêtre à fenêtre. Les cas considérés se baseront sur les six opérations *symétriques* suivantes :



Exemple simple

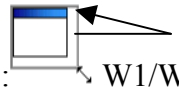
Reprenons l'exemple similaire à FlexClock que nous avons développé en QTk (cfr. page 21). Nous allons exposer les machines de Mealy et Moore pour cette application en ne considérant que l'opération d'agrandissement vertical et horizontal.

Machine de Mealy

Soit : Q = {  } ;
∑ = Γ = {W1 ; W2 ; W3} (W1 indique la fenêtre la plus petite et W3, la plus grande) ;

δ( ,Wi) = 

λ ( ,W1) = W2 ; λ ( ,W2) = W3

On obtient la machine suivante :  W1/W2 ; W2/W3

Machine de Moore

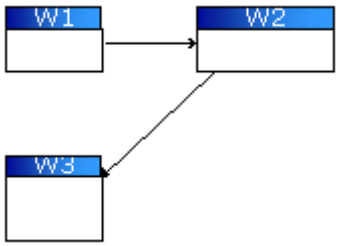
Soit : Q = {W1 ; W2 ; W3}

∑ = Γ = {  }

δ(Wi, ) = Wi

λ (W1, ) = W2 ; λ (W2, ) = W3

On obtient la machine suivante:



4.2.2 Résultats obtenus

Remarque préalable : Il faut mentionner que chaque fenêtre n'est pas réellement « fermée » en tant que tel à chaque transition : dans FlexClock, tout l'environnement est observable, le *conteneur* de chaque fenêtre peut changer mais l'ancien contenu pourra à nouveau être accessible.

4.2.2.1 Modélisation par machine de Moore

Voici quelques-uns des résultats obtenus pour la modélisation des transitions de fenêtres de FlexClock par machine de Moore. Les autres se trouvent en annexes. Il est à noter que chacune des transitions n'a été représentée que par une simple flèche pour réaliser un gain d'espace. La véritable transition se trouve en légende de chaque machine.

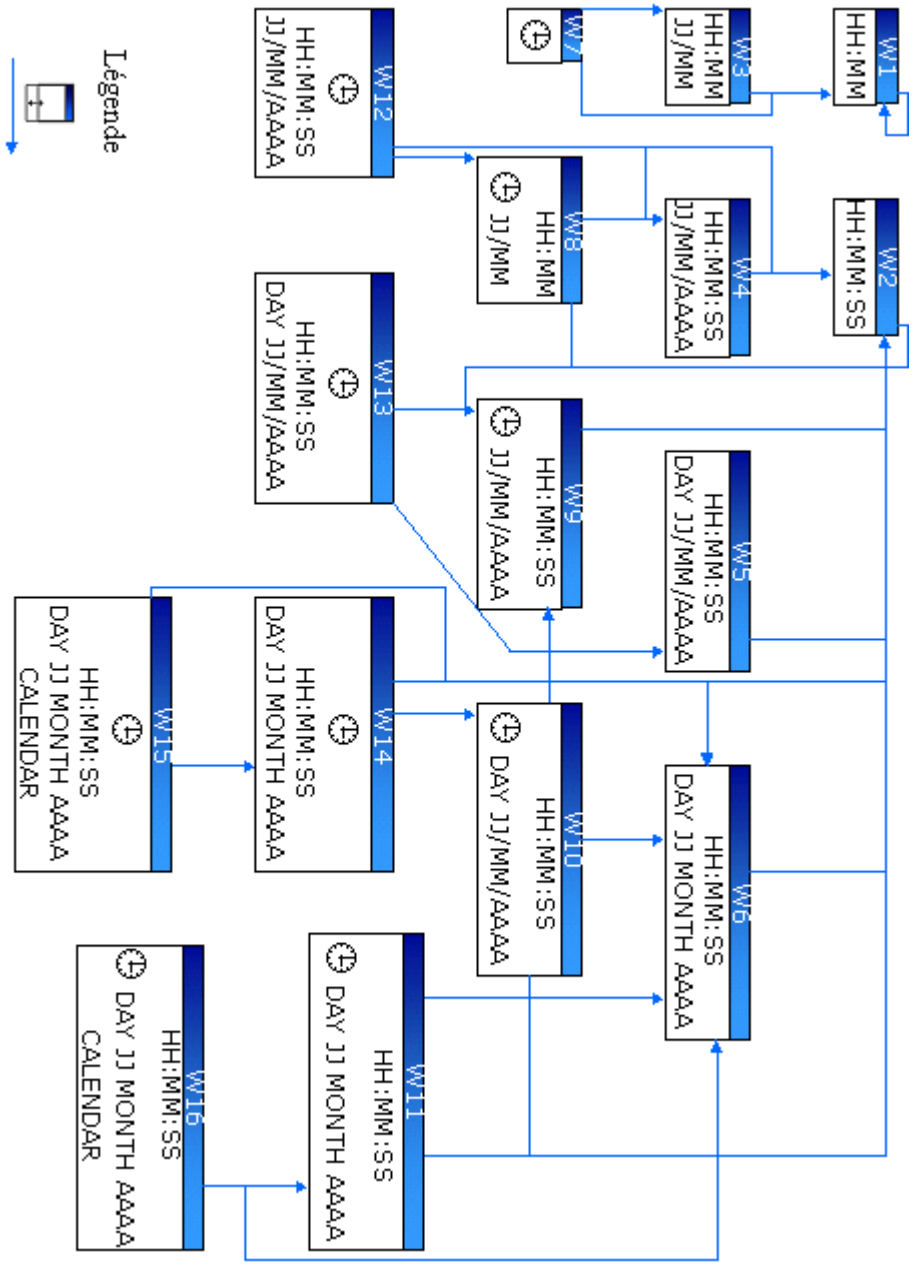


Figure 13 – FlexClock : Modélisation en machines de Moore – Quelques résultats

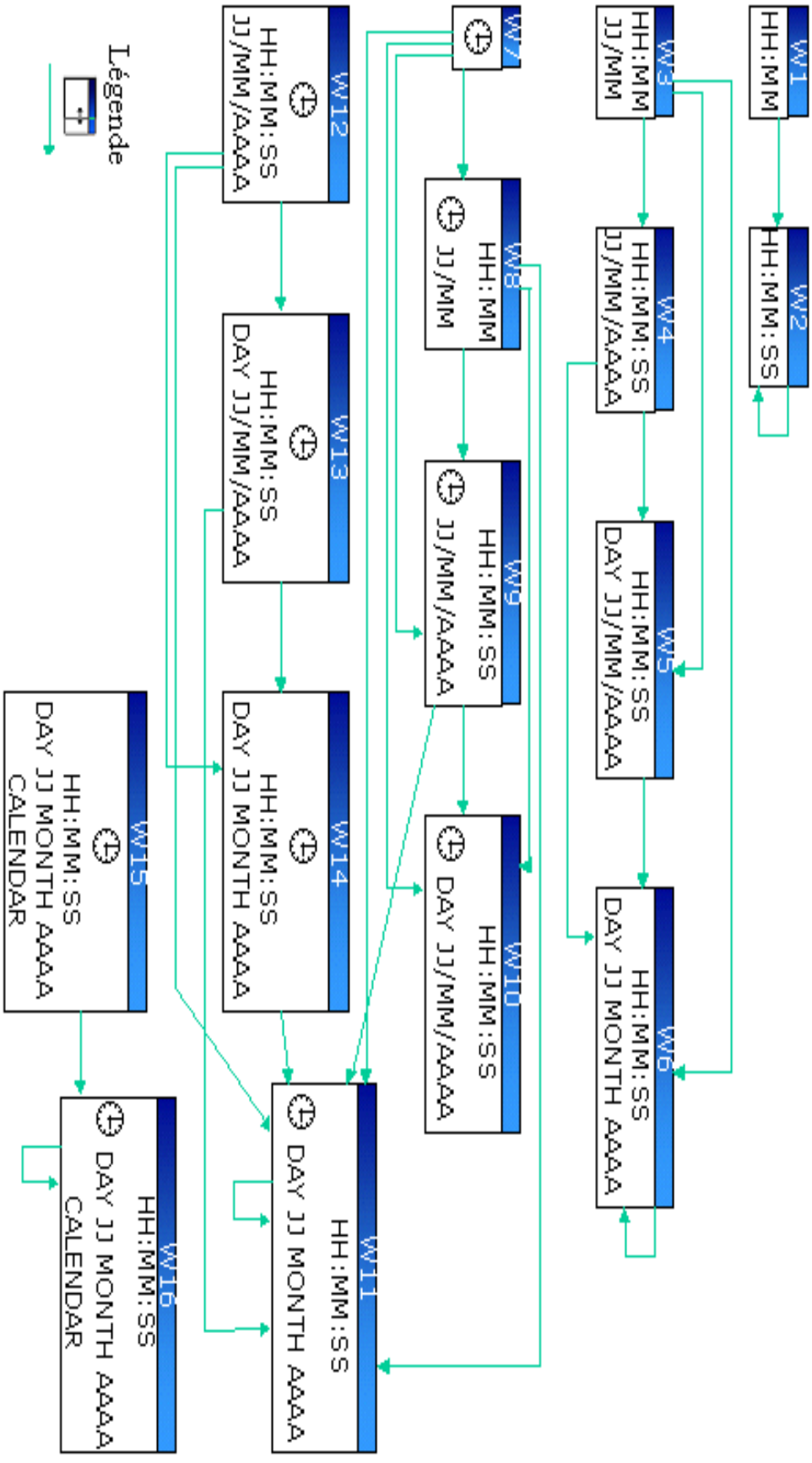


Figure 14 – FlexClock : Modélisation en machines de Moore – Quelques résultats

Voici toutes les transitions réunies sur un seul graphique. On notera l'illisibilité du graphique au vu du nombre très important de transitions possibles.

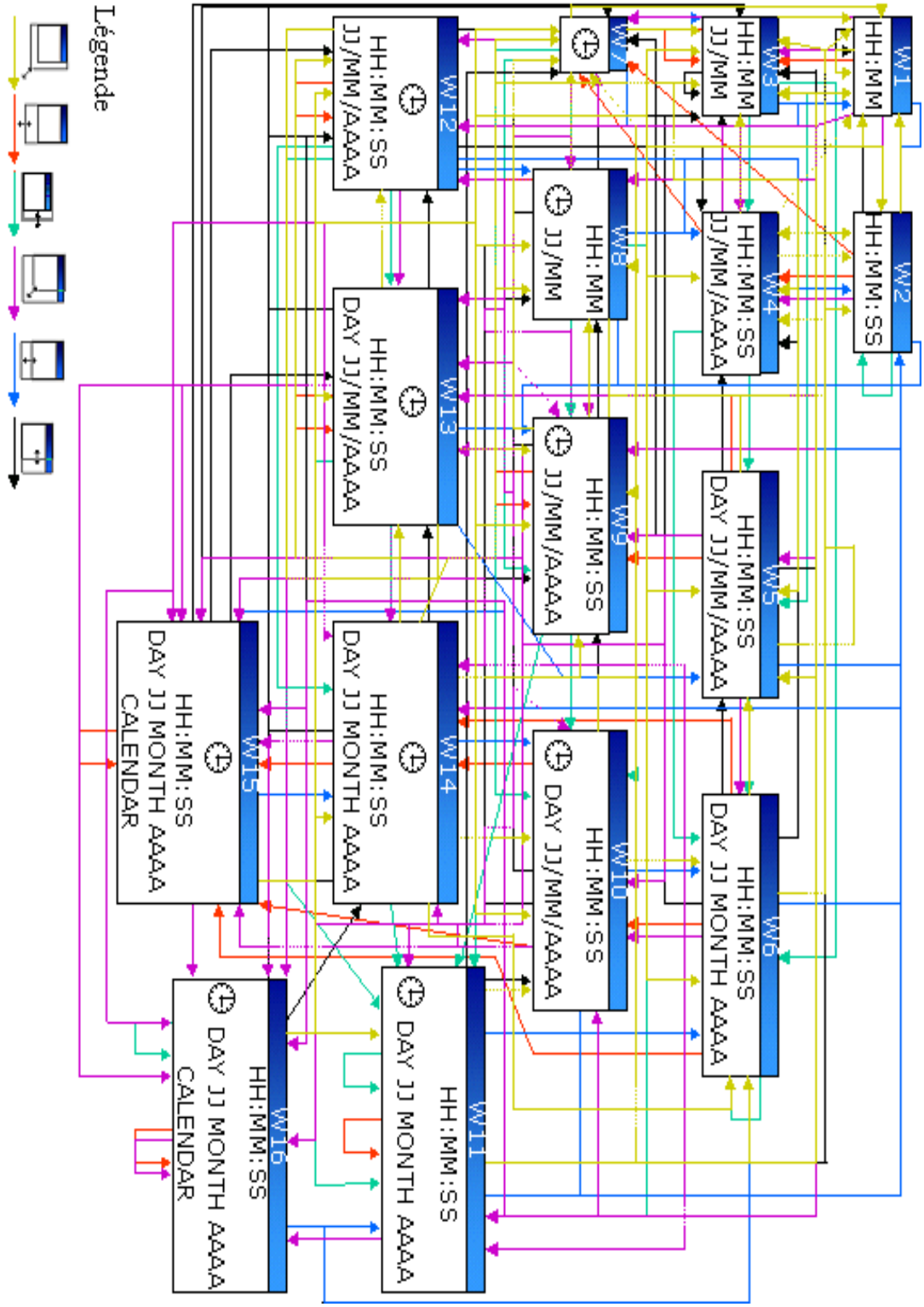


Figure 15 : FlexClock : Modélisation en Machine de Moore (toutes les transitions)

4.2.2.2 Modélisation par machine de Mealy

Voici l'ensemble des résultats obtenus pour la modélisation en machine de Mealy. Il est à noter que toutes les transitions n'ont pas été indiquées sur le graphique (figure 16), également dans un souci de clarté. Ainsi, les ensembles de flèches numérotés indiquent le nombre de possibilités pour les transitions après application des opérations choisies. Le chiffres situées à gauche indique le nombre de transitions possibles avant factorisation à gauche et/ou à droite (fusion des transitions ayant une même fenêtre source ou conduisant au même résultat), ceux à droite indiquent le nombre de transitions restant après factorisation.

L'ensemble des transitions possibles est indiqué de manière plus précise sur la page suivante.

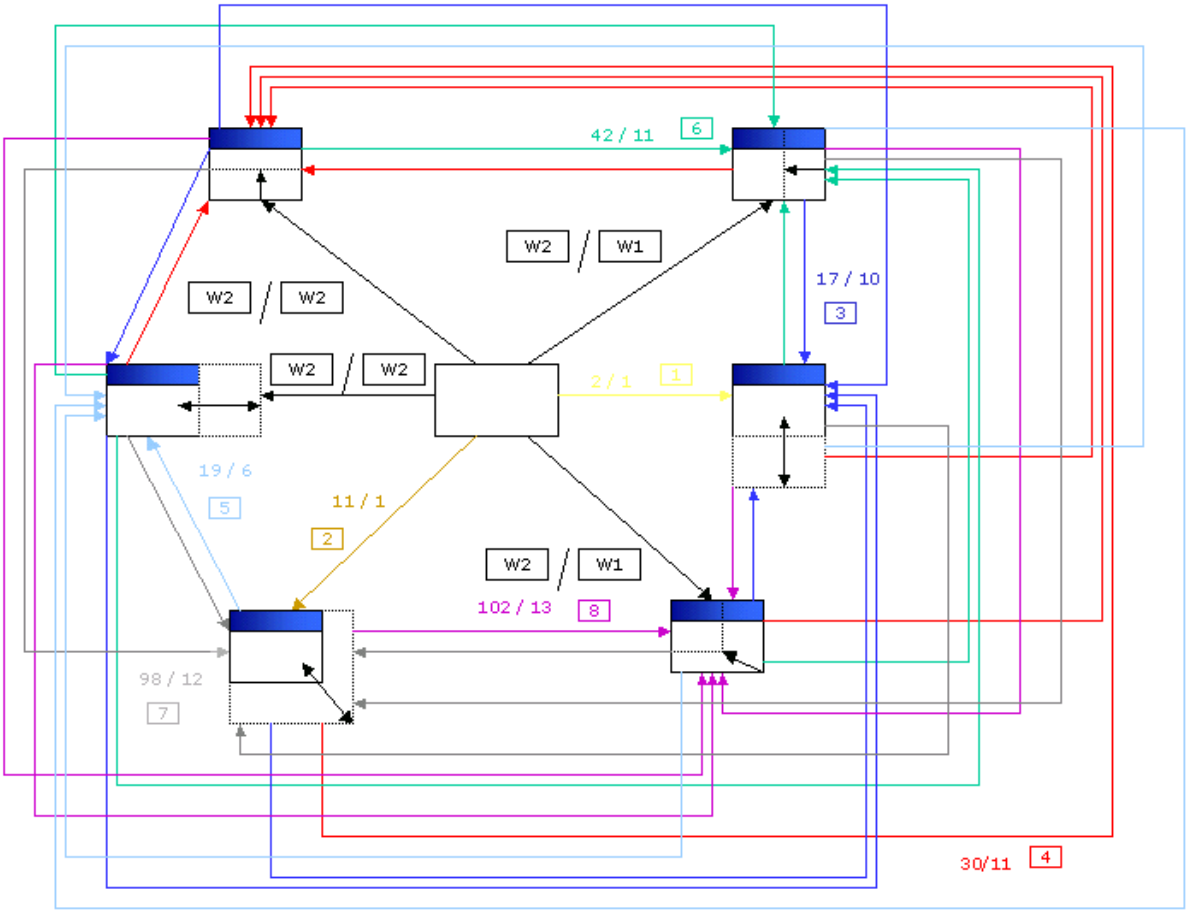


Figure 16 – FlexClock : Modélisation par Machine de Mealy

FlexClock – Description des transitions par machine de Mealy

Remarques préliminaires :

- la partie gauche de chaque transition (avant '/') représente ce qui est connu en entrée (la fenêtre courante) et la partie droite, ce qui est généré en sortie (la future fenêtre).
- ALL représente l'ensemble complet de toutes les fenêtres qu'il est possible d'obtenir après une transition quelconque.
- Le point (A) bien que, décrivant normalement les transitions non factorisées, intègre déjà les factorisations pour les transitions ayant la même fenêtre source (dans un souci de concision). Pour avoir l'ensemble des transitions non factorisées, il suffit au lecteur de séparer chacune d'entre-elles en considérant chaque point virgule comme séparateur.

(A) Description des transitions avant factorisation

1. *Agrandissement vertical de la fenêtre après ouverture.*

W2 / W4 ; W7

2. *Agrandissement vertical et horizontal de la fenêtre après ouverture.*

W2 / W5 ; W6 ; W8 ; W9 ; W10 ; W11 ; W12 ; W13 ; W14 ; W15 ; W16

3. *Agrandissement vertical.*

W1 / W3
W2 / W4 ; W7
W3 / W3
W4 / W7
W5 / W9 ; W13
W6 / W10 ; W14 ; W15
W8 / W12
W10 / W14 ; W15
W12 / W12
W13 / W13
W14 / W15
W15 / W15
W16 / W16

4. *Rétrécissement vertical.*

W1 / W1
W2 / W2
W3 / W3
W4 / W2
W5 / W2
W6 / W2
W7 / W3 ; W1
W8 / W4 ; W2
W10 / W6 ; W2
W11 / W6 ; W2
W12 / W8 ; W4 ; W2
W13 / W9 ; W5 ; W2
W14 / W10 ; W6 ; W2
W15 / W14 ; W10 ; W6 ; W2
W16 / W11 ; W6 ; W2

5. *Agrandissement horizontal.*

W1 / W2
W2 / W2
W3 / W4 ; W5 ; W6
W4 / W5 ; W6
W7 / W8 ; W9 ; W10 ; W11
W12 / W11 ; W13 ; W14
W13 / W11 ; W14
W14 / W11
W15 / W16
W16 / W16

6. *Rétrécissement horizontal.*

W1 / W1
W2 / W1
W3 / W3
W5 / W4 ; W3
W6 / W5 ; W4 ; W3
W7 / W3
W8 / W7 ; W3
W9 / W8 ; W7 ; W3
W10 / W9 ; W8 ; W7 ; W3
W11 / W10 ; W9 ; W8; W7 ; W3
W12 / W7 ; W3
W13 / W12 ; W7 ; W3
W14 / W13 ; W12 ; W7 ; W3
W15 / W13 ; W12 ; W7 ; W3
W16 / W14 ; W13 ; W12 ; W7 ; W3

7. *Agrandissement vertical et horizontal.*

W1 / ALL
W2 / W4; W5 ; W6 ; W8 ; W9; W10 ; W11 ; W12 ; W13; W14 ; W15 ; W16
W3 / W4 ; W5 ; W6 ; W7; W8 ; W9 ; W10 ; W11 ; W12 ; W13 ; W14 ; W15; W16
W4 / W5 ; W6 ; W8 ; W9; W10 ; W11 ; W12 ; W13; W14 ; W15 ; W16
W5 / W6 ; W10 ; W11 ; W14 ; W15 ; W16
W6 / W11 ; W16
W7 / W8 ; W9 ; W10 ; W11 ; W12 ; W13 ; W14; W15 ; W16
W8 / W9 ; W10 ; W11 ; W12 ; W13 ; W14 ; W15; W16
W9 / W10 ; W11 ; W13 ; W14 ; W15 ; W16
W10 / W11 ; W14 ; W15; W16
W11 / W16
W12 / W13 ; W14 ; W15; W16
W13 / W15 ; W16
W14 / W15 ; W16
W15 / W16
W16 / W16

8. *Rétrécissement vertical et horizontal.*

W1 / W1
W2 / W1
W3 / W1
W4 / W3 ; W2 ; W1
W5 / W4 ; W3 ; W2 ; W1
W6 / W5 ; W4 ; W3 ; W2; W1
W7 / W3 ; W1
W8 / W7 ; W4 ; W3 ; W2; W1
W9 / W8 ; W7 ; W4 ; W3; W2 ; W1
W10 / W6 ; W5 ; W9 ; W8 ; W7 ; W4 ; W3 ; W2; W1
W11 / W10 ; W6 ; W5; W9 ; W8 ; W7 ; W4 ; W3; W2 ; W1
W12 / W8 ; W7 ; W4 ; W3 ; W2 ; W1
W13 / W9 ; W5 ; W12; W8 ; W7 ; W4 ; W3 ; W2; W1
W14 / W10 ; W6 ; W13 ; W9 ; W5 ; W12 ; W8 ; W7 ; W4 ; W3 ; W2 ; W1
W15 / W14 ; W10 ; W13; W9 ; W5 ; W12 ; W8 ; W7 ; W4 ; W3 ; W2 ; W1; W6
W16 / ALL except W15

(B) Description des transitions après factorisation

La factorisation effectuée consiste à fusionner les transitions ayant les mêmes fenêtres destinations (la fusion pour les transitions ayant la même source a préalablement été réalisée dans le point (A)).

Dans ce cas-ci, nous ne citerons que les changements effectués et pas l'entièreté des transitions. Remarquons que cette factorisation n'est pas la seule possible. En effet, certaines transitions ayant des points communs avec plusieurs autres, il a fallu pour certains cas choisir de manière arbitraire quelles transitions mettre ensemble, et quelles autres dissocier. En jouant sur les différences, plusieures combinaisons sont donc possibles.

1. *Agrandissement vertical.*

W1; W3 / W3 W8; W12 / W12 W14; W15 / W15

2. *Rétrécissement vertical.*

W2; W4; W5; W6/ W2 W10; W11 / W6 ; W2

3. *Agrandissement horizontal.*

W1; W2 / W2 W15; W16 / W16

4. *Rétrecissement horizontal.*

W1; W2 / W1 W3; W7 / W3 W8; W12 / W7 ; W3

5. *Agrandissement vertical et horizontal.*

W11; W15; W16 / W16
W13; W14 / W15 ; W16

6. *Rétrécissement vertical et horizontal.*

W1; W2; W3 / W1
W9; W12 / W8 ; W7 ; W4 ; W3 ; W2 ; W1

4.2.3 Critique de la modélisation par machine à états finis

4.2.3.1 Modélisation par Machine de Moore

Avantages

L'avantage principal de cette technique réside dans la mise en évidence des différents états possibles, et donc des différents écrans affichables durant l'exécution de FlexClock. Cet aspect met par ailleurs l'accent sur la facilité d'analyse d'une telle machine à états finis : nous voyons de manière instantanée quelles peuvent être les fenêtres *successeurs* d'une fenêtre choisie (en tous cas, lorsque le nombre de transitions, et donc de successeurs possibles n'est pas trop élevé).

Inconvénients

L'inconvénient majeur réside dans l'explosion du nombre de transitions lorsque le nombre de fenêtres (représentations) possibles augmente. FlexClock, qui pourtant, ne comporte que 16 représentations possibles, pose déjà de gros problèmes de lisibilité. L'exemple le plus flagrant concerne l'agrandissement/rétrécissement dans deux dimensions simultanément (voir annexes). Dès lors, si nous superposons toutes les transitions possibles afin d'avoir la machine complète, le graphique devient impossible à analyser (voir figure 15).

4.2.3.2 Modélisation par Machine de Mealy

Avantages

L'avantage premier de la technique de représentation par machine de Mealy réside dans la mise en évidence des opérations et donc transitions possibles entre fenêtres. Dans ce cas-ci, contrairement aux machines de Moore, nous sommes en mesure d'analyser plus facilement les différentes transitions car la lisibilité est meilleure. De plus, n'oublions pas que dans notre cas d'étude, nous n'avons considéré que certains types d'opérations bien spécifiques. Ceci pour insister sur le fait, qu'en augmentant le nombre d'opérations possibles, la machine ne sera pas beaucoup plus difficile à analyser.

Inconvénients

L'inconvénient principal transparaît dans l'analyse non usuelle des machines construites suivant cette modélisation. Bien que la machine soit moins lourde graphiquement. Les fenêtres sources et destinations ne peuvent pas être facilement mises en évidence, à moins de disposer, comme dans le point précédent, des transitions sous format scriptural.

4.2.3.3 Résumé des avantages et inconvénients

	Représentativité	**Lisibilité** *- cas simple -* **(peu de fenêtres ; peu d'opérations)**	**Lisibilité** *- cas complexe -* **(beaucoup de fenêtres ; beaucoup d'opération)**
MACHINE DE MOORE	**Excellent**	**Excellent**	**Mauvais**
MACHINE DE MEALY	**- Excellent pour les opérations** **- Mauvais pour les fenêtres source et destination**	**Modéré**	**Modéré**

4.2.3.4 Conclusions

Ces deux techniques, bien qu'ayant chacune des avantages importants, posent un réel problème d'analyse. Elles ne permettent pas de représenter toutes les transitions et représentations possibles de fenêtres de manière efficace et concise. Bien que ne disposant que de 16 fenêtres de présentations possibles, FlexClock nous a clairement montré que ces deux modélisations ne conviennent pas pour atteindre notre objectif. Il nous faut donc une méthode plus appropriée.

4.2.4 Modélisation des transitions de fenêtres par surface

4.2.4.1 Description de la technique de modélisation

La technique consiste à représenter sur un plan à deux dimensions (hauteur, largeur en pixels) la *portée* visible de chacune des différentes vues/fenêtres possibles. Ainsi, ce plan contiendra, *in fine*, une série de rectangles indiquant les parties de ce plan (symbolisant les capacités de résolution de l'écran de l'utilisateur) dans lesquelles chacune des fenêtres est susceptible d'être affichée.

Voici un exemple avec une des vues de FlexClock : le coin inférieur droit de la fenêtre choisie est disposée à l'origine du plan en (0,0) et le rectangle bleu indique donc la portion de l'écran dans lequel on verra cette vue (remarque : à ce moment, nous n'avons pas encore pris connaissance des autres vues, ce qui implique que la portée de la représentation choisie est maximale et unique).

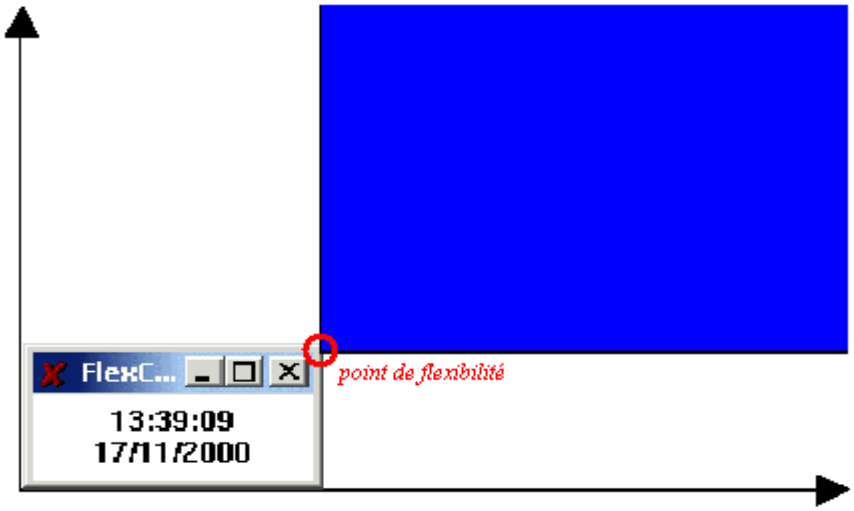


Figure 17 – FlexClock : exemple de modélisation par surface

Le cercle de couleur rouge (que nous appellerons *point de flexibilité*) représente le point à partir duquel nous passerons d'une présentation à une autre. Dans le cas présenté à la figure 17, une seule présentation est considérée. Ce qui implique que cette dernière sera la seule pour toutes les opérations de redimensionnement appliquée par l'utilisateur. Si nous voulions être plus précis, le rectangle bleu devrait normalement recouvrir l'entièreté du plan. En effet, avec une seule présentation possible, un rétrécissement dans une ou dans les deux dimensions possibles (à moins que la fenêtre ne soit pas rétrécissable) devrait conduire à cette même unique présentation.

4.2.4.2 Résultat obtenus

Voici les différents résultats obtenus pour la modélisation des transitions de FlexClock par surface (nous ne les montrerons cependant pas tous : ceux manquant sont disponibles en annexes). Sur les différentes figures, vous pourrez remarquer que les différents points de flexibilité des différentes fenêtres sont déjà indiqués dans un but de meilleure perception de l'entièreté de l'application.

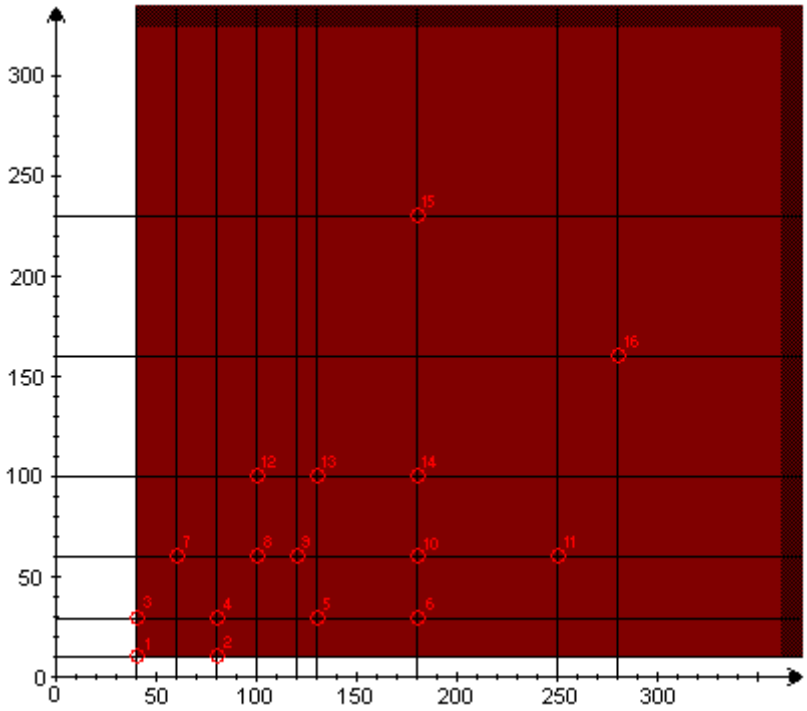


Figure 18 – FlexClock : Modélisation des transitions par surface pour la présentation la plus petite

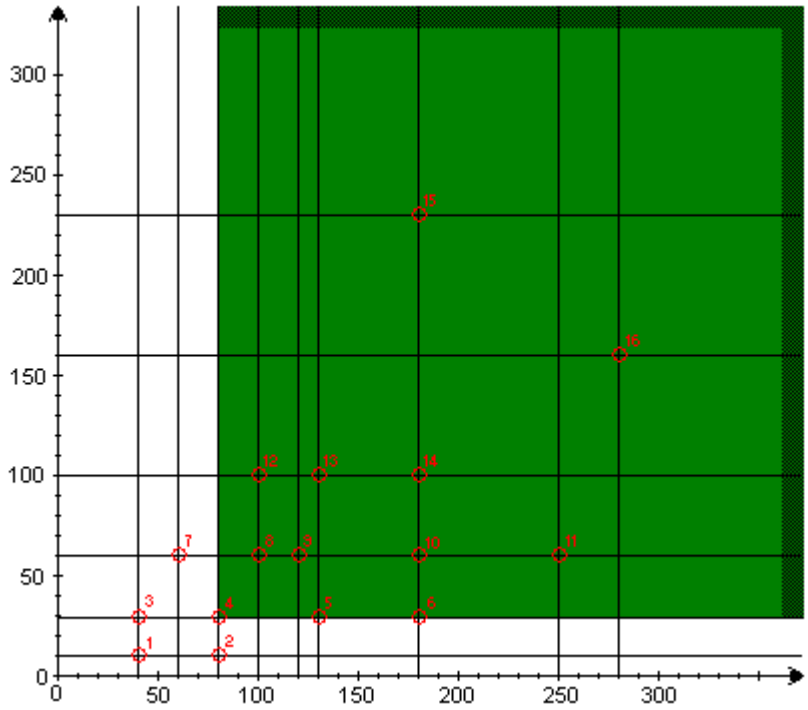


Figure 19 – FlexClock : Modélisation des transitions par surface pour la présentation W4

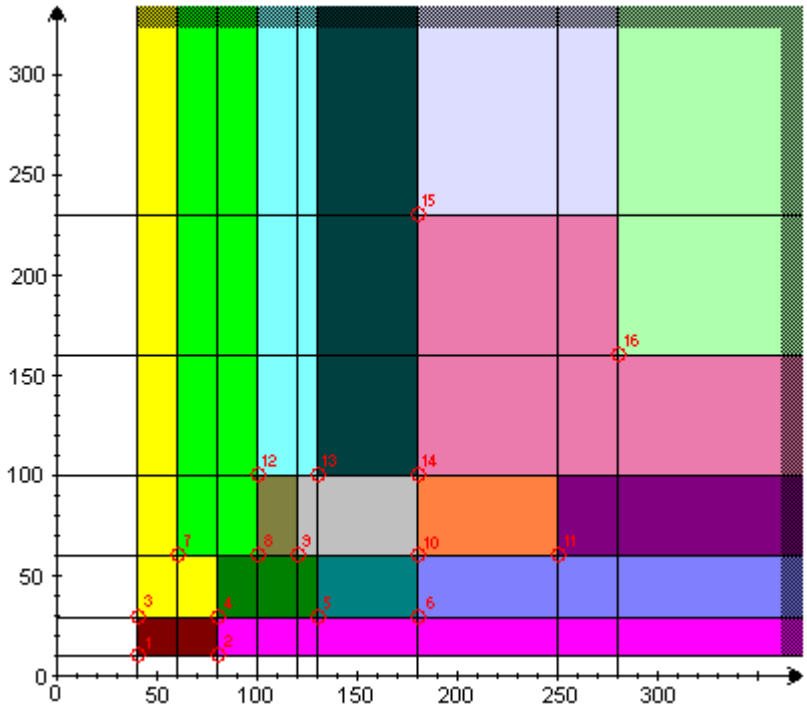


Figure 20 – FlexClock : modélisation des transitions par surface pour toutes les fenêtres possibles par superposition

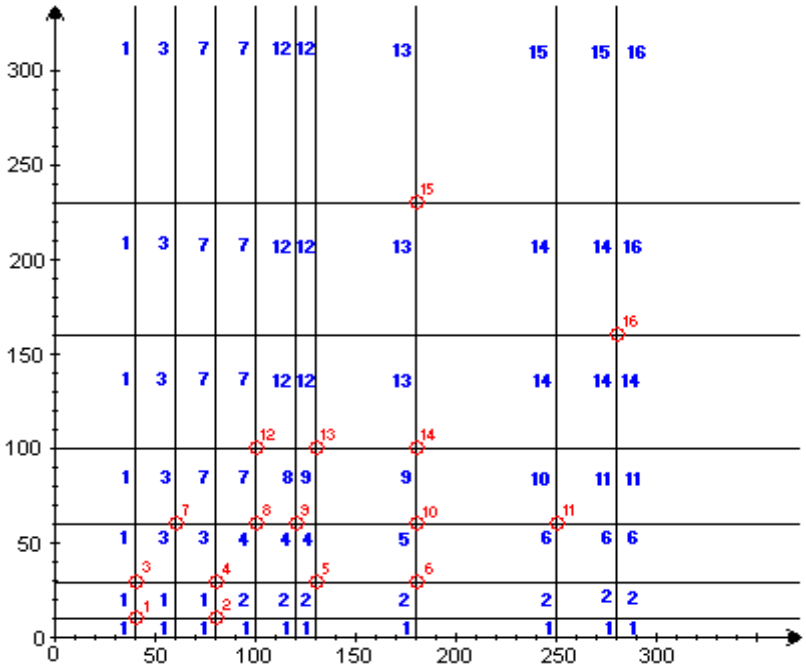


Figure 21 – FlexClock : Domaine de plasticité de chacune des interfaces

Nous avons inclus le domaine de plasticité *[Calvary & al. 2002]* sur la figure 21. Par domaine de plasticité, on entend l'ensemble des contextes d'usage pour lesquels l'interface reste opérationnelle et utilisable. La rupture de plasticité survient dès lors que le nouveau contexte est situé au-delà de ce domaine de plasticité. En fait, dans notre cas, chacun des changements de contexte peut s'interpréter comme un changement de taille de fenêtre, conduisant à afficher une nouvelle fenêtre. Notons qu'il n'y a pas de réelle différence, si ce n'est que le domaine de plasticité de la fenêtre un est étendu sur les parties initialement vides.

4.2.4.3 Critique et Comparaison par rapport aux machines à états finis

Cette technique semble tout à fait répondre aux désavantages rencontrés lors de la modélisation par machine à états finis. Elle permet de modéliser plus rapidement l'ensemble des transitions et la lisibilité ainsi que la précision, sont supérieures à celles des machines de Moore et de Mealy. Chacun des états possibles pour ces deux modélisations peuvent être transposés facilement à chacune des surfaces et points de flexibilité et nous savons exactement quand il faudra changer de présentation (rapport aux pixels). Les points de flexibilité indiquent les endroits cruciaux où une transition d'une présentation à une autre est possible et les surfaces montrent les dimensions limites autorisées pour chacune des fenêtres.

De plus, même si le nombre de fenêtres possibles augmente, la lisibilité du graphique ne diminue que très légèrement ou pour ainsi dire, pas du tout. En effet, il sera toujours possible de savoir à quelles fenêtres *destination* nous pouvons nous rendre depuis une fenêtre *source* sans devoir *éplucher* à la loupe le graphique, contrairement aux modélisations par machine à états finis.

4.3 Ebauche d'une application de génération de modélisations des transitions de fenêtres

Au départ, il n'était pas dans notre objectif de réaliser une application de génération automatique de modélisation de transitions de fenêtres. Néanmoins, après avoir trouvé une technique de modélisation efficace (par surface), il semblait intéressant de pouvoir concrétiser cela en imaginant une future application capable de la manipuler.

L'application décrite ci-dessous pourra générer à partir d'une modélisation par surface, les modélisations en machine de Moore et/ou de Mealy ainsi que le code de l'application, ce qui ne devrait pas être trop difficile à faire puisque, rappelons le, cette dernière ne dépend en réalité que de sa procédure d'adaptation *Place* (en supposant que la *sémantique pure* des objets des différentes présentations de l'application générée soit spécifiée à la main).

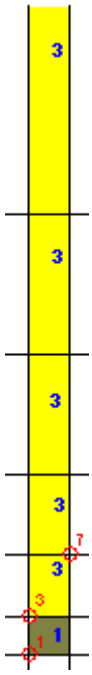
4.3.1 Règles de transformations entre modélisations pour l'application à générer

Nous savons qu'à partir de n'importe quelle fenêtre, nous pouvons passer à une autre. Ceci parce que les surfaces allouées à chaque présentation seront toujours des rectangles. Il est, en effet, difficile d'imaginer des résolutions pour lesquelles aucune fenêtre n'est disponible (mis à part peut-être le cas de la fenêtre pour la résolution minimale qui pourrait, dans le cas extrême, ne pas posséder la propriété de rétrécissement). De plus, la méthode choisissant la meilleure présentation devrait choisir la fenêtre se rapprochant le plus de la résolution spécifiée (ce qui rend obsolète ces « trous »). La seule contrainte réside dans le type de transitions autorisées. Ainsi, la seule chose à faire est de déterminer quelles sont les types d'opération possibles entre fenêtres et donc, quelles sont les transitions autorisées.

Passage de la modélisation par surfaces à la modélisation en machines de Moore

Pour ce faire, nous pouvons nous baser sur les informations stockées dans le tableau général et celui des points de flexibilité. Pour chacune des opérations considérées, nous observons le tableau général contenant toutes les parties du plan dans différents sens de lecture. Voici comment nous procédons :

1) Agrandissement/Rétrécissement vertical

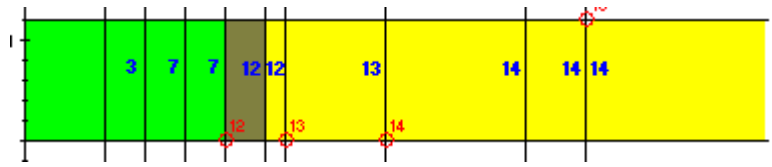


Dans ce cas-ci, il nous suffit, pour chaque présentation possible, et donc pour chaque endroit du tableau où se trouve un point de flexibilité, d'observer quelles sont les présentations possibles qui découlent de l'agrandissement (resp. rétrécissement) vertical de la fenêtre considérée.

Pour ce faire, nous limitons notre analyse aux seules colonnes du tableau où se trouvent un point de flexibilité, et nous mémorisons les différentes fenêtres possibles en haut (resp. en bas) de la case sélectionnée.

Dans l'exemple de gauche, nous voyons que pour un agrandissement vertical de la fenêtre numéro une, il n'est possible d'atteindre que la fenêtre trois.

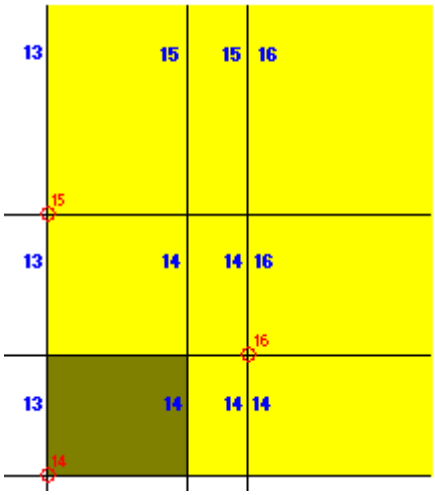
2) Agrandissement/Rétrécissement horizontal



Pour ces opérations, nous regardons, par contre, chacune des lignes du tableau. Pour l'agrandissement, nous voyons que dans le cas de la fenêtre 12, nous pouvons passer aux fenêtres 13 et 14. Remarquons que nous avons ignoré la case indiquant un passage vers la même fenêtre.

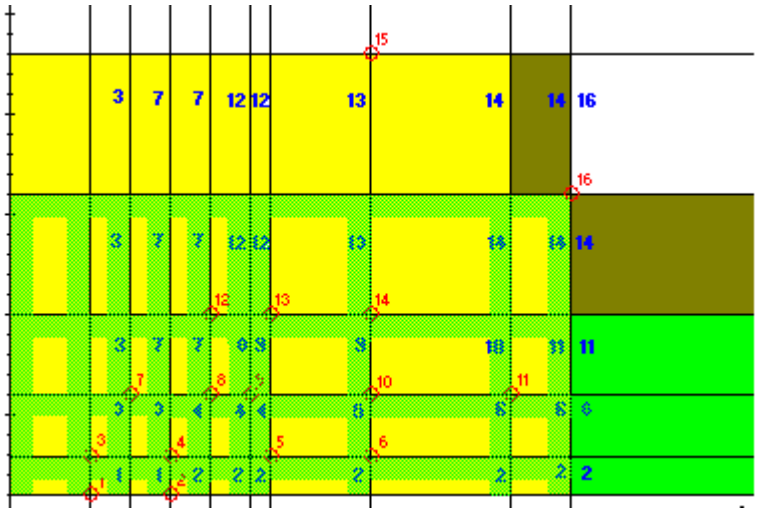
Dans le cas du rétrécissement, nous pouvons passer aux fenêtres 7, 3 et la plus petite, 1. Remarquons par rapport à la figure, que nous prenons également comme point de départ la case contenant le point de flexibilité et non celle qui lui est adjacente (qui devrait normalement être celle à considérer), car c'est totalement équivalent du point de vue du résultat. Nous verrons cependant que cette affirmation n'est pas vérifiée pour toutes les opérations.

3) Agrandissement dans les deux dimensions



Dans ce cas-ci, il nous faut analyser un tableau inclus dans le plan. Pour la fenêtre 14, nous pouvons transiter vers les fenêtres 16 et 15.
Comme pour l'exemple de l'agrandissement horizontal, nous ne mémoriserons pas les transitions vers les mêmes présentations que celle de départ.

4) Rétrécissement dans les deux dimensions



La partie concernant le rétrécissement dans deux dimensions est sans doute la partie la plus complexe à réaliser. Ici, contrairement aux opérations précédentes, nous ne prenons plus les cases où se trouvent les points de flexibilité comme points de départ mais les coins supérieurs droits du polygone délimitant les endroits du plan où telle présentation sera affichée (comme pour la présentation 14 sur la figure). Nous balayons alors chacun des tableaux pour voir quelles sont les différentes présentations auxquelles nous aboutissons en rétrécissant Ensuite, nous effectuons l'union de résultats obtenus. Ici également, nous n'incluons pas dans les résultats les transitions amenant à la fenêtre de départ.

Ensuite, lorsque toutes les informations sont collectées, nous construisons un tableau de vecteurs triés par valeur (application d'un algorithme de tri) pour chacune des opérations considérées ayant comme index les différentes fenêtres possibles pour l'application.

Cette structure finale représente la machine de Moore : on voit à partir de chaque fenêtre où on peut aboutir.

Exemple : tableau pour une opération d'agrandissement horizontal

W_1	1	3	9
W_2	1	4	
W_3	5		
…	…	…	…

Passage de la modélisation par surfaces à la modélisation en machines de Mealy

Il n'y a aucune différence par rapport à la transformation vers une modélisation en machine de Moore. Nous pouvons réutiliser la même structure : nous avons déjà les différents tableaux classés par opération. La seule chose à mentionner en plus sera la fenêtre ouverte par défaut lors du lancement de l'application afin de compléter la machine. Nous pourrions d'ailleurs utiliser l'algorithme constructif exposé dans la démonstration du théorème d'équivalence des machines.

4.3.2 Esquisse graphique d'une application indépendante et définition du modèle de la tâche

Voici l'apparence que nous donnerions à notre application s'il fallait la considérer comme indépendante. Nous allons décrire son fonctionnement ainsi que les différentes opérations possibles.

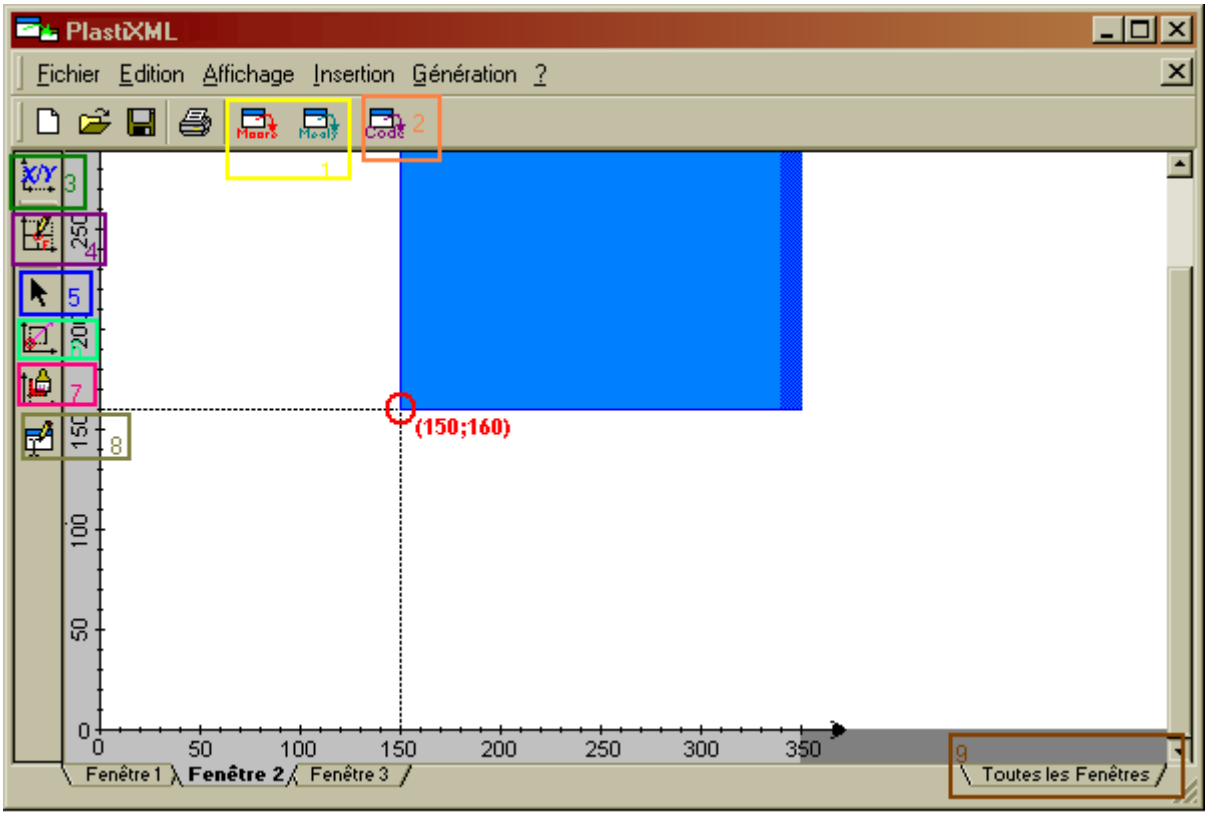


Figure 22 – Représentation graphique de PlastiXML : Notre application de modélisation de transitions

Nous allons pour cela définir un modèle de la tâche de l'application. Nous n'allons cependant pas rentrer dans les détails, nous nous limiterons aux tâches essentielles, notamment en ce qui concerne la construction d'un plan (au sens des transitions) et la manière de générer les différents modèles ainsi que le code. Il faut également signaler que, même si toutes les icônes semblent être actives sur la figure 22, il n'en sera normalement rien lors de l'utilisation de cette application. Il suffit pour cela de noter les nombreuses tâches séquentielles du modèle.

Pour construire le modèle de tâche, nous utiliserons la syntaxe utilisée par l'outil CTTE *[Paternó 97]*. Nous reviendrons plus en détail sur la notation ConcurTaskTree dans ce travail par la suite.

Voici le modèle que nous avons retenu pour notre application :

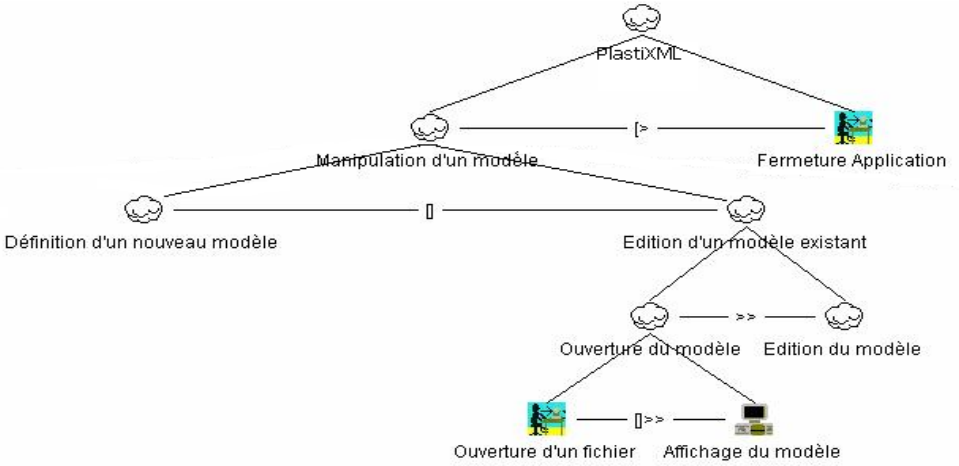


Figure 22 – Modèle de tâche non détaillé de PlastiXML

Nous allons maintenant détailler l'opération abstraite « Définition d'un nouveau modèle ».

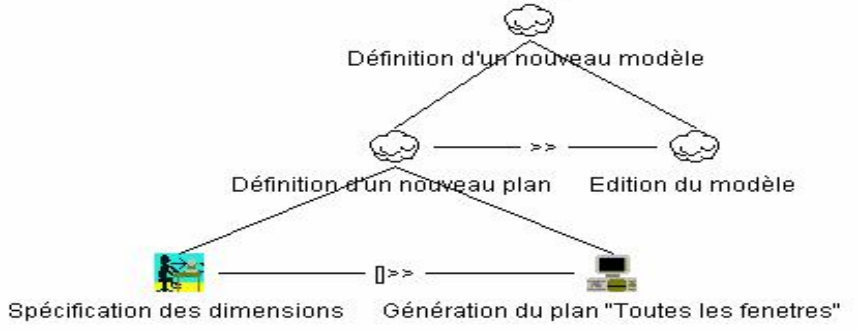


Figure 23 – Détail de la tâche Définition d'un nouveau modèle

Cette opération comprend l'opération « Edition du modèle », celle-ci est représentée de la manière suivante :

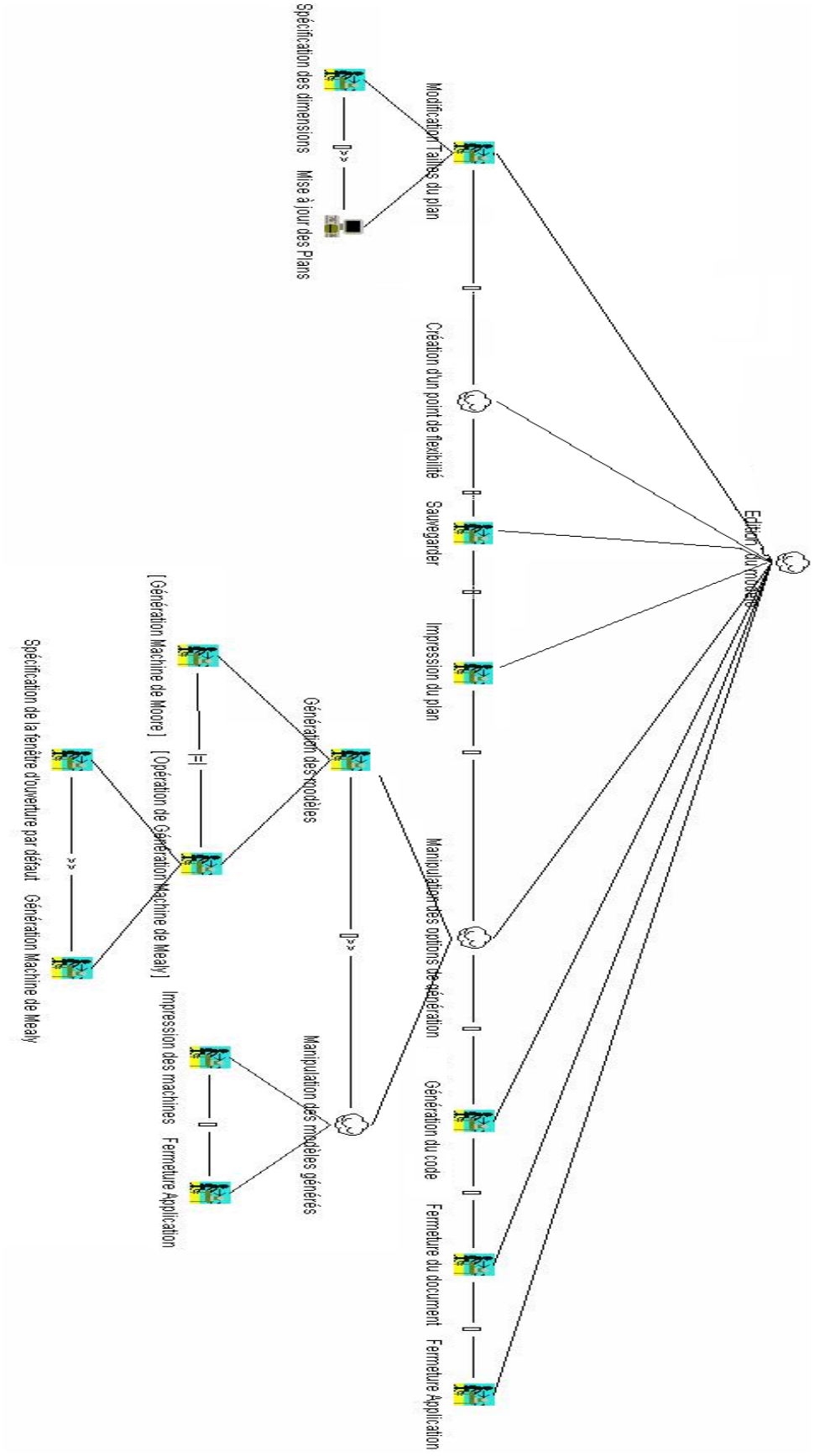


Figure 24 - Détail de la tâche Edition d'un modèle

Parmi les tâches filles de cette dernière, on peut citer :

- la tâche interactive *Modification Taille du Plan* qui sera déclenchée via l'icône numéro trois sur la figure 22 et aura une influence sur tous les onglets « Fenêtre »;
- la tâche abstraite *Manipulation des options de générations* qui pourra être déclenchée via les deux icônes contenues dans le rectangle numéro 1 de la figure 22 ;
- la tâche interactive *Génération de code* déclenchée par l'icône numéro deux ;
- la tâche abstraite *Création d'un point de flexibilité* que nous allons détailler immédiatement.

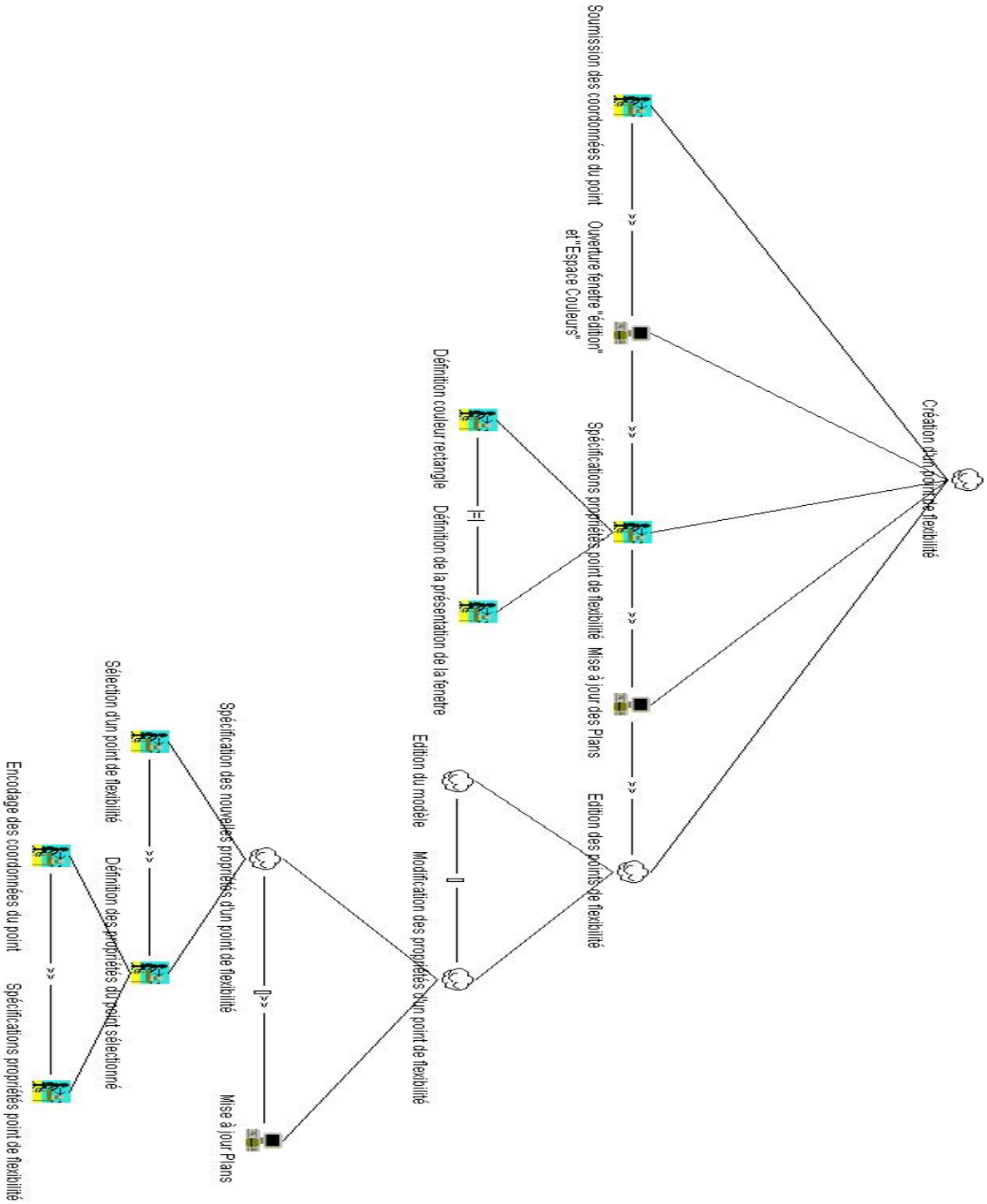


Figure 26 - Détail de la tâche Création du point de flexibilité

Parmi les tâches filles de cette dernière, on peut citer :

- la tâche interactive *Soumission des coordonnées du point* qui sera effectuée via l'icône numéro quatre de la figure 22
- la tâche système *Ouverture fenêtre 'édition' et 'Espace Couleurs'* qui se traduira par l'ouverture de quatre fenêtres : une pour le choix de la couleur du rectangle indiquant l'espace où la nouvelle fenêtre sera visible, trois pour la création de la présentation de la fenêtre (boîte de construction style *Interface Builder* et une fenêtre montrant la présentation que l'utilisateur est en train de construire ainsi qu'une fenêtre d'édition pour le code de l'application à générer) ;
- la tâche abstraite *Modification des propriétés d'un point de flexibilité* déclenchée qui, comme on peut le voir sur la figure 26, est raffinée en tâches séquentielles : ainsi, nous devrons d'abord sélectionner un point (icône 5). Ensuite, nous pourrons effectuer la tâche *Définition des propriétés du point sélectionné*, qui feront appel en premier lieu à l'icône 6 et ensuite aux icônes 7 et 8 (le lecteur pourra se rendre compte que la tâche *Spécification des propriétés du point de flexibilité* devrait normalement être raffinée en deux sous-tâches optionnelles pour chacune de ces icônes).

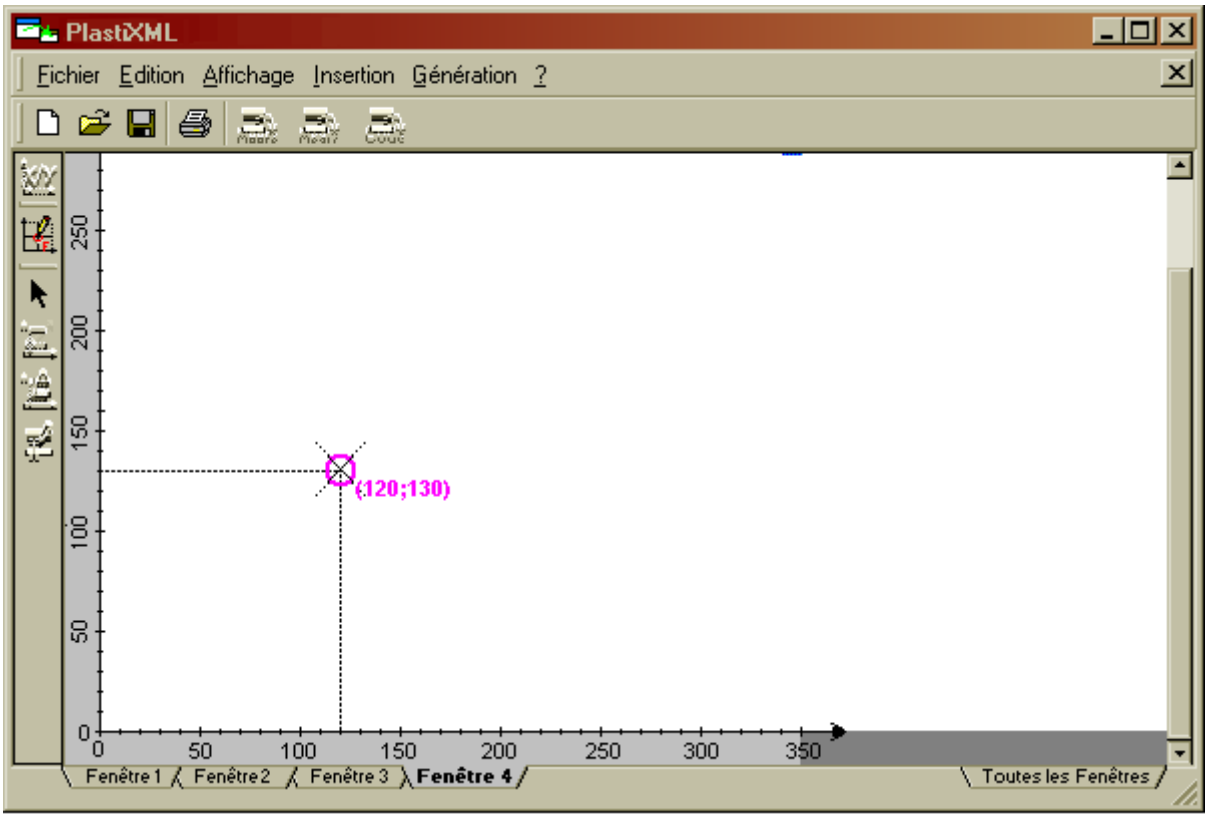


Figure 27 – PlastiXML : Soumission des coordonnées du point de flexibilité

4.3.3 Intégration de PlastiXML comme plugin au sein d'une application de génération d'interfaces basée modèle

L'idée de devoir concevoir une application spécialisée avec l'obligation de créer un éditeur d'interface (graphique et textuel) constitue une charge supplémentaire. D'autant plus qu'il vaille mieux que le langage généré ait un caractère « universel » afin de pouvoir l'utiliser sur une grande quantité de plate-formes. De même, il serait intéressant de pouvoir disposer d'une définition d'interface qui soit compatible avec la technique de génération basée modèle.

Or, il existe déjà un éditeur générant des interfaces en USiXML (cfr. infra), baptisé GraphiXML. Cet outil permettant le support de différents plugins, il serait donc utile d'envisager l'intégration de notre application dans cet environnement.

Les fichiers renfermant l'information sur les transitions comporteraient alors deux parties :

- une partie statique, comportant la définition des différentes fenêtres en USiXML
- une partie dynamique, stockant les transitions possibles entre les différentes fenêtres (nous pourrions utiliser le mécanisme de RuleTerm[1] ou chaque *action* spécifierait une transition selon une machine de Moore (même si cela comporte les inconvénients soulevé plus haut))

De même, nous pourrions également étendre le champ de notre plugin aux interfaces à présentations purement dynamique, en plus des interfaces à présentations pré-calculées. Au lieu d'avoir comme *action* une transition entre présentations, nous aurions alors une **fonction de représentation** qui reconstruirait l'interface souhaitée après transition.

[1] http://api.openoffice.org/docs/common/ref/com/sun/star/ucb/RuleTerm.html

Chapitre intermédiaire : Discussion : FlexClock et le cas de la dégradation harmonieuse

Comme nous avons pu nous en rendre compte, FlexClock repose sur un principe assez simple. Dès lors, pourrions-nous continuer dans la même optique pour la partie pratique du mémoire ? La question vaut la peine d'être abordée.

La réponse à cette question est malheureusement négative : dans le cas de FlexClock, nous ne pouvons pas parler d'adaptation *at runtime* à proprement parler. Chacune des présentations n'est pas calculée dynamiquement lors du redimensionnement mais est déjà connue à l'avance. Voilà pourquoi nous qualifions l'environnement dans lequel nous situons pour cet application de *totalement observable*.

Toutes les vues sont en effet spécifiées et préconçues à l'avance, avant l'exécution du système. La procédure *Place* ne fait donc que choisir la présentation qui satisfera le plus aux exigences (la résolution) de la fenêtre. Et voilà donc pourquoi, même s'il y a un principe d'adaptation du système en fonction de la résolution, nous ne pouvons pas parler de dégradation harmonieuse au sens strict puisque nous définissons plusieures présentations plutôt qu'une seule (celle qui sera destinée à l'écran de plus haute résolution).

Néanmoins, le principe même de lier un événement <*Configure*> à un *placeholder* pour la sélection de la meilleure vue est essentiel dans notre approche. Ainsi, nous pourrons peut-être recréer une procédure se rapprochant de *Place* par certains aspects. La seule différence (mais qui est de taille !) consistera à reconstruire l'interface de manière automatisée en fonction de certains critères et règles de transformation plutôt qu'à sélectionner celle qui conviendra le mieux dans un ensemble prédéfini.

Chapitre 5 : La dégradation harmonieuse d'UIG dans le cadre d'une approche orientée modèle

5.1 Etude de cas : la génération automatique d'interfaces utilisateur.

Comme nous l'avons mentionné dans le chapitre intermédiaire, nous allons devoir revoir la manière dont nous allons choisir et construire les présentations à afficher à l'écran. Ainsi, une bonne manière de commencer notre analyse serait de considérer, dans une premier point, le cas de la génération automatique d'interfaces utilisateur indépendamment d'une règle de sélection d'écran à afficher.

Pour cela, nous allons nous baser sur l'exemple *Roger Rabbit* exposé dans le chapitre 10 du livre *[Van Roy & Aridi 2003]*. Nous allons réaliser une application du même style et essayer d'en retirer les avantages et inconvénients, notamment en ce qui concerne les structures de données utilisées pour modéliser la future interface à construire. L'exemple fourni permet d'afficher deux présentations possibles (une en mode lecture et une en mode écriture) en actionnant un bouton. Ainsi, chaque fois que le bouton est actionné, une procédure de construction d'interface est lancée et le résultat affiché dans un *placeholder*.

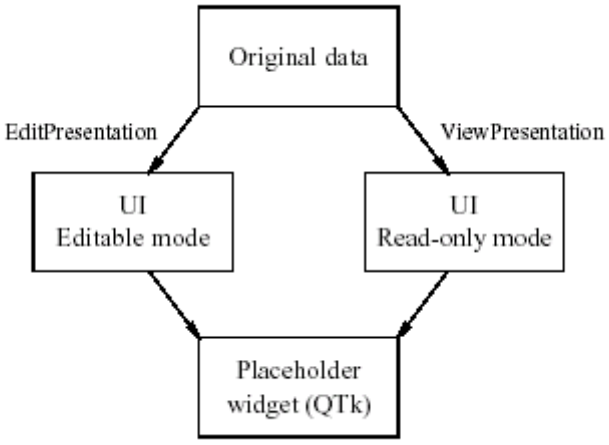


Figure 28 – Etapes de construction de l'interface à partir des données (figure tirée de [Van Roy & Aridi 2003])

Nous avons vu précédemment que l'approche basée modèle pouvait être d'une grande utilité dans la spécification d'interface. Il sera donc utile de l'utiliser pour notre future application. Néanmoins, dans quelle mesure devons nous appliquer cette modélisation? L'exemple exposé ci-dessus ne fait appel qu'au modèle du domaine et aux seules valeurs attribuées aux éléments constituants.

```
Data = [name#"Roger" surname#"Rabbit" age#14]
```

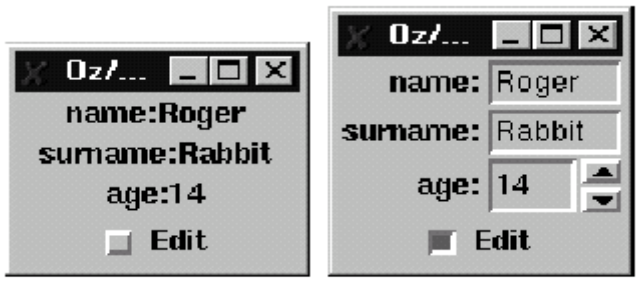


Figure 29 – Les deux présentations de l'application Roger Rabbit (figure tirée de [Van Roy & Aridi 2003])

Néanmoins, cela est-il suffisant pour atteindre notre objectif final et ne risquons-nous pas de limiter le nombre de règles de transformations qu'à des aspects limités de notre interface (concrètement, les éléments du domaine) ?

L'application de génération automatique que nous allons construire dans un premier temps va nous permettre de déterminer quels sont les points à mettre en évidence et ceux qui sont superflus ou de moindre importance.

Notre application : description sommaire

Le but de notre application de génération automatique d'IU est d'afficher de sélectionner des noms de pays dans un accumulateur et de mettre en évidence ceux sélectionnés sur une carte. Un petit carré de couleur sera attribué à chacun des pays sélectionnés sur la carte et en déplaçant le curseur sur ce dernier, on peut vérifier la position géographique du pays choisi.

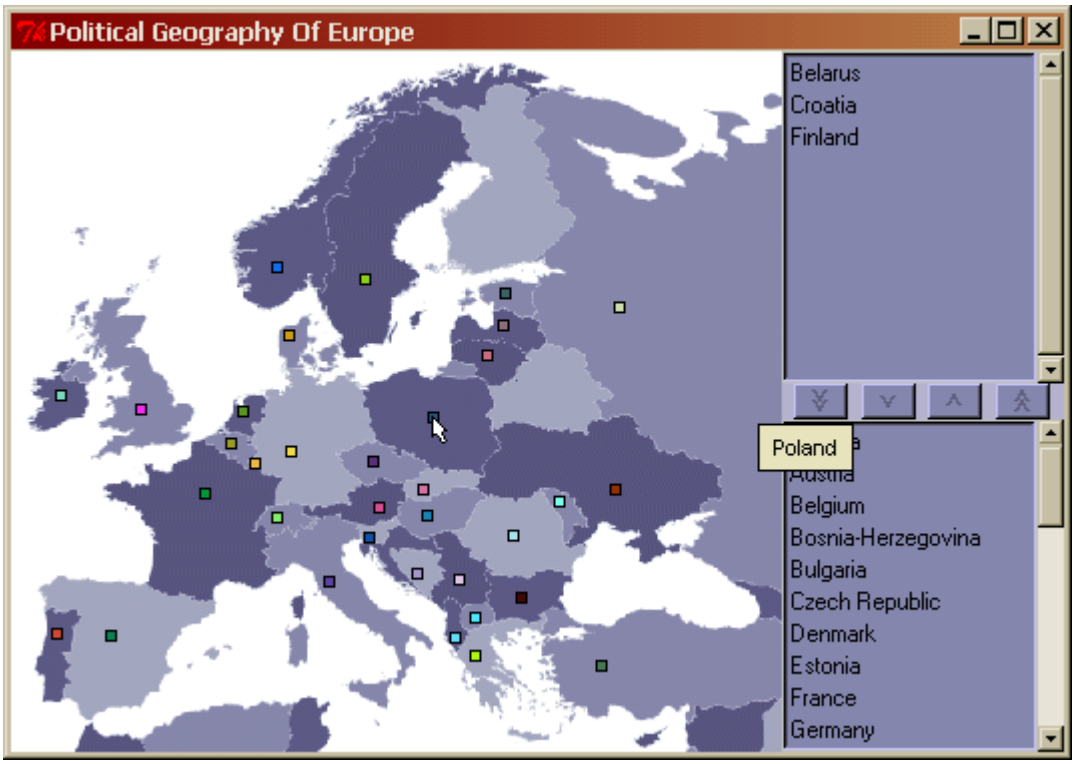


Figure 30 – Apparence graphique de notre application de génération automatique d'IU

Notre modèle du domaine comprend une image (nom de fichier GIF) pour représenter la carte et un dictionnaire dont les clefs sont les noms des pays et les valeurs sont les coordonnées de chaque carré à représenter sur la carte, ainsi qu'un *Tag* leur faisant référence.

```
Data = [picture#'map-europe_385x350.gif' countriesdatas#Dict]

Ex : Dict.'Latvia' := 243#134#_
     Dict.'Lithuania' := 235#149#_
     Dict.'Luxemburg' := 119#203#_
     Dict.'Macedonia' := 229#280#_
     Dict.'Moldova' := 271#222#_
     Dict.'Switzerland' := 130#230#_
```

Dans notre cas, nous n'aurons qu'une seule présentation à construire. Dans la fonction qui la construit, chaque paire `identificateur#valeur` est liée à un widget spécifique. Ainsi, nous avons (le code complet se trouve en annexes) :

```
fun {ViewPresentation Data}
   H
   fun {Loop X}
      case X of D#V|Xs then
    if (D==picture andthen {IsAtom V}) then

        Widget1|{Loop Xs}

    elseif (D==countriesdatas andthen {IsDictionary V}) then

           % Accumulator Functions and Procedures
      proc{AccumuleOneElem Handle1 Handle2 DrawAction}
         …
      end

      proc{AccumuleAllElem Handle1 Handle2}
         …
      end

          %Drawing/UnDrawing Procedure
      proc {DrawSquares CanvasHandle Dict List DrawAction}
         …
      end

      proc {ToolTipMaker CanvasHandle List}
         …
      end
   in
         …
Widget2 (-> accumulateur -> ensemble de widgets)|{Loop Xs}
   end
       else nil end
    end
in
    r(desc:{Record.adjoin {List.toTuple lr {Loop Data}}
     lr(glue:nswe background:c(180 179 203) handle:H)}
       handle :H)
end
```

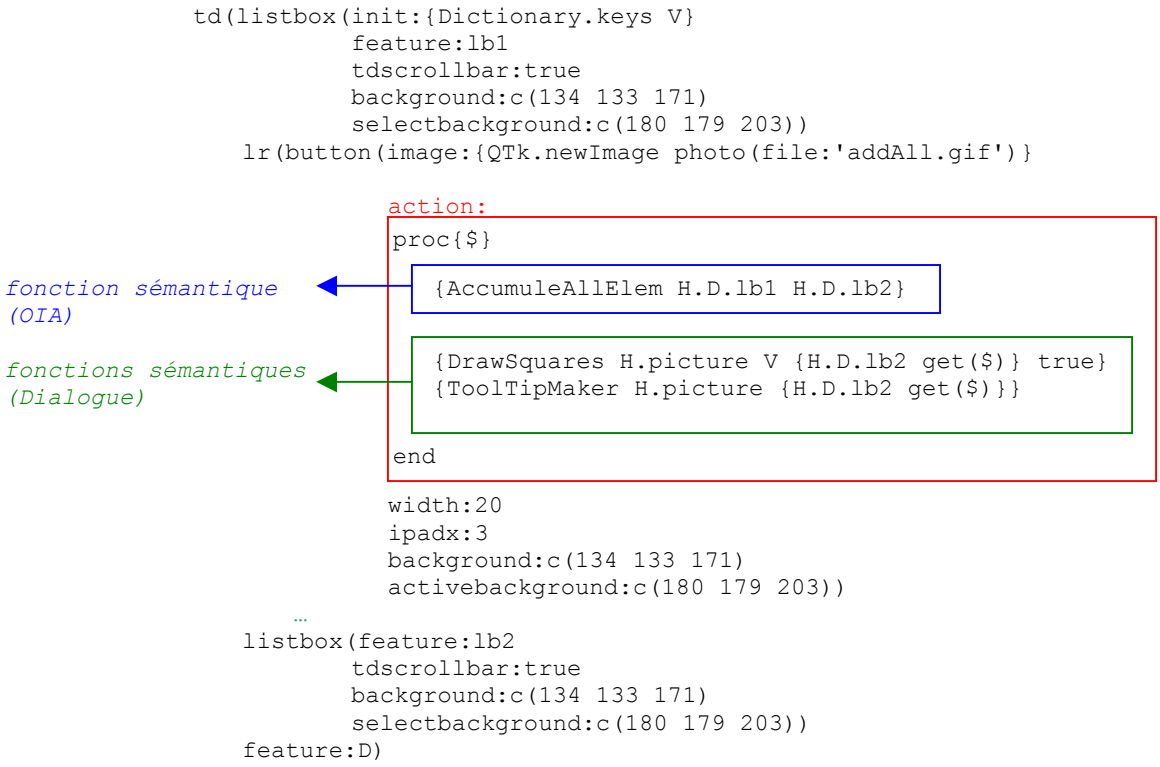
Notre application : critique de l'approche suivie

Comme nous nous en doutions avant de réaliser notre application, le gros désavantage de cette technique réside dans la pauvreté de la définition/spécification de l'interface. En ne disposant uniquement d'un record avec les données, le type de widget choisi doit toujours être spécifié à la main dans la fonction de construction (les encadrés rouges dans le code de cette dernière le montre clairement).

Avec une définition d'interface comme celle-ci, le choix reste donc arbitraire de bout en bout. Or, dans notre processus de dégradation d'interfaces, il ne nous sera pas donné la possibilité de spécifier nos choix de widgets de cette manière pour toutes les présentations possibles.

Il va donc sans dire que notre définition devra être enrichie si nous voulons avoir des règles de sélection plus fines, dont le fonctionnement dépend de plusieurs niveaux d'abstraction de l'interface.

Enfin, nous devrons veiller à ce que les fonctions déclenchées lors de manipulation de widgets respectent la propriété de continuité de l'interface. En effet, la manière dont les actions sont gérées en QTk ne permet pas forcément de bien séparer les fonctions sémantiques spécifiques à un widget/groupe de widgets de celles qui sont spécifiques au modèle du dialogue d'une interface, notamment en ce qui concerne les widgets composés. L'accumulateur utilisé dans notre exemple est là pour le montrer :



Or, comme nous le verrons plus loin dans ce travail, certains widgets seront plus appropriés dans certains cas spécifiques de présentation. Ceci implique que nous devrons non seulement nous arranger pour bien séparer les fonctions sémantiques des autres fonctions de manière à pouvoir les réinjecter dans les widgets choisis lors de la construction de la présentation, mais aussi adapter leur mode de déclenchement puisque la manière de déclencher une action n'est pas la même pour tous les widgets.

5.2 Enrichissement de notre définition d'interface : méthode suivie

L'utilisation du simple modèle du domaine n'est pas suffisante. Nous allons donc dans un premier temps décrire de manière succinctes les différents modèles qui sont le plus souvent considérés dans le développement d'interfaces utilisateurs.

Ensuite, nous verrons et classifierons les règles de dégradation harmonieuse et leur incidence sur les différents modèles considérés. Pour réaliser cette classification, nous introduirons tout d'abord les quatre niveaux d'abstraction décrit par le *framework* Cameleon, ce qui nous permettra ainsi de structurer les modèles décrits précédemment par rapport à une structure plus générale. Et enfin, nous déterminerons les différentes règles de dégradation retenues pour notre application en QTk et donc, par implication, les différents modèles à retenir également, ainsi que le formalisme choisi pour les spécifier.

5.3 Description des modèles constituants

Les modèles généraux que nous pourrions considérer pour notre problème sont au nombre de quatre : le modèle du domaine, le modèle de la tâche, le modèle de la présentation et le modèle de la plate-forme.

5.3.1 Le modèle du domaine

Le modèle du domaine capture les objets du domaine et leurs inter-relations. Une étude réalisée sur différents MB-UIDEs (Model-based User Interface Development Environments) *[Limbourg 2001]*, *[Souchon 2002]* a montré que le formalisme utilisé pour décrire ce modèle pouvait prendre quatre formes différentes :

1. Les diagrammes de classes (e.g. GENOVA, TADEUS, TEALLACH)
2. Les diagrammes d'entité-relation (e.g. TRIDENT)
3. Les classes implémentées (e.g. MasterMind et les classes CORBA)
4. Les langages ad-hoc (e.g. le langage TIMM dans MOBI-D)

5.3.2 Le modèle de la tâche

Le modèle de la tâche consiste en une description des tâches qu'un utilisateur est dans la capacité d'accomplir en interagissant avec un système. Cette description se traduit par une décomposition hiérarchique d'une tâche globale, avec des contraintes exprimées sur et entre plusieures sous-tâches.

Le modèle de tâche est utile durant tout le cycle de développement d'une application : on l'utilise de la découverte des requis du futur software demandé par l'utilisateur (user requirements elicitation) jusqu'à la phase de design, de développement et d'évaluation. Voilà pourquoi il doit être compréhensible par tous les intervenants du processus de développement du logiciel (utilisateurs, développeurs de l'application, développeurs de l'interface, …)

5.3.3 Le modèle de la présentation

Le modèle de la présentation représente la partie statique de l'interface utilisateur. Elle consiste en :

- une décomposition hiérarchique de l'écran en un ensemble d'éléments de présentations appelés objets interactifs jusqu'à ce qu'il n'y ait plus aucun élément composé

- la description des relations entre ces différents éléments, e.g. les relations de placement (ex : tel objet est placé au-dessus d'un autre) et les relations logiques (ex : un groupe d'objets interactifs dans lequel on accorde plus d'importance à un d'entre eux).

Les Objets Interactifs

Dans le cadre de l'approche de dégradation harmonieuse, les objets interactifs devraient posséder les propriétés suivantes :

1. Ils devraient être indépendants du toolkit utilisé. Ceci permettrait notamment au designer de spécifier l'interface sans devoir connaître de manière précise les détails sur chaque élément de présentation pour chaque toolkit, comme le nom de l'objet et les noms de ses attributs, par exemple.

2. Le(s) rôle(s) de chaque objet interactif devrait être inclus dans la description qui lui est faite. Classifier les objets interactifs par rôle peut nous permettre de voir quels sont ceux qui sont équivalents conceptuellement (dans le but de pouvoir faire de la substitution quand le moment sera voulu).

3. Ils devraient spécifier les types de données abstraits qu'ils sont capables de manipuler.

4. Ils devraient spécifier leur *coût de présentation [Thevenin & Coutaz 99]*, c'est-à-dire la quantité de ressources physiques à mettre en place pour leur instanciation. Dans le cas qui nous intéresse, les ressource physiques sont essentiellement la hauteur et la longueur des objets affichables à l'écran.

5. La description des objets interactifs devrait permettre de voir si un OI est approprié pour une combinaison donnée de rôle/type de données abstrait/plate-forme. Ainsi, dans notre cas, la description devrait participer à l'évaluation de l'utilisabilité entre un interface source et une interface destination.

La plupart des *toolkits* et UIMSs utilisent les widgets comme point d'appui pour développer les interfaces utilisateurs. Néanmoins, cette approche ne satisfait pas au premier critère : l'indépendance des toolkits. Aussi, il faut introduire une autre manière, qui soit commune à tous, pour représenter les objets interactifs pour les systèmes basé modèle.

[Vanderdonckt & Bodart 93] se sont intéressés au problème en faisant la distinction entre Objets interactifs concrets (OIC) et Objets interactifs abstraits (OIA).

Un OIC (widget, interacteur physiques) représente tout objet visible, manipulable de l'IHM, utilisé pour l'acquisition et/ou la restitution d'informations relatives à la tâche interactive de l'utilisateur au sein d'un contexte d'utilisation donné. C'est donc l'objet « réel » se rapportant à un toolkit particulier. La modélisation d'un OIC comprend :

- ses attributs concrets : ensemble des attributs physiques matérialisant les états internes et externes de l'OIC dans l'environnement physique.
- ses événements concrets : ensemble des événements asynchrones physiques reçus ou générés par l'OIC suite à tout changement d'état au sein de l'environnement physique.
- ses primitives concrètes : ensemble des primitives propres à l'environnement physique régissant le comportement interne et externe de l'OIC.
- sa représentation graphique : ensemble des apparences physiques de la composition spatiale de l'objet ou du groupe de ses objets composants

Un OIA (interacteur logique) consiste en une abstraction de l'ensemble des OICs de même type indépendamment des environnements physiques qui l'accueillent. Sa modélisation comporte des attributs, des événements abstraits, ainsi que des primitives abstraites. Sa nature est soit, d'acquisition, soit de restitution, soit une combinaison de deux et il peut également être élémentaire ou composé (décomposable en d'autres objets élémentaires ou composés).

Néanmoins, cet OIA comme proposé ne permet pas de spécifier une interface à un niveau sémantique très élevé. Par exemple, il est possible de spécifier dans le modèle de la présentation qu'un *bouton* doit être utilisé, mais il n'est pas possible de définir un objet de présentation capable de déclencher une fonction indépendamment de cet objet (bouton, item de menu, …).

Plusieurs catégories d'OIA ont été proposées (OIA d'action, de défilement, statiques, de contrôle (action/information), de dialogue et de feed-back) mais celles-ci ne correspondent pas réellement aux rôles que nous voudrions représenter dans notre modèle des objets interactifs. En effet, les critères de classification employés ne sont pas appropriés à notre objectif : un menu ou un bouton permettent tous deux de déclencher des fonctions et sont pourtant séparés. De plus, les objets appartenant à une même catégorie ne sont pas substituables mutuellement. Il nous faudra donc trouver une autre technique de classification.

Les relations de placement (Layout Relationships)

Les relations de placements peuvent être définies de différentes façons et il existe, d'après *[Thevenin 2001]* trois niveaux d'abstraction :

1. *le niveau lexical*, qui positionne les objets interactifs dans un espace coordonné
2. *le niveau syntaxique*, qui définit les relations géométriques entre les objets interactifs (par contraintes)
3. *le niveau sémantique*, qui vérifie des propriétés de l'interface. Ces types de placement vérifient, par exemple, l'équilibre de l'interface, l'uniformité de la densité (le gris typologique en mise en page), …

Nous nous limiterons cependant à l'étude des points un et deux dans notre application.

Les relations logiques (Logical Relationships)

Ces relations entre objets interactifs se situent à un niveau beaucoup plus abstrait. Quatre types de relation logique ont été identifiés :

1. l'*ordonnancement*
2. la *hiérarchie logique*
3. le *groupement*
4. la *séparation*

Remarquons que groupement n'est pas synonyme de composition d'OI (comme mentionné plus haut). Le groupement concerne des objets interactifs distincts, contrairement à la composition, qui sont liés par différentes tâches *feuille* du modèle de tâche.

Chacune de ces relations logiques peut être traduite en *contraintes géométriques* entre objets interactifs ou en *différences au niveau de l'apparence externe* entre objets interactifs.

Contraintes géométriques

a) ordonnancement : l'adjacence horizontale et verticale entre objets interactifs; la relation à droite *de* (l'abscisse gauche d'un OI est strictement supérieure à l'abscisse droite d'un autre) ou *est inférieur par rapport à* (la coordonnée supérieure d'un OI est strictement supérieure à la coordonnée inférieure d'un autre) entre objets interactifs.

b) hiérarchie logique : le renfoncement à gauche (un OI est inférieur rapport à un autre et son abscisse gauche est supérieure à l'abscisse gauche de l'autre d'une certaine distance d)

c) groupement : l'inclusion d'OI dans un même conteneur (group box,…)

d) séparation : l'accroissement de distance entre OI

Différences au niveau de l'apparence externe

a) ordonnancement : une liste avec puces ou avec numéros

b) hiérarchie logique : le type de fonte utilisé, sa taille, son style, la couleur utilisée, …

c) groupement : l'inclusion d'OI dans un même conteneur (group box,…)

d) séparation : l'insertion d'objets statiques du type *séparateur* (lignes, retour à la ligne…)

5.3.4 Le modèle de la plate-forme

La plate-forme se définit comme l'ensemble des moyens logiciels et matériels mis à disposition pour supporter la tâche de l'utilisateur.

Lorsque nous considérons le développement d'interfaces utilisateur, il est souvent très utile d'ajouter à la notion de plate-forme d'autres éléments tels que les navigateurs utilisés (pour les interfaces Web) ou encore les toolkits graphiques disponibles. Dans cette optique de modélisation de plate-forme, nous pouvons citer comme exemple les profiles W3C CC/PP[2].

Ces stéréotypes consiste en une description des capacités de l'appareil et des préférences de l'utilisateur, ceci dans le but de guider la manière dont la présentation devra s'effectuer et donc être adaptée sur chaque appareil.

CC/PP n'est cependant pas un vocabulaire à part entière nous permettant de décrire une plate-forme mais se base sur un langage générique basé sur XML/RDF (Resource Description Format) permettant d'en écrire. Il prend donc en compte les préférences de l'utilisateur (langue préférée, son on/off, images on/off, scripting on/off, cookies on/off, etc…). Ces profiles mettent également l'emphase sur les attributs de la plate-forme physique (distributeur, modèle, la classe d'appareil {téléphone, pda, imprimante, etc…}, taille de l'écran, couleurs, la bande passante disponible, CPU, mémoire, moyens de stockage secondaires, baffles, etc…) mais également sur les attributs logiciels (la marque et la version du logiciel, le niveau d'HTML supporté, le vocabulaire XML supporté, le niveau de CSS supporté, le vocabulaire RDF supporté, le niveau supporté WAP, les langages pour le scripting, etc…).

[2] Composite Capability/Preference Profiles (CC/PP): A user side framework for content negotiation accessible at http://www.w3.org/TR/NOTE-CCPP

5.4 Mise en évidence des règles de dégradation harmonieuse

Dans cette partie, nous introduirons la structure et les quatre niveaux d'abstraction des modèles instanciés du framework Cameleon sur lequel nous baserons notre modélisation, ainsi que le formalisme utilisé pour définir les différents modèles (tâches, domaine, …) qui en font partie. Ensuite, nous expliciterons en détails les différentes règles de transformation retenues pour notre application et leur place par niveau d'abstraction.

5.4.1 le framework Cameleon : 4 niveaux d'abstraction

La structure Cameleon [Calvary & al. 2003] est composée de modèles *ontologiques* et de modèles *instanciés*. Les modèles ontologiques sont des descriptions abstraites des concepts qui entrent en jeu dans tous les développements d'IU sensibles au contexte. Il s'agit de méta-modèles indépendant de n'importe quel secteur d'activité humaine (médical, comptable,…) et de n'importe quel système interactif. Il existe trois modèles ontologiques concernant la sensibilité au contexte :

- les modèles du *Domaine* qui donnent une description des concepts et tâches de l'utilisateur relative à un domaine ;
- les modèles du *Contexte* qui donnent une description du contexte d'utilisation, décomposables en trois modèles concernant le profile de l'utilisateur, la plate-forme et l'environnement (il est donc nécessaire de disposer d'au moins un de ces modèles pour construire le modèle du contexte) ;
- les modèles de l'*Adaptation* qui spécifient la manière dont une IU peut être adaptée après un changement de contexte d'utilisation suffisamment significatif.

Une fois *instanciés*, les modèles produits seront alors dépendants d'un système particulier (domaine d'activité, etc…). Il existe ainsi quatre niveaux d'abstraction pour les modèles instanciés, de la spécification de la tâche de l'utilisateur jusqu'à l'interface finale :

- Le niveau *Tâches et Concepts* : ce niveau donne les spécifications des systèmes interactifs en terme de tâches utilisateur à gérer et en terme des objets du domaine qui seront manipulés par ces tâches.
- Le niveau *AUI* : ce niveau exprime l'IU en terme d'unités de présentation, et ceci indépendamment des objets interactifs qui sont disponibles. (une unité de présentation représente un environnement de présentation permettant l'exécution d'un ensemble de tâches interconnectées)
- Le niveau *CUI* : ce niveau exprime l'IU en terme d'objets interactifs abstraits et leur position. Ainsi, la CUI représente encore une sort de maquette, à ce stade du développement.
- Le niveau *FUI* : ce niveau traduit la génération de code source dans un langage de programmation donné d'une CUI. Celui-ci pourra être interprété ou compilé.

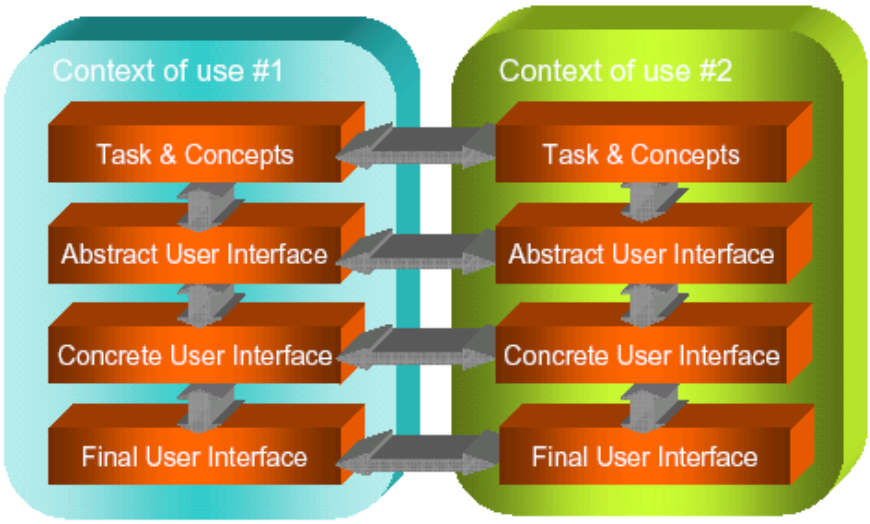


Figure 31 – Les quatre niveaux d'abstraction retenus pour les modèles instanciés du framework Cameleon

5.4.2 le framework Cameleon : le formalisme USIXML

USIXML est un langage basé XML/XSD construit de telle manière à respecter les quatre niveaux d'abstraction définis par la structure Cameleon. Le niveau *Tâches et Concepts* comporte ainsi trois modèles : le modèle de la tâche, le modèle du domaine et le modèle du contexte. Le niveau AUI comporte un seul modèle, de même que le niveau CUI. Le niveau de l'interface finale correspondant au code pour la plate-forme finale, il n'a donc pas de modèle retenu.

Ceci étant dit, nous n'expliciterons cependant pas tout le formalisme et l'entièreté de la syntaxe utilisée pour définir les différents modèles considérés. Nous nous limiterons aux seuls modèles de la tâche et du domaine que nous utiliserons pour notre application finale (notre modèle de la présentation sera explicité plus tard et déterminé par analogies avec QTk) et nous attirerons également l'attention sur la propriété de *mapping* entre modèles. Pour de plus amples informations sur les différents modèles et leur syntaxe précise, nous invitons le lecteur à consulter le document de référence *[USIXML]*.

5.4.2.1 le modèle de tâche

Le formalisme utilisé pour décrire le modèle de tâche se base sur CTT (ConcurTaskTree), et ceci pour plusieures raisons, parmi lesquelles :

- CTT est plus orienté vers le software engineering que les méthodes d'analyse psycho-cognitive
- CTT a un ensemble riche d'opérations temporelles (basé sur le standard ISO LOTOS)
- CTT est manipulable via un outil graphique facilitant sa communication et dissémination entre professionnels

Concrètement, un modèle de tâche CTT consiste en une structure de tâches hiérarchique graphiquement représentable et incluant les éléments suivants :

- un identificateur (et un nom) : chaque tâche se voit attribuer un identifiant unique et un nom (sans utilité réelle).

- une catégorie : il existe quatre catégories de tâche, suivant l'acteur qui va la réaliser : utilisateur, système, abstraite et interactive. Une tâche utilisateur consiste en une tâche « cognitive », faite par l'utilisateur. Une tâche système apparaît typiquement lorsque le système est chargé d'effectuer une opération (calcul, …). Les tâches abstraites sont des tâches complexes décomposables en plusieurs sous-tâches de type différent. Et enfin, les tâches interactives sont réalisées par l'utilisateur lorsqu'il interagit avec le système (ex : sélection d'un élément, édition d'un champ, déclenchement d'une action en cliquant sur un bouton).
- un type : nous venons de voir que l'attribution d'une tâche à un acteur déterminait la catégorie de cette dernière. Néanmoins, dans chaque catégorie, il existe différents types de tâches possibles. Malheureusement, il n'existe pas de liste de types de tâches dans l'environnement CTT. Cependant, considérer plusieurs types de tâches pour la catégorie des tâches interactives peut se révéler très importante, notamment si nous considérons la sélection de widget en fonction des informations sur le modèle de la tâche. Cela est moins important pour les tâches système et utilisateur car elles ne font aucune référence à des éléments de présentation. Aussi, nous utiliserons la typologie hiérarchique explicitée dans *[Florins 2003]*.

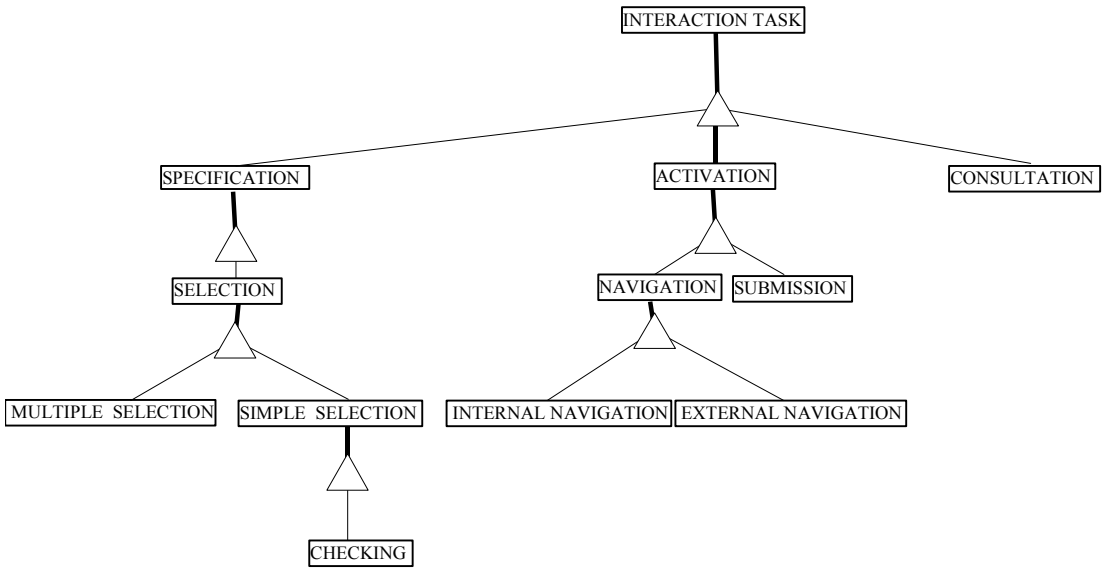


Figure 32 – La Typologie des tâches interactives

Définitions

- Une *tâche interactive* est une tâche réalisée par l'utilisateur interagissant avec le système dans le but de consulter et/ou modifier l'état du système.
- Une tâche de *spécification* se rapporte à la communication d'information au système par l'utilisateur (sélection, encodage de texte, …).
- Une tâche de *sélection* est une tâche de spécification où l'utilisateur a la possibilité de choisir un élément parmi plusieurs dans les options proposées.
- Une tâche de *sélection multiple* indique que l'utilisateur a la possibilité de choisir plusieurs éléments parmi les options proposées.
- Une tâche de *sélection simple* indique que l'utilisateur a la possibilité de choisir un et un seul élément parmi les options proposées.

- Une tâche de *contrôle* constitue le cas extrême où une seule option est proposée à l'utilisateur, et celui-ci n'a d'autre possibilité que d'indiquer s'il est d'accord ou pas pour l'appliquer.
- Une tâche d'*activation* consiste à appeler une fonction sémantique du système.
- Une tâche de *navigation* est une tâche d'activation qui place une autre section de l'information visible dans la présentation courante (ex : toute la fenêtre courante, une partie de la fenêtre) sans modifier l'état du système.
- Une tâche de *navigation interne* est une tâche de navigation gérée dans la même unité de présentation (ex : un lien HTML faisant référence à un autre endroit au sein de la même page).
- Une tâche de *navigation externe* est une tâche de navigation gérée entre deux unités de présentation distinctes (ou encore entre unités de présentation d'applications différentes).
- Une tâche de *soumission* est une tâche d'activation qui permet d'envoyer de l'information spécifiée préalablement par l'utilisateur (ex : soumission d'un formulaire web)
- Une tâche de *consultation* est une tâche interactive ne modifiant ni l'état du système, ni la présentation.

D'autres concepts importants inclus dans CTT sont :

- la fréquence : la fréquence d'une tâche (faible, moyenne élevée) est utile car elle nous indique par quel degré de facilité d'utilisation et de mise en évidence elle devra être caractérisée.
- la performance temporelle : indique le temps nécessaire pour accomplir cette tâche (minimale, moyenne, maximale).
- les relations hiérarchiques : chaque tâche CTT peut être décomposée en deux ou plus de tâches. Ceci implique donc, qu'à l'exception de la tâche *racine*, chacune des tâches possède une tâche mère de laquelle elle hérite des propriétés temporelles (cfr. relations temporelles).
- les relations temporelles : les relations temporelles entre tâches sont spécifiées par des opérateurs temporels qui peuvent être de deux types : unaires ou binaires.

Il y a trois opérateurs unaires possibles :

- l'opérateur d'*itération*, noté T*, indiquant qu'une tâche peut être répétée plusieurs fois jusqu'à ce que d'autres tâches la désactive.
- l'opérateur d'*itération finie*, noté T(n), lorsque l'utilisateur connaît à l'avance le nombre de fois que la tâche sera lancée.
- L'opérateur *optionnel*, noté [T], indiquant que l'utilisateur n'est pas obligé de réaliser la tâche T.

Il y a également huit opérateurs binaires possibles entre deux tâches T1 et T2:

- la *concurrence indépendante* (T1 ||| T2) : T1 et T2 peuvent être effectuées dans n'importe quel ordre et suivant n'importe quelles contraintes. (ex : remplissage des champs 1 et 2 dans un formulaire).
- la *concurrence avec passage d'information* (T1 |[]| T2) : T1 et T2 peuvent être effectuées dans n'importe quel ordre mais doivent se synchroniser dans le but de

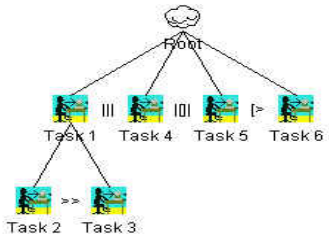
s'échanger de l'information. (ex : remplissage des champs 1 et 2 dans un formulaire où ceux-ci doivent assurer une certaine cohérence (ex : ville et code postal))

- le *choix* (T1 [] T2) : l'utilisateur a le choix de lancer soit T1 ou T2, mais pas les deux. Une fois que le choix est effectué, il n'est plus possible de déclencher l'autre, à moins que celle-ci vienne de se terminer et que la hiérarchie des tâches fasse en sorte que le choix soit à nouveau possible (ex : ouvrir un nouveau document ou un document existant dans une application quelconque).
- l'*indépendance ordonnée* (T1 |=| T2) : cette opération indique que l'utilisateur a le choix concernant la tâche à accomplir en premier mais quoiqu'il arrive, ces deux tâches doivent être réalisées. Dans le cas où l'utilisateur choisit T1, il devra obligatoirement accomplir T2 lorsque T1 sera terminée.
- la *désactivation* (T1 [> T2) : T1 est définitivement interrompue lorsque T2 ou la première sous-tâche de T2 a été lancée (ex : l'envoi d'un formulaire désactive toutes les tâches possibles pour le remplissage de ce formulaire).
- la *mise en suspend* (T1 |> T2) : T1 est interrompue temporairement lorsque T2 ou sa première sous-tâche est lancée. Ensuite, lorsque cette dernière a été effectuée, T1 reprend son cours. (ex : un message de feed-back indiquant que l'e-mail envoyé est arrivé à son destinataire)
- l'*activation* (T1 >> T2) : indique un ordre séquentiel entre T1 et T2. T2 ne sera accessible que lorsque T1 aura été effectuée et complètement terminée. (ex :l'introduction d'un mot de passe avant d'accéder aux services fournis).
- L'*activation avec passage d'information* (T1 []>> T2) : T1 active T2 et lui fournit certaines informations. (ex : une requête à une base de données (T1 permet de spécifier la requête et T2 affiche les résultats que l'utilisateur pourra ensuite consulter)).

Toutes ces opérateurs unaires et binaires ne peuvent être mis sur un même pied d'égalité. Certains sont, en effet, plus prioritaires que d'autres. Voici le classement qu'on peut leur donner, par ordre décroissant de priorité :

Figure 33 – Ordre de priorité des opérateurs temporels dans CTT

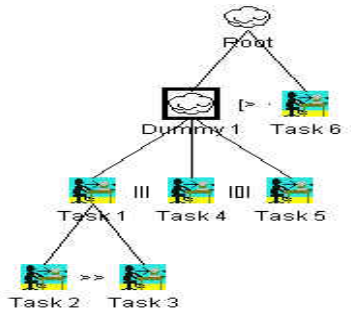
Aussi, en regard de ces différences de priorités, il sera sans aucun doute utile de ne travailler qu'avec des modèles de tâches *respectant cette priorité* par niveau. En effet, il pourrait être assez compliqué de faire des manipulations sur un arbre non prioritaire, notamment si nous devons diviser cet arbre en plusieurs parties tout en respectant la priorité des tâches entre elles. Or, conceptuellement, un arbre prioritaire est tout à fait équivalent à un arbre non prioritaire : le premier sera en fait le deuxième complété par des tâches factices comme dans l'exemple suivant :



Voici notre modèle de tâche CTT normal.

Et voici le même arbre, cette fois prioritaire, dans lequel nous aurons rajouté une tâche factice.

Cette technique de transformation, introduite par *[Luyten & al. 2003]* sera particulièrement utile par la suite.



Les dernières propriétés à souligner à propos de CTT sont :

- la possibilité de *tâches récursives* : pour ce faire, il suffit d'utiliser le même nom pour deux ou plus de tâches. La tâches appelée sera celle qui comporte des sous-tâches et/ou est située plus haut dans la hiérarchie complète. La/Les tâches appelantes seront les autres.
- l'*ordonnancement graphique* : la syntaxe graphique du modèle CTT détermine un ordre graphique entre tâches liées par un même opérateur binaire. Cet ordonnancement graphique est qualifié de symétrique, dans le sens où il lie des tâches voisines qui peuvent être effectuées selon deux sens de lecture. Remarquons que cette propriété nous est très utile parce qu'elle donne déjà une idée de l'agencement des tâches pour la présentation de l'interface utilisateur.

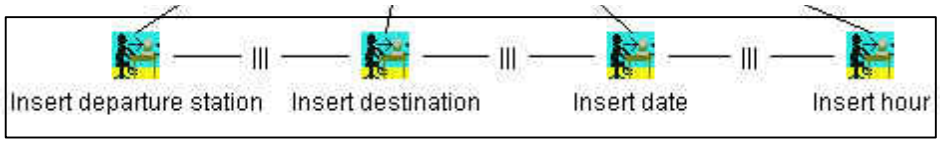


Figure 34 – Ordonnancement graphique de tâches concurrentes

- la propriété de *mapping* avec d'autres modèles : ceci sera développé dans le point 4.4.2.3.

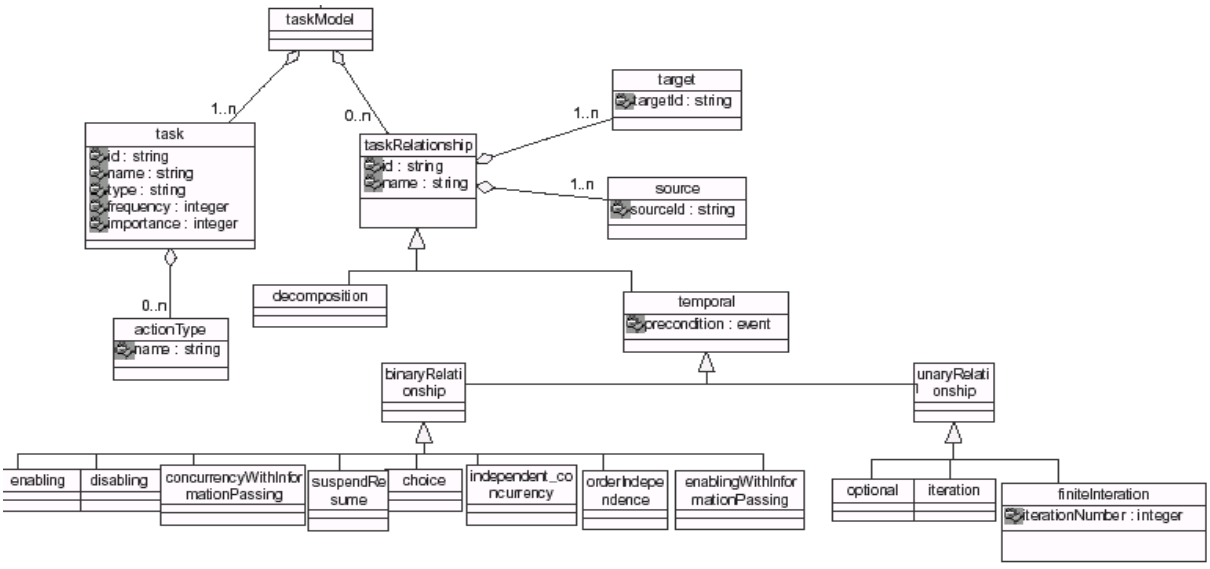


Figure 35 – Modèle de tâche complet

5.4.2.2 le modèle du domaine

Le formalisme utilisé pour décrire le modèle du domaine se base sur les diagrammes de classe en UML. Les concepts principaux à retenir des diagrammes de classes sont :

- les *classes* qui décrivent des groupes d'objets possédant des caractéristiques (attributs) et comportements (opérations) similaires.
- les *attributs* comportant : un nom, une multiplicité (maximale et minimale), un type (qui peut être défini par l'utilisateur lui-même), une valeur par défaut, ainsi que les propriétés éventuelles sur cette valeur (type énuméré, …).
- les *opérations* comportant : un nom et une signature (la liste des paramètres utilisés ainsi que leur type et le type de résultat renvoyé)
- les *relations inter-classes* : ces relations portent le nom d'association. Une association a la plupart du temps un nom, des attributs et des rôles (suivant leur multiplicité). Certaines associations portent un nom plus spécifique, comme l'agrégation, la composition et la généralisation car elles possèdent quelques propriétés. La généralisation permet de structurer des classes suivant une hiérarchie de super-classes (plus générales) et sous-classes (plus spécifiques). Ainsi, une sous-classe hérite des attributs et opérations de sa super-classe, en plus de ses propres caractéristiques. L'agrégation est une relation entre un ensemble de classes et une de ses classes (ex : classe *voiture* et classe *volant*). La composition consiste en une relation d'agrégation plus forte : les parties composées sont entièrement dépendantes de l'ensemble et ne peuvent exister sans lui.

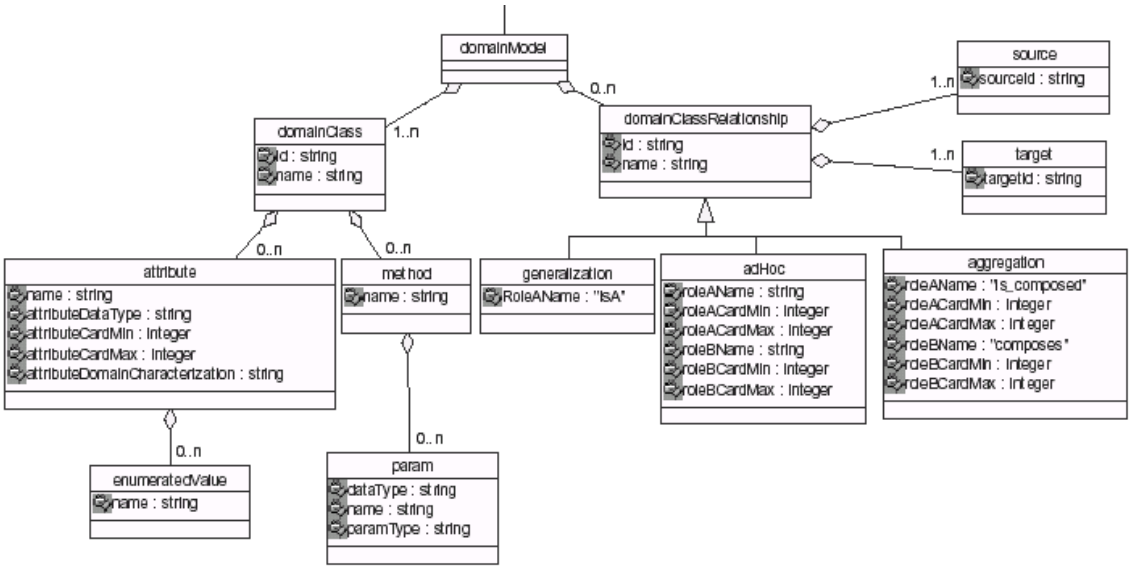


Figure 36 – le modèle du domaine complet

5.4.2.3 le mapping entre modèles

Plutôt que de disposer d'une collection importante de modèles sans rapport entre eux, il peut être utile de créer des liens et des relations entre modèles et entre composants de modèles différents. Ceci permet, par exemple, de préciser la manière dont devront être réalisées certaines tâches (nature des éléments du domaine manipulé, objet interactif concret souhaité,…). Ainsi, nous pouvons ainsi définir plus précisément certains contextes d'utilisation et nous assurer de la traçabilité au long du cycle de développement.

USIXML fournit ainsi un ensemble de relations prédéfinies permettant de créer des liens entre éléments de modèles différents. Ces *mappings* peuvent être de plusieurs types (nous nous contenterons de ceux que nous utiliserons par la suite) :

- *Manipulates* : lie une tâche à un concept du domaine : une classe, un attribut, une opération, ou une combinaison de plusieurs types. Cette relation a un attribut 'centrality' qui spécifie l'importance relative d'un élément du domaine dans l'exécution de sa tâche. Cet attribut est évalué sur une échelle de un à cinq. Un signifiant que ce concept de domaine n'est pas central, Cinq qu'il est réellement nécessaire à l'exécution de la tâche.
- *Is Executed In* : lie une tâche et un objet interactif abstrait ou concret. Il indique que la tâche sera exécutée à travers l'utilisation de cet (ensemble d') objet(s) interactif(s).

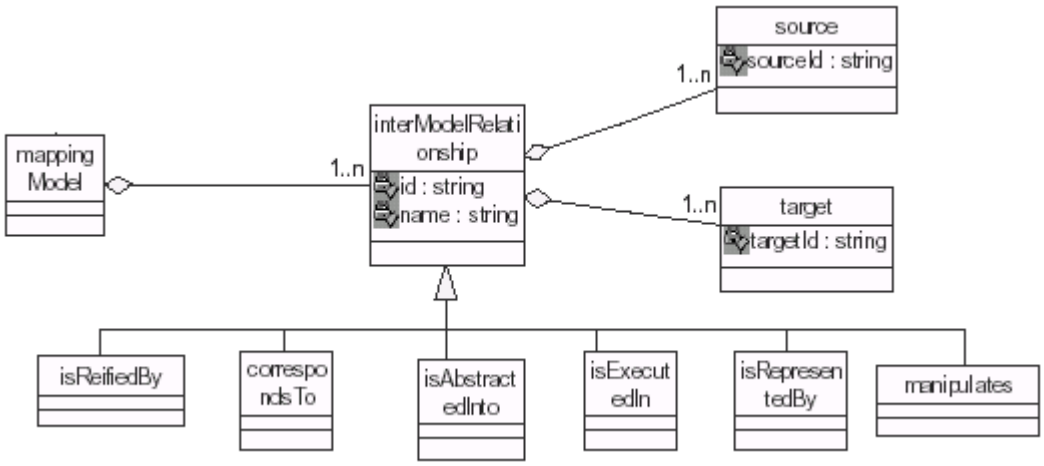


Figure 37 – Modèle du *mapping* complet

5.4.3 Règles de dégradation harmonieuse par niveau d'abstraction

Nous allons présenter dans ce point du chapitre les quelques règles de dégradation harmonieuse que nous retiendrons pour notre application finale. Nous n'expliciterons cependant pas toutes les règles de dégradation déjà répertoriées. Pour de plus amples informations à ce sujet, nous invitons le lecteur à consulter *[Florins 2003]* et *[Florins & Vanderdonckt 2004]*. Chacune des règles choisies sera classée suivant le niveau d'abstraction (au sens du *framework* Cameleon) auquel elle se rapporte.

5.4.3.1 Règles de transformation au niveau de l'interface abstraite

Concepts

Interface utilisateur abstraite : une interface utilisateur abstraite (AUI) consiste en un ensemble d'*unités de présentation* et de transitions entre ces unités de présentation. Ce niveau d'abstraction est indépendant du point de vue des objets interactifs utilisés.

Unité de présentation : une unité de présentation (PU) est un ensemble de tâches logiquement liées qui seront effectuées dans la même présentation (fenêtre, panneau, boîte de dialogue, ..). Les tâches contenues dans l'unité de présentation seront arrangées selon l'ordre décrit par l'arbre du modèle des tâches. Les tâches retenues pour chaque unité de présentation sont les *tâches feuille* (ce sont les seules qui peuvent être directement déclenchées dans la présentation concrète) qui sont interactives (aucune tâche utilisateur ou système n'est représentée dans la présentation concrète. De plus, les tâches système affichant un résultat peuvent être remplacées par des tâches interactives de consultation).

Définitions

Les définitions suivantes vont nous permettre de préciser la notation que nous utiliserons pour nos règles de transformation à ce niveau :

Definition 1 Une *relation temporelle* entre deux tâches t_i et t_j est un triplet $r = (t_i, t_j, x)$ où $x \in \mathcal{TO} = \{ [], |=|, |||, |[]|, [>, |>, >>, []>> \}$.

Definition 2 Une *relation temporelle unaire* est une relation temporelle dans le cas où $t_i = t_j$

Definition 3 Une *relation temporelle binaire* est une relation temporelle dans le cas où $t_i \neq t_j$

Definition 4 Une *relation temporelle symétrique* est une opération temporelle binaire $r = (t_i, t_j, x) \mid (t_j, t_i, x)$ a la même sémantique. Les *opérateurs temporels symétriques* sont {[], |=|, |||, |[]|}

Definition 5 Une *relation temporelle asymétrique est une relation temporelle binaire* $r = (t_i, t_j, x) \mid (t_j, t_i, x)$ n'a pas la même sémantique. Les *opérateurs temporels assymétriques* sont {[>, |>, >>, []>>}

Definition 6 Un *modèle de tâche* $\mathcal{TM}$ est une structure composée d'un arbre ordonné dirigé où les nœuds sont les tâches et de relations temporelles entre nœuds du même *niveau de* décomposition.

Definition 7 Un *arbre prioritaire* d'un modèle de tâche $\mathcal{TM}$, noté $\mathcal{PTM}$, est un arbre ayant la même sémantique que $\mathcal{TM}$ mais où les relations temporelles d'un même niveau dans la hiérarchie des tâches ont la propriété d'avoir le même opérateur temporel

Definition 8 Le *niveau* d'une tâche dans un modèle de tâches ou un arbre prioritaire est le nombre d'ancêtres que comporte cette tâche.

Some functions have to be defined on the tree structures (task tree and priority tree):

Definition 9 *mother-of*: tâche → tâche, renvoie la tâche mère d'une tâche.

Definition 10 *children-of*: tâche → liste de tâches, renvoie la liste (ordonnée) des fils d'une tâche

Definition 11 *right-siblings-of*: tâche → liste de tâches, renvoie la liste (ordonnée) des tâches voisines se trouvant à droite de la tâche sélectionnée.

Definition 12 *left-siblings-of*: tâche → liste de tâches, renvoie la liste (ordonnée) des tâches voisines se trouvant à gauche de la tâche sélectionnée

Definition 13 *first*: tâche → liste de tâches, renvoie la liste des filles d'une tâche qui peuvent être les premières à effectuer lorsque la tâche est exécutée (i.e. qui apparaît avant la première occurrence d'un opérateur >>, []>>, [> ou |>)

Objectif

Le but des règles de transformation à ce niveau sera de permettre à la présentation finale d'être affichée sur plusieures fenêtres lorsque l'écran devient plus petit. Ainsi, si nous disposons d'une seule unité de présentation au départ, nous aurons alors plusieurs unités de présentation après transformation.

Cette transformation implique donc que les tâches de l'unité de présentation initiale seront redistribuées en plusieurs unités de présentations séparées. Or, cette redistribution ne doit pas se faire n'importe comment : il faut respecter la priorité qui découle des opérateurs temporels entre les tâches du modèle. Nous devrons donc analyser les différentes types de scission possibles pour les opérateurs binaires entre tâches et l'implication de telles modifications sur le modèle source.

5.4.3.1.1 La découpe en unités de présentation au niveau des tâches séquentielles

Exemple simple

Les tâches séquentielles sont liées par les opérateurs d'activation et d'activation avec passage d'information (>> et []>>). Nous allons montrer, via un exemple simple, comment les tâches séquentielles peuvent être réparties dans des différentes unités de présentation.

Soit l'arbre de tâche suivant, modélisant un système simple de consultation dans une bibliothèque pour déterminer si des exemplaires de livre sont disponibles. Celui-ci est composé de trois tâches séquentielles : la première, interactive, permet à l'utilisateur d'encoder/sélectionner le livre recherché ; la seconde est une tâche système qui traduit la recherche d'information ; la dernière permet à l'utilisateur de consulter les résultats obtenus.

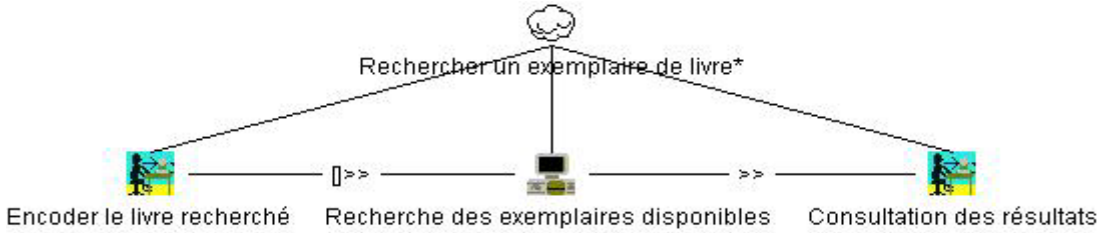


Figure 38 – Exemple de tâches séquentielles

Nous avons plusieurs cas possibles de répartition entre unités de présentation :

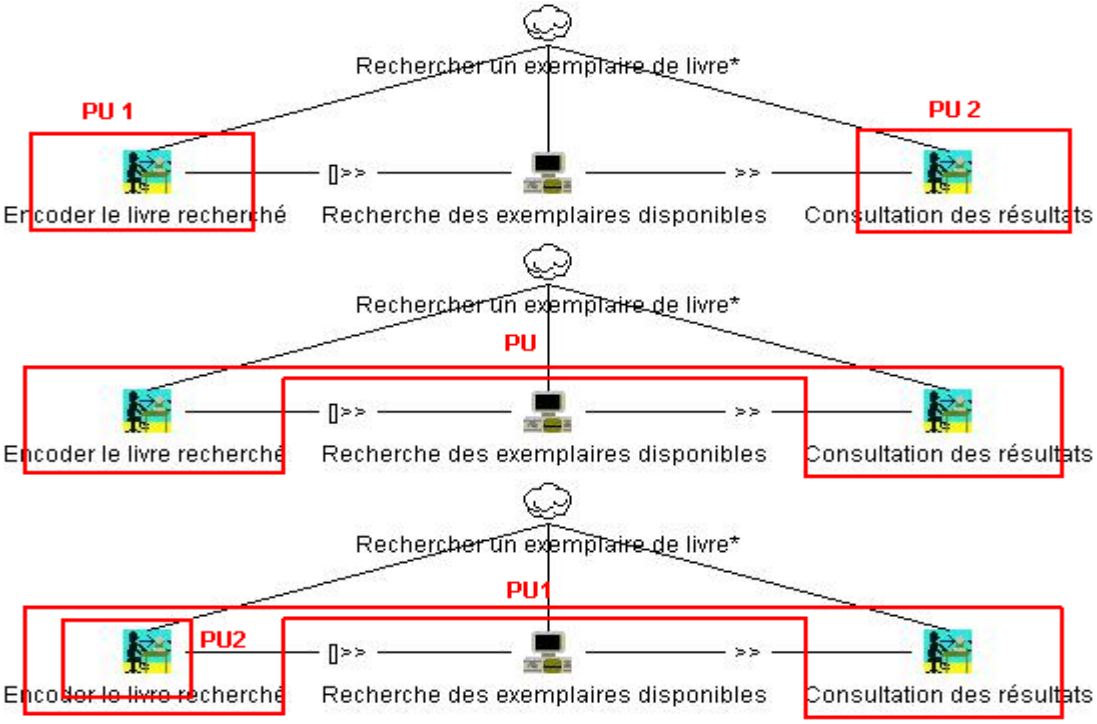


Figure 39 – Différentes distributions possibles de tâches entre unités de présentation

Ainsi, après analyse, nous pourrions donc dire qu'il existe trois types de transformation applicables à une unité de présentation initiale lorsque nous coupons notre arbre de tâche au niveau des opérateurs séquentiels.

Soit une unité de présentation source $PU_{source} = (t_1,\ldots, t_n)$

$\forall\ i \in [1..n\text{-}1]$: If $\forall\ r \in TM : r = (t_i, t_{i+1}, o) \rightarrow o \in \{>>, []>>\}$

1. *la découpe simple (sans redondance)*

$PU_{source} \rightarrow PU_{target1} = (t_1,\ldots, t_i) \wedge PU_{target2} = (t_{i+1},\ldots, t_n)$

2. *la découpe avec redondance à droite*

$PU_{source} = (t_1,\ldots, tn) \rightarrow PU_{target1} = (t_{1,\ldots,} t_i) \wedge PU_{target2} = (t_{1,\ldots,} t_n)$

3. *la découpe avec redondance à gauche*

$PU_{source} = (t_1,\ldots, tn) \rightarrow PU_{target1} = (t_{1,\ldots,} t_n) \wedge PU_{target2} = (t_{i+1,\ldots,} t_n)$

Néanmoins, cette règle ne suffit pas pour spécifier entièrement toutes les réorganisations possibles. En effet, nous n'avons considéré qu'un cas extrêmement simple ne prenant en compte que l'unique opération d'*activation*. Lorsque d'autres opérateurs se rencontrent dans le modèle de la tâche, la réorganisation entre PU s'avère totalement différente. Nous allons expliciter ces différents cas et la manière d'effectuer la réorganisation entre unités de présentation.

La découpe de tâches séquentielles situées dans la portée d'un opérateur d'interruption

Lorsque nous découpons une unité de présentation au niveau d'un opérateur séquentiel, si les tâches séquentielles sont situées dans la portée d'un opérateur d'interruption, nous devons distribuer la tâche d'interruption parmi toutes les unités de présentation. A partir de ce moment, nous utiliserons des arbres prioritaires.

Soit l'unité de présentation suivante $PU_{source} = (t_1,\ldots, t_n)$, nous avons donc :

$\forall\ i \in [1..n\text{-}1] : \exists\ r_1 \in \mathcal{PTM} \mid r_1 = (t_i, t_{i+1}, o) \wedge o \in \{>>, []>>\} \wedge \exists\ j \in [i+2..n\text{-}1]\ \exists\ r_2 \in \mathcal{PTM} \mid (\text{mother-of}(t_i), t_j, [>)$: $PU_{source} \rightarrow PU_{target1} = \text{Concaténation}[((t_1,\ldots, t_i), \text{right-siblings}(\text{mother-of}(t_i))] \wedge PU_{target2} = (T_{i+1},\ldots, T_n)$

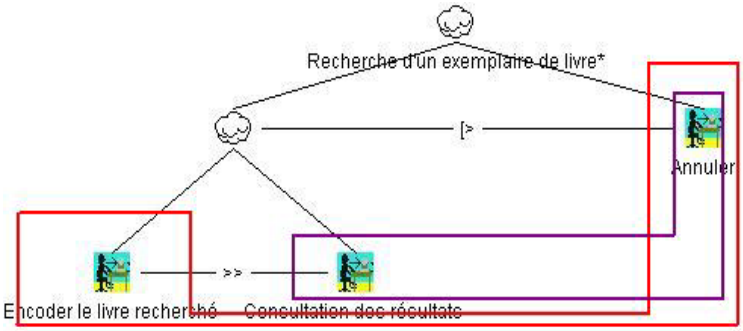


Figure 40 – Répartition des tâches entre PUs de tâches séquentielles situées dans la portée d'un opérateur d'interruption

La découpe de tâches séquentielles situées dans la portée d'opérateurs concurrentiels

Lorsque nous découpons une unité de présentation au niveau d'un opérateur séquentiel, si les tâches séquentielles sont situées dans la portée d'un ou plusieurs opérateurs de concurrence (||| ou ||[]||), nous devons distribuer la tâche d'interruption parmi toutes les unités de présentation.

Puisque nous travaillons avec un arbre prioritaire, si toutes les tâches séquentielles situées dans la portée d'un opérateur concurrentiel se trouve au niveau L dans l'arbre, cela veut donc dire que les tâches du niveau L+1 sont concurrentes et que celles-ci doivent être distribuées parmi les unités de présentations.

$\forall\ i \in [1..n\text{-}1]: \exists\ r_1 \in \mathcal{PTM} \mid r_1 = (t_i, t_{i+1}, o) \wedge o \in \{>>, []>>\}$

Nous appliquons les étapes suivantes:

1) Scission simple au niveau de l'opérateur séquentiel : $PU_{source} \rightarrow PU_{target1} = (t_1, t_i) \wedge PU_{target2} = (t_{i+1}, t_n)$.

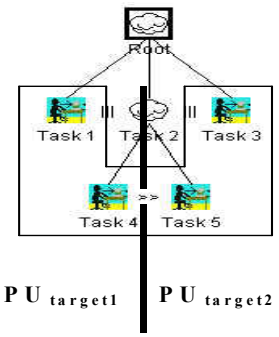


Figure 41 – Première étape de découpe de tâches séquentielles situées dans la portée d'opérateurs concurrentiels

2) Distribution des tâches concurrentes situées à ga. de la tâche mère des tâches séquentielles :

$\exists\, h \in [1..i\text{-}1]\,,\ o_1 \in \mathcal{TO}\,,\ r_1 \in \mathcal{PTM}\ |\ r_1 = (t_h,\ \text{mother-of}(t_i),\ o_1) \wedge o_1\ \in \{|||,\ |[]|\}$:
$PU_{target2} \leftarrow$ Concaténation[(left-siblings(mother-of(t_i)), $PU_{target2}$)]

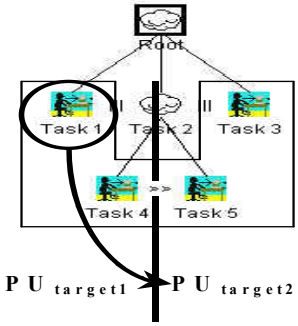


Figure 42 – 2ème étape de découpe de tâches séquentielles situées dans la portée d'opérateurs concurrentiels

3) Distribution des tâches concurrentes situées à dr. de la tâche mère des tâches séquentielles :

$\exists\, j \in [i+2,\ n],\ o_2 \in \mathcal{TO}\,,\ r_2 \in \mathcal{PTM}\ |\ r_2 = (\text{mother-of}(t_i),\ t_j,\ o_2) \wedge o_2\ \in \{|||,\ |[]|\}$:
$PU_{target1} \leftarrow$ Concaténation[($PU_{target1}$, right-siblings(mother-of(t_i)))]

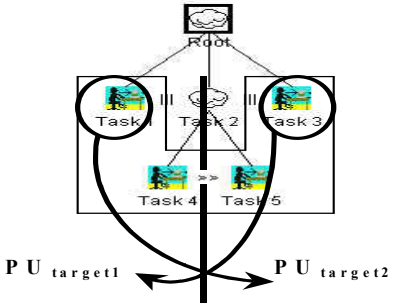


Figure 43 – 3ème étape de découpe de tâches séquentielles situées dans la portée d'opérateurs concurrentiels

5.4.3.1.2 La découpe en unités de présentation au niveau des tâches concurrentes

Exemple simple

Les tâches concurrentes sont liées par les opérateurs de concurrence et de concurrence avec passage d'information (||| et |[]|). Nous allons montrer, via un exemple simple, comment les tâches concurrentes peuvent être réparties dans des différentes unités de présentation. Reprenons notre exemple de recherche d'exemplaires de livres disponibles dans une bibliothèque. Nous rajoutons quatre sous-tâches à « Encoder le livre recherché », toutes concurrentes : insérer le titre du livre, insérer le nom de l'auteur, sélectionner la catégorie à laquelle se rapporte le livre (sciences, économie,…), insérer l'année de publication.

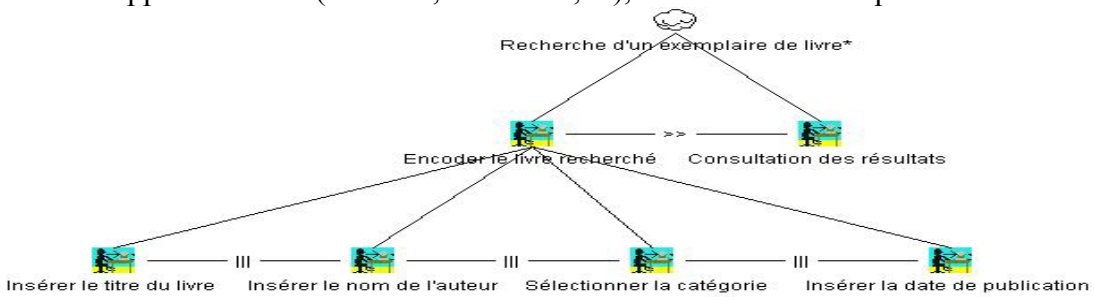


Figure 44 - Exemple de tâches concurrentes

Nous allons nous focaliser sur les tâches concurrentes. Plusieurs cas de distributions de ces tâches par unité de présentation sont possibles, en voici quelques-unes :

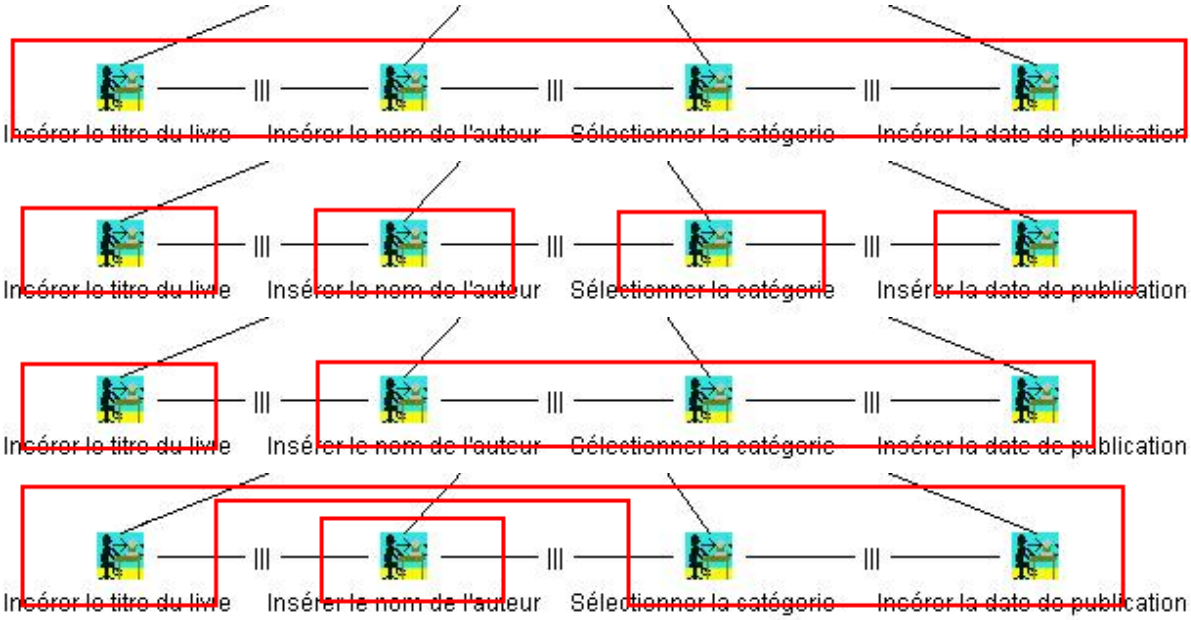


Figure 45 – Différentes distributions possibles de tâches concurrentes entre unités de présentation

Etant donné $PU_{source} = (t_1,\ldots, tn)$, notre règle de répartition devient alors celle-ci :

$$\forall\, i \in [1..n-1] : \forall\, r \in \mathcal{TM} : r = (t_i, t_{i+1}, o) \rightarrow o \in \{|||, |[]|\} : PU_{source} \rightarrow PU_{target1} = (t_1, t_i) \wedge PU_{target2} = (t_{i+1}, tn)$$

Des Variantes sont possibles:

$$PU_{source} = (t_1,\ldots, tn) \rightarrow PU_{target1} \wedge PU_{target2} \mid (PU_{target1} \cup PU_{target2} = PU_{source}) \wedge (PU_{target1} \cap PU_{target2}) = \varnothing$$

Dans ce dernier cas, nous considérons que l'ordre graphique entre tâches concurrentes ne doit pas forcément être respecté. On peut même supprimer la dernière contrainte $(PU_{target1} \cap PU_{target2}) = \varnothing$, si nous considérons la duplication de tâches concurrentes entre plusieurs unités de présentation comme permise.

Conséquence sur le niveau abstrait des Tâches et Concepts

Ces transformations ont des conséquences sur le modèle de tâche. En effet, en répartissant des tâches entre unités de présentation, celles-ci perdent leur propriété de tâches concurrentielles. Ainsi, nous avons deux cas possibles :

- Ajout d'un opérateur d'*activation* entre les deux sous-ensembles obtenus : nous insérons un nouveau niveau dans l'arbre entre les tâches concurrentes et la tâche mère. Ensuite, nous créons deux tâches abstraites dans ce niveau reliées par un opérateur d'activation. Après quoi, nous leur donnons le rôle de mère de chacun des sous-ensembles obtenus après scission.

- Insertion de la possibilité de choix entre les deux sous-arbres : nous insérons deux nouveaux niveaux dans l'arbre entre les tâches concurrentes et la tâche mère. Ensuite, nous créons deux tâches abstraites A et B dans le niveau le plus bas, reliées par un opérateur d'indépendance. Nous insérons une tâche interactive qui jouera le rôle de sélection entre A et B et une tâche abstraite factice qui sera la mère de A et B, reliées par un opérateur d'activation avec passage d'information.

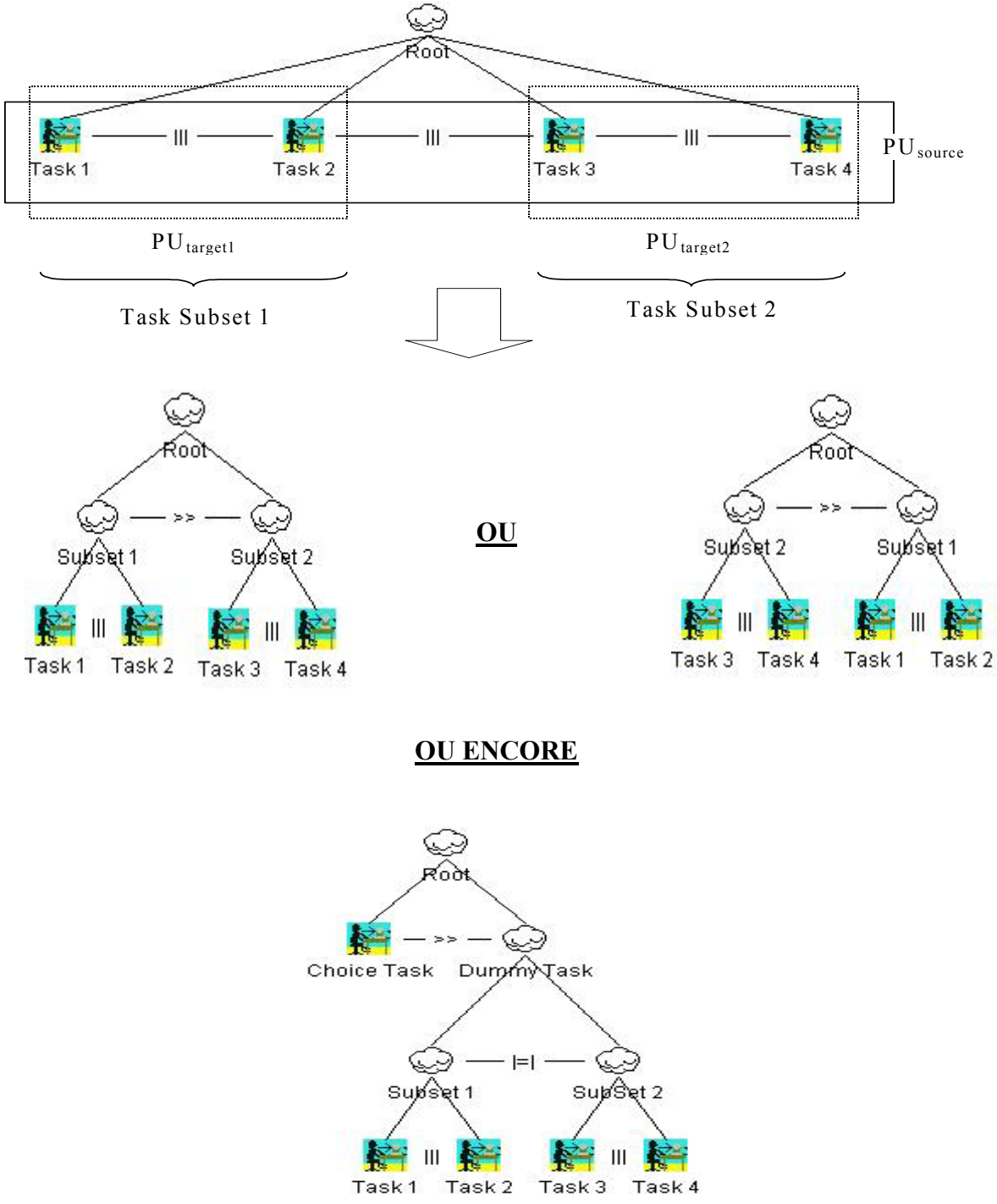


Figure 46 – Conséquences de la répartition en PU pour les tâches concurrentes

Au même titre que pour la scission avec les opérateurs séquentiels, le cas n'est pas aussi simple qu'il semble l'être. Nous allons donc améliorer notre règle de scission en considérant plus d'opérateurs dans notre arbre.

La découpe de tâches concurrentes combinées avec des tâches séquentielles

Nous avons déjà abordé ce sujet dans la partie faisant référence à la scission de tâches séquentielles. L'unité de présentation sera donc d'abord découpée au niveau des tâches séquentielles et ensuite, nous continuerons la découpe au niveau des tâches concurrentes.

La découpe de tâches concurrentes combinées avec des tâches d'interruption

Dans ce cas précis, nous procéderons, à l'instar de la technique pour les tâches séquentielles, en répartissant dans toutes les unités de présentations les tâches d'interruption ([> ou |>).

La découpe de tâches concurrentes combinées avec d'autres tâches

D'autres opérateurs temporels sont possibles : notamment celui du choix ([]) et celui de l'indépendance (|=|). Cependant, comme ces deux opérateurs possède une priorité plus faible que les opérateurs de concurrence, ils n'auront aucune influence sur la manière de répartir les tâches entre unités de présentation.

5.4.3.2 Règles de transformation au niveau de l'interface concrète

Concepts

Une *interface utilisateur concrète* consiste en une description de l'interface utilisateur en termes d'*objets interactifs abstraits* et leur *relations de placement* dans l'espace.

Différents types de règles peuvent contribuer à la dégradation harmonieuse à ce niveau d'abstraction, et les changements seront donc appliqués sur les objets interactifs même, ou leurs relations de placement.

5.4.3.2.1 Règles appliquées aux objets interactifs

Règles de substitution

Une règle de substitution remplace un objet interactif par un autre objet interactif qui assure les même fonctionnalités. Une règle de substitution peut être réalisée pour deux principales raisons :

- *l'indisponibilité* : lorsque un objet interactif n'est plus disponible sur la plate-forme destination, il doit être remplacé par un autre qui s'y rapproche. Par exemple, les boîtes à cocher et boutons radio n'existe pas dans le langage WML sur les téléphones portables, et sont alors remplacés par des listes.

- L'*inadéquation de la taille d'écran* : si un objet interactif ne convient pas à la plate-forme cible parce qu'il prend trop de place, il doit également être remplacé. La figure suivante montre les substitutions possibles pour un accumulateur. L'accumulateur peut être remplacé par une version simplifiée du même objet (les libellés des boutons ont été raccourcis) ou encore par l'utilisation d'autres objets interactifs supportant la sélection multiple (typiquement une liste de sélection, une liste de boîte à cocher, une liste résumée à un seul élément). Un exemple similaire est exposé sur la figure suivante pour la sélection simple.

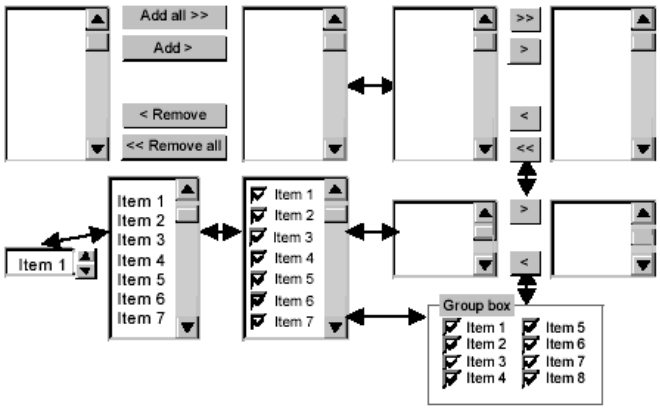


Figure 47 – Objets interactifs susceptibles de convenir à la sélection multiple

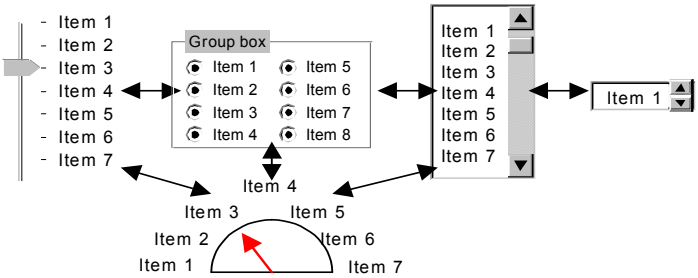


Figure 48 – Objets interactifs susceptibles de convenir à la sélection simple

Différent types de substitution peuvent être réalisés :

- *la simple substitution* (1⇒1) : l'objet interactif X provenant de la plate-forme source est remplacé par un objet interactif de la plate-forme destination (ex : un bouton est remplacé par un item de menu)
- *le regroupement* (N⇒1) : l'ensemble d'objets interactifs de la plate-forme source sont remplacés par un objet interactif unique (ex : les boîtes à cocher et l'accumulateur de la figure 47)
- *le découpage* (1⇒N) : un seul objet interactif de la plate-forme source est remplacé par un ensemble d'objets interactifs de la plate-forme destination. (ex : différents champs d'édition indiquant respectivement le jour, le mois et l'année sont remplacés par un calendrier)

Notons que toutes les alternatives n'ont pas forcément la même qualité ergonomique :

- tous les objets interactifs ne sont pas forcément facile à utiliser en fonction de la plate-forme sur laquelle nous nous trouvons. (ex : les boîtes à cocher sont plus difficiles à manipuler sur un écran tactile)
- certains objets interactifs satisfont mieux au critère de guidage (ex : un accumulateur indique clairement que la sélection multiple est autorisée, tandis qu'une liste de sélection le dénote moins bien (il faut parfois appuyer sur une touche supplémentaire pour sélectionner plusieurs éléments))
- certains objets interactifs sont plus appropriés en vue du nombre de valeurs qu'il faudra manipuler (ex : un groupe de boîtes à cocher devrait être normalement limité à sept éléments pour assurer et optimiser la lisibilité)

Règles de suppression

Lorsque les règles de substitution ne satisfont pas parce que l'espace est trop limité, il peut être souhaitable, dans le cas extrême, de recourir à la suppression d'objets interactifs. C'est notamment le cas en ce qui concerne les images sur les GSMs.

5.4.3.2.2 Règles appliquées au niveau des relations de placement

Règles de repositionnement

Les règles de repositionnement modifient la localisation d'un objet interactif (sa position dans la fenêtre) en terme de coordonnées ou de relations géométriques avec d'autres objets interactifs (alignement, …)

Elles sont très utiles dans ne nombreux cas, notamment lorsque :

- les composants ne conviennent pas à l'écran dans une des deux dimensions (ils dépassent l'espace autorisé horizontalement ou verticalement) et qu'il reste de l'espace dans l'autre dimension.
- les composants ne conviennent pas horizontalement et que nous voulons éviter le scrolling latéral.
- certaines règles ergonomiques doivent être respectées sur la plate-forme cible (les menus doivent être par exemple placés dans le haut de la fenêtre)

Un bon exemple de règle de déplacement est illustré dans Skweezer (produit de GreenLight Wireless Corporation). Cette application permet de reformater les pages Web afin d'éviter un scrolling horizontal et d'avoir un contenu lisible lorsque des personnes veulent consulter un site Web sur des dispositifs comme les PDAs.

Règles de redimensionnement

Cette règle consiste à agrandir ou le plus souvent, réduire la taille et donc l'espace pris par n'importe quel composant d'une interface utilisateur. Théoriquement, tout composant peut être redimensionné. Néanmoins, certaines conditions doivent être vérifiée :

- l'objet interactif est-il redimensionnable ? En effet, certains objets ont des tailles fixées par les toolkits dans lesquels ils ont été implémentés et ne sont pas redimensionnables (ex : les boutons radios).
- les limites de la perception humaine sont-elle respectée ? Certains résultats expérimentaux ont montré qu'une icône ne pouvait pas descendre en dessous de la résolution de 8x7 pixels. En dessous de cette taille, elle devient illisible.
- les contraintes imposées par les toolkits permettent-elles de redimensionner des objets ? Certaines toolkit donnent directement des dimensions arbitraires aux objets interactifs concrets (HTML ou encore QTk).
- Le designer veut-il réellement redimensionner ses éléments ?

Lorsqu'un composant est redimensionnable, nous devons exprimer la taille minimum qu'il peut avoir. Ainsi, on peut définir par exemple sa hauteur minimale ou maximale lorsque la longueur minimale est atteinte et vice-versa. C'est notamment ce qui est fait pour l'algorithme de *[Keränen et Plomp 2002]*.

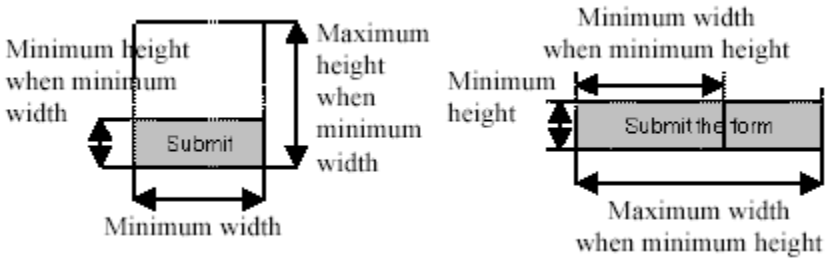


Figure 49 – L'espace minimal requis pour un bouton

Pour certains types d'objets interactifs concrets, il n'est pas possible de définir une taille minimale. Leur longueur et/ou hauteur minimale sont en réalité déterminés par d'autres facteurs comme, par exemple, les objets du domaine manipulés (la largeur d'une liste de sélection dépend de la longueur maximale parmi tous les éléments proposés) ou la longueur d'un libellé qui sera inclus dans le widget (la longueur minimale d'un bouton est souvent déterminée par la longueur de son libellé). Ainsi, on exprimera cette taille minimale via une fonction de la longueur de chaînes de caractères.

5.5 Notre Application en QTk

5.5.1 Modèles et Règles de transformation retenues

Règles de transformation choisies

Les règles de transformation que nous appliquerons seront celles que nous avons explicitées dans le point précédent, à savoir :

- la substitution d'objets interactifs
- la scission de la fenêtre de départ en plusieures fenêtres
- le redimensionnement d'objets interactifs
- le repositionnement d'éléments (nous appliquerons cependant des stratégies simples (tout dans une colonne, tout dans deux colonnes, etc…)
- la suppression d'images
- le choix d'images dans un format réduit

Modèles nécessaires à l'application de ces règles

Nous aurons besoin des modèles du domaine, de la tâche et de la présentation pour appliquer ces différentes règles. Le modèle de la plate-forme, bien que très important ne sera pas réellement soulevé ici. Ainsi, notre règle de substitution d'objets interactifs n'inclura pas, par exemple, la substitution liée aux capacités de la plate-forme. La seule contrainte abordée en ce qui concerne la plate-forme sera la résolution de l'écran.

Pour les deux premiers modèles, nous nous baserons sur le formalisme décrit dans USIXML. Pour le modèle de la présentation, il nous reste encore à le définir, et c'est ce que nous allons faire maintenant.

5.5.2 Notre modèle de la présentation

Nous allons construire notre modèle de la présentation en réalisant des analogies avec QTk, surtout en ce qui concerne les relations de placement, ce qui nous facilitera la tâche par la suite.

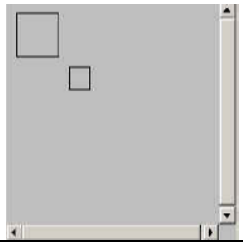
La représentation des objets interactifs

Nous allons nous inspirer essentiellement de la représentation utilisée dans *[Florins 2003]*. Dans cette représentation, le modèle des objets interactifs est représenté par trois niveaux d'abstraction :

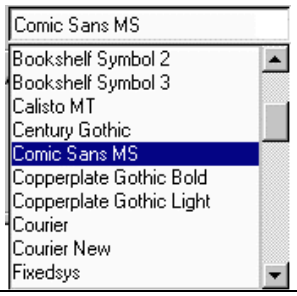
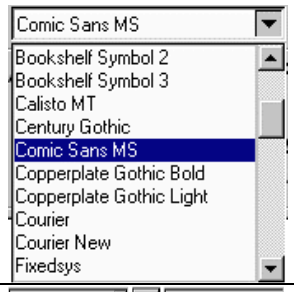
- les *a-modal AIOs* représente le niveau supérieur (ex : Number Inputter, Date Choser, Action Item, Separator ou Container)
- le second niveau est représenté par les *modal/visual AIOs*, qui constitue une particularisation des a-modal AIOs pour les interfaces graphiques. (ex :edit box, radio button, list box, …)
- Le troisième niveau représente les objets interactifs concrets (CIO). A ce niveau, les éléments sont dépendant de la plate-forme.

Les *a-modal AIOs* sont décrits par leur nom, leur cardinalité (indiquant le nombre de concepts que les objets interactifs peuvent manipuler), les types de tâches qu'ils permettent de réaliser (voir 4.4.2.1) et les types de données abstraits qu'il sont en mesure de manipuler.

Ainsi, ce modèle permet de représenter les widgets du toolkit QTk. En voici un exemple :

CIO - QTk toolkit **(name – graphical representation)**		**Modal AIO**	**A-Modal AIO** **(name, type of task, abstract data type, cardinality)**
Button		Push Button	ActionItem T: activation D: _ Card :_
Canvas		Canvas	Graphical Editor T: specification & consultation D : graphics Card : N

Nous avons également rajouté quelques objets interactifs au modèle de *[Florins 2003]*. En effet, bien que déjà très complet, le modèle proposé n'abordait pas le cas des OIAs composés, tels que l'accumulateur par exemple. Ceux-ci n'étant cependant pas présents en tant que tel, dans le modèle QTk, nous n'attribuerons pas de nom aux widgets faisant référence aux OIAs que nous avons rajouté :

/		Combo Box	Text Inputter + Simple Text Chooser T : specification & simple selection D : string Card : N
/		Drop-Down Combo Box	Text Inputter + Simple Text Chooser T : specification & simple selection D : string Card : N
/		Accumulator	Text Chooser T: simple & multiple selection D : string Card : N

Remarquons que l'apparence graphique de l'accumulateur n'est pas unique, il existe plusieures variantes (on peut notamment sans rendre compte en consultant la figure 47).

Cette description nous sera particulièrement utile en ce qui concerne la substitution de widget. Néanmoins, comme nous le verrons dans la description du programme, nous n'utiliserons pas toute l'information véhiculée par ce modèle.

La représentation des relations de placement

Pour représenter nos relations de placement, nous allons fortement travailler en analogie avec QTk. Ainsi, notre représentation *au niveau lexical* proposera un système coordonné où l'origine se place dans le coin supérieur gauche du système.

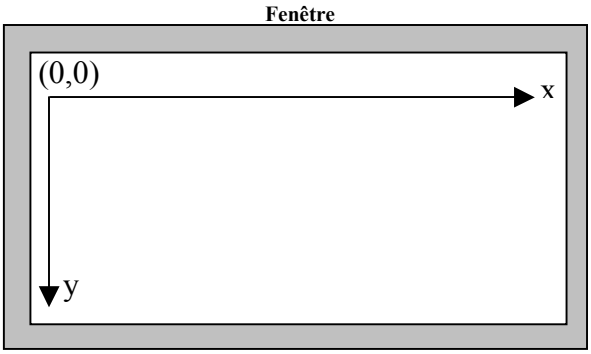


Figure 50 – Notre système de coordonnées au niveau lexical

En ce qui concerne *les relations syntaxiques*, QTk offre l'avantage de disposer de deux widgets spéciaux (lr et td) qui servent de conteneurs et d'aide aux placement entre différents autres widgets. Ceux-ci disposent également de code spécifique (newline, empty, continue) qui permettent d'organiser tous les widgets qui sont à l'intérieur d'être organisés selon une structure de grille, ou encore du paramètre *glue* spécifiant le placement des widgets contenus lorsque la fenêtre est redimensionnée. Ainsi, nous baserons toutes ces relations suivant ces deux widgets.

Plusieures notations ont été proposées pour spécifier ces relations de manière abstraite (*[Allen83]*, *[Vanderdonckt 1997]*) mais nous ne les exprimerons cependant pas de cette manière et utiliserons directement ces deux widgets pour construire nos interfaces de manière similaire à l'exemple du point 4.1.

5.6 Architecture de l'application

Architecture logique

Notre architecture logique se compose de huit couches : la couche *Description des interfaces utilisateur*, la couche *Mise en évidence d'unité de présentation et des fonctions sémantiques associées aux tâches*, la couche *Description des objets interactifs*, la couche *Structuration d'unités de présentation et de fonctions sémantiques*, la couche *Génération d'interfaces semi-concrètes*, la couche *Transformation d'interfaces*, la couche *Génération d'interfaces concrètes*, et enfin, la couche *Visualisation d'interfaces concrètes*. Le schéma de la page suivante montre leur agencement (les différentes relations entre MODULES sont des relations USES). Nous allons les décrire brièvement une par une en commençant par la plus basse.

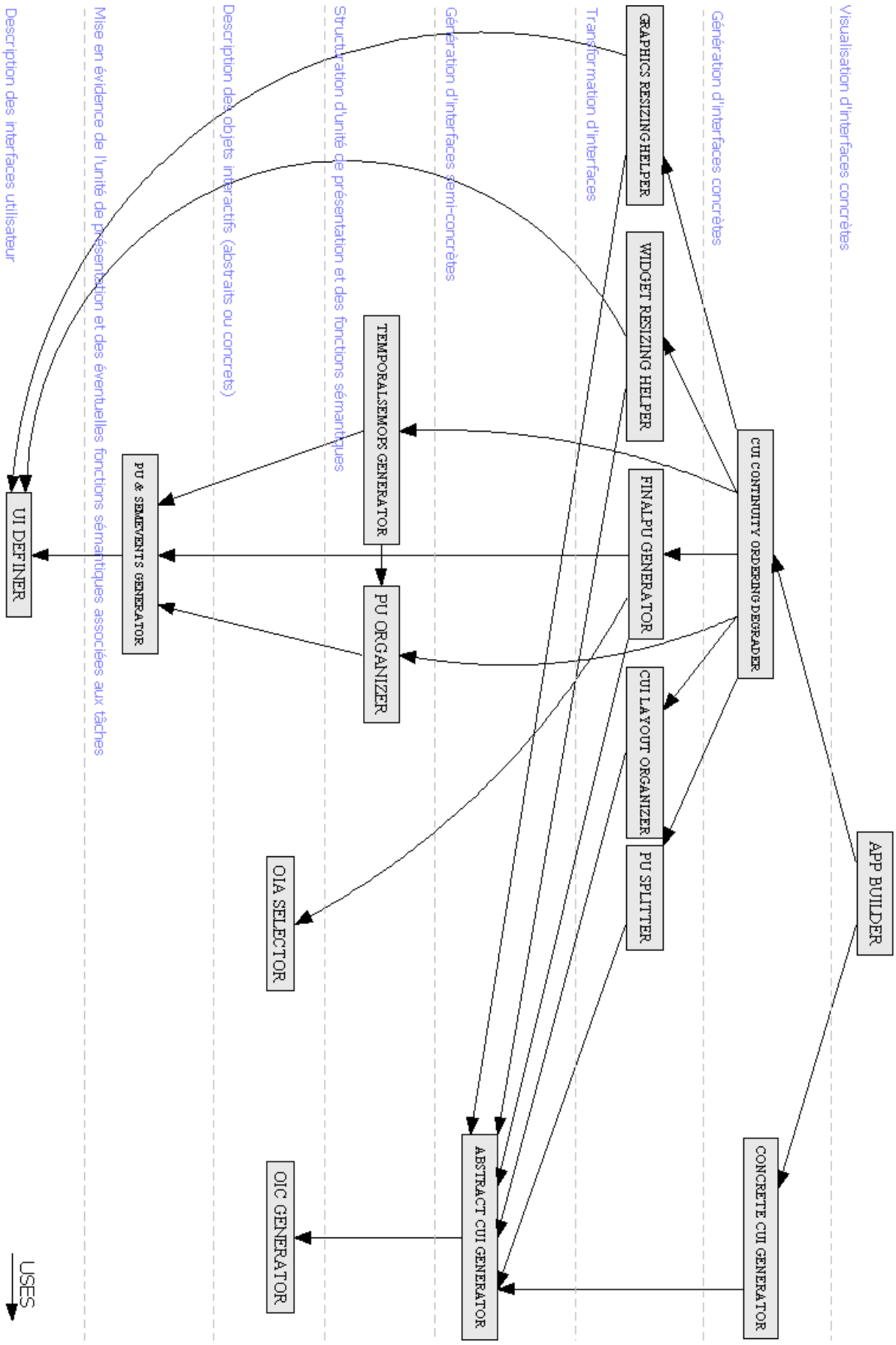


Figure 51 – Architecture Logique de notre application de Dégradation Harmonieuse

(A) Description des interfaces utilisateur

Cette couche a pour but de décrire les interfaces utilisateur utilisées comme données dans l'application. Elle est composée d'un module **UI Definer**, qui comporte les fonctions décrivant chacune des interfaces à construire/dégrader. Nous reviendrons plus en détail sur ce module par la suite.

(B) Mise en évidence d'unités de présentation et des fonctions sémantiques associées aux tâches

Cette couche se concentre sur la construction de deux structures, une qui stockera l'unité de présentation (unique à ce moment précis) d'une interface utilisateur représentée par son modèle de tâche et du domaine, l'autre qui stockera les noms des tâches qui déclencheront des fonctions sémantiques et les noms de celles qui sont associées à des fonctions sémantiques (exemple : les tâches systèmes)

(C) Description des objets interactifs

Cette couche est composée de deux modules. Le premier est chargé de fournir des listes d'OIAs disponible pour un type de tâche et un type de donnée manipulée précis. L'autre s'occupe de la traduction des OIAs en OICs *semi-concrets*. Nous reviendrons sur cette dénomination dans le point (E).

(D) Structuration d'unités de présentation et de fonctions sémantiques

Dans cette couche, nous disposons de deux modules afin de mieux isoler les informations concernant les unités de présentations et les fonctions sémantiques liées à l'activation/désactivation de tâches interactives pour différents opérateurs temporels du modèle de tâches. Le module **PU Organizer** sera chargé d'isoler les tâches d'un même niveau à partir d'une unité de présentation pour refléter au mieux l'arbre de tâches. L'autre module construira, quant à lui, des listes de fonctions sémantiques supplémentaires afin de mettre en évidence dans la future application les tâches séquentielles, concurrentes, de choix, …

(E) Génération d'interfaces semi-concrètes

Cette cinquième couche va permettre de construire des interfaces concrètes à un détail près, les fonctions sémantiques qui peuvent être déclenchées par les OICs seront exprimées sous une forme abstraite. En effet, il suffit qu'un widget change dans la présentation finale pour que les fonctions sémantiques qui le manipulent ne soient plus valides. Aussi, plutôt que de générer des fonctions sémantiques qui soient directement liées à un widget précis, nous les exprimeront en fonctions des identificateurs de tâche auxquels ils se rapportent, ces derniers étant fixes.

(F) Transformation d'interfaces

Le but de cette couche est de transformer et d'organiser les différentes unités de présentation suivant différentes règles de transformation (les règles de dégradation retenues). Le module **FinalPU Generator** va permettre d'attribuer de manière définitive des OIAs aux tâches d'une unité de présentation (qu'il s'agisse d'une attribution par défaut ou une attribution

après substitution). Les autres modules se chargeront de calculer différentes valeurs susceptibles d'assurer que les conditions sur la résolution de la plate-forme soient satisfaites pour chacune des règles de dégradation ciblée. On remarquera que chacun des modules **utilise** le module de génération d'interfaces semi-concrètes afin de vérifier à chaque étape de dégradation que la résolution se rapprochera de celle souhaitée. On notera aussi l'utilisation du module **UI Definer** par les modules **Graphics Resizing Helper** et **Widget Resizing Helper** car ceux-ci ont besoin de connaître les informations sur la taille admissible des objets interactifs pour les tâches retenues et celles sur les éléments graphiques de présentation pure (qui ne sont pas liées au modèle de la tâche).

(G) Génération d'interfaces concrètes

Cette couche se compose de deux modules. Le premier (**CUI Continuity Ordering Degrader**) vise à générer une interface *semi-concrète* finale sur base de l'application de différentes règles de dégradation. Le deuxième transforme cette interface *semi-concrète* en interface totalement concrète en transformant les fonctions sémantiques abstraites en fonctions sémantiques concrètes.

(H) Visualisation d'interfaces concrètes

Cette dernière couche se compose d'un seul module qui va se charger de construire les fenêtres des applications choisies pour dégradation éventuelle.

Architecture physique

L'architecture physique est en grande partie équivalente à l'architecture logique. Les relations USE deviennent des relations CALL et nous ajoutons une interface graphique développée en QTk au dessus du module de la couche supérieure.

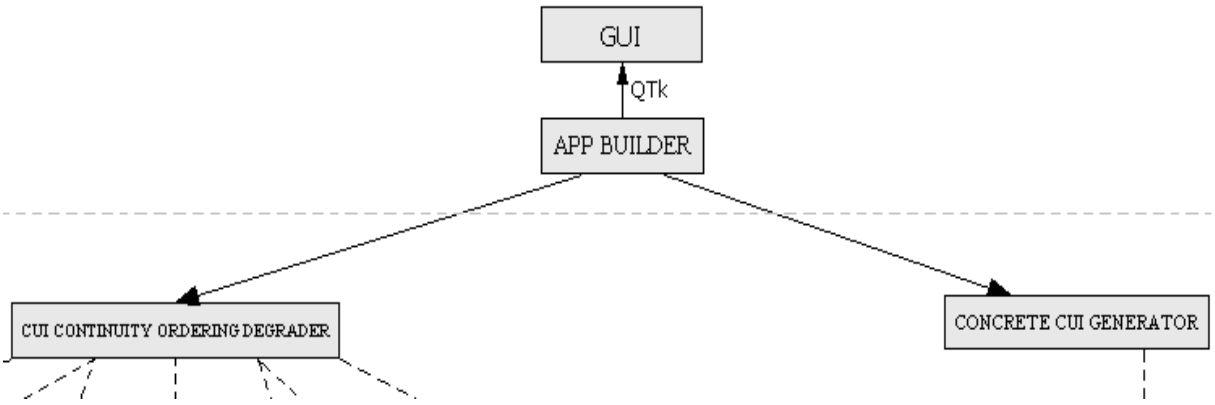


Figure 52 – Une partie de l'architecture physique de l'application de dégradation harmonieuse

A noter que nous n'utilisons pas dans notre application de structures de stockage physique. Ainsi, le module **UI Definer** n'est relié à aucun support de stockage et le modèle de la tâche et du domaine sont donc compris comme des variables dans le code. Nous justifierons pourquoi un tel choix (semblant introduire des limitations dans notre application) a été fait par la suite.

5.7 Description du fonctionnement des modules et des structure de données retenues pour l'application des règles de dégradation

De manière à rester fidèle à la démarche suivie pour la hiérarchie en couches de notre architecture, nous décrirons chaque structure de données utilisée et algorithme de notre application par couche et par module, en commençant par la plus basse.

5.7.1 La Couche Description des interfaces utilisateur

Pour l'unique module qui compose cette couche, nous nous contenterons de décrire la manière de représenter les éléments pertinents qui seront utilisés par la suite.

Les modèles de la tâche et du domaine

Nous utiliserons pour modéliser les modèles de la tâche et du domaine une structure unique, permettant de résumer l'information utile et nécessaire à l'application des règles de dégradation harmonieuse que nous avons retenues, ceci permettant également de réaliser dans la foulée un mapping entre ces deux modèles. Ainsi, nous gagnons en rapidité, toute l'information restant concentrée en un seul modèle. Bien entendu, le fait de vouloir tout réunir dans une seule structure peut introduire une certaine limitation dans les possibilités de l'application mais nous allons montrer que ce n'est pas le cas en ce qui concerne la nôtre.

Voici de manière plus détaillée la manière dont nous représenterons notre modèle :

Notre modèle consistera en un modèle de tâche prioritaire (voir point 5.4.2.1), pour lequel chaque tâche comportera, en plus des informations qui s'y rapportent, le type de données manipulé (et donc des éléments du domaine). Pourquoi inclure ces données dans le modèle de tâche plutôt que de consacrer un modèle dédié aux éléments du domaine ? Simplement car dans le cadre de notre travail, il n'est pas nécessaire de connaître toute l'information sur le domaine manipulé. Seul le type de donnée et les valeurs possibles sont utiles.

Ceci pourrait introduire quelques inconvénients car, ne disposant plus que de l'information sur les attributs et non plus sur les classes, il nous est alors impossible de mettre en liaison deux tâches manipulant le même attribut. On peut citer par exemple, une tâche de consultation dans une liste de sélection qui serait suivie d'une tâche de sélection sur le même objet interactif. Néanmoins, nous sommes partis de l'heuristique qu'une tâche interactive manipulait un unique objet interactif. De plus, il est très simple de contourner ce problème en fusionnant les deux tâches de l'exemple en une seule.

Le problème serait plus difficile à résoudre si les tâches n'étaient pas adjacentes. Cependant, on peut se convaincre qu'une application d'un tel type n'est franchement pas ergonomique car elle suggère à l'utilisateur de revenir sur un objet interactif déjà « visité ». A moins que cet objet ne se répète sur plusieures présentations séparées, il est réellement peu crédible d'imaginer un tel cas.

En réalité, le seul inconvénient de ce modèle transparaît si nous décidions d'appliquer une règle de dégradation sur le niveau Tâches & Concepts. Si nous supprimons une tâche, il n'y a pas de problème. Par contre, si nous décidons de supprimer une classe ou un attribut, il n'y a pas moyen de retrouver quelle tâche supprimer. Ceci pourrait néanmoins être résolu en spécifiant, en plus du type de donnée manipulé, le nom de la classe ou de l'attribut cible.

Au final, ceci reste donc tout à fait compatible avec la notation USIXML. Nous utilisons en effet un modèle de la tâche prioritaire et il a été montré qu'un modèle de tâche prioritaire a son équivalent en mode simple. De plus, on peut imaginer un traitement initial, transformant un modèle de tâche, du domaine et le mapping entre ces modèles en une structure unique de manière très simple.

Voici maintenant la syntaxe que nous utiliserons pour le définir (pour rappel, ce modèle sera compris comme une variable dans notre application et non comme un document de type XML. Comme déjà énoncé précédemment, nous justifierons par la suite un tel choix) :

Chaque arbre sera décrit comme un record Oz. Nous avons donc un modèle du type :

```
<model> ::= tree(id: <id>
                 type: <type>
                 childsBType: <tempOp>
                 childs :<listOfChilds>)

<id> ::= atome oz

<type> ::= 'abstract' | 'interactive' | 'system' | 'user'

<tempOp> ::= '>>' | '[]>>' | '|||' | '|[]|' | '|>' | '[>' | '[]'

<listOfChilds> ::= liste oz de <sub-model>

<sub-model> ::= <model> <sub-modelHP> | <leafTask> <sub-modelHP>

<sub-modelHP> ::= <model> <sub-modelHP> | <leafTask> <sub-modelHP> | <model> | <leafTask>

<leafTask> ::= leaf(id: <id>
                    name: <id>
                    type: 'interactive'
                    actionType: <someActionType>
                    manipulDataType: <someDataType>
                    frequency: <fvalue>
                    importance: <fvalue>
                    manipulDataValue: <dvalue>
                    manipulCardMin: <cardmin>
                    manipulCardMax: <cardmax>) |

                leaf(id: <id>
                     name: <id>
                     type: 'user' | 'system' | 'abstract') |

                leaf(id: <id>
                     name: <id>
                     type: 'interactive'
                     actionType: 'selection' | 'simple selection' | 'specification'
                     manipulDataType: 'alphanumeric'
                     frequency: <fvalue>
                     importance: <fvalue>
                     manipulDataValue: none
                     manipulDomainCharacterization: estimated_length(entier)) |

                leaf(id: <id>
                     name: <id>
                     type: 'interactive'
                     actionType: 'selection' | 'simple selection' | 'specification'
                     manipulDataType: <someDataType>
                     frequency: <fvalue>
                     importance: <fvalue>
                     manipulDataValue: liste oz avec un domaine entièrement connu
                     manipulDomainCharacterization: 'discrete interval' |
                                                    'continuous interval'
                     manipulCardMin: entier supérieur ou égal à 2
                     manipulCardMax: <cardmax>) |
```

```
                leaf(id: <id>
                     name: <id>
                     type: 'interactive'
                     actionType: 'multiple selection'
                     manipulDataType: <someDataType>
                     frequency: <fvalue>
                     importance: <fvalue>
                     manipulDataValue: <valuelist>
                     manipulCardMin: entier supérieur ou égal à 2
                     manipulCardMax: <cardmax>) |

                leaf(id: <id>
                     name: <id>
                     type: 'interactive'
                     actionType: 'selection' | 'simple selection' | 'multiple selection'
                     manipulDataType: 'graphics'
                     frequency: <fvalue>
                     importance: <fvalue>
                     manipulDataValue: <grValuelist>
                     manipulCardMin: entier supérieur ou égal à 2
                     manipulCardMax: <cardmax>) |

                leaf(id: <id>
                     name: <id>
                     type: 'interactive'
                     actionType: 'specification'
                     manipulDataType: 'graphics'
                     frequency: <fvalue>
                     importance: <fvalue>
                     manipulDataValue: none) |

                leaf(id: <id>
                     name: <id>
                     type: 'interactive'
                     actionType: 'consultation'
                     manipulDataType: 'graphics'
                     frequency: <fvalue>
                     importance: <fvalue>
                     manipulDataValue: <grValue>
                     manipulCardMin: 1
                     manipulCardMax: 1) |

                leaf(id: <id>
                     name: <id>
                     type: 'interactive'
                     actionType: 'consultation'
                     manipulDataType: 'graphics'
                     frequency: <fvalue>
                     importance: <fvalue>
                     manipulDataValue: <grValueList>
                     manipulCardMin: entier supérieur ou égal à 2
                     manipulCardMax: <cardmax>)

<someActionType> ::= 'activation' | 'submission' | 'checking' | 'navigation' |
                     'external navigation' | 'internal navigation' | 'consultation'

<someDataType> ::= 'alphanumeric' | 'integer' | 'real' | 'date' | 'time' | 'graphics'

<fvalue> ::= entier compris entre 1 et 5

<cardmin> ::= entier supérieur ou égal à 1
<cardmax> ::= entier supérieur ou égal à manipulCardMin

<dvalue> ::= valeur ou liste oz de valeurs du type déterminé par manipulDataType

<valuelist> ::= liste oz de valeurs du type déterminé par manipulDataType

<grValueList> ::= liste oz d'atomes du type <grfile> ou de couples bibliothèque d'images QTk #
liste d'atomes du type <grfile> représentant des fichiers se trouvant dans la bibliothèque

<grValue> ::= atome oz du type <grfile>

<grfile> ::= liste de caractères alphanumériques <res> x <res> . <ext>

<res> ::= entier

<ext> ::= extension fichier image
```

Remarquons que nous ne gérons pas l'opérateur temporel |=| car celui-ci suggère l'utilisation d'un état supplémentaire indiquant quelle(s) tâches(s) parmi celles liées par cet opérateur ont déjà été réalisées.

Nous ne prenons également pas en compte les tâches récursives et itératives, de même que celles qui sont optionnelles.

La taille admissible des objets interactifs

La taille des objets interactifs redimensionnables sera spécifiée de la manière suivante :

```
[MinW#[MinH MaxH] [MinW MaxW]#MinH]

 avec :- MinW, la largeur minimale de l'objet
       - MaxW, la largeur maximale de l'objet lorsque MinH est atteint
       - MinH, la hauteur minimale de l'objet
       - MaxH, la hauteur maximale de l'objet lorsque MinW est atteint
```

Les éléments spécifiques de présentation

Certains éléments liés uniquement à la présentation devront être pris en compte (par exemple, des libellés décrivant des objets interactifs (critère ergonomique de guidage)). Nous décrirons chacun d'entre-eux par un triplet du type :

```
id. de tâche sur laquelle il faudra appliquer un élément de présentation ou l'atome
head si cet élément de présentation se rapporte à un ensemble de tâches d'un même
niveau #

élément de présentation (<leafTask> #'OIA')#

un booléen indiquant si cet élément de présentation est optionnel ou non dans le
cas où l'élément de présentation s'articule sur une/des données graphiques
```

Les fonctions sémantiques abstraites

Chacune des fonctions sémantiques abstraites sera décrite de la manière suivante :

```
procédure Oz # liste de couples (variable libre utilisée comme poignée QTk # Feat
(id. de tâche servant de feature à un objet interactif concret QTk))
```

Dans chaque procédure Oz, les poignées manipulées seront les variables libres du deuxième membre. Par exemple, si X est utilisé comme une poignée dans la procédure et H la poignée principale de la présentation, X sera égale au chemin partant de H jusque la feature Feat.

Comme le lecteur aura pu s'en rendre compte, disposer d'un modèle de tâche et du domaine n'est pas suffisant pour décrire l'interface utilisateur minimale dont nous disposerons pour réaliser nos opérations de dégradation. Aussi, en regard des différents points qui viennent d'être développés, on peut alors comprendre pourquoi nous avons décidé de garder nos modèles à l'intérieur du module et non comme une ressource externe.

5.7.2 La Couche Mise en évidence d'unités de présentation et des fonctions sémantiques associées aux tâches

Ici, nous nous attarderons uniquement sur les structures de données résultantes. Les unités de présentation seront du type :

```
liste de couples ((<leafTask>#OIA)#'tâches mères de la tâche interactive' (des
tâches <model> sans l'attribut childs))
```

remarque : OIA n'est pas forcément spécifié à ce moment-ci. Il peut être égal à un OIA si un mapping existe entre un OIA et une tâche précise (*is executed in*) ou à 'not yet specified'.

Les références aux fonctions sémantiques se feront par l'intermédiaire de couples du type :

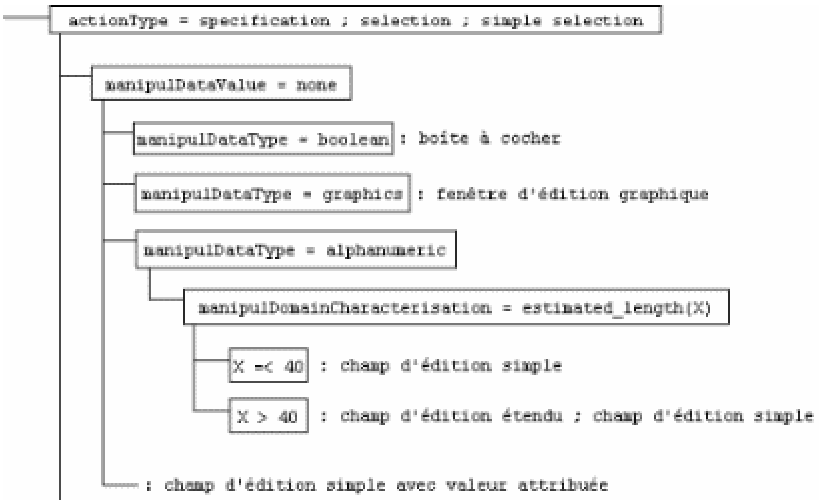
```
identificateur tâche système/interactive # identificateur tâche interactive
```

A noter le traitement spécial que nous devrons effectuer en ce qui concerne les tâches liées par un opérateur de choix. En effet, nous avons pris comme heuristique le fait que les tâches systèmes soient toujours déclenchées par la tâche feuille les précédents. Or, pour cet opérateur, une tâche système ne sera plus forcément déclenchée par une seule tâche mais par un ensemble de tâches.

5.7.3 La Couche Description des objets interactifs

Pour cette couche, nous décrirons dans un premier temps le module **OIA Selector** (l'autre consistant à transformer un OIA en OIC). Ce module est composé d'une unique fonction décrivant l'arbre conditionnel de sélection d'OIAs en fonction des données d'une tâche.

Notre arbre se base essentiellement sur la typologie des tâches interactives introduite dans la section 5.4.2.1 et sur les arbres conditionnels découlant des règles ergonomiques de sélection d'objets interactifs introduites par *[Vanderdonckt 1998]*. Notre arbre constitue en ce sens un mélange des deux car il n'est pas toujours possible d'obtenir un rendu ergonomique pour tout objet et tout type de tâche, comme nous nous devons de choisir un OIA approprié pour un type d'action. De même, nous devrons nous limiter aux OIAs compatibles avec QTk. Voici une partie de l'arbre que nous avons obtenu (l'arbre complet se trouve en annexes) :



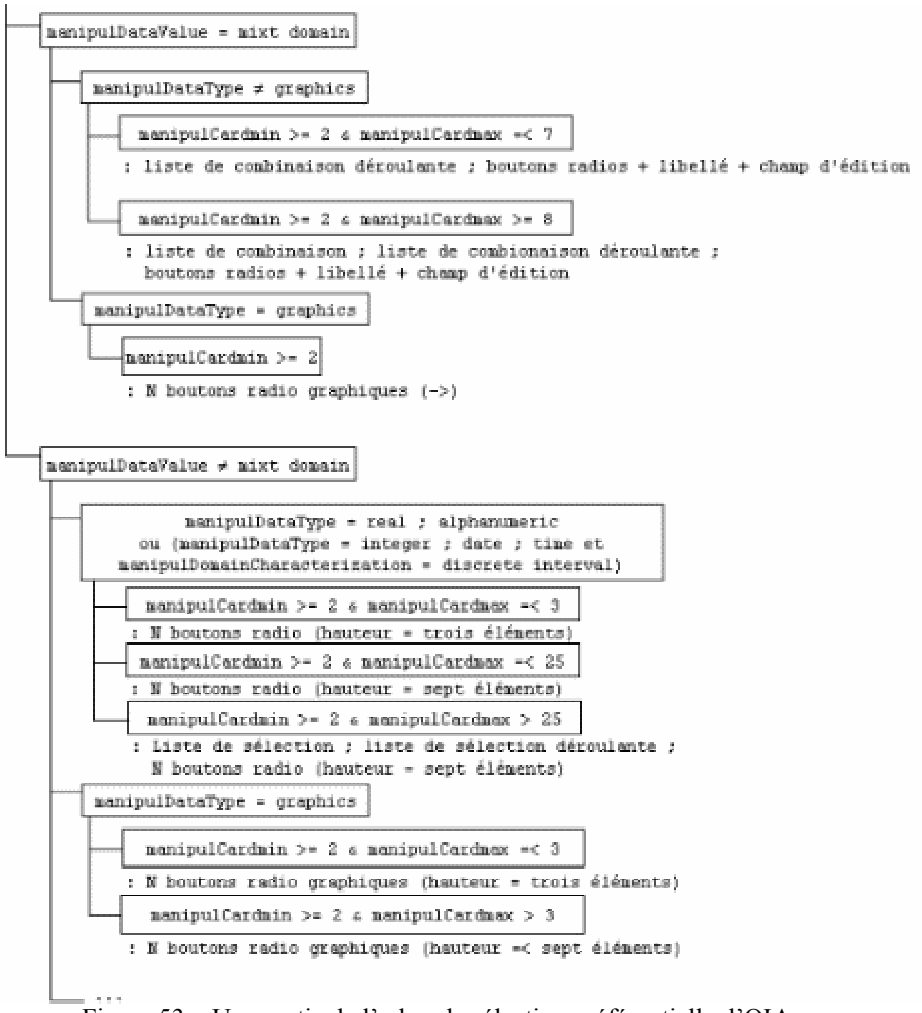


Figure 53 – Une partie de l'arbre de sélection préférentielle d'OIA

On peut noter quelques problèmes liés aux capacités de QTk, tous les OIAs considérés dans *[Vanderdonckt 1998]* n'ont pas pu être adaptés ou auraient demandé plus de travail sur un point non central de la problématique soulevée par ce mémoire. On peut citer comme exemple l'OIA désigné par le signet (->) dans la figure 53. Il aurait normalement fallu ajouter aux boutons radio un véritable éditeur graphique. De même, certains OIAs compatibles ont été écartés car ils violaient les priorités des opérateurs temporels après une substitution (ex : le tableau à onglets permet de revenir en arrière même si nous avons un opérateur de séquence).

Nous ne discuterons que très brièvement du second module. Les seuls points importants à retenir de ce module de transformation d'OIA en OIC concernent le déclenchement des fonctions sémantiques. En effet, il n'est pas toujours souhaitable de déclencher les fonctions sémantiques dès que l'état de l'objet interactif change mais plutôt à la demande spécifique de l'utilisateur.

On peut citer comme exemple les OIAs supportant la sélection multiple, pour lesquels nous ajouterons lors de la construction un bouton de validation, celui-là même déclenchant les fonctions sémantiques. En effet, si l'utilisateur sélectionne par exemple un élément dans une liste de sélection, les fonctions sémantiques ne doivent pas être déclenchées automatiquement mais seulement lorsque l'utilisateur a fini de faire ses choix. De même, si nous utilisons des OIAs du type échelle ou bouton de variation, il ne faut pas non plus que les fonctions

sémantiques soient déclenchées lorsque l'utilisateur bouge le curseur de sélection mais lorsque ce dernier valide son choix.

5.7.4 La Couche Structuration d'unités de présentation et de fonctions sémantiques

Cette couche se compose tout d'abord du module **PU Organizer**. Son but a déjà été brièvement décrit. Nous nous contenterons donc de donner une idée des données produites en sortie. Chaque unité de présentation source est du type de celle décrite dans le point 5.7.2. la version organisée de l'unité de présentation se présente comme suit :

```
liste de paires (<leafTask>#OIA) ou de sous-unités de présentations organisées #
tâche mère du type <model> (sans l'attribut childs)
```

Le second module vise, d'une part, à générer de manière correcte les fonctions sémantiques liées aux opérateurs temporels et d'autres parts, à isoler les tâches inactives à l'ouverture de l'application. La première fonctionnalité a un déroulement en plusieurs étapes.

La première étape consiste à nettoyer une unité de présentation organisée en remontant d'un niveau les sous-unités de présentation pour lesquelles la mère a un opérateur temporel binaire identique à celui de ses tâches filles.

La deuxième étape consiste à isoler chaque opération en générant une liste de triplets :

```
tâche(s) Source#tâche(s) Destination#Opérateur
```

Le fait que plusieurs tâches (des listes en réalité) puissent jouer le rôle de source ou destination implique qu'il faudra également effectuer, le cas échéant, un sous-traitement sur ces tâches.

Une fois cette étape réalisée, nous effectuons alors une troisième étape, consistant à simplifier certains des triplets obtenus et à modifier l'unité de présentation source. Voici les différentes modifications à réaliser.

Soit X#Y#Op, un triplet où : - X est une (liste de) tâche(s) Source
- Y est une (liste de) tâche(s) Destination
- Op est un opérateur temporel parmi `<tempOp>`

Si X se réduit à une seule tâche et si Y est une liste de tâches :

- Si Op est un opérateur concurentiel et Op_Y, l'opérateur temporel commun aux tâches de Y (rappelons que notre modèle utilise un arbre prioritaire), est un opérateur de séquence, alors X n'est en réalité concurrent qu'avec la première tâche de Y

Si X est une liste de tâches (peu importe la nature de Y) :

- Si Op est un opérateur de séquence et Op_X est un opérateur de concurrence, nous devrons modifier l'unité de présentation principale en ajoutant un bouton de validation permettant d'indiquer quand ces tâches concurrentes seront terminées afin de continuer la séquence.
- Si Op est un opérateur de séquence et si Op_X est un opérateur de choix, nous devrons modifier ce triplet en une liste de triplet pour que chaque tâche contenue dans X puisse déclencher les tâches de Y

- Si Op est un opérateur de concurrence et Op_X est un opérateur de séquence, nous pouvons simplifier ce triplet en indiquant que seul la première tâche de X active celles de Y.

Remarquons que l'ajout d'un bouton de validation impliquera un changement dans la liste de couples reflétant les tâches responsables du déclenchement des fonctions sémantiques (voir point 5.7.2). C'est en effet ce bouton qui sera responsable dorénavant du déclenchement des fonctions sémantiques et plus la dernière tâche de l'ensemble de tâches concurrentes (cfr. heuristique utilisée).

La dernière étape consistera à créer un ensemble de couple du même type que ceux du point 5.7.2 en ce qui concerne les fonctions sémantiques mais où le premier membre est un record dont la fonction est l'activation ou la désactivation des tâches qui y sont contenues. Ceci étant bien entendu déclenché par la tâche dont l'identificateur se trouve dans le deuxième membre.

La deuxième fonctionnalité se base sur les résultats obtenus lors de la troisième étape pour la première fonctionnalité. Puisque tous les opérateurs de séquence ont été isolés, il est alors facile de déterminer quelles seront les tâches inactives à l'ouverture.

5.7.5 La Couche Génération d'interface semi-concrètes

Comme dit précédemment, cette couche consiste à générer une interface QTk mais où les fonctions sémantiques ne sont pas encore tout à fait concrètes, le type d'objet interactif sur lesquelles elles s'appliquent ayant la possibilité de varier. Ces dernières seront alors exprimées selon la terminologie expliquée dans le point 5.7.1.

5.7.6 La Couche Transformation d'interface

C'est probablement la couche la plus intéressante car c'est à ce niveau que nous appliquons nos règles de dégradation harmonieuse.

Le module FinalPU Generator

Ce module va permettre de générer une unité de présentation où tous les OIAs seront spécifiés de manière définitive. Deux cas d'affectation d'un OIA à une tâche sont possibles :

- le cas par défaut : on choisira l'OIA dont l'OIC aura la taille la plus élevée.
- le cas spécifique à une résolution précise : on choisira l'OIA dont l'OIC permet de satisfaire les conditions de résolution.

Pour cela, nous devons, dans une première étape, organiser des listes d'OIAs candidats pour leurs tâches respectives en fonction de l'espace qu'ils (leurs OICs) prendront. Ensuite, nous attribuons le premier candidat de chaque liste à chaque tâche. Si nous ne considérons que le cas par défaut, nous gardons cette configuration. Sinon, si la taille de l'ensemble est supérieure à ce qui est souhaité, nous devrons substituer certains OIAs par d'autres.

Pour réaliser cette dernière étape, nous attribuons une importance relative à chaque tâche (le facteur multiplicateur *importance*fréquence* de la tâche) et nous substituons les OIAs par ordre croissant d'importance relative. Ainsi, les tâches les moins importantes subiront d'abord une substitution.

Remarquons néanmoins que cette technique pose un léger problème. En effet, nous n'attribuons d'*ordre de passage* aux OIAs candidats que pour le facteur de taille. Il aurait peut-être été utile d'incorporer en plus un facteur ergonomique car, une taille supérieure ne signifie pas forcément que l'OIA sera plus ergonomique que les autres candidats.

Le module PU Splitter

Nous n'entrerons pas énormément dans les détails en ce qui concerne ce module car les règles de scission ont déjà été explicitées dans la section 5.4.3.1. Nous nous contenterons donc de décrire la démarche générale suivie.

La première étape consistera à déterminer l'endroit où commencer la scission. Ainsi, nous déterminerons l'ancêtre (la tâche mère) commun à toutes les tâches feuilles de l'unité de présentation et se situant au niveau le plus bas possible dans la hiérarchie des tâches.

Ensuite, nous commencerons par tenter une scission au niveau des opérateurs de séquence. Si cette scission est concluante, nous devrons alors vérifier que nous ne devons pas créer de redondance pour certaines tâches (concurrentes ou de désactivation) en les joignant aux différentes unités de présentation obtenues. Si l'ancêtre commun le plus bas n'a par contre pas de fils liés à un opérateur de séquence, nous essayons alors de couper au niveau d'un de ses voisins qui a des descendants.

Si la scission n'a toujours pas eu lieu, nous essayons de découper l'unité de présentation suivant un opérateur de concurrence. La technique suivie est essentiellement la même, si ce n'est que, dans le cas où nous devons créer des redondances entre tâches est réduit car le cas de répartition de tâches concurrentes n'a pas de sens ici. La seule différence transparaît dans la modification de nature des opérateurs concurrents en opérateurs de séquence après une scission.

Une fois une éventuelle scission réalisée, nous vérifions la taille de chaque unité de présentation obtenue et réessayons de couper si la scission n'a pas été suffisante pour chacune d'entre-elles.

Remarquons le cas spécifique supplémentaire que nous n'avions pas spécifié dans la section 5.4.3.1 concernant les opérateurs de choix. En effet, si nous coupons sur un opérateur de séquence ou de concurrence inférieur à une tâche dont l'opérateur temporel est un opérateur de choix, nous perdons la possibilité de choix. Aussi, nous ne coupons pas à ce niveau.

Le module CUI Layout Organizer

Le but de ce module est de déterminer la meilleure combinaison possible niveau (en terme de hiérarchie de tâches d'un arbre) à partir duquel réorganiser les tâches # nombre de colonnes souhaitées par groupe de tâches d'un même niveau.

Pour ce faire, nous calculons dans un premier temps le niveau maximal et le nombre de tâches maximal qu'il est possible d'atteindre. Ensuite, pour chacune des combinaisons possibles (on commence par le couple 1#1), nous retenons celle pour laquelle la présentation se rapproche le plus de la résolution souhaitée.

A chaque phase du traitement, on vérifie que la nouvelle présentation candidate à une taille qui satisfait à la résolution souhaitée. Si c'est le cas, on choisit cette dernière (on prend toujours celle en cours et pas une des suivantes car c'est celle-ci qui nécessite le moins de réorganisation). Sinon, on compare la différence de taille obtenue entre la fenêtre candidate et la fenêtre de résolution souhaitée et celle pour la fenêtre précédente. En effet, si la différence diminue, on prendra la nouvelle présentation et on gardera l'ancienne dans le cas contraire.

Cette technique, bien qu'originale, pose parfois quelques problèmes. Le premier repose sur la vitesse d'exécution. En effet, si le nombre de niveaux et de tâches par niveaux de l'arbre augmentent, le temps de calcul augmentera également car nous devons envisager tous les cas possibles.

Une solution serait peut-être de transformer ce problème de sélection en problème d'optimisation. Nous utiliserions alors une heuristique qui suggérerait à quel moment on arrête le choix.

Le deuxième problème réside dans le fonctionnement de la technique. En effet, si deux groupes de tâches feuilles indépendants se situent à un même niveau dans la hiérarchie de tâches, toutes deux seront réorganisées, ce qui n'est pas toujours souhaitable. De plus, elles partageront le même nombre de colonnes. Ainsi, même si le comportement général reste acceptable, certains résultats ne sont pas du tout optimaux.

Aussi, une solution encore plus complète serait de réaliser un mélange entre la première solution proposée et l'algorithme de Wong *[Wong 2002]*. En effet, l'algorithme de Wong ne réorganise les tâches que lorsque la largeur du groupe de tâches est supérieure à celle de la résolution choisie. Aussi, dans notre cas, ceci serait totalement inutile car notre méthode de génération automatique d'interfaces semi-concrètes se base sur l'usage d'une seule colonne par défaut.

Alors, plutôt que de calculer la meilleure combinaison possible dès le départ, nous pourrions également partir du facteur de coût des tâches pour savoir quel groupe (niveau) réorganiser en premier (ceci en additionnant les différentes valeurs obtenues). Et ensuite, nous nous chargerions de continuer la transformation jusqu'à ce que la taille de la présentation soit la plus proche de celle souhaitée.

Le module Widget Resizing Helper

Ce module a pour but de déterminer une liste de tâche pour lesquelles les OIAs seront redimensionnés pour atteindre les objectifs de résolution d'écran attendus. Cette détermination passe par plusieures étapes.

La première consiste à déterminer quels sont les OIAs qui sont redimensionnables et donc, quelles tâches seront à traiter par la suite. Remarquons que nous n'avons pas admis la possibilité de redimensionnement pour le widget *Canvas* car si celui-ci doit afficher des éléments graphiques, ceux-ci ne seront pas redimensionnés (à moins que la fonction sémantique d'affichage ne prenne en compte la résolution de l'écran, cas extrême que nous avons décidé d'ignorer). La deuxième consiste à créer la présentation complète et déterminer la taille des différents widgets qui la composent. De manière parallèle, nous construisons une liste de tâches organisées, de manière décroissante cette fois, en fonction de leur importance relative et récupérons les tailles *admissibles* pour les widgets de la présentation.

Ensuite, nous comparons la taille obtenue avec les limites autorisées pour chacune des tâches dans l'ordre fraîchement déterminé lors de la création de la liste décroissante. Si la taille est inférieure, nous changeons le statut de la tâche à 'non redimensionnable', ou laissons cela comme tel dans le cas contraire.

Dans la pratique malheureusement, cette opération ne fonctionne pas. En effet, bien que le principe soit acceptable, la récupération des tailles ne se fait pas de manière correcte. Ceci serait apparemment lié à un problème situé sur une couche inférieure à QTk.

Concrètement, plutôt que de renvoyer la taille correcte des objets, la méthode QTk de récupération des dimensions renvoie la plupart du temps la taille 1 pixel sur 1 pixel, ce qui est assez ennuyeux puisque notre méthode croit alors que l'objet interactif est trop petit et change son statut à 'non redimensionnable'. Même en utilisant {QTk.flush} censée bloquer les opérations jusqu'à ce que toutes les modifications opérées sur une fenêtre aient été réalisées, les résultats sont incorrects. En effet, en temps normal, tous les appels sur une fenêtre sont appliqués en les combinant tous en un seul lot pour des raisons d'efficacité. Aussi, les valeurs renvoyées pourraient être de 1 pixel sur 1 pixel et cette dernière méthode a donc pour objectif d'empêcher cela et récupérer les bonnes dimensions. Or, ceci n'est pas toujours le cas.

L'inconvénient supplémentaire est qu'il n'existe pas, à notre connaissance, d'autres techniques pour effectuer le redimensionnement d'objets interactifs sans avoir la possibilité de récupérer la taille des objets interactifs.

Le module Graphics Resizing Helper

La méthode de choix d'images contenue dans ce module a un fonctionnement en deux étapes. La première consiste à choisir la taille des images pour les éléments de présentation purs (qui ne se réfèrent à aucune tâche). Si ce choix permet d'atteindre les limites autorisées, nous arrêtons le traitement à cet endroit. Sinon, nous continuons en choisissant la taille des éléments graphiques du domaine qui sont manipulés par des tâches à représenter.

La première étape se déroule de la manière suivante : nous déterminons tout d'abord le nombre maximal d'images disponibles pour l'entièreté des éléments spécifiques de présentation. Ensuite, nous commençons la réduction du nombre d'images disponibles.

Nous vérifions la taille de la présentation. Si celle-ci n'est pas valable, nous prenons, pour tous les éléments, une image de format réduit (lors de la déclaration du modèle unique et des éléments spécifiques de présentation, la liste des images disponibles est déjà organisée de manière décroissante en fonction de la taille de l'image). Et nous continuons la réduction jusqu'à ce qu'il n'y ait plus de possibilité de le faire ou que la résolution cible soit atteinte (*remarque* : certains éléments spécifiques de présentation peuvent être optionnels. Ceci implique que certaines images peuvent être carrément supprimées).

Si la première étape n'a pas mené à un résultat concluant, nous passons à la deuxième étape. Nous générons une liste croissante de tâches en fonction de leur importance relative (coût). Ensuite, nous appliquons la réduction dans l'ordre obtenu jusqu'à ce que la résolution cible soit atteinte ou qu'il n'y ait plus de possibilité de choix d'images dans un format réduit.

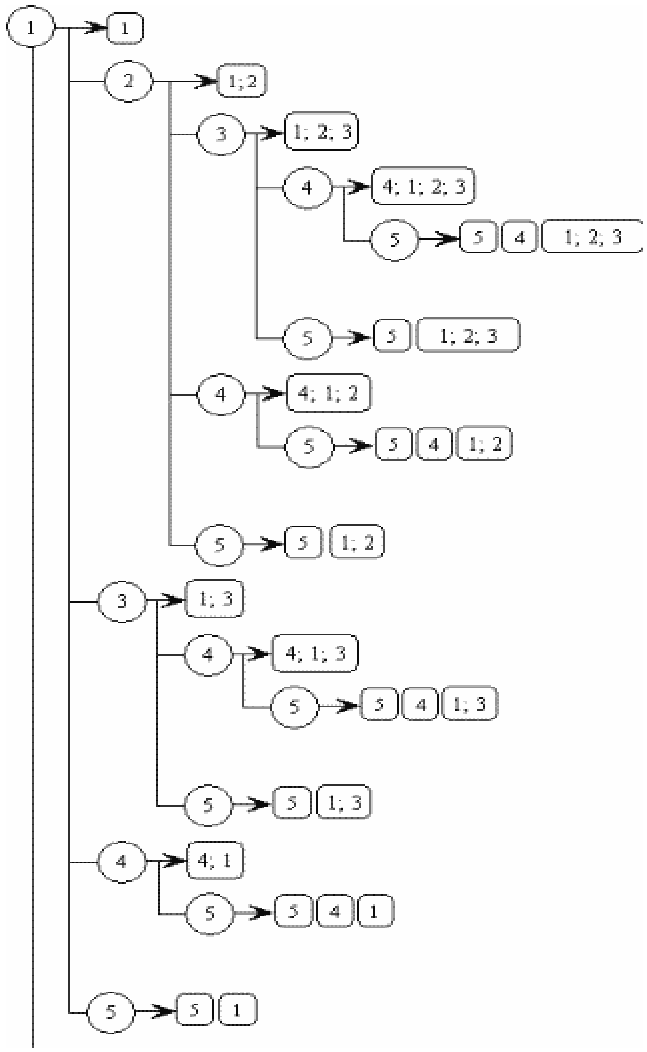
5.7.7 La Couche Transformation d'interface

Cette couche est composée de deux modules, nous ne décrirons en détails que l'un d'entre eux.
Le premier module, **Concrete CUI Generator**, construit des CUIs où les fonctions sémantiques sont transformées pour que les procédures se basent sur de réelles poignées QTk et non plus sur des variables libres, puisque les OIAs ont été affectés de manière définitive.

Le deuxième module est celui sur lequel nous allons le plus nous attarder. Sa fonctionnalité consiste à appliquer les règles de dégradation souhaitées par l'utilisateur de manière respectueuse par rapport à la propriété de continuité, explicitée dans *[Florins & Vanderdonckt 2004]*.

Nous avons donc réalisé un arbre de décision d'application *suffisante* de ces règles (comprenons ici par *suffisante* que toutes les règles choisies ne seront pas forcément appliquées si la résolution atteinte par la fenêtre cible est inférieure ou égale à celle de la plate-forme).

L'arbre exposé ici est celui que nous avons retenu pour notre application. Celui-ci n'est pas réellement complet, certaines *décisions* ayant été simplifiées quelque peu. Aussi, le lecteur pourra trouver une version plus complète en annexes de ce mémoire.



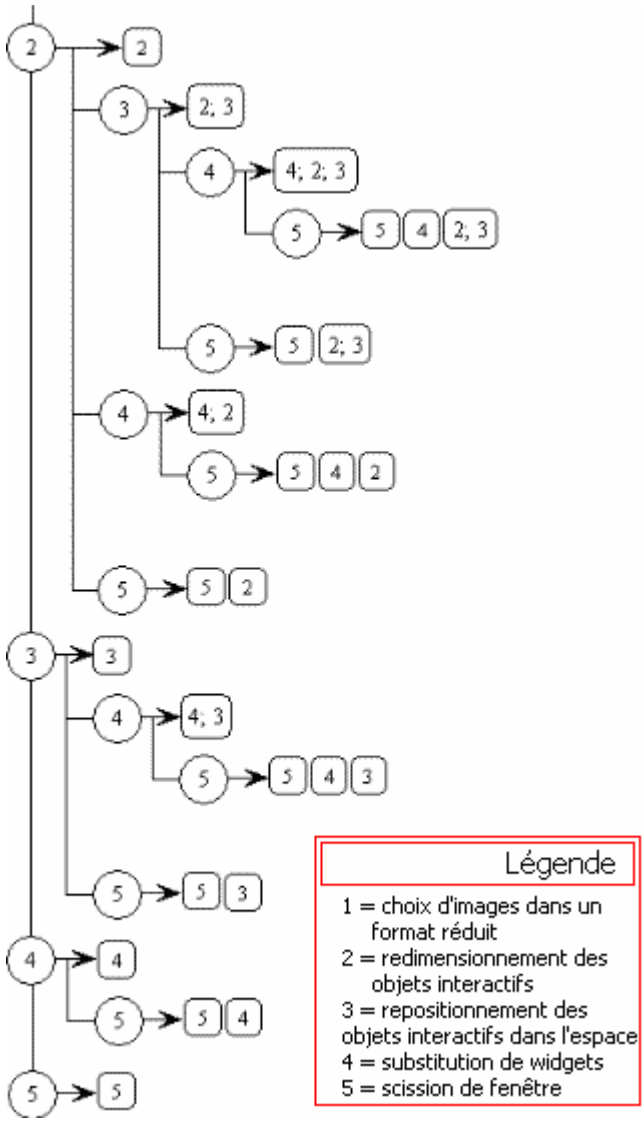


Figure 54 – Arbre de décision d'application de règles de dégradation harmonieuse sur une présentation

Pour cet arbre de décision, chacun des cercles signifie qu'une règle a été choisie par l'utilisateur pour application. Les rectangles aux coins arrondis signifient l'application d'une règle. Remarquons que lorsque la règle de scission est considérée, nous n'appliquons pas forcément aux différentes unités de présentation résultantes les autres règles de dégradation (c'est uniquement fait si cela s'avère nécessaire). Ainsi, par exemple, dans l'avant dernier cas, nous n'appliquons la substitution que si cela aura une incidence sur la résolution de l'unité de présentation considérée.

5.7.8 La Couche Visualisation d'interfaces concrètes

Cette couche ne se compose que d'un seul module qui a pour but de construire une fenêtre composée d'un *placeholder* QTk qui contiendra la présentation si elle est unique, ou encore d'un *placeholder* qui contiendra toutes les présentations possibles et d'un bouton qui permet de naviguer de présentation en présentation si la multiplicité des présentations est supérieure à un.

5.8 Cas d'études et résultats obtenus

Nous allons exposer dans cette section les deux cas d'études que nous avons retenus pour notre application de dégradation harmonieuse ainsi que quelques-uns des résultats que nous avons obtenus en leur appliquant nos règles de dégradation harmonieuse.

5.8.1 Cas d'études

Premier cas d'étude : un formulaire d'abonnement pour un magazine

Ce cas-ci est relativement simple, il s'agit d'un simple formulaire que l'utilisateur remplira et validera ou annulera par la suite. Le modèle du domaine comportera une seule classe : Client.

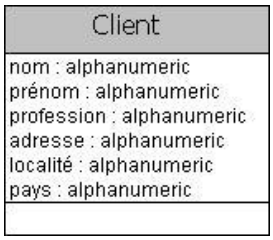


Figure 55 – Modèle du domaine du cas d'étude numéro un de notre application de dégradation harmonieuse

Voici le modèle de la tâche retenu pour notre formulaire d'abonnement :

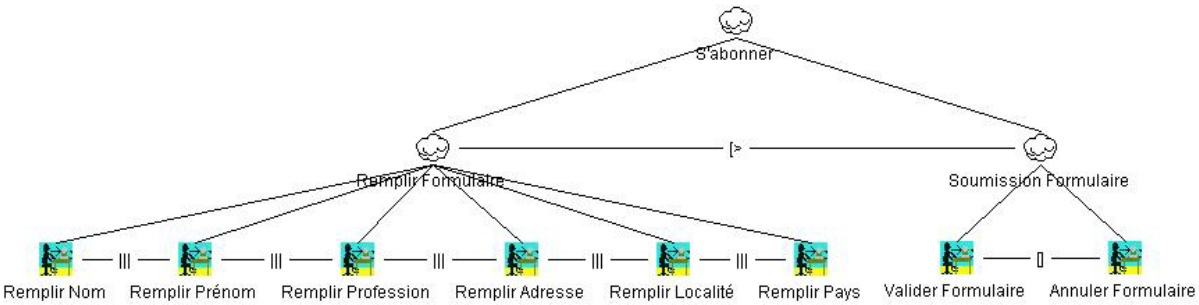


Figure 56 – Modèle de tâche du cas d'étude numéro un de notre application de dégradation harmonieuse

Deuxième cas d'étude : un guide de l'aquariophile débutant

Le deuxième cas se veut être un guide pour le débutant en aquariophilie. Il va permettre à l'utilisateur de déterminer quels sont les poissons adéquats aux caractéristiques de son aquarium et aux caractéristiques du futur nouveau pensionnaire. Le modèle du domaine est composé de deux classes principales: Aquarium et Poisson.

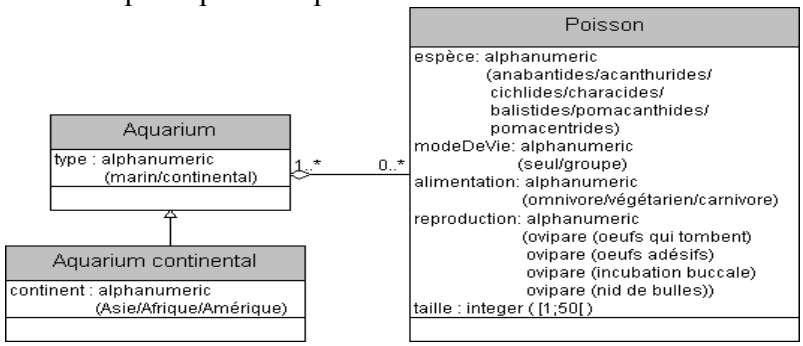


Figure 57 – Modèle du domaine du deuxième cas d'étude de notre application de dégradation harmonieuse

Le modèle de la tâche est, par contre, beaucoup plus complexe que dans le premier cas d'étude :

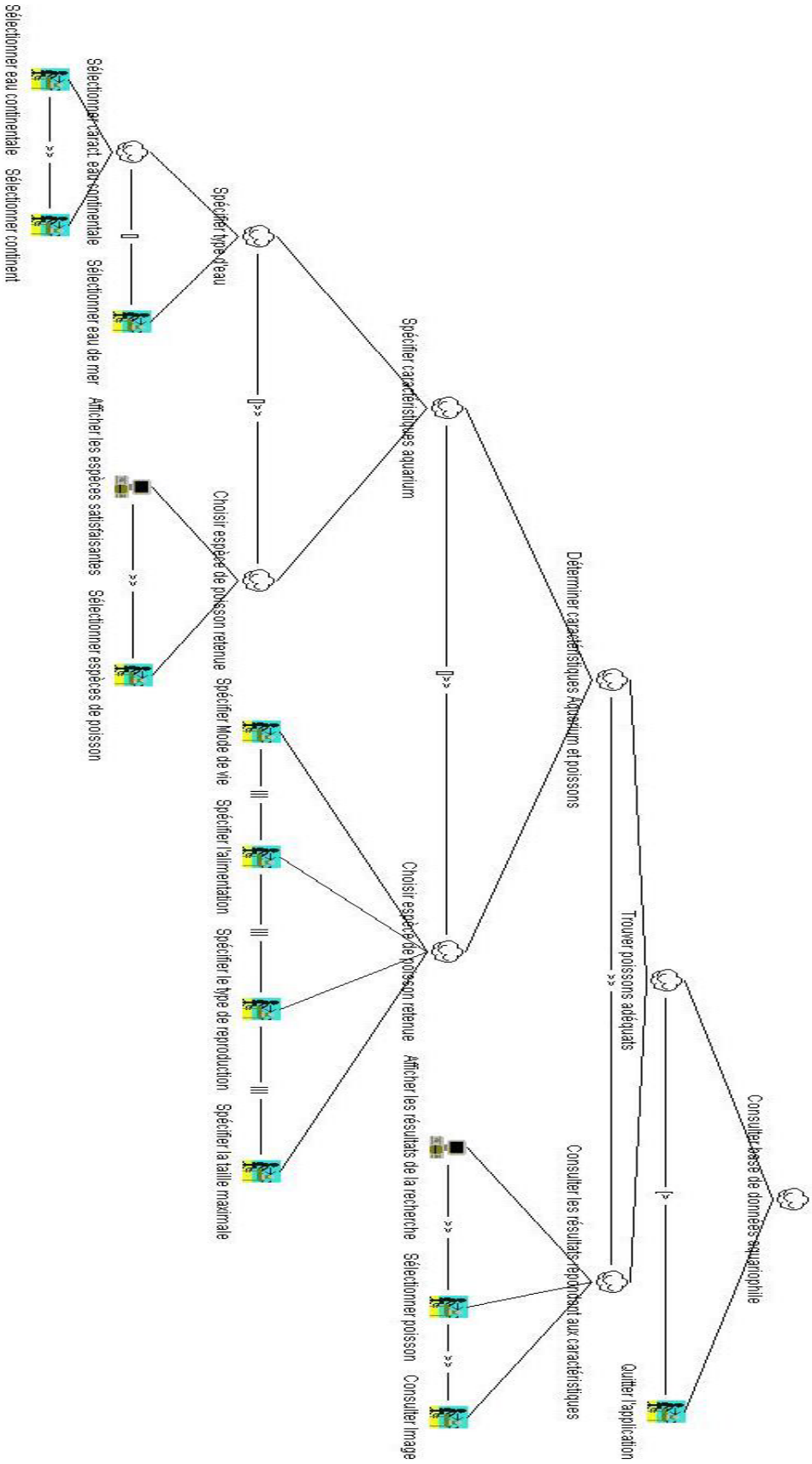


Figure 58 - Modèle de tâche du deuxième cas d'étude de notre application de dégradation harmonieuse

5.8.2 Quelques résultats

Nous articulerons et commenterons quelques résultats choisis parmi ceux obtenus en fonction des règles de dégradation utilisées. Le premier donne uniquement une idée des présentations générées par l'application. Nous n'aborderons pas le cas de la règle de redimensionnement d'objets étant donné les problèmes déjà cités (cfr. supra).

Génération simple (aucune règle de dégradation)

Un seul regret concerne l'alignement des widgets (c'est par exemple le cas en ce qui concerne les boîtes d'édition pour le nom et le prénom sur la figure 59). C'est sans aucun doute un des points qu'il faudra améliorer dans le cadre d'une future mise à jour. Néanmoins, n'oublions pas qu'il s'agit d'une application de dégradation harmonieuse d'interfaces utilisateur et pas d'une application de génération automatique d'IU. Chose curieuse, lors d'une génération et un affichage, parfois les boutons radios d'un même groupe ont un élément mis à *true*, parfois pas. Ceci explique, entre autres, pourquoi le bouton « Valider » du premier exemple se trouve parfois dans l'état sélectionné.

Figure 59 – Cas de génération simple du premier cas d'étude (320x320)

Figure 60 - Cas de génération simple du deuxième cas d'étude (640x480)

Premier test : Choix d'images dans un format réduit / Suppression d'images

L'image de titre a été réduite et celle de guidage « Veuillez … », optionnelle, a été supprimée. On voit donc que les éléments spécifiques à la présentation sont réduits au maximum pour satisfaire aux contraintes de résolution.

De même, les éléments spécifiques au modèle de la tâche ont également été réduits puisque les boutons Valider et Annuler ont également été réduits.

Figure 61 – Règle de choix et de suppression d'images pour le cas d'étude numéro un (320x320)

Nous ne présenterons pas d'exemple concernant le deuxième cas, car les résultats vont dans le même sens.

Deuxième test : Repositionnement des objets interactifs dans l'espace

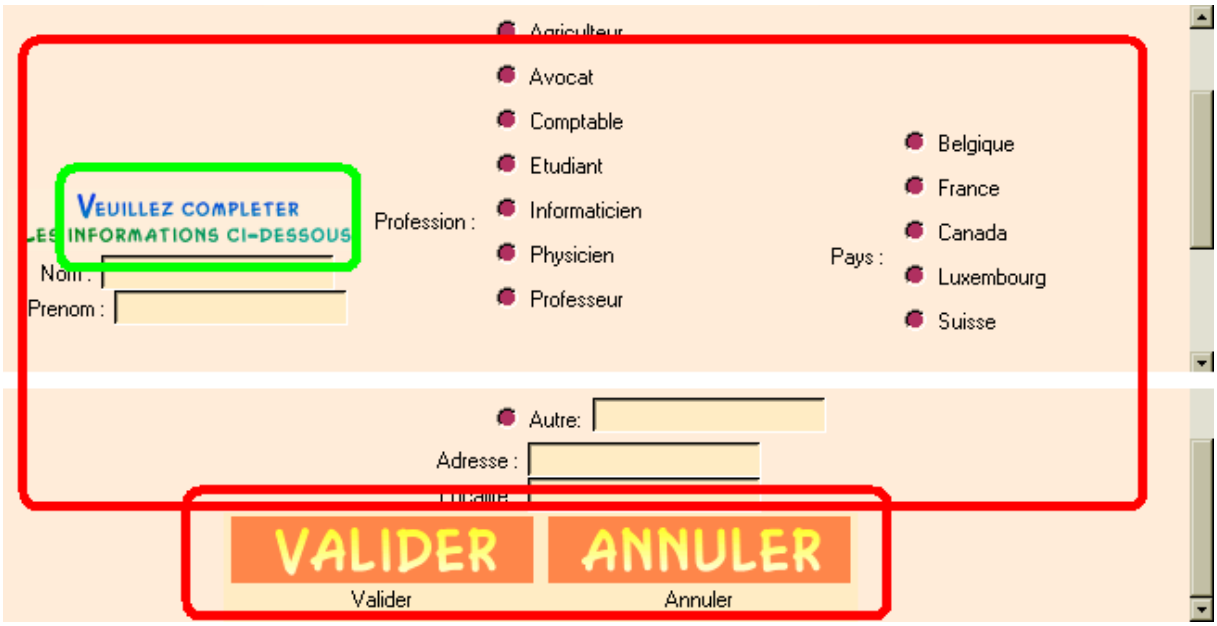


Figure 62 - Règle de repositionnement d'objets interactifs pour le cas d'étude numéro un (640x200)

Les résultats montrent que les tâches du niveau 2 ont été réorganisées et que le nombre de colonnes choisi est de trois (cfr. grand encadré rouge). De plus, puisque les tâches de validation du formulaire se situent au niveau 2 également, nous les réorganisons aussi (puisque le nombre d'éléments est inférieur à trois, il n'y a que deux colonnes).

Un autre problème, moins ennuyeux certes, mais qui nuit à la lecture réside dans le repositionnement de l'élément spécifique à la présentation encadré en vert. C'est encore un autre point qu'il faudra surveiller dans une future mise à jour.

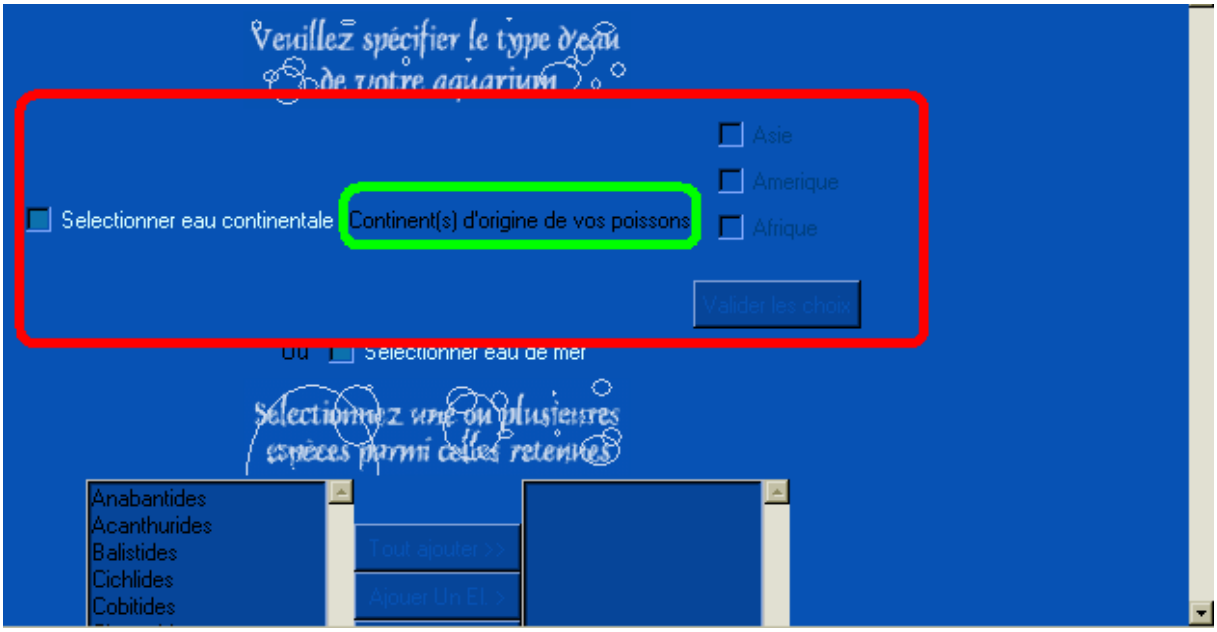


Figure 63 - Règle de repositionnement d'objets interactifs pour le cas d'étude numéro deux (640x480)

Dans le cadre du cas d'étude numéro deux, on constate que seul le dernier niveau a été réorganisé et que le nombre de colonnes est de deux (et non pas trois) comme suggéré dans l'encadré en rouge. On constate également que le libellé dans l'encadré vert n'a pas été repositionné. Cela peut s'expliquer en sachant que le libellé et la tâche de sélection d'un continent se situent déjà, à la base dans un autre conteneur en plus du conteneur principal du niveau.

Ceci n'est toutefois pas réellement gênant car cela nous permet de garder le même sens de lecture. Néanmoins, on peut s'interroger sur l'intérêt, dans ce cas-ci, de réorganiser les tâches horizontalement, pour réaliser un gain relativement faible en hauteur. Bien entendu, cela satisfait entièrement aux contraintes de largeur de la fenêtre mais d'un point de vue esthétique, cette réorganisation n'est pas des plus réussies.

Troisième test : Substitution d'objets interactifs concrets

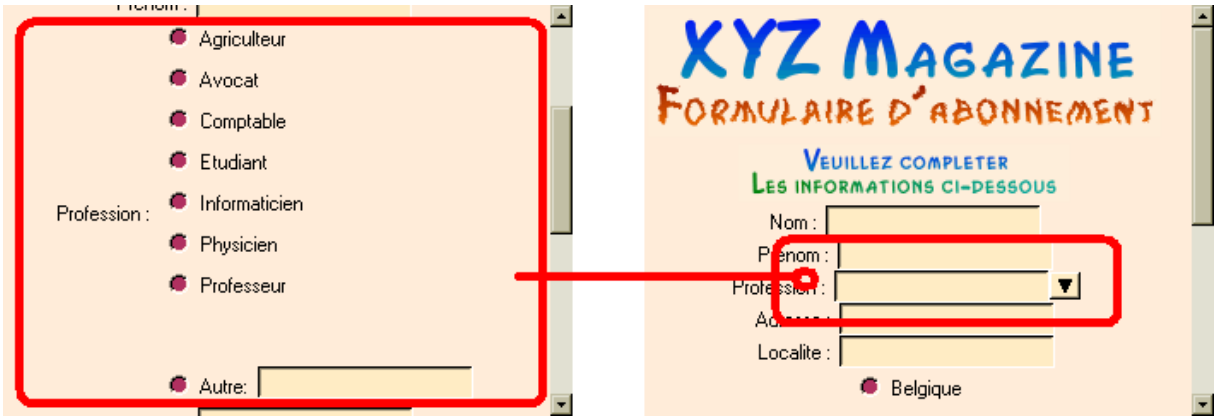


Figure 64 - Règle de substitution d'objets interactifs pour le cas d'étude numéro un (320x240)

Dans le premier cas, on voit que le widget de spécification d'une profession (boutons radios + libellé + champ d'édition) a été remplacé par une liste de combinaisons déroulante.

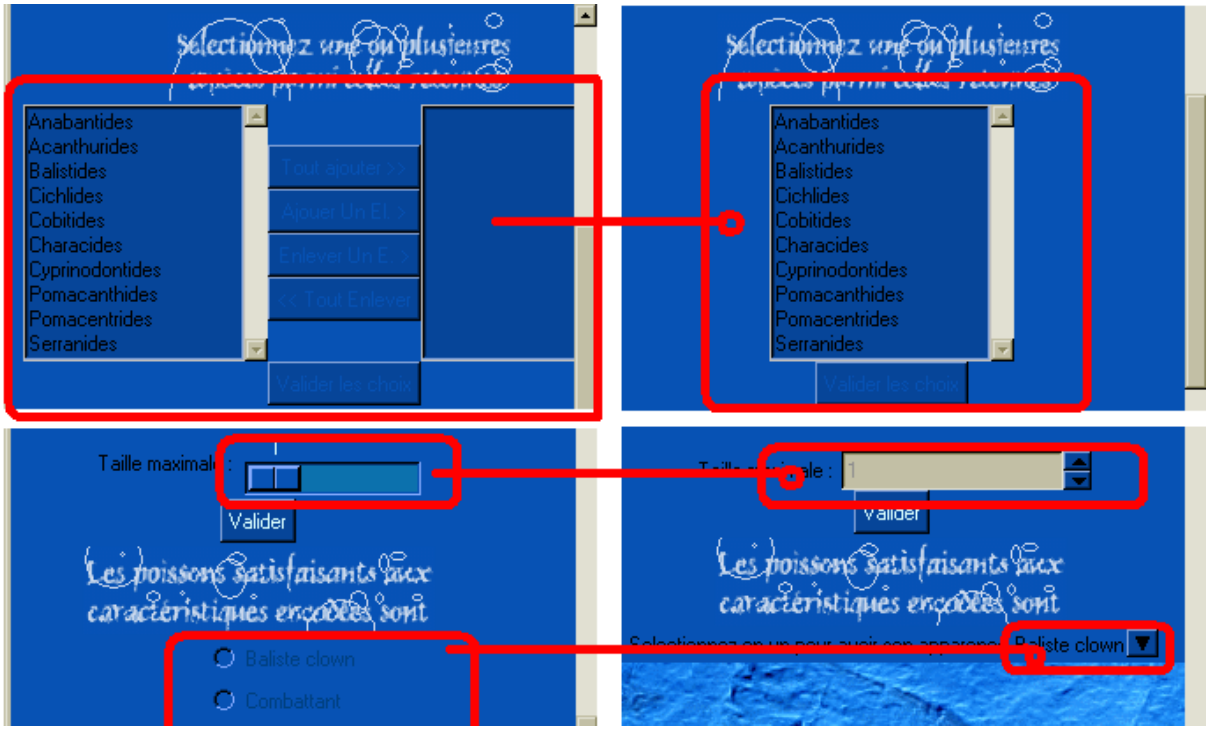


Figure 65 - Règle de substitution d'objets interactifs pour le cas d'étude numéro deux (320x240)

Dans le deuxième cas, plusieurs substitutions ont été réalisées. Un accumulateur s'est transformé en liste de sélection multiple, une échelle horizontale s'est fait substituer par un bouton de variation et enfin, une liste de boutons radio a été remplacée pour laisser la place à une liste de sélection déroulante.

Notons que le fait de substituer les widgets peut apporter des effets indésirables. En effet, après avoir substitué l'accumulateur par une liste de sélection multiple et l'échelle par le bouton de variation, nous obtenons quelques anomalies en ce qui concerne le fonctionnement de l'application.

Le problème se situe au niveau de la sélection d'éléments par l'utilisateur. Imaginons le cas suivant : l'utilisateur sélectionne les éléments de la liste de sélection multiple et valide ses choix. Peu après, l'utilisateur arrivant à l'étape de spécification de la taille du poisson, celui-ci clique sur le bouton de variation. Pas de problème jusque là, si ce n'est que les éléments précédemment sélectionnés dans la liste sont oubliés car le curseur de sélection n'est plus positionné sur ces derniers. Aussi, lorsque l'utilisateur déclenchera la recherche, il n'aura aucun résultat.

Voici donc encore un problème auquel il faudra trouver une solution dans le cadre d'une future mise à jour.

Quatrième test : Scission de fenêtres

La scission en plusieurs fenêtres en ce qui concerne le premier cas d'étude mène à six fenêtres destination pour une fenêtre source.

Remarquons la répartition des tâches de validation ou annulation des formulaires entre toutes les présentations, puisque celles-ci sont liées à l'envoi d'un formulaire (opérateur de désactivation).

De plus, les tâches de remplissage du formulaire ne conservent plus leur statut de tâches concurrentes et deviennent des tâches séquentielles. Il faut valider chaque tâche complétée (dans le cas du champ d'édition, nous avons affaire à un mode de déclenchement explicite non affiché (il faut appuyer sur la touche `<Return>` pour valider l'opération)).

Figure 67 – Règle de scission de fenêtres pour le cas d'étude numéro deux (640x480)

Dans ce second cas d'étude, la scission mènera à huit fenêtres séparées. Remarquons, dans le cas précis de la figure 67, que les tâches de sélection du type d'eau n'ont pas été séparées puisqu'elles se partagent un opérateur de choix.

Remarquons également que ce qui a été cité précédemment en ce qui concerne la mise à *vrai* aléatoire pour le premier élément d'une liste de boutons radio lors de la génération automatique des interfaces peut poser de réels problèmes à l'utilisateur de l'application testée. En effet, si celui-ci veut justement sélectionner le premier élément, il ne cliquera pas forcément sur le bouton radio puisque celui-ci semble déjà sélectionné. Or, si cette tâche est séquentielle, il n'aura pas la possibilité d'effectuer la tâche suivante, après avoir appuyé sur le bouton `Suite>>`.

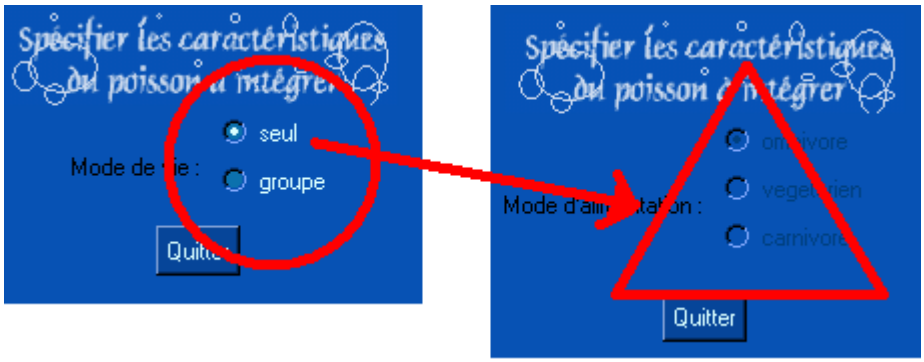


Figure 68 – Problèmes liés aux boutons radio dans le cadre d'une scission de fenêtre

Autres tests

Nous n'allons pas décrire l'entièreté des résultats obtenus pour toutes les combinaisons possibles de règles de dégradation harmonieuse (d'autant plus qu'il existe en tout 26 possibilités !). Néanmoins, les cas de base donnent un bon aperçu des possibilités de l'application. Aussi, nous invitons l'utilisateur curieux à tester lui-même les combinaisons qu'il souhaite appliquer.

Chapitre 6 : Conclusions.

Le travail réalisé dans ce mémoire nous a permis de nous rendre compte de la complexité qui ressort de l'étude de système de transitions de fenêtres et de la mise en place de règles de dégradation harmonieuse au sein d'une application. C'est, en effet, en creusant plus profondément cette problématique, simple en apparence, que l'étude réalisée a fait apparaître une série de nouveaux problèmes supplémentaires auxquels il a fallu attribuer un statut privilégié. On peut citer, par exemple, la nécessité de disposer d'une technique de représentation de transitions de fenêtres robuste et lisible dans la partie théorique ou encore l'obligation de représenter certaines structures de manière abstraite dans notre application QTk.

Ce travail nous a également permis de nous rendre compte de l'étendue du problème et nous sommes conscients que nous n'avons abordé qu'une infime partie. Néanmoins, cette étude s'est révélée très enrichissante et nous a permis de soulever les points suivants :

- Elle nous a notamment permis de réfléchir à la mise en place d'*une structure adéquate pour les transitions de fenêtres* d'une application du type FlexClock
- Ceci nous a également fait entrevoir le développement d'*une future application* permettant de construire de telles représentations sur base du modèle de la tâche que nous avons imaginé.
- Elle a renforcé *l'utilité d'une approche basée modèle* dans le cadre de l'étude de règles de dégradation, et de l'utilisation d'une structure caractérisant différent niveaux d'abstraction d'une interface utilisateur.
- Nous avons pu mettre en place *un arbre de sélection d'OIA* pour l'environnement étudié.
- Nous avons pu évaluer *l'implémentatibilité* d'un tel système et des quelques règles que nous avons décidé de considérer pour l'environnement choisi.
- Nous avons aussi pu nous rendre compte de l'application non-systématique de ces règles de dégradation par rapport aux cas d'études considérés et des *différents raffinements* qu'il reste encore à réaliser pour avoir de meilleurs résultats.
- Enfin, nous avons eu l'opportunité de développer *un arbre de décision* d'application de règle suivant *la propriété de continuité* des interfaces.

Aussi, au regard des résultats obtenus et de ce qui a été abordé, les perspectives sont multiples :

- Implémentation d'un système de génération de diagrammes d'état-transition sur base du travail préliminaire réalisé dans la partie théorique
- Amélioration des algorithmes de traitement et d'application de règles de dégradation
- Ajout de fonctionnalité non couverte dans notre modèle unique
- Ajout de règles de dégradation
- Création d'un éditeur générant du code Oz à partir de fichiers USIXML de manière à permettre à l'utilisateur de tester la dégradation sur ses propres interfaces
- Amélioration de l'arbre de décision d'application de règles
- Création de plusieurs arbres de sélection d'OIA en fonction des possibilités de différentes plate-formes
- …

Cette liste non exhaustive montre bien les possibilités qui s'offrent à nous dans l'approfondissement du sujet. Cependant, si nous devions attribuer un certain degré d'importance à chacune des idées émises, les travaux futurs à réaliser sur lesquels il faudra s'attarder en premier sont sans aucun doute la réalisation du plugin de génération de diagrammes d'état-transition et l'amélioration des algorithmes consacrés à l'application des règles.

La partie théorique envisageant le développement du plugin PlastiXML n'est en effet développée que de manière très succincte, et il vaut la peine de s'attarder d'avantage sur cet partie dans le cadre d'un futur travail supplémentaire.

De même, il serait plus judicieux d'accorder du temps à l'amélioration des règles déjà étudiées dans le cadre de ce mémoire que de vouloir directement en augmenter le nombre. Les résultats ont fait apparaître quelques cas où l'apparence des présentations dégradées était assez discutable sur le plan de l'ergonomie ou encore sur le plan de la légitimité d'appliquer les règles de transformation choisies pour ces cas spécifiques. Il reste donc encore du travail à réaliser si nous voulons obtenir de meilleurs résultats.

Enfin, lorsque tous ces objectifs auront été atteints, il sera intéressant d'accroître l'espace des règles de dégradation et celui des modèles considérés.

Références bibliographiques

[Allen 83]
Allen J.F., Maintaining knowledge about temporal intervals, Communications of the ACM, Vol. 26, No. 11, November 1983, pp. 832-843

[Ali 2003]
Ali M.F., Pérez-Quiñones M.A. and Abrams M., Building Multi-Platform User interfaces with UIML, in A. Seffah & H. Javahery (eds.) Multiple User Inetrfaces: Engineering and Application Framework, Jhon Wiley and Sons, 2003.

[Calvary & al. 2002]
Gaëlle Calvary, Joëlle Coutaz, *Plasticité des interfaces : une nécessité !*, Actes des deuxièmes assises nationales du GdR I3, 2002, pp 248-260.

[Calvary & al. 2003]
Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J., *A Unifying Reference Framework for Multi-Target User Interfaces*, Interacting with Computers, Vol. 15, No. 3, June 2003, pp. 289-308.

[Crease et al. 2000]
M. Crease, P. D. Gray, S. A. Brewster, A Toolkit of Mechanism and Context Independent Widgets. Actes du workshop Design, Specification, and Verification of Interactive Systems, DSVIS'00, 2000, pp. 121-133.

[Dieterich et al. 1993]
H. Dieterich, U. Malino wski, T. Kühme, M. Schneider - Hufschmidt. State of the Art in Adaptive User Interfaces, Adaptative User Interfaces : Principles and practice, Schneider-Hufschmidt et al., 1993, pp.13-48.

[Florins 2003]
Florins M., *Graceful Degradation of User Interfaces,* , DEA Interuniversitaire en Informatique, Mémoire, Université Catholique de Louvain, September 2003.

[Florins & Vanderdonckt 2004]
Florins M. and Vanderdonckt J., *Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems*, in Proceedings of the 2004 International Conference on Intelligent User Interfaces IUI 2004 (Funchal, Madeira Island, Port.,January 13-16, 2004)

[Genova]
E. Arisholm, Incorporating Rapid User Interface Prototyping in Object-Oriented Analysis and Design with Genova, Department of Informatics, University of Oslo, PO Box 1080 Blindern, N-0316 Oslo, Norway (e-mail: erikar@ifi.uio.no)
Hans Christian Benestad, Jon Skandsen, Hege Fredhall, Genera A/S, Marcus Thranes gate 2, N-0473 Oslo, Norway (e-mail: {hcb,jsk,hfr}@genera.no)

[Grolaux & al. 2002]
Grolaux D., Van Roy P., and Vanderdonckt J., FlexClock : A Plastic Clock Written in Oz with the QTk Toolkit, *in Proceedings of 1st International WorkShop on Task Models and Diagrams for User Interface Design Tamodia'2002 (Bucharest 18-19 July 2002)*, Academy of Economic Studies of Bucharest, 2002, pp. 135-142

[Keränen & Plomp 2002]
Keranen H., Plomp J., Adaptive Runtime Layout of Hierarchical UI Components, in *Proceedings of NordiCHI'2002,* ACM Press, New York, pp. 251-254.

[Limbourg 2001]
Quentin Limbourg, *Introducing the mapping Problem in a Model-Based Approach of User Interfaces Development*, DEA en Sciences de Gestion, Universtié Catholique de Louvain, Sept. 2001

[Luyten & al. 2003]
Luyten K., Clerckx T., Coninx K. & Vanderdonckt J., Derivation of a Dialog Model from a Task Model by Activity Chain Extraction, In *Preliminary Proceedings of DSV-IS'2003 (Funchal, Madeira Island, Portugal, 4-6 June 2003)*, pp. 191-220.

[Mozart]
Mozart Consortium: The Mozart Programming System (Oz 3), accessible at :
http://www.mozart-oz.org/documentation

[Myers92-1]
B. Myers, Survey on User Interface Ace Programming. Porceeding of Human Factors in Conputing Systems (CHI'92), Addison Wesley, 1992, pp.195-202.

[Myers92-2]
Myers, Brad A. and Zanden, Brad Vander. Environment for Rapidly Creating Interactive Design Tools , The Visual Computer, 1992

[Paternó 97]
F.Paternó, C.Mancini, S.Meniconi, "ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models", Proceedings Interact'97, pp.362-369, July'97, Sydney, Chapman&Hall.

[Phanariou 2000]
C. Phanariou, UIML: a Device-Independent User Interface Markup Language. Thèse pour l'obtention du titre de docteur en informatique, Institut Polytechnique de Virginie (faculty of Virginia Polytechnic Institute and State University), Blacksburg, Virginie, septembre, 2000.

[QTk]
Donatien Grolaux, Peter Van Roy, Jean Vanderdonckt, QTk : An integrated Model-Based Approach to Design Executable User Interfaces, Université catholique de Louvain, Belgium, 2001.

[Schneider 2002]
Schneider, K., and Cordy, J. Aui: A programming language for developing plastic interactive software. In Proc. HICSS-35 - Hawaii Int'l Conf. on the System Sciences (Waikoloa, Hawaii, Jan. 2002), pp. 281–291.

[Souchon 2002]
Nathalie Souchon, *Towards a Computational Notation for Supporting Context-Sensitive User Interface Development*, DEA en Sciences de Gestion, Université Catholique de Louvain, Sept. 2002

[Szekely95]
P. Szekely, P. Sukaviriya, P. Castells, J. Muthukumarasamy, E. Salcher, Declarative interface models for user interface construction tools: the Mastermind approach. In EHCI'95, 1995, pp.120-150.

[Szekely96]
P. Szekely, Retrospective and challenges for Model-Based Interface Development in Proceedings of Computer-Aided Design of User Interfaces (CADUI'96), Presses Universitaires de Namur, 1996, pp. xxi-xliv.

[Thevenin & Coutaz 99]
Thevenin D., Coutaz, J., Plasticity of User Interfaces: Framework and Research Agenda, *Proceedings of 7th IFIP Int. Conf. on Human-Computer Interation INTERACT'99* (Edinburgh, 30 August-3 September 1999), IOS Press, Amsterdam, pp.110-117

[Trident]
F. Bodart, A.M Hennebert, J.M Leheureux, B. Sacré, I. Provot, J. Vanderdonckt, An Overview of the TRIDENT Project. Transparent sur le projet TRIDENT, Facultés Universitaires Notre-Dame de la Paix, Institut d'Informatique, Namur, Belgique.
http://www.info.fundp.ac.be/~emb/Trident.html

[Ullman 79]
Hopcroft, J. E. and Ullman, J. D. (1979), *Introduction to automata theory, languages, and computation*, Addison-Wesley, Reading, MA

[USIXML]
Limbourg Q., Vanderdonckt J., Michotte B., Bouillon L., Florins M., Trevisan D., *USIXML: A User Interface Description Language for Context-Sensitive User Interfaces*, in Proceedings of the ACM AVI'2004 Workshop "Developing User Interfaces with XML: Advances on User Interface Description Languages" (Gallipoli, May 25, 2004), K. Luyten, M. Abrams, Q. Limbourg, J. Vanderdonckt (Eds.), Gallipoli, 2004, pp. 55-62
http://www.usixml.org

[Vanderdonckt & Bodart 93]
Vanderdonckt J., Bodart F., Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection, *in Proceedings of the Conference on Human Factors in Computing Systems INTERCHI'93 (Amsterdam, 24-29 April 1993)*, ACM Press, New York, 1993, pp. 424-429

[Vanderdonckt 1997]
Vanderdonckt J., *Conception assistée de la présentation d'une interface homme-machine ergonomique pour une application de gestion hautement interactive*, Thèse pour l'obtention du titre de docteur en informatique, Facultés Universitaires Notre-Dame de la Paix, 1997.

[Vanderdonckt 1998]
Vanderdonckt J., *Règles ergonomiques de sélection d'objets interactifs,* Actes du 6ème Colloque Ergonomie et Informatique Avancée (Biarritz, 4-6 novembre 1998), M.-F. Barthet (éd.), ESTIA/ILS, Bidart, 1998, pp. 250–259.

[Vanderdonckt & al. 2003]
Vanderdonckt J., Limbourg Q., Florins M., *Deriving the Navigational Structure of a User Interface*, Proc. of 9th IFIP TC 13 Int. Conf. on Human-Computer Interaction Interact2003 (Zurich, 1-5 September 2003), M. Rauterberg, M. Menozzi, J. Wesson (Eds.), IOS Press, Amsterdam, 2003, pp. 455-462

[Van Roy & Aridi 2003]
Van Roy P., Aridi S., Concepts, techniques, and models of computer programming, MIT Press, 2003

[Watson 2003]
Watson, M. Paui syntax and server: a status report. Tech. Rep. SE Lab WP03-201-MDW, University of Saskatchewan Software Engineering Lab, 2003.

[Watson 2004]
Watson, M. Interactions, Transformations and AUI3D, Tech. Rep. SE Lab CMPT856, University of Saskatchewan Software Engineering Lab, April 29, 2004

[Wong 2002]
Wong C., Chu H.H. and Katagiri M. A single-Authoring technique for Building Device-independent Presentations, in proceedings of W3C Workshop on Device Independent Authoring Techniques (St. LeonRot, 15-16 September 2002)

[Xweb]
Olsen, D.R., Jefferies, S., Nielsen, T., Moyes,P., Fredrickson, P., Cross Modal Interaction using XWEB, in *Proc. of the 13th annual ACM symposium on User interface software and technology UIST 2000* (San Diego, United States), ACM Press, New York, pp. 191-200.

Annexes

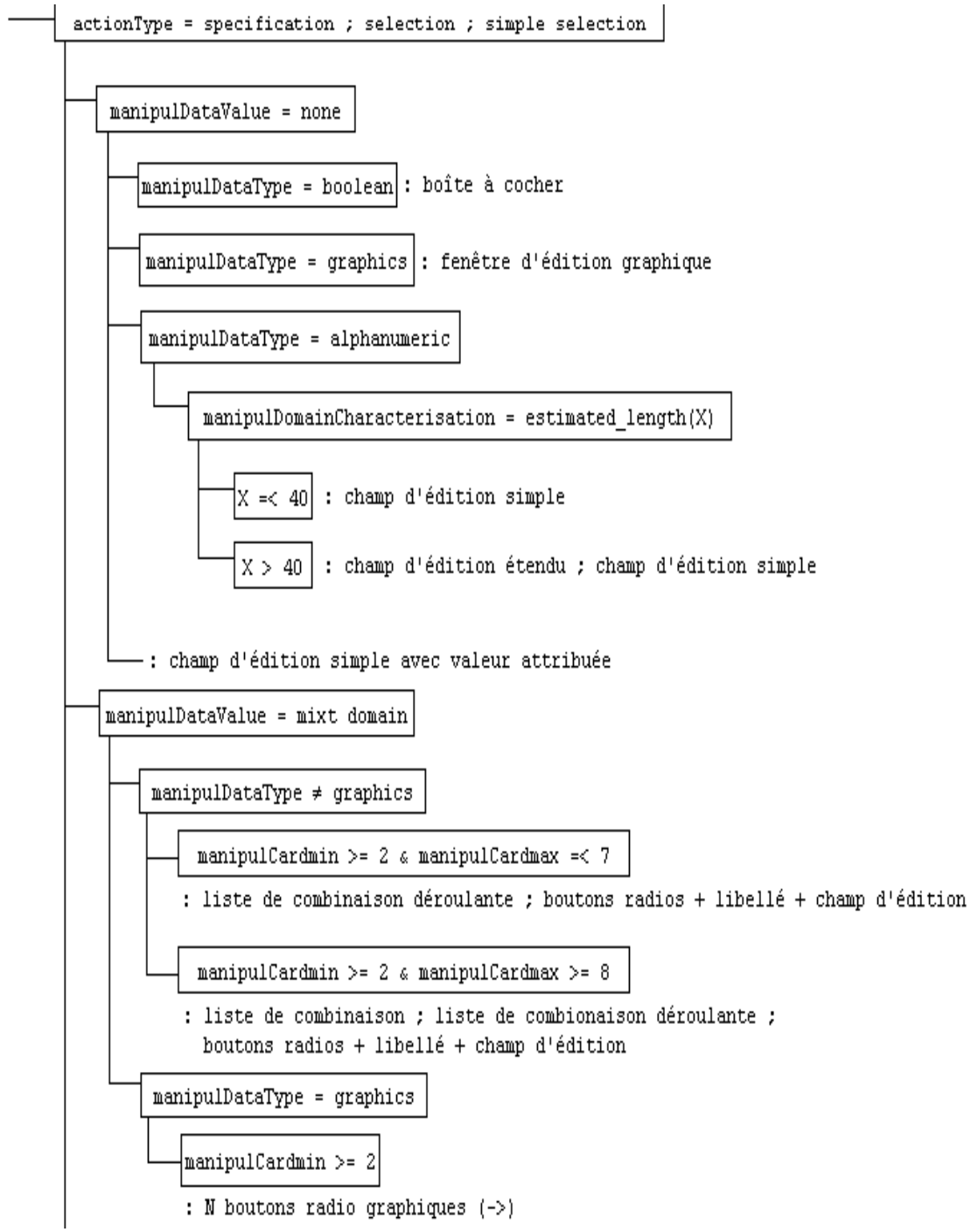
Code de l'application éducative au fonctionnement similaire à FlexClock

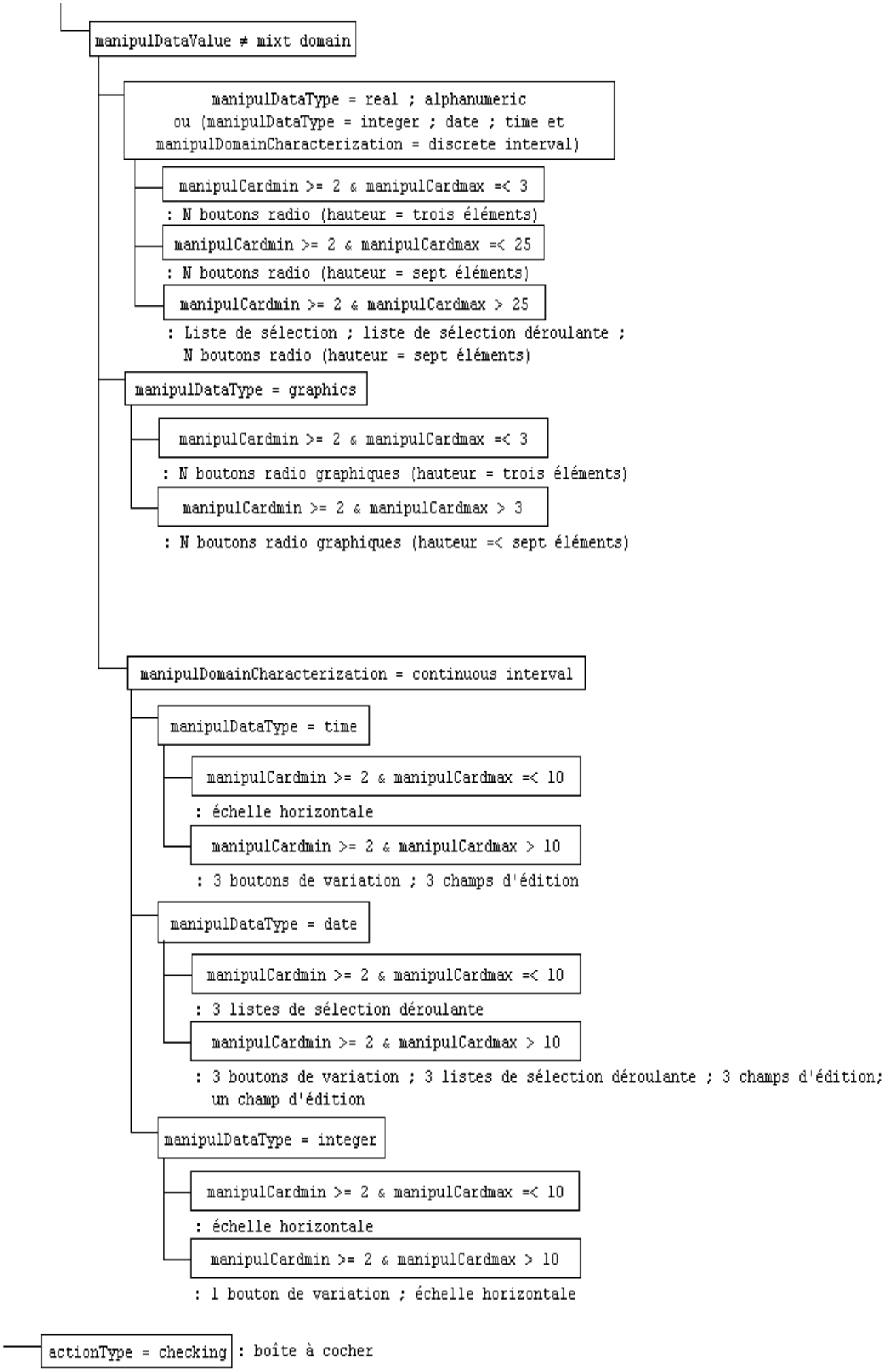
(consulter le fichier FlexD.oz sur le cd-r)

Code d'automatic Country Selector

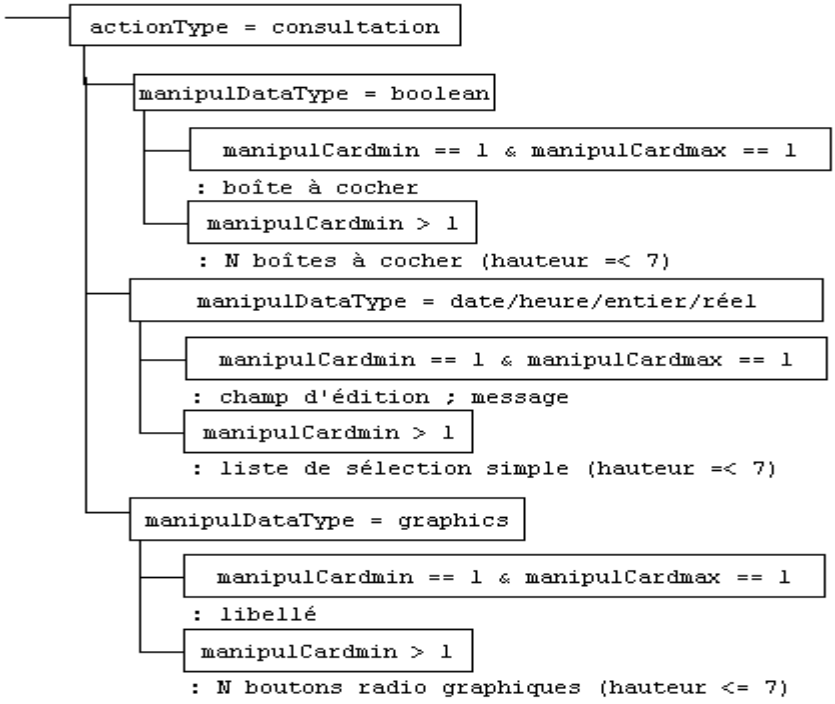
(consulter le fichier AutomaticCS.oz sur le cd-r)

Arbre de sélection préférentielle d'OIA pour une tâche donnée

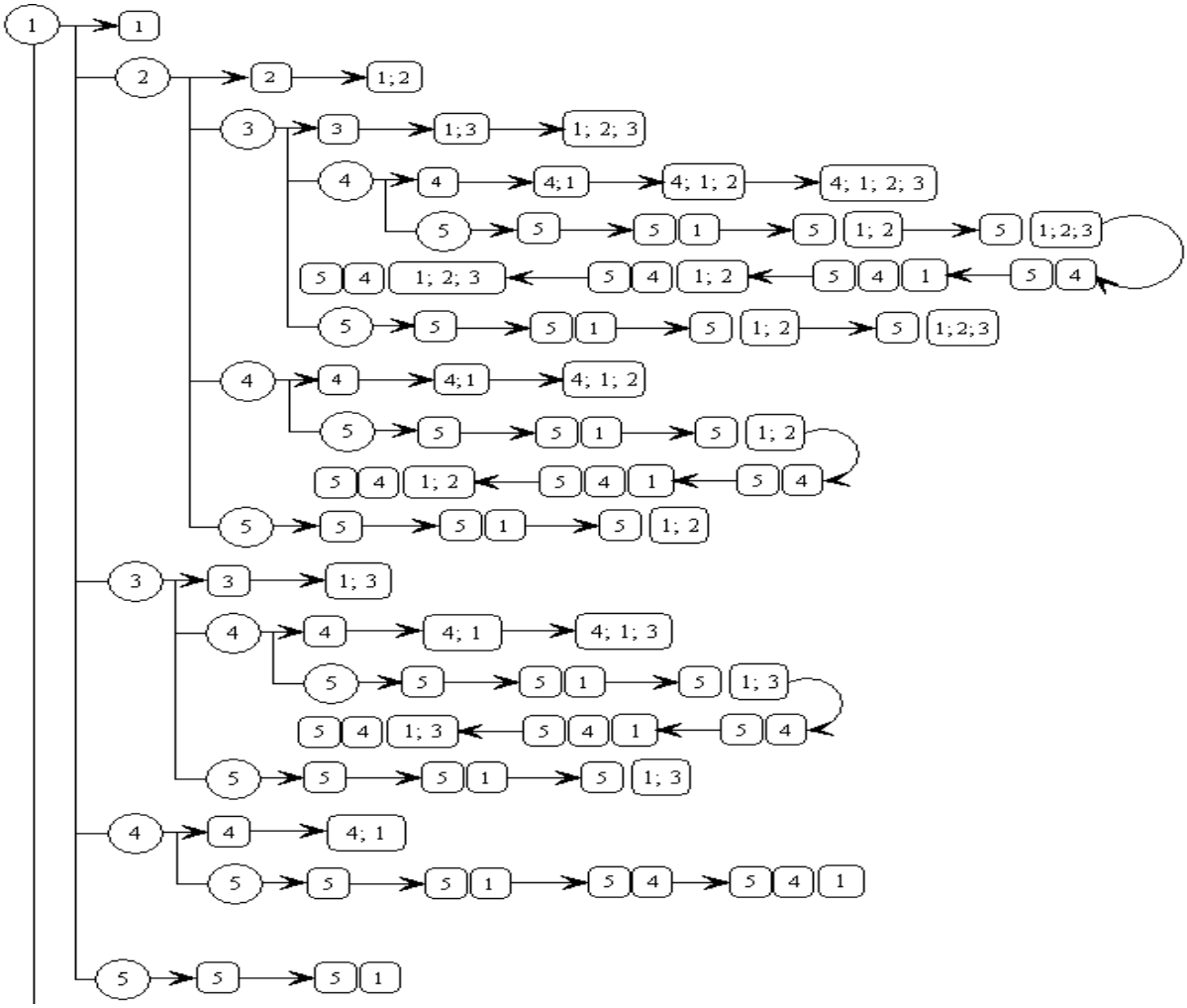


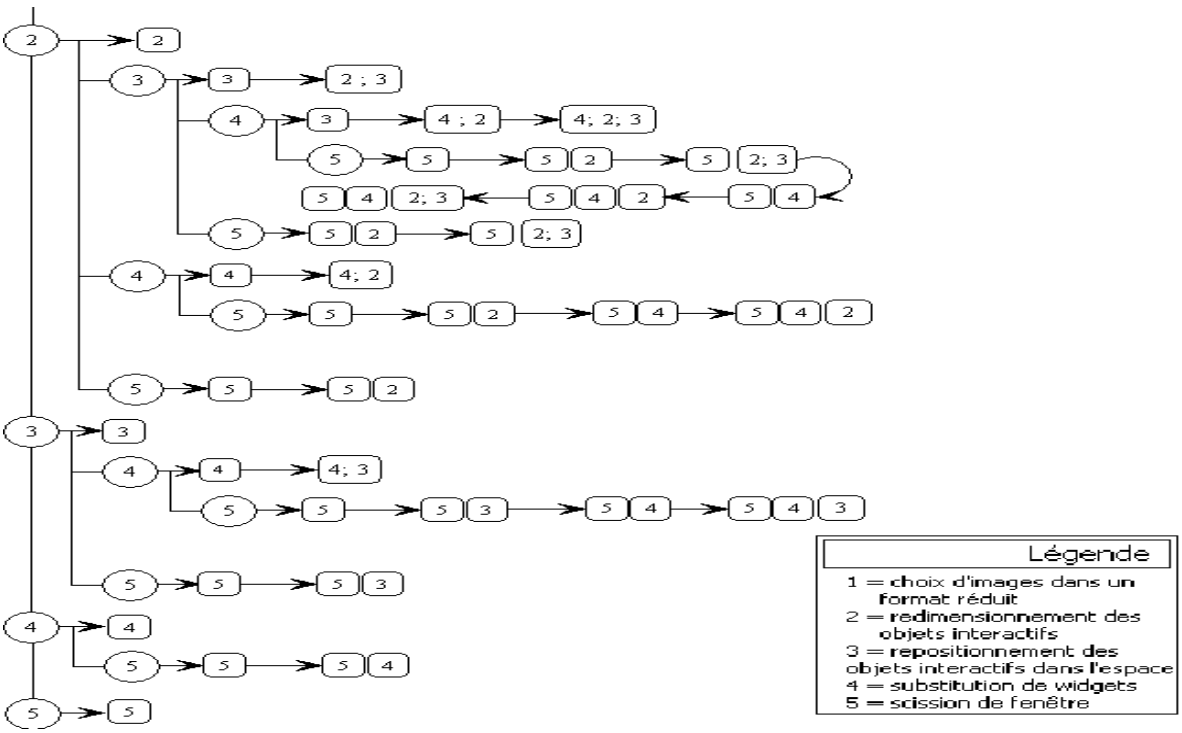

manipulDataValue ≠ mixt domain
manipulDataType = real ; alphanumeric
ou (manipulDataType = integer ; date ; time et
manipulDomainCharacterization = discrete interval)
manipulCardmin >= 2 & manipulCardmax =< 3
: N boutons radio (hauteur = trois éléments)
manipulCardmin >= 2 & manipulCardmax =< 25
: N boutons radio (hauteur = sept éléments)
manipulCardmin >= 2 & manipulCardmax > 25
: Liste de sélection ; liste de sélection déroulante ;
N boutons radio (hauteur = sept éléments)
manipulDataType = graphics
manipulCardmin >= 2 & manipulCardmax =< 3
: N boutons radio graphiques (hauteur = trois éléments)
manipulCardmin >= 2 & manipulCardmax > 3
: N boutons radio graphiques (hauteur =< sept éléments)
manipulDomainCharacterization = continuous interval
manipulDataType = time
manipulCardmin >= 2 & manipulCardmax =< 10
: échelle horizontale
manipulCardmin >= 2 & manipulCardmax > 10
: 3 boutons de variation ; 3 champs d'édition
manipulDataType = date
manipulCardmin >= 2 & manipulCardmax =< 10
: 3 listes de sélection déroulante
manipulCardmin >= 2 & manipulCardmax > 10
: 3 boutons de variation ; 3 listes de sélection déroulante ; 3 champs d'édition;
un champ d'édition
manipulDataType = integer
manipulCardmin >= 2 & manipulCardmax =< 10
: échelle horizontale
manipulCardmin >= 2 & manipulCardmax > 10
: 1 bouton de variation ; échelle horizontale
actionType = checking
: boîte à cocher


actionType = multiple selection
manipulDataValue = none
: accumulateur éditable (trois types possibles)
manipulDataValue = mixt domain
manipulDataType ≠ graphics
manipulCardmin >= 2
: accumulateur éditable (trois types possibles) ;
liste de combinaisons à sélection multiple
manipulDataType = graphics
manipulCardmin >= 2
: éditeur graphique (->)
manipulDataValue ≠ mixt domain
manipulDataType ≠ graphics
manipulCardmin >= 2
: accumulateur simple (trois types possibles) ;
liste de sélection multiple ; N boîtes à cocher
manipulDataType = graphics
manipulCardmin >= 2
: N boîtes à cocher graphiques
actionType = activation
manipulDataValue = none
: barre de défilement (horizontale ou verticale) ; bouton simple ;
bouton radio graphique (nom de tâche affichée ou pas)
manipulDataType = graphics
: bouton simple ;
bouton radio graphique (nom de tâche affichée ou pas)
manipulDataType ≠ graphics
: bouton simple ; menu
actionType = submission/external navigation
manipulDataType = graphics
: bouton simple ; bouton radio graphique
(nom de tâche affichée ou pas)
manipulDataType ≠ graphics
: bouton simple
actionType = navigation
manipulDataType = graphics
: bouton simple ; bouton radio graphique
(nom de tâche affichée ou pas)
manipulDataType ≠ graphics
: bouton simple ; barre de défilement (horizontale/verticale)
actionType = internal navigation
: barre de défilement (horizontale/verticale)

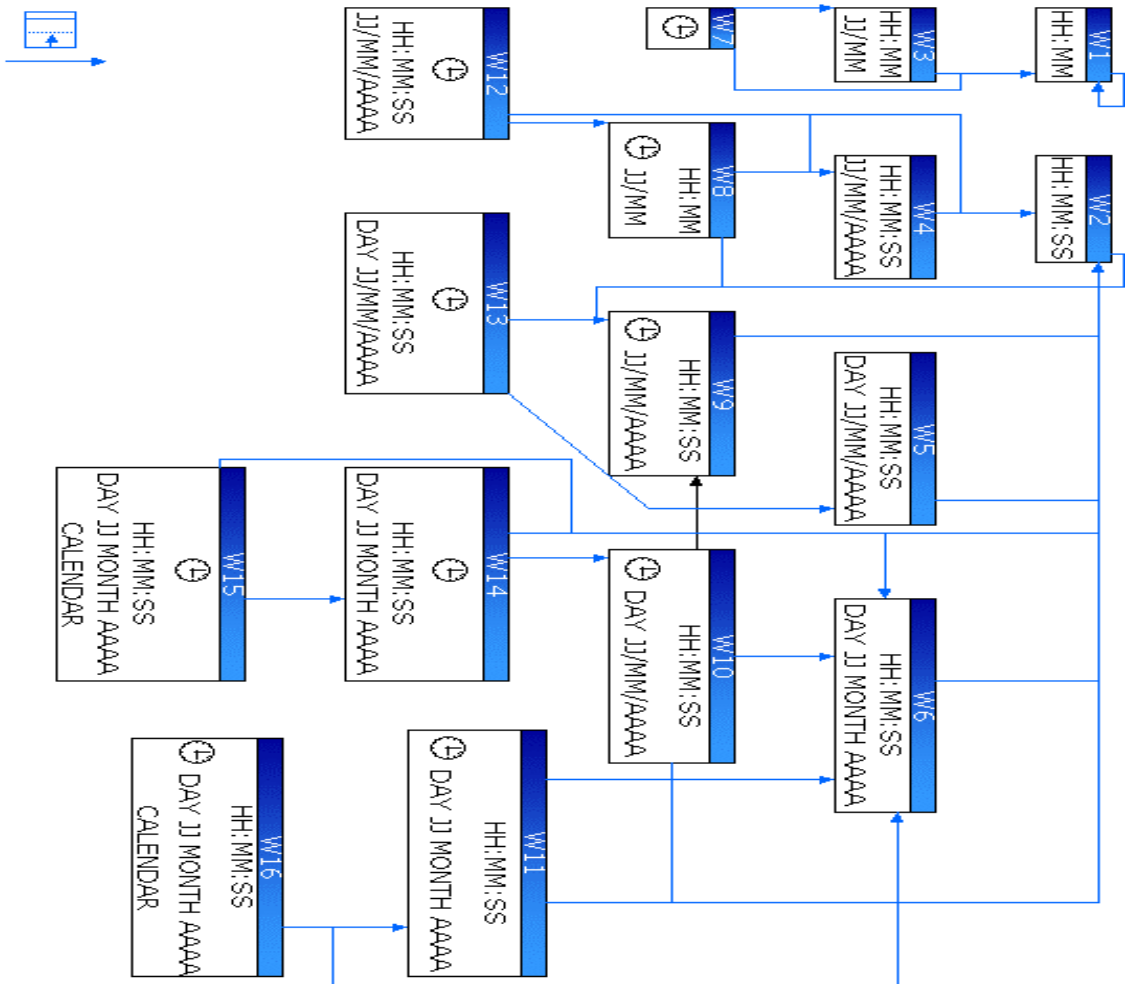


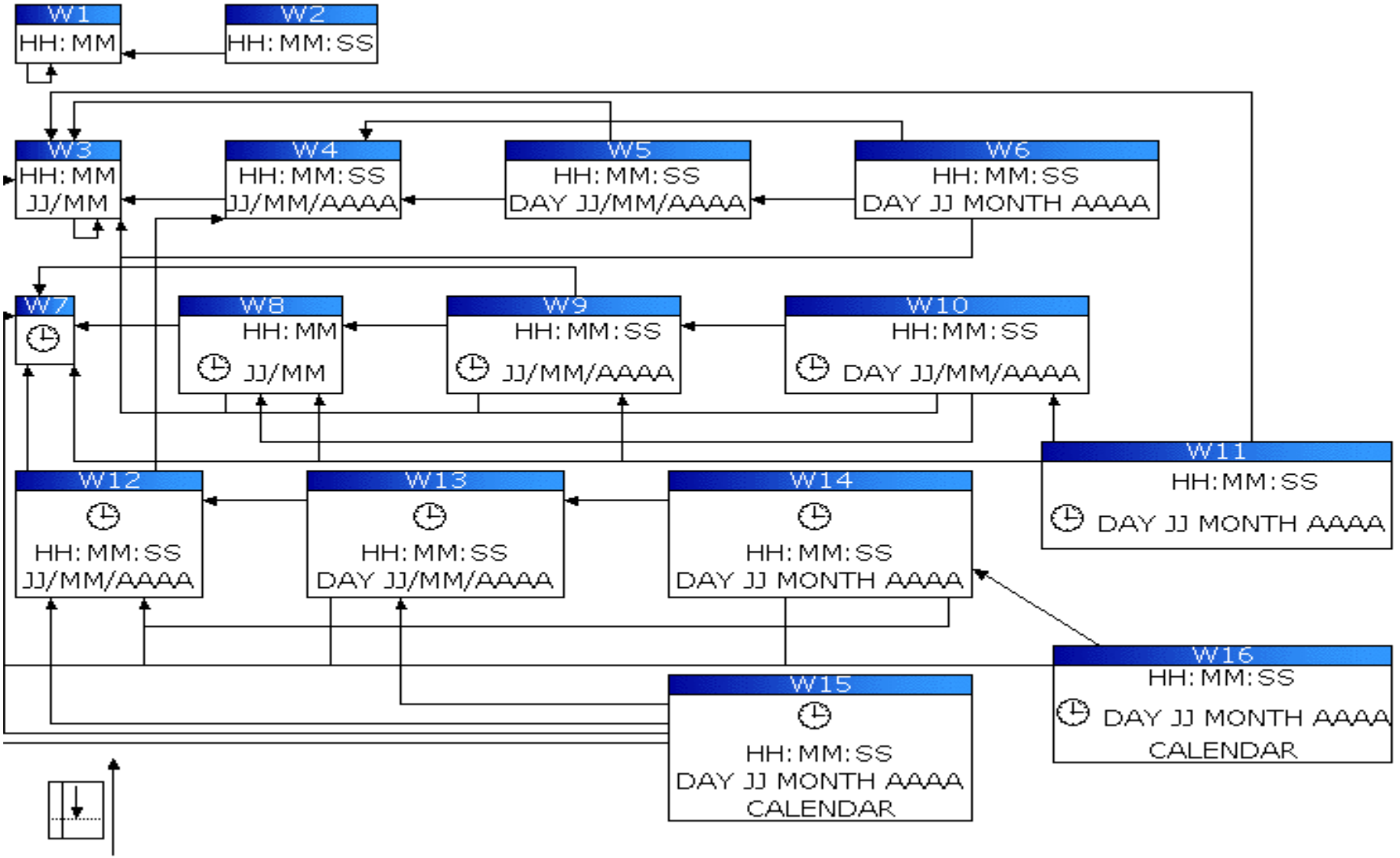
Arbre de sélection d'application de règles de dégradation harmonieuse.

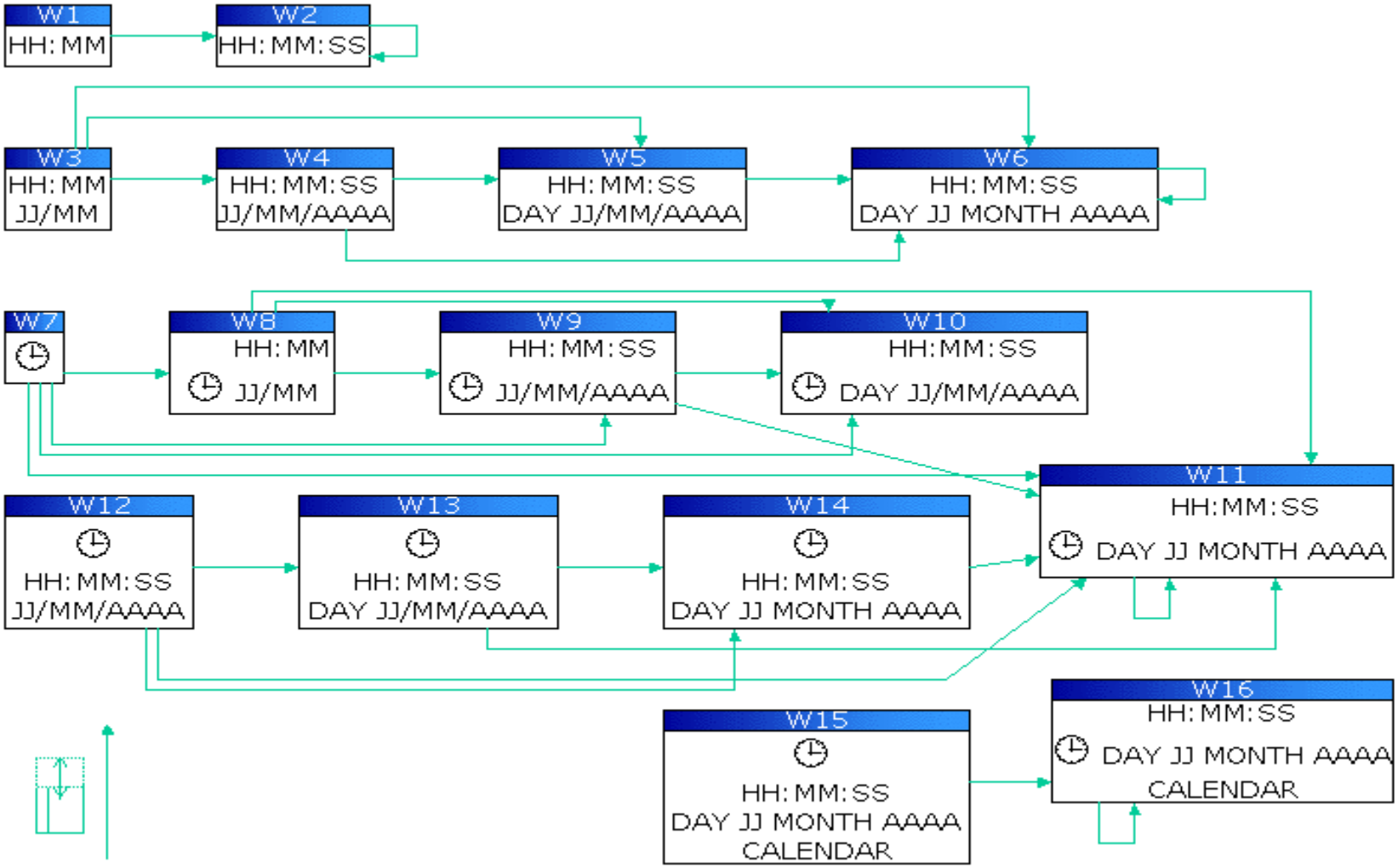


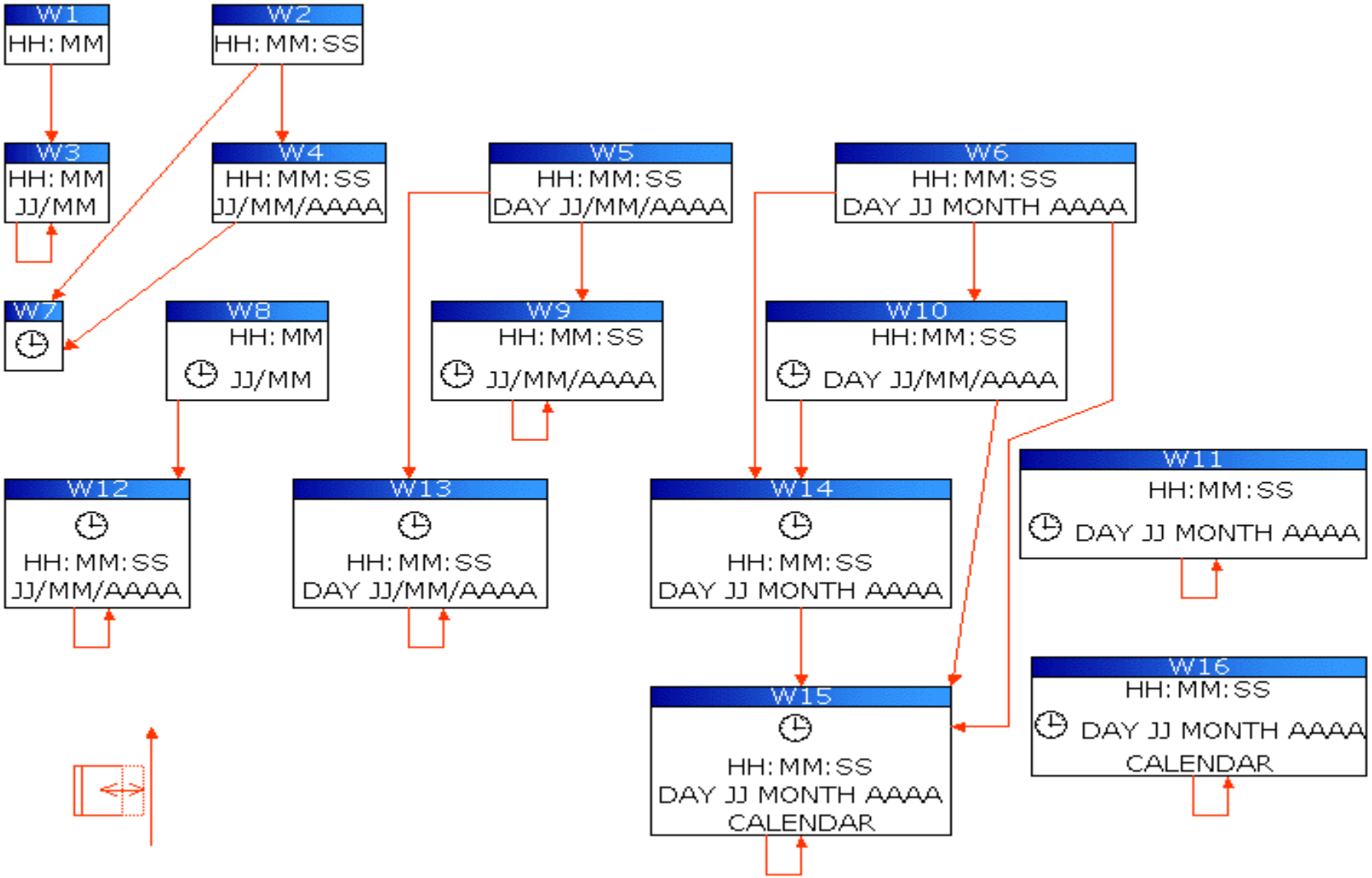


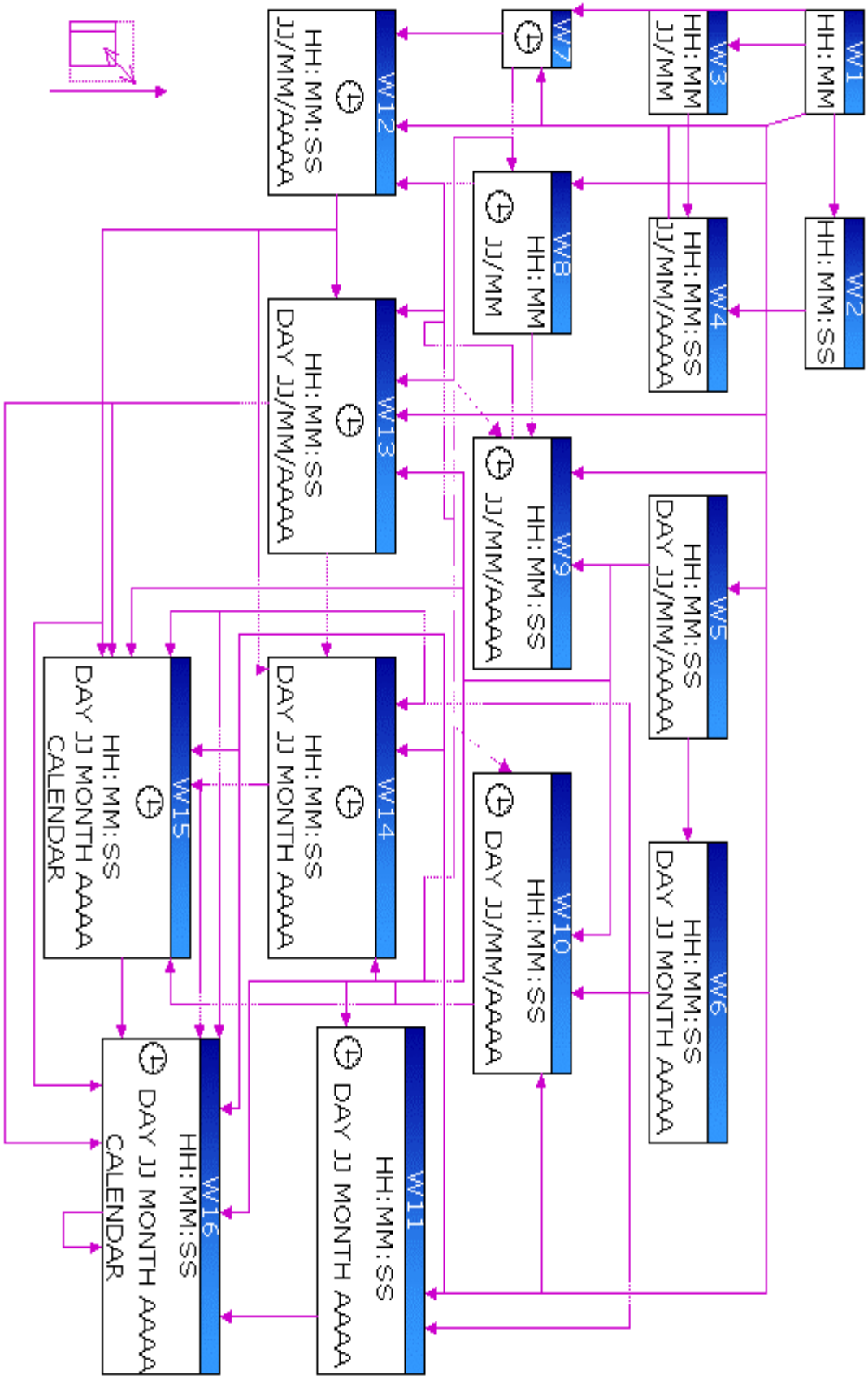
FlexClock : Modélisation des transitions liées aux opérations de redimensionnement de fenêtres par Machine de Moore

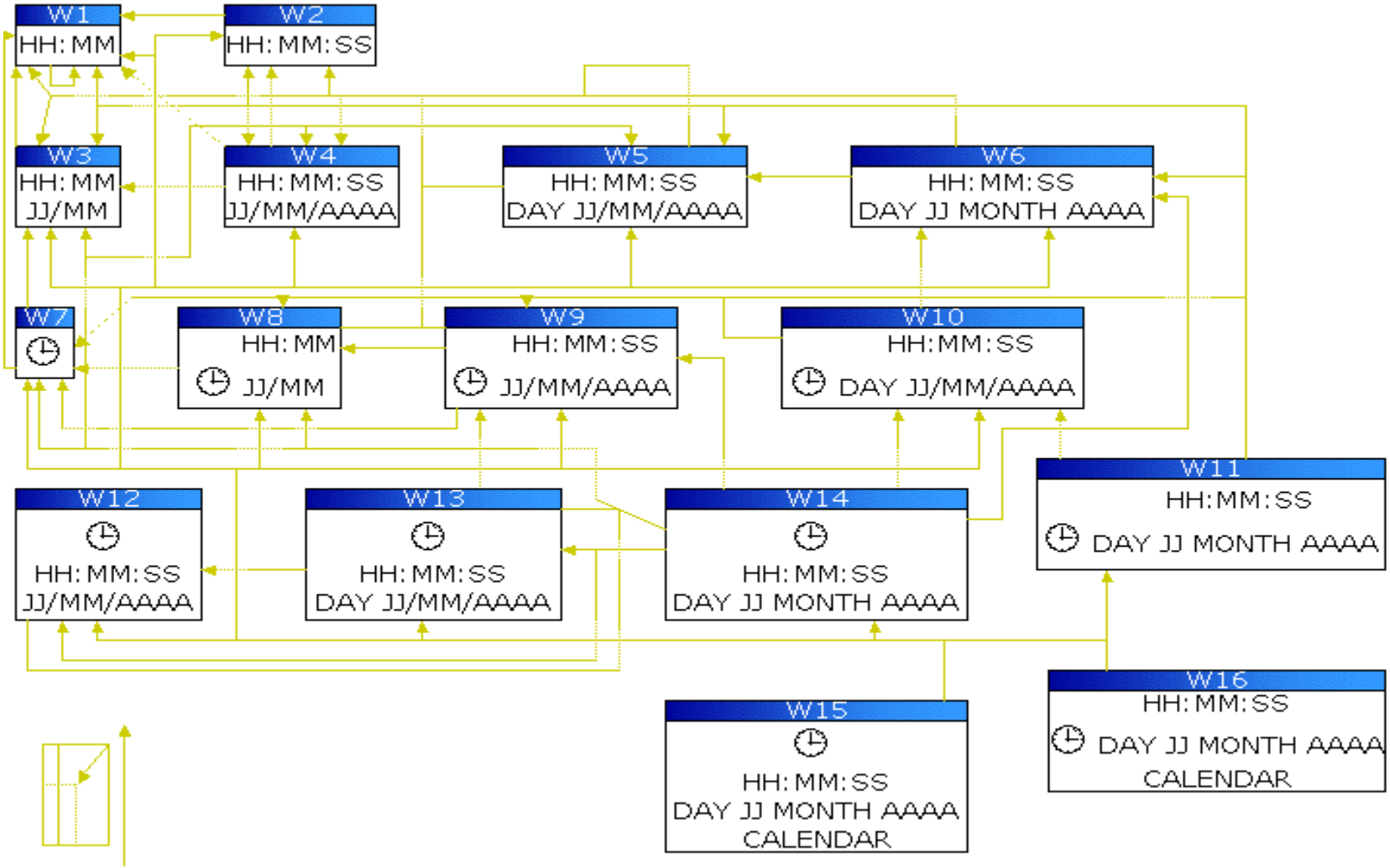


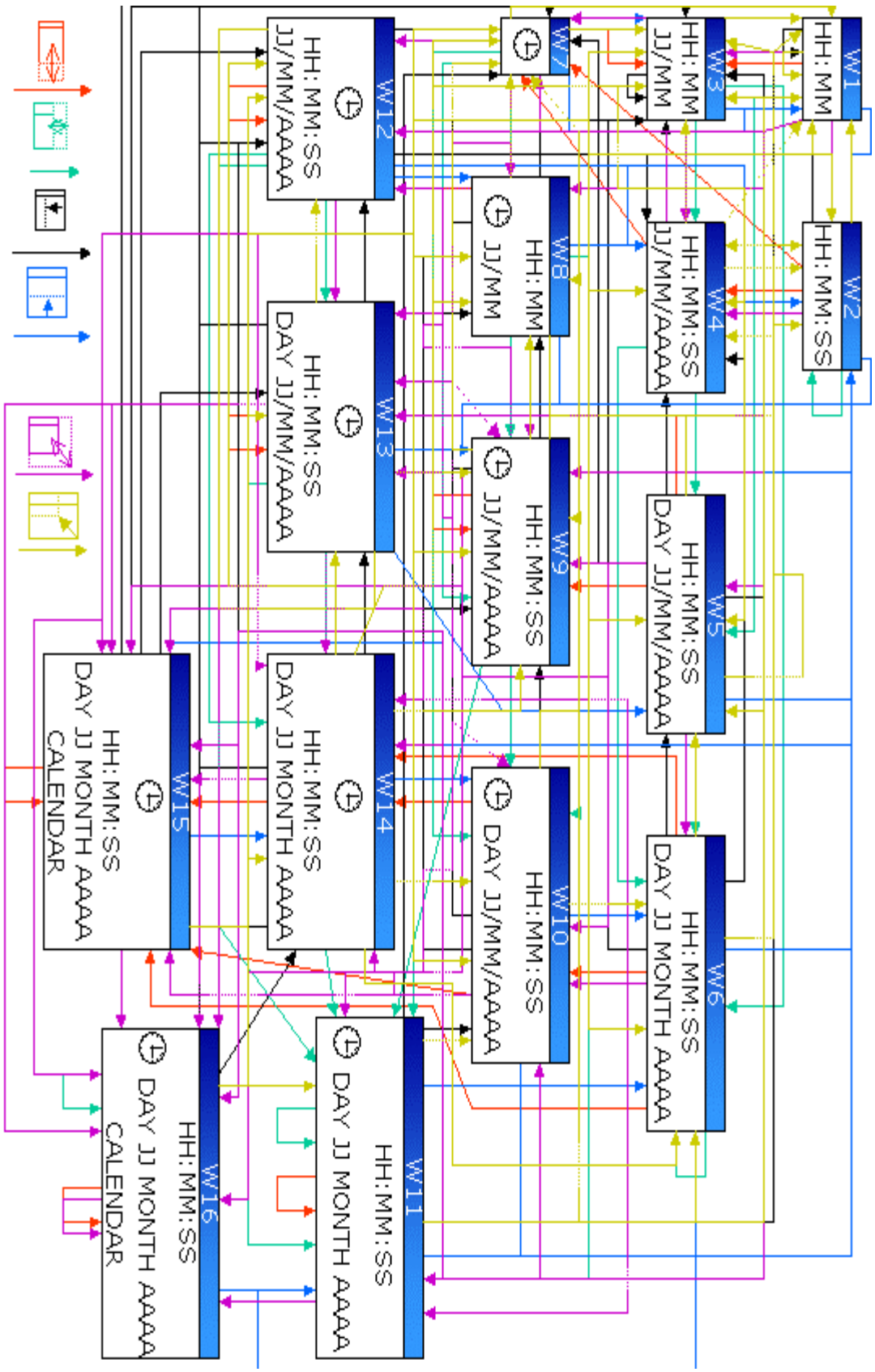

W1
HH:MM
W2
HH:MM:SS
W3
HH:MM
JJ/MM
W4
HH:MM:SS
JJ/MM/AAAA
W5
HH:MM:SS
DAY JJ/MM/AAAA
W6
HH:MM:SS
DAY JJ MONTH AAAA
W7
W8
HH:MM
JJ/MM
W9
HH:MM:SS
JJ/MM/AAAA
W10
HH:MM:SS
DAY JJ/MM/AAAA
W11
HH:MM:SS
DAY JJ MONTH AAAA
W12
HH:MM:SS
JJ/MM/AAAA
W13
HH:MM:SS
DAY JJ/MM/AAAA
W14
HH:MM:SS
DAY JJ MONTH AAAA
W16
HH:MM:SS
DAY JJ MONTH AAAA
CALENDAR
W15
HH:MM:SS
DAY JJ MONTH AAAA
CALENDAR


W1
HH:MM
W2
HH:MM:SS
W3
HH:MM
JJ/MM
W4
HH:MM:SS
JJ/MM/AAAA
W5
HH:MM:SS
DAY JJ/MM/AAAA
W6
HH:MM:SS
DAY JJ MONTH AAAA
W7
W8
HH:MM
JJ/MM
W9
HH:MM:SS
JJ/MM/AAAA
W10
HH:MM:SS
DAY JJ/MM/AAAA
W11
HH:MM:SS
DAY JJ MONTH AAAA
W12
HH:MM:SS
JJ/MM/AAAA
W13
HH:MM:SS
DAY JJ/MM/AAAA
W14
HH:MM:SS
DAY JJ MONTH AAAA
W15
HH:MM:SS
DAY JJ MONTH AAAA
CALENDAR
W16
HH:MM:SS
DAY JJ MONTH AAAA
CALENDAR

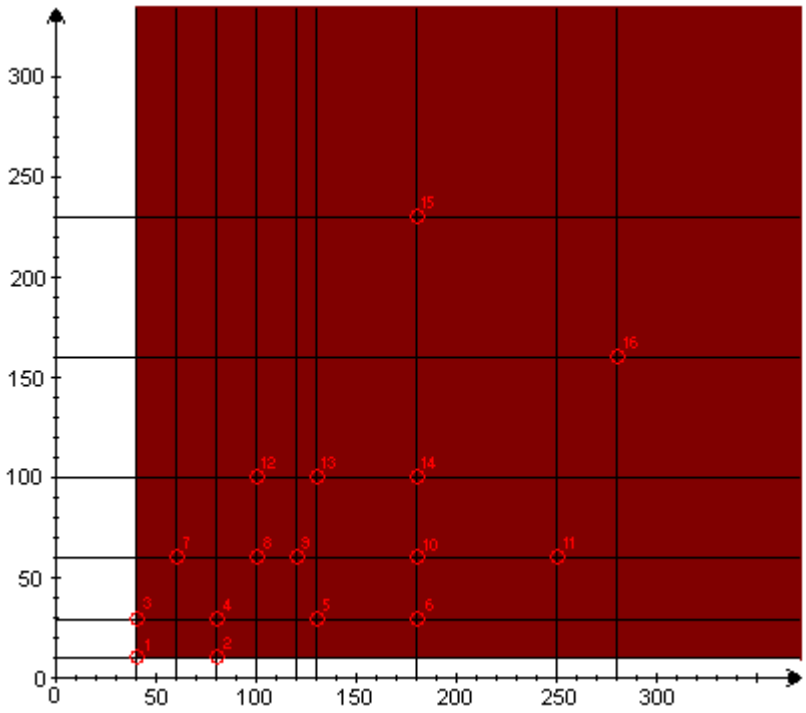

W1
HH:MM
W2
HH:MM:SS
W3
HH:MM
JJ/MM
W4
HH:MM:SS
JJ/MM/AAAA
W5
HH:MM:SS
DAY JJ/MM/AAAA
W6
HH:MM:SS
DAY JJ MONTH AAAA
W7
W8
HH:MM
JJ/MM
W9
HH:MM:SS
JJ/MM/AAAA
W10
HH:MM:SS
DAY JJ/MM/AAAA
W11
HH:MM:SS
DAY JJ MONTH AAAA
W12
HH:MM:SS
JJ/MM/AAAA
W13
HH:MM:SS
DAY JJ/MM/AAAA
W14
HH:MM:SS
DAY JJ MONTH AAAA
W15
HH:MM:SS
DAY JJ MONTH AAAA
CALENDAR
W16
HH:MM:SS
DAY JJ MONTH AAAA
CALENDAR

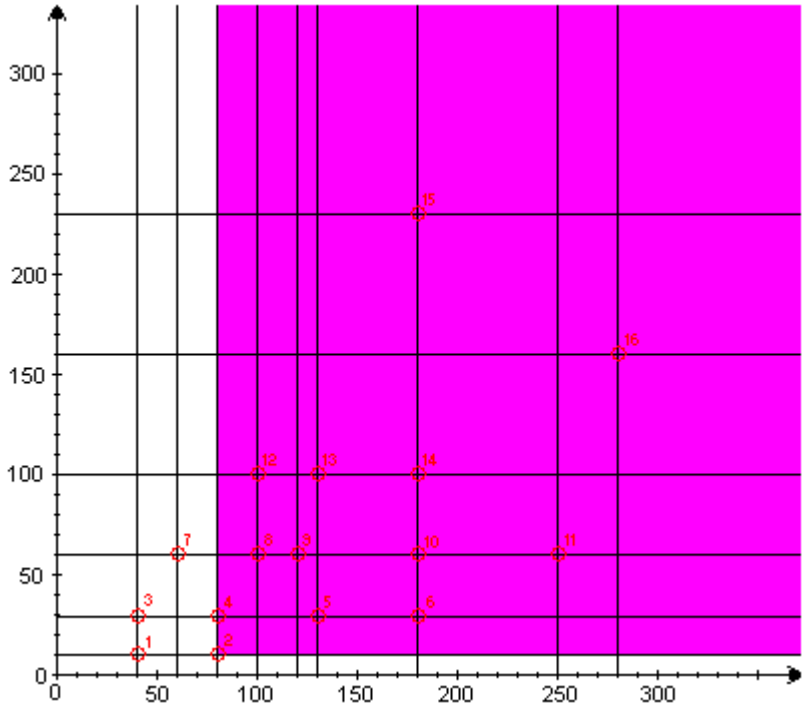

W1
HH:MM
W2
HH:MM:SS
W3
HH:MM
JJ/MM
W4
HH:MM:SS
JJ/MM/AAAA
W5
HH:MM:SS
DAY JJ/MM/AAAA
W6
HH:MM:SS
DAY JJ MONTH AAAA
W7
W8
HH:MM
JJ/MM
W9
HH:MM:SS
JJ/MM/AAAA
W10
HH:MM:SS
DAY JJ/MM/AAAA
W11
HH:MM:SS
DAY JJ MONTH AAAA
W12
HH:MM:SS
JJ/MM/AAAA
W13
HH:MM:SS
DAY JJ/MM/AAAA
W14
HH:MM:SS
DAY JJ MONTH AAAA
W15
HH:MM:SS
DAY JJ MONTH AAAA
CALENDAR
W16
HH:MM:SS
DAY JJ MONTH AAAA
CALENDAR

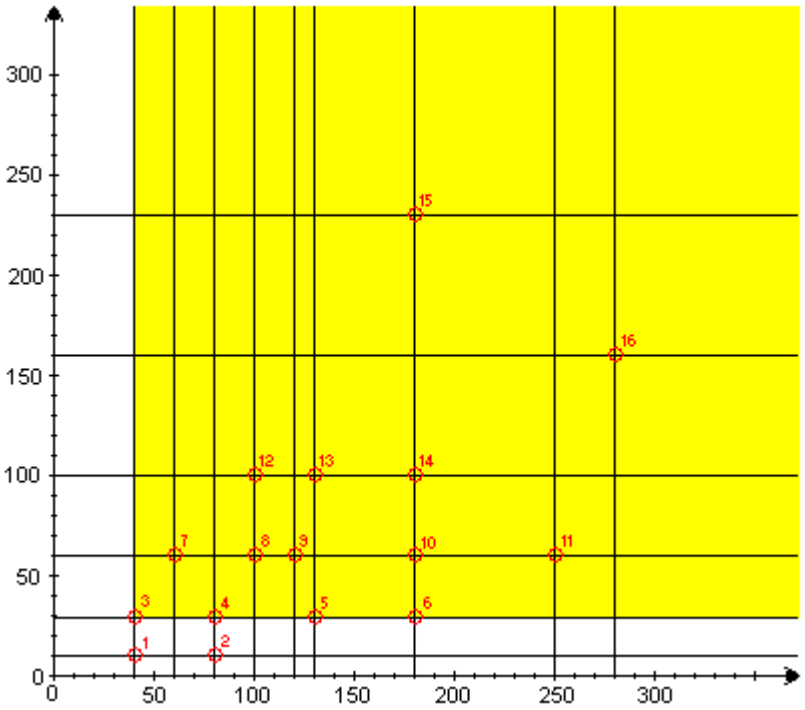

W1
HH:MM
W2
HH:MM:SS
W3
HH:MM
JJ/MM
W4
HH:MM:SS
JJ/MM/AAAA
W5
HH:MM:SS
DAY JJ/MM/AAAA
W6
HH:MM:SS
DAY JJ MONTH AAAA
W7
W8
HH:MM
JJ/MM
W9
HH:MM:SS
JJ/MM/AAAA
W10
HH:MM:SS
DAY JJ/MM/AAAA
W11
HH:MM:SS
DAY JJ MONTH AAAA
W12
HH:MM:SS
JJ/MM/AAAA
W13
HH:MM:SS
DAY JJ/MM/AAAA
W14
HH:MM:SS
DAY JJ MONTH AAAA
W15
HH:MM:SS
DAY JJ MONTH AAAA
CALENDAR
W16
HH:MM:SS
DAY JJ MONTH AAAA
CALENDAR

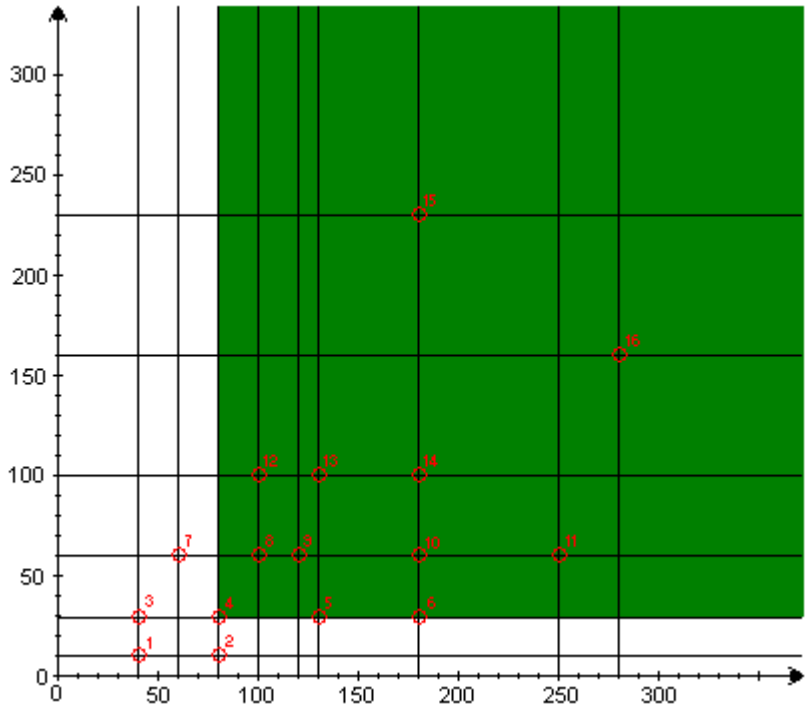

W1
HH:MM
W2
HH:MM:SS
W3
HH:MM
JJ/MM
W4
HH:MM:SS
JJ/MM/AAAA
W5
HH:MM:SS
DAY JJ/MM/AAAA
W6
HH:MM:SS
DAY JJ MONTH AAAA
W7
W8
HH:MM
JJ/MM
W9
HH:MM:SS
JJ/MM/AAAA
W10
HH:MM:SS
DAY JJ/MM/AAAA
W11
HH:MM:SS
DAY JJ MONTH AAAA
W12
HH:MM:SS
JJ/MM/AAAA
W13
HH:MM:SS
DAY JJ/MM/AAAA
W14
HH:MM:SS
DAY JJ MONTH AAAA
W15
HH:MM:SS
DAY JJ MONTH AAAA
CALENDAR
W16
HH:MM:SS
DAY JJ MONTH AAAA
CALENDAR

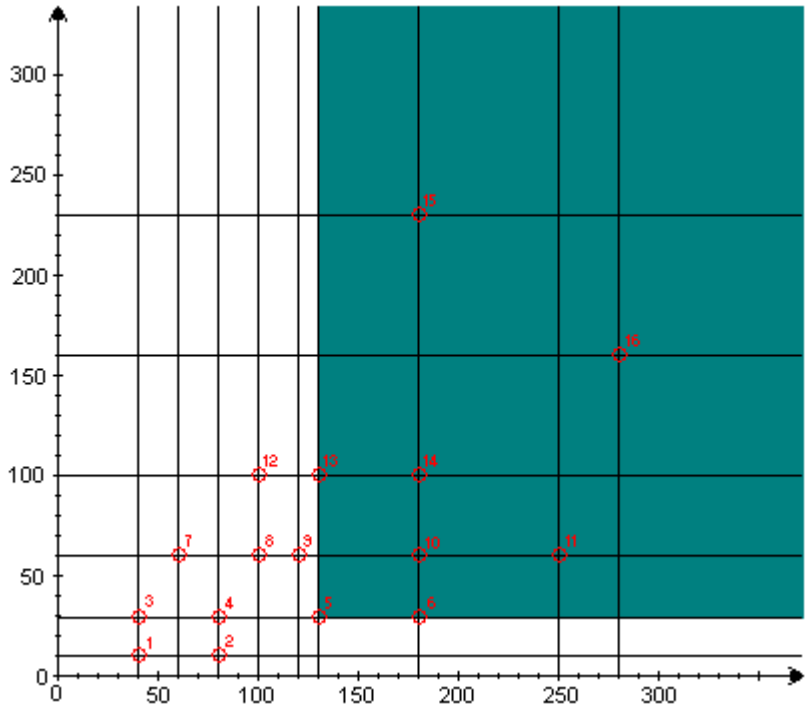
FlexClock : Modélisation des transitions liées aux opérations de redimensionnement de fenêtres par surface

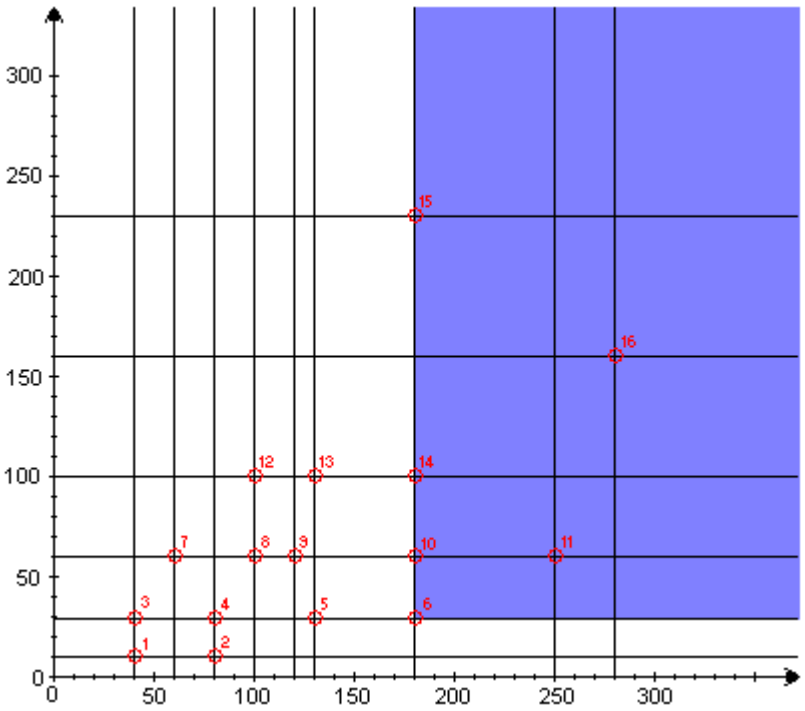


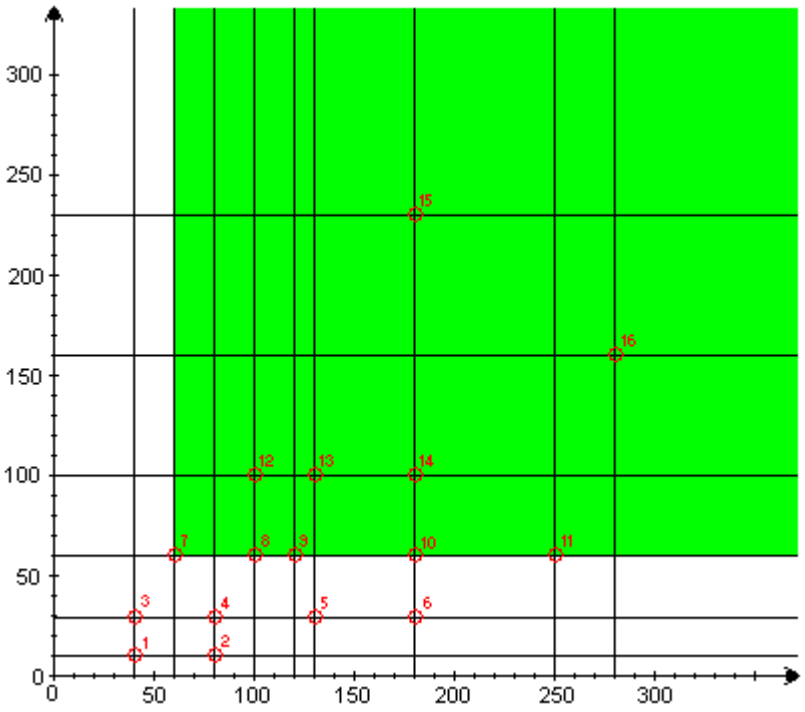


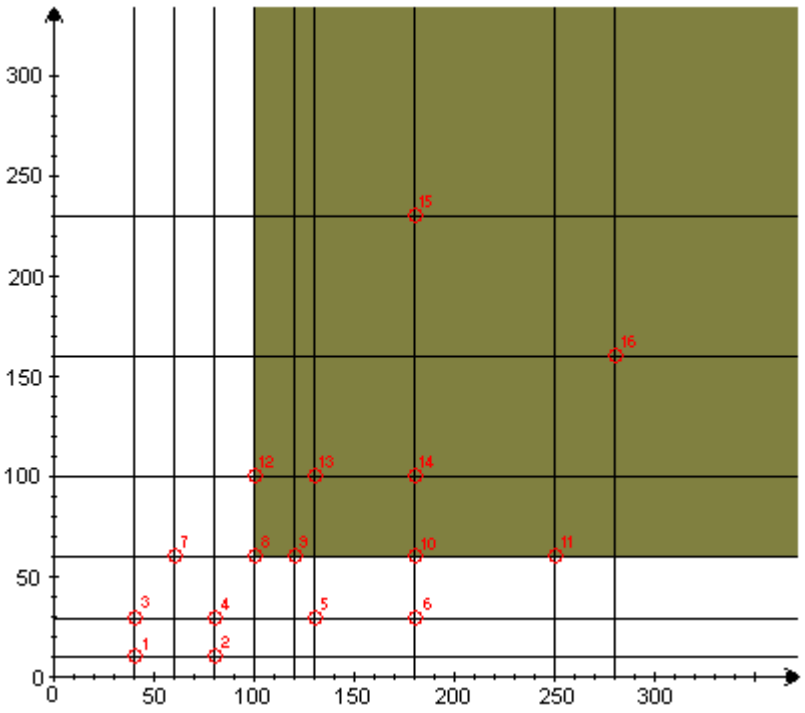

300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

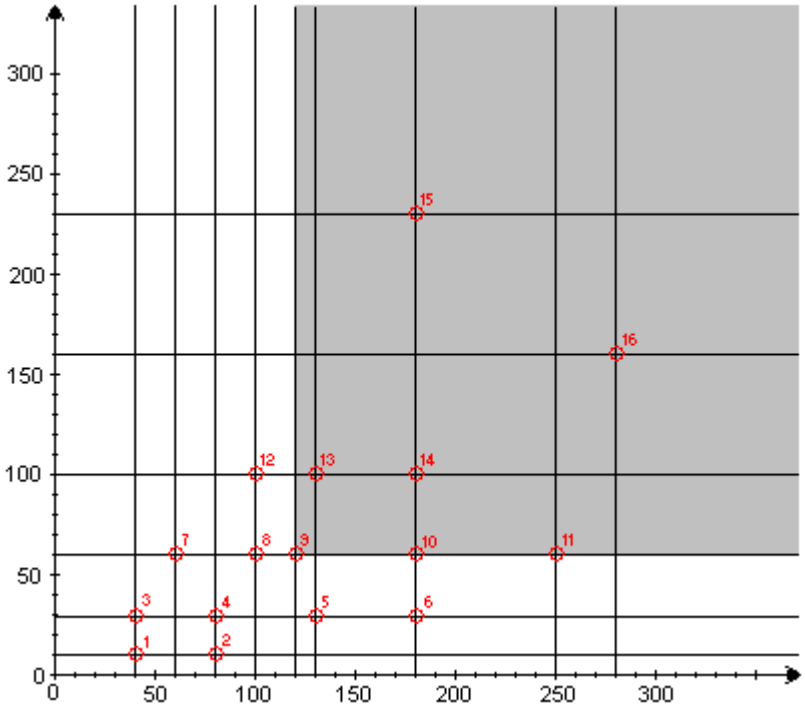

300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

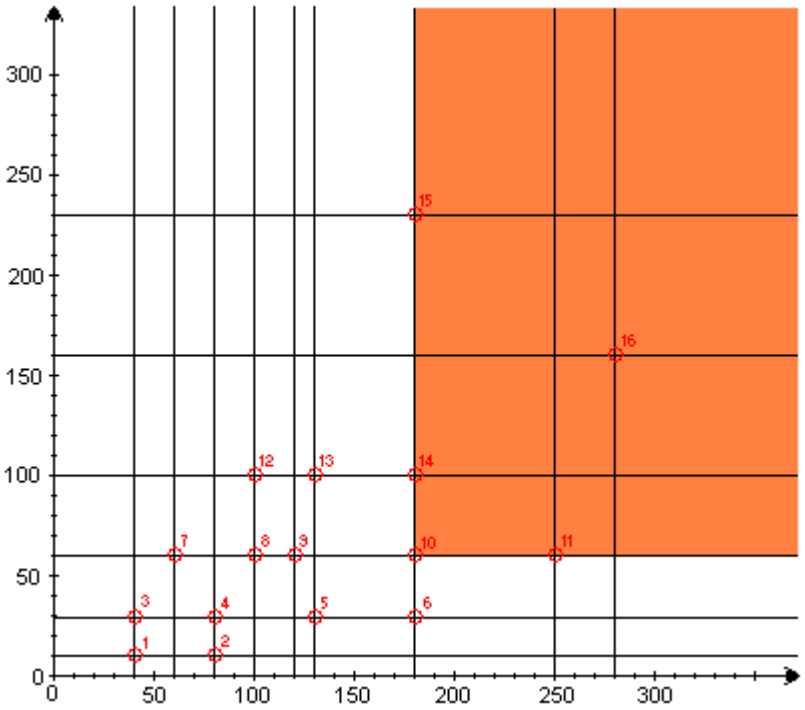

300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

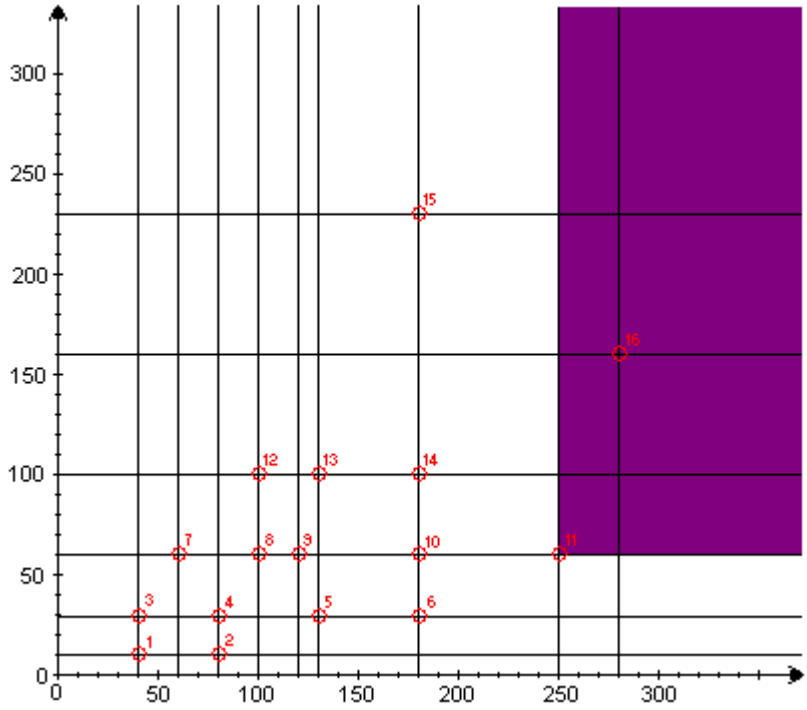

300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

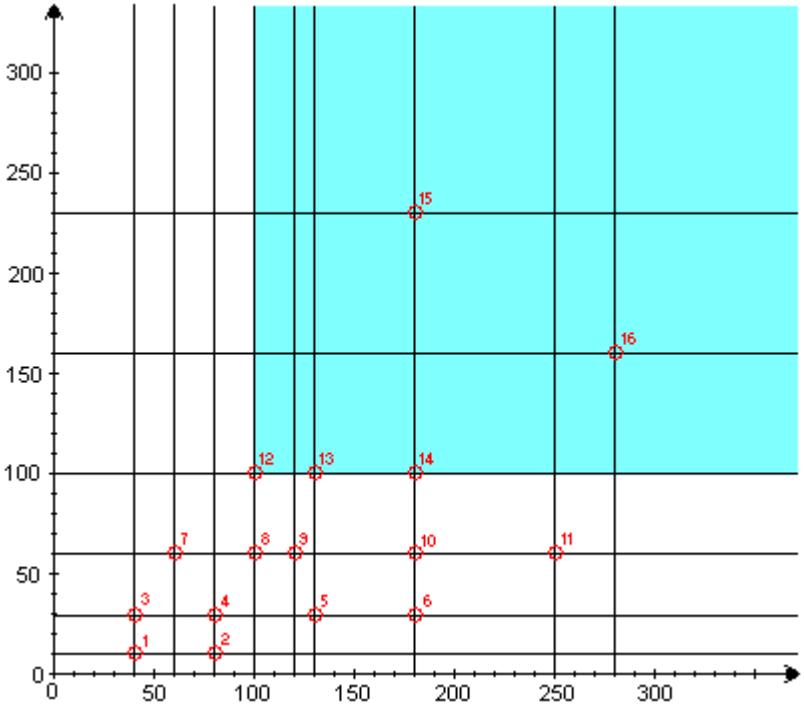

300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

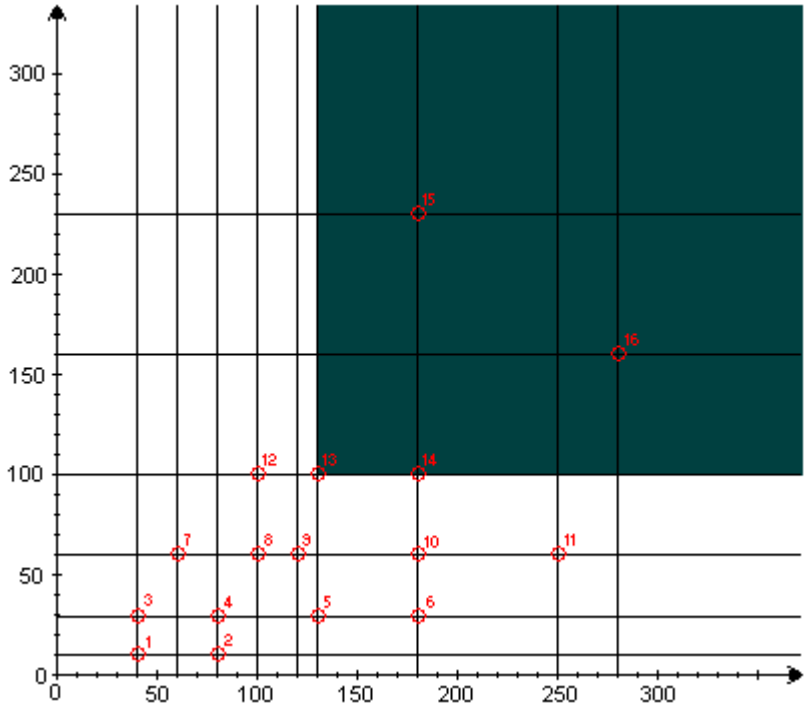

300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

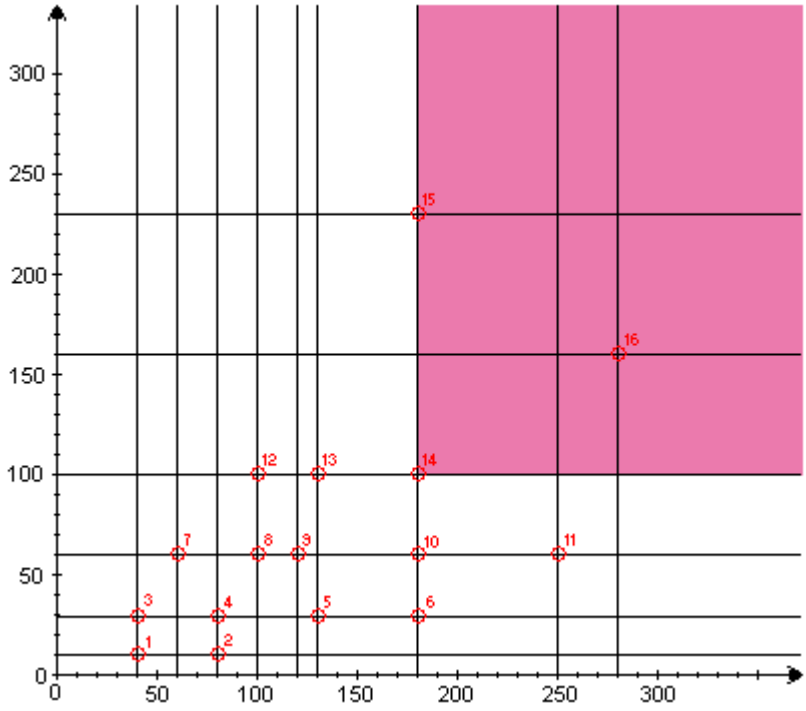

300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

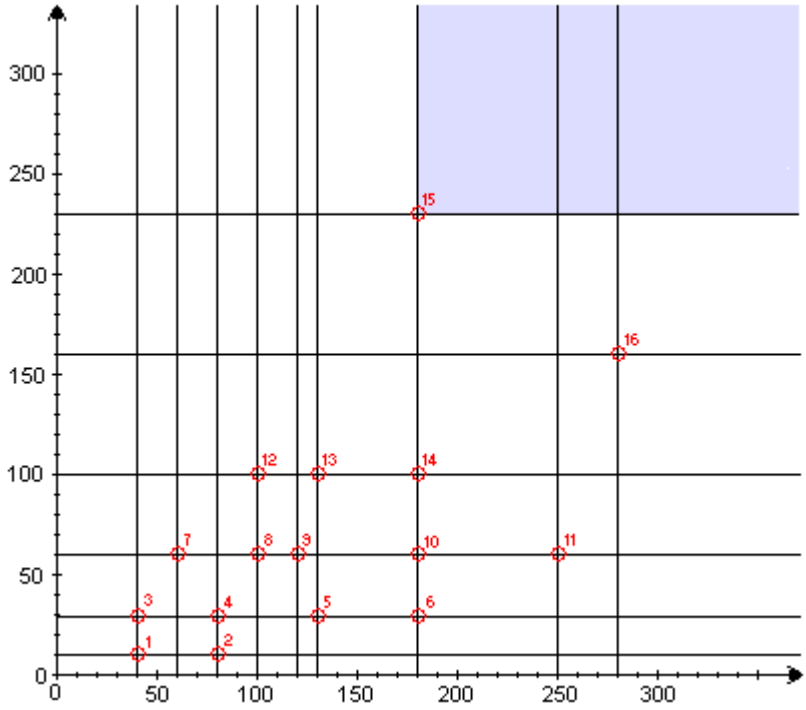

300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

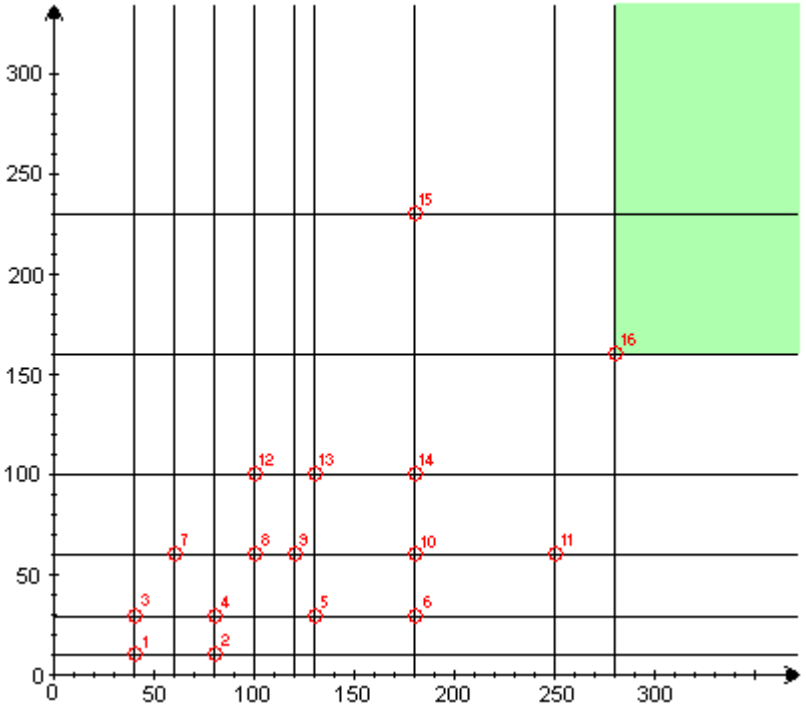

300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16

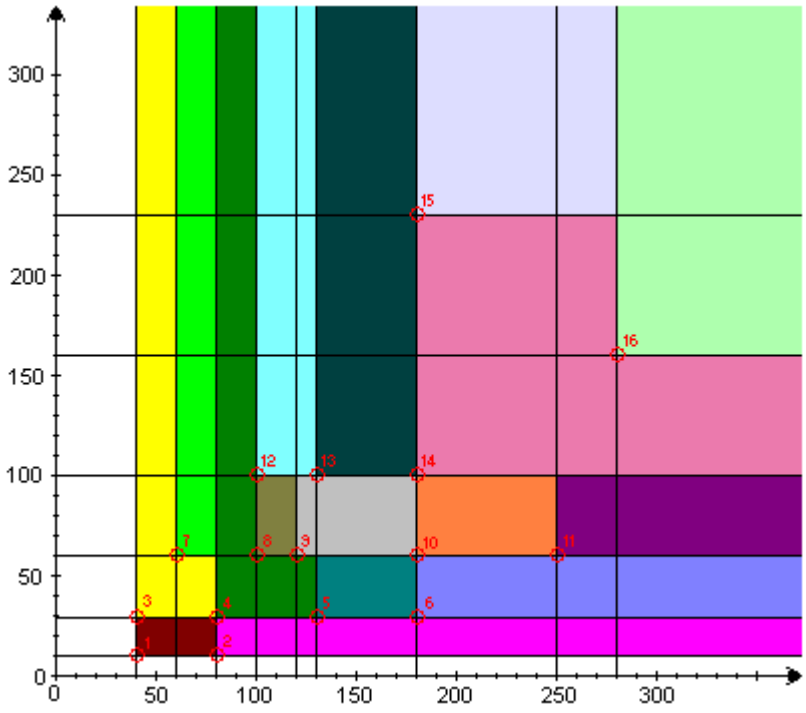

300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16


300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16


300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16


300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16


300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16


300
250
200
150
100
50
0
0
50
100
150
200
250
300
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16