**Laboratoire CLIPS-IMAG**
**Unité Mixte de Recherche CNRS N° 5524**
**Equipe IIHM**

# RAPPORT TECHNIQUE

# PLASTICITE DES INTERFACES : UNE APPROCHE, UN OUTIL

**Gaëlle CALVARY, Joëlle COUTAZ, David THEVENIN**

Date : 3 Mars 2001
Version : 2
Révision : 1

Ce rapport technique traite de la Plasticité des Interfaces. Il en présente une approche et un outil, illustrés dans le cadre du Gestionnaire d'Energie. Le rapport s'articule en quatre parties :

- La première partie introduit la propriété de plasticité. Elle en présente l'espace problème.
- La seconde décrit l'approche préconisée pour la production d'interfaces plastiques. Elle en présente le processus de développement et les modèles impliqués.
- La troisième partie présente ArtStudio, un outil implémentant une instance de ce processus de développement. Le processus et les heuristiques de génération y sont décrits.
- La quatrième partie conclut en jugeant de l'opportunité pour EDF de recourir à l'approche et l'outil ainsi étudiés.

En annexes, sont consignés :

- Une définition et une formalisation de la notion de contexte (annexe A) ;
- Des compléments quant à l'implémentation d'ArtStudio (annexe B) ;
- La liste des publications de l'équipe sur le sujet (annexe C).

**TABLE DES MATIÈRES**

## 1 ESPACE PROBLEME

Avec les avancées des réseaux sans fil et les prouesses en miniaturisation, l'ordinateur devient évanescent. Nos objets quotidiens sont augmentés [6] devenant supports possibles à l'interaction. Citons, par exemple, le tableau magique [2] ou le mètre augmenté [11]. En même temps, nos ordinateurs de poche s'adaptent au contexte pour nous offrir une information située et mieux nous assister dans nos tâches. Pour exemple, le Cyberguide [1], l'assistant de bureau [21] ou encore l'assistant conversationnel Welbo [12].

D'un point de vue de l'usage, ce qui semble séduisant c'est de permettre à l'utilisateur de profiter au mieux de ces avancées technologiques, pour mieux se concentrer sur sa tâche et en oublier le dispositif : faire de l'assistant personnel une télécommande universelle [17] ; permettre à l'utilisateur, par simple tirer-lâcher, de migrer des données d'un dispositif à l'autre [14]. Les applications sont nombreuses : passer du portable à l'assistant personnel, sans perte de contexte d'interaction, lorsque la batterie du portable faiblit ; ou passer du portable au tableau pour une explication de groupe avec conservation des données.

Ces applications sensibles au contexte, communicantes, diversifiant les dispositifs d'interaction, ouvrent de nouveaux pans de recherche en Ingénierie de l'Interaction Homme-Machine. Nous nous intéressons ici à l'adaptation des interfaces au contexte, que nous baptisons plasticité des interfaces [20] [4]. Nous illustrons ces travaux sur le gestionnaire d'énergie, cadre d'étude proposé par EDF R&D.

Un gestionnaire d'énergie permet de régler le confort d'une habitation par la définition de programmes journaliers ou hebdomadaires. Il s'apparente à un boîtier électronique muni de sondes de température. Ces sondes contrôlent la température des zones identifiées dans l'habitation : salle de bain, salon, chambres, etc. Pour cette programmation, EDF souhaite offrir de nouveaux modes d'interaction : la télécommande sur dispositif d'interaction banalisé tels que les téléphones, PDA (Personal Digital Assistant) ou navigateurs Internet (Figure 1.1).

***Figure 1.1:*** Commercialisation à grande échelle de nouveaux dispositifs d'interaction.

L'usager programmera son confort en spécifiant son rythme de vie et la température souhaitée par zone. Cette programmation pourra se faire, à domicile, sur le boîtier dédié ou, à distance, via un butineur Internet, un PDA ou un téléphone portable. Il pourra ainsi, à distance, déclencher son chauffage avant d'arriver ; modifier sa programmation en fonction des événements climatiques ou vérifier le bon fonctionnement de son installation.

Mais à l'évidence, l'IHM de l'application ne peut être identique sur ces différentes plates-formes : les différences de ressources matérielles nécessaires à l'interaction personne-système, comme la taille de l'écran ou l'absence de clavier, imposent des solutions interactionnelles dédiées. La figure 1.2 présente des exemples d'interfaces pour plates-formes graphiques de taille d'écran variées (stations de travail / PDA) :

- En a), la taille de l'écran permet l'affichage simultané des températures des différentes pièces de l'habitation (salle de bain, salon, chambres) ;

· En b1) et b2), la surface d'affichage est trop restreinte : une tâche de navigation est nécessaire pour parcourir les différentes pièces. Un objet de navigation est introduit (un onglet en b1) ; un menu déroulant en b2)) pour choisir la pièce ciblée.

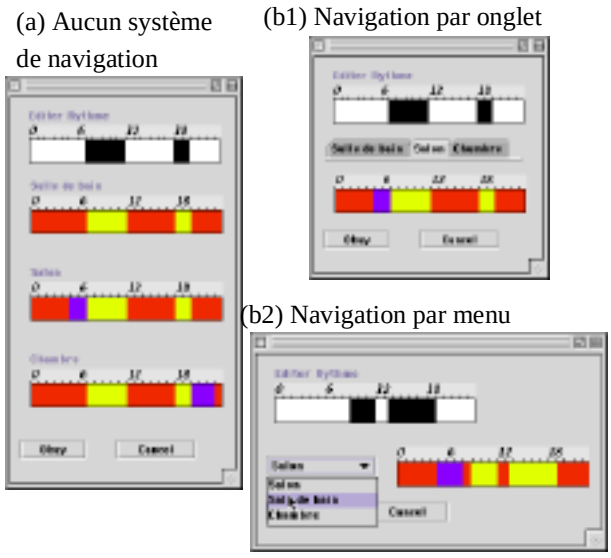



***Figure 1.2:*** En b1) et b2), contrairement à a), la surface d'affichage est trop restreinte pour afficher la température de toutes les pièces. Une navigation par onglet en b1), par menu déroulant en b2) permet à l'utilisateur de parcourir les différentes pièces.

Sur téléphones portables (Figure 1.3), une même tâche de navigation est nécessaire. A cette tâche de navigation s'ajoute une tâche articulatoire permettant l'entrée en mode édition ("donner ordre"). Cette tâche est introduite pour pallier l'absence de manipulation directe. On notera aussi le titre ajouté en entête de chaque page ("Chauf/Salon/Ordres") rappelant le contexte d'interaction.


***Figure 1.3:*** Sur téléphones portables, la taille de l'écran et l'absence de manipulation directe imposent des tâches articulatoires : une tâche de navigation pour parcourir les différentes pièces et une tâche d'entrée en mode édition pour pouvoir modifier la température des pièces.

Ces interfaces (Figures 1.2 et 1.3) ont été produites par ArtStudio [4], un assistant logiciel pour la réalisation d'interfaces plastiques. ArtStudio est décrit dans la troisième partie de ce rapport.

### 1.1 Plasticité : définition

Par analogie à la plasticité des matériaux, la plasticité d'une interface dénote sa capacité à s'adapter au contexte dans le respect de son utilisabilité [20]. Dans cette définition :

- Le contexte est un couple formé des entités "plate-forme / environnement" où la plate-forme est la plate-forme matérielle et logicielle sous-tendant l'interaction. Par exemple, un PalmPilot ou un téléphone portable. La taille de l'écran, les dispositifs d'interaction, les capacités de calcul et de communication y sont modélisés. L'environnement se réfère à l'environnement physique accueillant l'interaction. Il est décrit par un ensemble d'informations, périphériques à la tâche en cours, mais susceptibles de l'influencer. Par exemple, la luminosité, le bruit, la localisation géographique, la colocalisation sociale, etc. L'annexe A précise cette définition et en propose une formalisation mathématique.
- L'adaptation est une réaction au changement de contexte. Elle peut par exemple consister en un remodelage de l'interface (passage de 1.2a à 1.2b1 dans le cas d'une surcharge de l'écran). L'adaptation est faite pour le bien-être de l'utilisateur, mais elle ne tient pas compte des éventuels changements d'état (mental, physique, etc.) de l'utilisateur. L'utilisateur est un utilisateur stéréotype défini dans le cahier des charges.
- L'utilisabilité est évaluée sur la base de propriétés énoncées dans le cahier des charges.

Ainsi définie, la plasticité d'une interface est-elle une forme d'adaptation.

### 1.2 Espace d'adaptation

La plasticité mobilise un processus de type "Action  Réaction" où :

- L'action se réfère au changement de contexte (plate-forme et/ou environnement) ;
- La réaction dénote les mesures mises en œuvre par le système et/ou l'utilisateur pour préserver l'utilisabilité.

La réaction est caractérisée par un objet, un acteur et une occurrence [20] (Figure 1.4) :

- L'objet identifie les composants logiciels de l'interface affectés par l'adaptation. Il peut porter sur le dialogue et/ou la présentation.
- L'acteur précise qui, de l'homme et/ou du système, met en œuvre la réaction ;
- L'occurrence spécifie le caractère statique et/ou dynamique de l'adaptation. Une adaptation statique intervient entre sessions tandis qu'une adaptation dynamique s'effectue à la volée durant l'exécution.

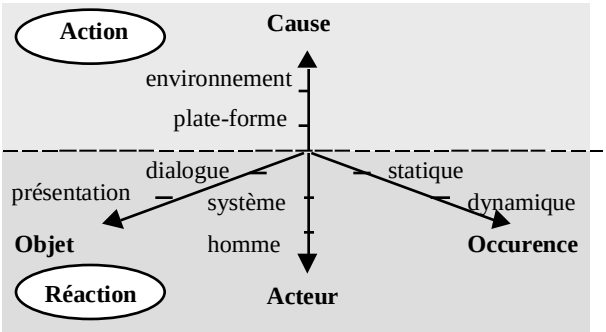


***Figure 1.4 :*** Espace d'adaptation.

Nous affinons cet espace d'adaptation en un processus d'adaptation.

## 1.3 Processus d'adaptation

Le processus d'adaptation que nous proposons structure la réaction en trois étapes : reconnaissance de la situation, calcul de la réaction puis mise en œuvre de la réaction.

### 1.3.1 Reconnaissance de la situation

La reconnaissance de la situation comprend :

- La capture du contexte (ex : température de 22°) ;
- La détection du changement (ex : passage de 21° à 22°) ;
- L'identification du changement (ex : transition du contexte *Normal* au contexte *Confort*). C'est cette identification qui va déclencher une éventuelle réaction. On l'appellera déclencheur. On distingue deux types de déclencheurs : l'entrée dans un contexte et la sortie d'un contexte. Les déclencheurs sont combinables par conjonction et disjonction. Ainsi "sortie(C1) et entrée(C2)" est un déclencheur exprimant la transition du contexte C1 au contexte C2. [19] suggère un troisième type de déclencheur : le fait d'être dans un contexte. Nous ne le considérons pas ici.

### 1.3.2 Calcul de la réaction

Le calcul de la réaction couvre :

- Le recensement des réactions possibles. On prévoit :
  - un changement de plate-forme ou d'environnement (ex passer du portable au Palm lorsque la batterie faiblit ou allumer la lumière lorsque la pièce s'obscurcit) ;
  - un changement d'exécutable. L'interface courante ne couvrant pas le nouveau contexte (elle ne peut pas être remodelée et garantir son utilisabilité), l'exécution d'une nouvelle interface s'impose. Cette solution engendre une discontinuité dans l'interaction ;
  - un remodelage de l'interface (ex passer de 2a à 2b1 en cas de surcharge à l'écran) sans changer d'exécutable;
  - l'exécution de tâches par le système et/ou l'utilisateur (ex déclencher automatiquement le chauffage à l'approche de l'occupant) sans modifier la présentation. Seul le dialogue en est éventuellement modifié.
- Le choix d'une solution parmi les candidates. Plusieurs critères peuvent être considérés, notamment le coût de la migration entre contextes. Ce coût de migration peut être évalué d'un point de vue système et utilisateur. La figure 1.5 en propose une représentation graphique. Cette représentation est à décliner pour chaque critère considéré. Une flèche entre deux contextes indique une migration possible entre le contexte source et le contexte cible. L'épaisseur de la flèche reflète le coût de la migration (un trait épais correspond à une migration coûteuse).

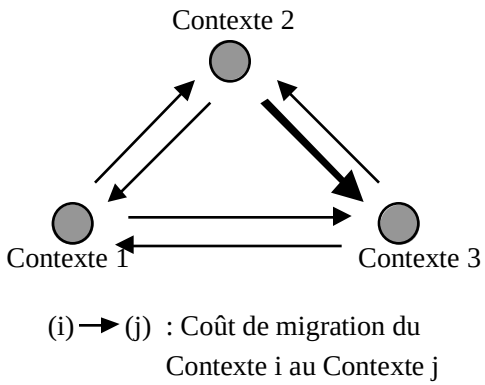



***Figure 1.5 :*** Sur ce graphe de contextes, une flèche d'un contexte source à un contexte cible indique une migration possible entre ces deux contextes. L'épaisseur de la flèche reflète le coût de la migration estimé pour un critère donné.

### 1.3.3 Mise en œuvre

La mise en œuvre consiste à :

- Préparer la réaction. Ce prologue comprend la gestion de la tâche en cours (terminaison, suspension ou avortement), la sauvergarde du contexte d'interaction (ex : température en cours de modification) et la production effective du changement (nouvelle présentation, nouveau dialogue) si cette nouvelle version n'a pas encore été élaborée ;
- Exécuter la réaction (ex : commuter vers la nouvelle présentation, le nouveau dialogue ou exécuter une tâche) ;
- Terminer la réaction. Cet épilogue comprend la restauration du contexte.

Chaque étape peut être mise en œuvre par le système et/ou l'utilisateur, à la volée ou en différé (dès lors qu'une étape se fait en différé, il en sera de même pour les suivantes). L'observabilité de chaque étape sera à étudier, au cas par cas, par le concepteur conformément aux propriétés définies en HCI [9].

Une réaction automatique suppose :

- la connaissance du degré de plasticité de chaque interface (sa couverture potentielle en termes de contextes) ;
- l'écriture d'un modèle d'évolution spécifiant la réaction à mettre en oeuvre en cas de changement de contexte.

Le modèle d'évolution est intégré dans le processus de développement, décrit en partie II. Le degré de plasticité est défini dans la section suivante.

## 1.4 Plasticité : degré de plasticité

Pour rappel, une interface plastique est une interface capable de s'adapter au contexte dans le respect de son utilisabilité (cf 1.1). Il y aura rupture de plasticité dès lors que le nouveau contexte sera situé au-delà de ce seuil de plasticité.

En supposant que l'on puisse "ordonner" les contextes selon les deux axes "plate-forme / environnement", la figure 1.6 représente le seuil de plasticité d'une interface. On y perçoit deux exemples de contextes : un exemple couvert par l'interface, un autre non couvert par cette même interface.

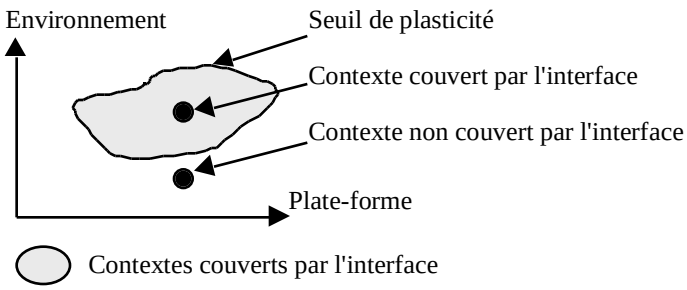



***Figure 1.6 :*** Le degré de plasticité d'une interface reflète la couverture des contextes qu'elle assure sans perte d'utilisabilité.

Il convient de connaître, pour chaque interface, son degré de plasticité. L'approche basée modèles décrite en partie II le permet.

# 2 APPROCHE

Nous présentons, dans la première section, un processus de développement fédérateur permettant l'adaptation d'une interface au contexte. Ce processus adopte une approche basée modèles, dont nous présentons les modèles dans la deuxième section.

## 2.1 Processus de développement

Le processus de développement que nous proposons s'apparente à un méta-processus donnant lieu à différentes mises en œuvre. Après l'avoir décrit dans la première section, nous en présentons une instanciation dans la deuxième section. C'est cette instanciation en Fermeture Eclair qui est mise en œuvre dans le cadre du gestionnaire d'énergie.

### 2.1.1 Processus de référence

La plasticité adopte le slogan « spécifier une fois, générer N fois ». Dans cette optique, le processus de référence propose (Figure 2.1) :

- une **capitalisation** des connaissances au sein de modèles (« spécifier une fois ») ;
- la **déclinaison** de ces modèles en différentes IHM selon le contexte[1] d'interaction ciblé (« générer N fois »).

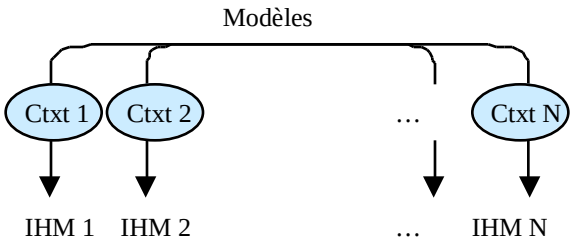



***Figure 2.1 :*** Capitalisation de modèles. Déclinaison en différentes IHM selon le contexte d'interaction ciblé.

La déclinaison s'effectue par génération. Nous en distinguons deux types :

- une **génération verticale** qui s'exerce par **réification ;**
- une **génération horizontale** qui procède par **traduction**.

#### 2.1.1.1 Réification et Traduction : Définitions

Par définition, la réification est une chosification. Dans le contexte de cette étude, la réification consiste à transformer par étapes un modèle (ou représentation) en une interface homme-machine concrète. Chaque étape produit par inférence une représentation à un niveau d'abstraction donné. Nous dirons que chaque étape définit un niveau de réification.

L'inférence s'exerce sur un modèle source : elle applique, à ce modèle, un ensemble de règles de production qui exploitent toutes connaissances déjà acquises par ailleurs. Les règles de production concernent l'organisation des informations, le style d'interaction, l'ergonomie, etc. ainsi que les règles de compilation de l'application. On distingue, dans le principe, deux possibilités (Figure 2.2) :

[1] Le contexte est défini (cf partie I) par la plate-forme cible et l'environnement de l'utilisateur.

- Soit les informations générées enrichissent le modèle source (cas a) ;
- Soit elles définissent un nouveau modèle (cas b).

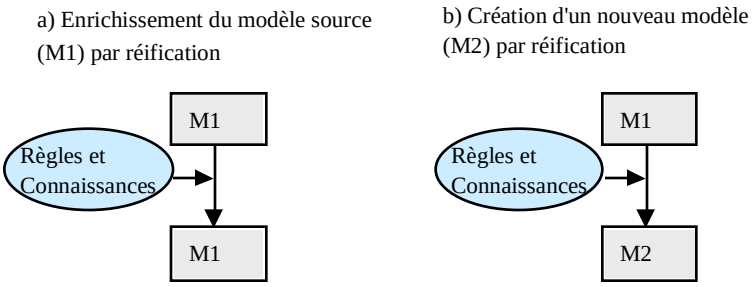



***Figure 2.2 :*** En pratique, les informations inférées lors de la réification (a) enrichissent le modèle source M1 ou (b) définissent un nouveau modèle M2.

Dans notre approche, seul le cas b) est appliqué.

Par opposition, la traduction conserve le niveau d'abstraction des informations. Elle traduit, à un niveau de réification donné, un ensemble de connaissances en des connaissances homologues adaptées à un nouveau contexte (Figure 2.3).

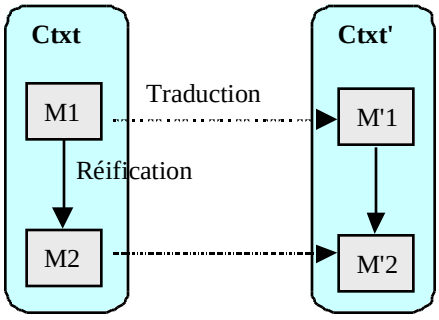



***Figure 2.3:*** La traduction s'apparente à une fonction de transfert entre contextes : elle s'exerce à un niveau de réification donné et convertit, à ce niveau, les modèles réifiés en de nouveaux modèles adaptés au nouveau contexte.

Réifications et traductions supposent un ensemble d'heuristiques. Notons :

- *Reification* $_{ctxt}$ (*M*) le modèle obtenu, dans le contexte *ctxt* par l'application des heuristiques de réification sur le modèle *M* ;
- *Traduction* $_{ctxt, ctxt'}$(*M*) le modèle obtenu par traduction pour passer du contexte *ctxt* à *ctxt'* à partir du modèle *M*.

Pour un ensemble donné d'heuristiques, le processus de génération garantit :

- L'identité par traduction et traduction inverse

$$\text{ctxt, ctxt' } Traduction_{ctxt, ctxt'} \circ Traduction_{ctxt', ctxt} = I$$

- L'identité par réification et réification inverse

$$\text{ctxt } Reification_{ctxt} \circ Reification_{ctxt}^{-1} = I$$

· L'équivalence entre "réification puis traduction" et " traduction puis réification"

$$\forall ctxt, ctxt' \; Traduction_{ctxt,\,ctxt'} \circ Reification_{ctxt'} = Reification_{ctxt} \circ Traduction_{ctxt,\,ctxt'}$$

Les heuristiques de génération sont décrites dans le paragraphe 2.

En résumé :

· la réification conserve le contexte d'interaction : elle infère, dans un contexte donné, des informations utiles à la conception de l'IHM ;
· la traduction conserve le niveau de réification : c'est une fonction de transfert entre contextes. Elle convertit, à un niveau de réification donné, les connaissances valides dans le contexte source en des connaissances homologues valides dans le nouveau contexte.

#### 2.1.1.2 Réification et traduction : Coopération

Le processus de référence combine réification et traduction (Figure 2.4) :

· En vertical, pour un contexte donné, la réification permet la synthèse d'informations, avec, à terme, la production de l'IHM finale ;
· En horizontal, la traduction offre un ensemble de passerelles entre contextes. Ces transferts peuvent s'effectuer à différents niveaux de réification : ceux prévus dans les branches verticales.

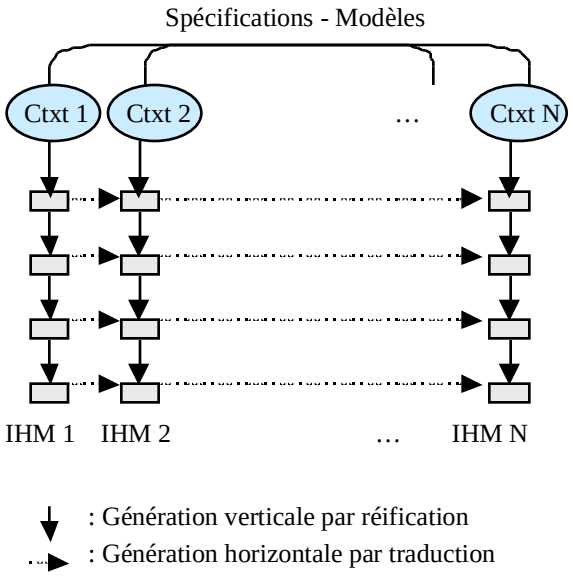



***Figure 2.4:*** Coopération entre réification et traduction : la réification, par son caractère itératif, définit les lieux possibles de traduction. La traduction s'exerce, de façon horizontale, à un niveau de réification donné.

La traduction s'appuyant sur les étapes de réification, nous présentons, en premier lieu, l'aspect réification du processus de référence.

Le processus de réification est un processus à étapes qui s'appuie sur les modèles. A chaque étape (c'est-à-dire chaque nouvelle réification), le processus prend en entrée des modèles et en produit de nouveaux. On distingue trois types de modèles : les modèles initiaux, transitoires et terminaux.

- Un **modèle** est dit **initial** s'il est établi par le concepteur et n'alimente le processus qu'en entrée ;
- Un **modèle** est dit **transitoire** s'il est le fruit d'une étape intermédiaire de réification ou traduction. Un modèle transitoire peut être produit par le concepteur, généré par le système voire retouché manuellement ;
- Un **modèle** est dit **terminal** s'il est le fruit final du processus.

Le processus de référence suggère des modèles initiaux/transitoires/finaux mais n'en impose pas l'usage (Figure 2.5). L'énumération n'est pas non plus limitative.

**Modèles initiaux**

Les modèles initiaux constituent une base de connaissance :

- Le modèle des concepts décrit les concepts faisant sens dans le domaine applicatif traité ;
- Le modèle des tâches formalise l'activité humaine classiquement observée dans le monde réel puis transposée sur support informatique ;
- Le modèle de la plate-forme décrit le matériel ciblé en termes de dispositifs d'interaction, de performances de calcul et de communication ;
- Le modèle des interacteurs spécifie les objets d'interaction ou widgets disponibles dans l'environnement de développement adopté ;
- Le modèle de l'environnement décrit le contexte d'utilisation sous ses aspects géographique, physique, social, etc. ;
- Le modèle d'évolution spécifie l'évolution inter-contextes de l'application pour des variations de plate-forme ou d'environnement (changement d'éclairage, batterie à plat, bande passante nulle, etc.).

Ces modèles sont établis manuellement, capitalisés puis référencés dans le processus de développement lors d'une phase de réification ou traduction. Le processus de référence ne contraint pas ces liens de dépendance : il laisse le développeur libre de référencer, à tout niveau de réification, ces modèles initiaux. Néanmoins, plus les références seront tardives (fort niveau de réification), plus les domaines de validité des modèles transitoires seront larges. La gestion de configuration en sera simplifiée. Dans l'exemple de la figure 2.5 :

- la plate-forme est référencée dès la spécification orientée tâches. En conséquence, tous les modèles transitoires en aval de ce lien de dépendance seront à revoir en cas de changement de plate-forme.
- L'environnement, par contre, n'intervient que dans l'IHM concrète : seul ce modèle transitoire risquera l'obsolescence dans un nouvel environnement.

**Modèles transitoires**

Les modèles transitoires sont des modèles intermédiaires produits semi-automatiquement :

- La spécification orientée tâches est un enrichissement du modèle des tâches. Elle précise, par exemple, les concepts manipulés dans les tâches. Elle décore les tâches d'attributs de criticité, complexité, fréquence, etc., dans le contexte d'interaction traité. Elle référence, pour ce faire, les modèles initiaux tâches et concepts ;
- L'interface abstraite structure l'IHM en espaces de travail et spécifie l'enchaînement entre espaces ;
- L'interface concrète concrétise ces espaces en termes d'objets d'interaction.

Ces modèles sont suggérés par le processus mais non obligatoires.

**Modèles terminaux**

· Les interfaces finales sont les seuls exemples de modèles terminaux. Il s'agit de modèles décrivant l'application finale indépendamment du choix de langage et du système d'exploitation. Ils permettent de générer des interfaces exécutables écrites en Java/Swing, C++/MacOS ou C++/PalmPilot par exemple.

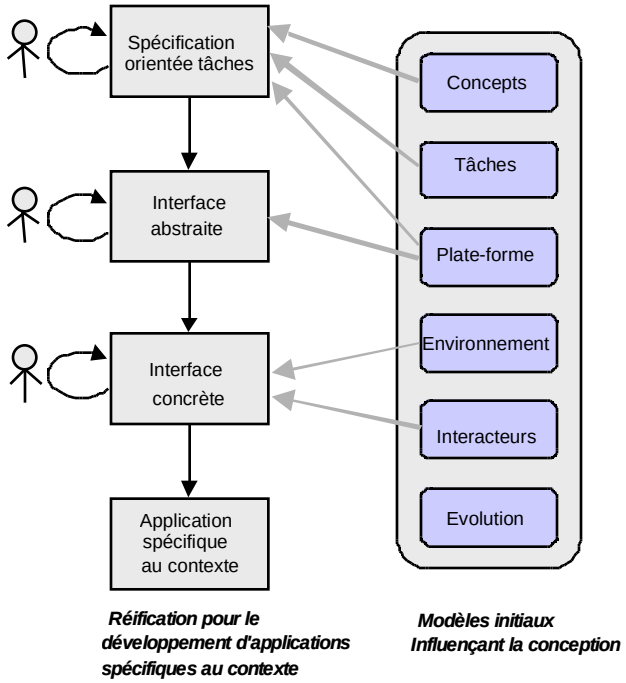



***Figure 2.5 :*** La réification dans le processus de référence.

Le processus de référence prévoit un brin (ou fil) de réification pour chaque contexte d'interaction. Les brins coopèrent via des opérations de traduction (Figure 2.6). Réifications et traductions y sont semi-automatiques : l'expertise humaine est volontairement préservée. Elle prévaut sur les heuristiques de génération. Les interventions humaines risquent, par contre, de compromettre la satisfaction des propriétés énoncées en 2.1.1. Au concepteur de vérifier leur satisfaction.

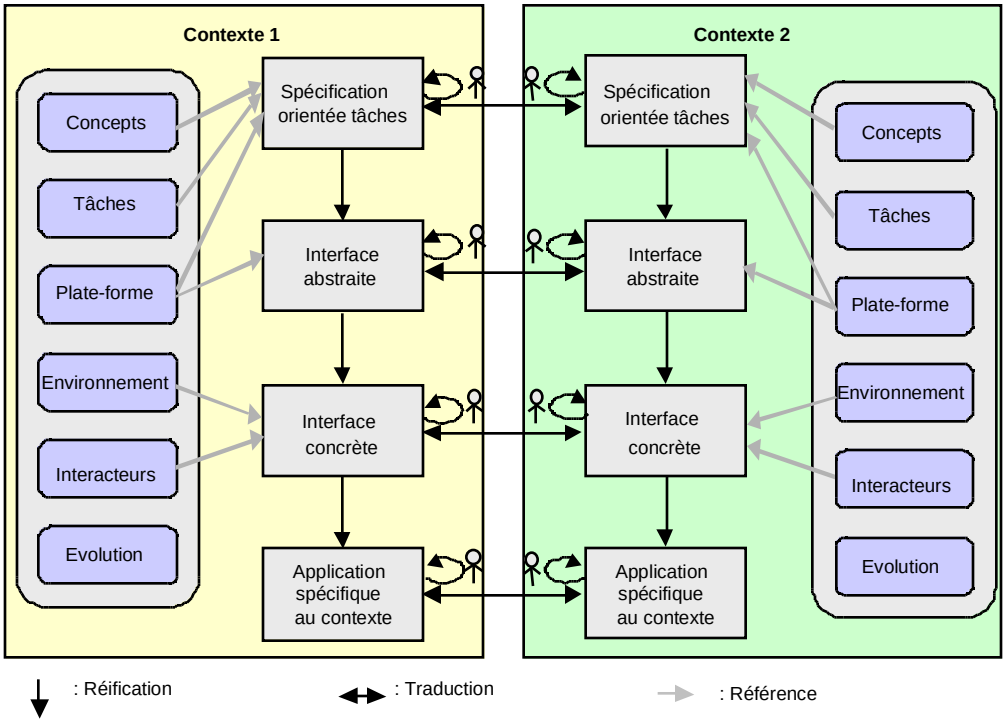



***Figure 2.6 :*** Processus de référence.

Ce processus s'apparente à un méta-modèle de processus donnant lieu à différents schémas de coopération entre réifications et traductions (Figure 2.7) :

- Réifications indépendantes (cas a) ;
- Traduction terminale (cas b) ;
- Traduction initiale (cas c) ;
- Entrelacement opportuniste de réifications et de traductions (cas d).

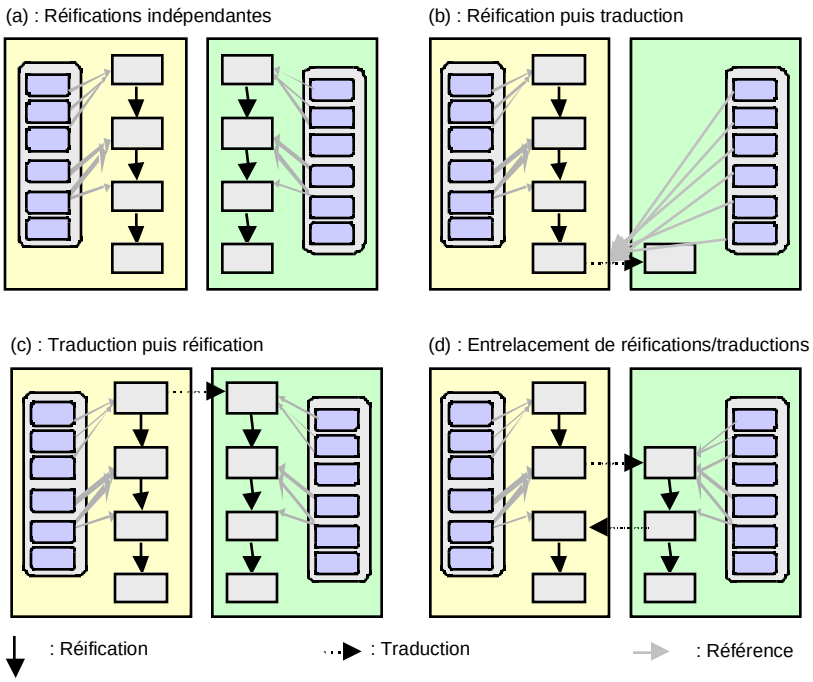



***Figure 2.7 :*** Schémas de coopération entre réifications et traductions.

En résumé :

- Le processus de référence préconise un ensemble de modèles initiaux. Ces modèles alimentent le processus de réification et traduction.
- Chaque réification et traduction se consigne en un modèle transitoire. Le processus préconise un ensemble de modèles transitoires. Ces modèles sont établis par le concepteur ou produits par le système. Le concepteur peut toujours remanier un modèle généré par le système support du processus.
- Plus les modèles initiaux seront référencés tard dans le processus, plus les modèles transitoires seront génériques. Le point d'injection reste à l'appréciation du concepteur.
- Réifications et traductions coopèrent selon des schémas non imposés. Quatre stratégies émergent.

### 2.1.2 Instanciation en Fermeture Eclair

Le processus de référence s'apparente à un méta-modèle de processus donnant lieu à différentes mises en œuvre. Parmi elles, le principe de la fermeture éclair que nous retenons pour une meilleure capitalisation. Il s'agit de :

- factoriser les modèles communs aux différents contextes ;
- retarder, au maximum, le point de divergence entre contextes.

Pour ce faire :

- Le modèle des concepts couvre le plus largement possible le domaine applicatif traité. Il ne raisonne pas en termes de contexte, ne se limite pas aux concepts utiles dans un contexte donné. Il couvre idéalement tous les concepts faisant sens dans le domaine, indépendamment des contextes d'interaction envisagés lors de la conception. Si certains concepts ne s'avèrent pertinents que pour certains contextes (i.e. ils ne font sens que dans ces contextes), alors des décorations peuvent être introduites pour exprimer ce domaine de "validité" du concept

(i.e. le rayonnement du concept). Les décorations s'apparentent à des attributs relatifs à la plate-forme et à l'environnement. Ce modèle des concepts se veut représentatif du domaine. Sa capitalisation est intéressante ;

2. Le modèle des tâches se veut, de la même façon, un modèle unique regroupant les modèles des tâches spécifiques aux différents contextes. Les spécificités sont exprimées par le biais de décorations. Elles spécifient, par exemple, que la tâche "Changer le réglage de la température du niveau Confort" (Tâche ACF7) n'est accessible que du boîtier électronique, les dispositifs satellites n'y donnant pas accès. Cette "factorisation" par le biais de décorations peut être déportée vers la spécification orientée tâches : mais plusieurs modèles de tâches sont alors à gérer (écriture, maintenance, etc.). Ainsi ces décorations permettent d'exprimer, dans un modèle général, des spécificités liées à un contexte donné. On peut supposer que ces décorations se retrouveront dans le modèle d'évolution qui spécifie la transition entre contextes.

L'idée sous-jacente est de repérer les similitudes en termes de tâches entre les différents contextes d'interaction recensés lors de la conception. Deux arbres seront dits similaires si leurs spécificités sont exprimables par le biais de décorations.

- Si des similitudes apparaissent entre contextes, le concepteur considère l'union de ces contextes. Il écrit les modèles pour cette union et exprime, si nécessaire, des spécificités par le biais de décorations ;
- Si les divergences sont trop importantes, les contextes sont traités tels quels en tant que contextes d'interaction (Figure 2.8). Ils justifient un brin de réification/traduction dans le processus de développement.

Les unions de contextes peuvent être recherchées à partir d'un contexte de référence, par exemple, le contexte le plus fréquent (Figure 2.8).

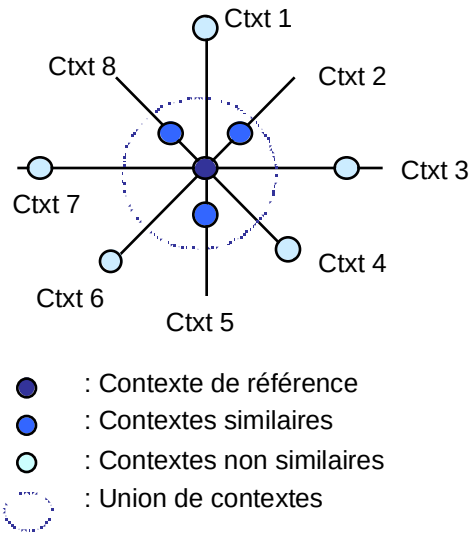



***Figure 2.8 :*** Diagramme de Kyviatt comparant les contextes d'interaction à un contexte de référence. En cas de similitude, les contextes concernés sont amalgamés en une union de contextes, devenant le contexte d'interaction à traiter par le concepteur.

Les unions de contextes permettent de factoriser les efforts déployés en amont du processus. Leur validité reste temporaire, les spécificités des contextes pouvant justifier la divergence ultérieure du processus en brins de réification/traduction dédiés. Le processus s'ouvre alors en Fermeture Eclair (Figure 2.9) : c'est le processus mis en œuvre dans le gestionnaire d'énergie. Dans ce processus :

- Le modèle des concepts est unique, commun aux différents brins de réification/traduction ;

- Le modèle des tâches est unique, les spécificités entre plates-formes et environnements étant exprimées par le biais de décorations ;
- Les décorations introduites dans le modèle des tâches vivent tout au long du processus. Elles se retrouvent dans les spécifications orientées-tâches et tout autre modèle transitoire ;
- Les modèles de la plate-forme et de l'environnement peuvent être référencés tout au long du processus, l'idée étant de : (a) ventiler, à bon escient, les spécificités des contextes et (b) forcer la conscience du concepteur quant aux dépendances de son application envers le contexte traité ;
- La scission de l'union des contextes en contextes peut s'effectuer à tout moment. Dans un objectif de capitalisation, l'idée est de retarder au maximum ce point de divergence. Si notamment le modèle des interacteurs diffère entre plates-formes, l'idéal n'est d'ouvrir la fermeture éclair qu'au niveau des interfaces concrètes (Figure 2.9). C'est ce qui est pratiqué dans le gestionnaire d'énergie.

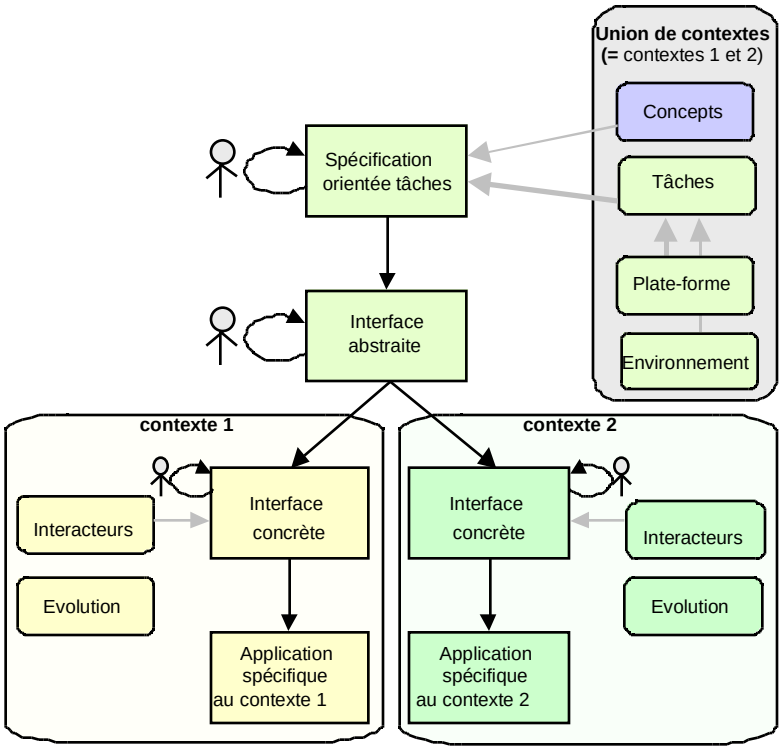



***Figure 2.9 :*** Instanciation du processus de référence en Fermeture Eclair.

Dans ce processus, cinq stratégies sont envisageables pour appréhender un nouveau contexte (Figure 2.10) :

- les quatre stratégies recensées pour le processus de référence (Figure 2.7) ;
- plus une stratégie de réification partielle résultant de la capitalisation des modèles.

Ces stratégies sont les suivantes (Figure 2.10) :

1. procéder à une réification totale à partir des modèles initiaux, indépendamment des réifications précédentes (cas a Figures 2.7 et 2.10). Cette stratégie ne met pas à profit la capitalisation propre au processus en Fermeture Eclair ;

2. procéder à une réification partielle à partir du point de divergence (cas a' - Figure 2.10). Cette stratégie est une optimisation de la stratégie précédente (cas a). Elle met à profit la fermeture éclair. C'est la stratégie aujourd'hui appliquée dans le gestionnaire d'énergie, sauf pour la plate-forme WAP ;

· traduire l'interface finale en une version homologue adaptée au nouveau contexte (cas b - Figures 2.7 et 2.10) ;

· traduire l'interface concrète puis réifier dans le nouveau contexte (cas c - Figure 2.10). Cette stratégie est l'homologue du cas c - Figure 7 mais s'apparente, du fait de la capitalisation, à un entrelacement de réifications/traductions (cas d Figure 2.7) ;

· l'entrelacement (cas d - Figure 2.7) peut prendre d'autres formes si le développeur personnalise son processus en introduisant d'autres modèles transitoires.

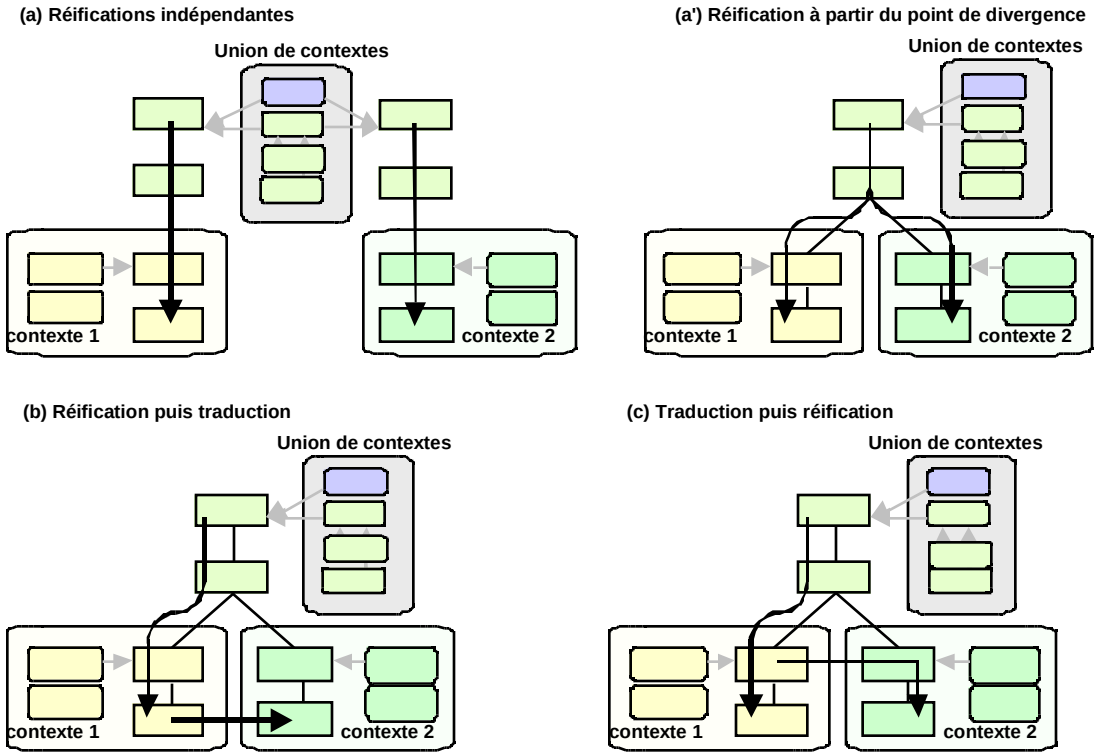



***Figure 10 :*** Stratégies de réification/traduction dans le processus en Fermeture Eclair.

En résumé, le processus en Fermeture Eclair est une instance du processus de référence. Il s'apparente lui-même à un méta-modèle de processus. Il incite à la capitalisation des connaissances par l'introduction d'unions de contextes mais il laisse libre :

· le nombre de modèles transitoires ;

· le point d'ouverture de la fermeture éclair ;

· les stratégies de réifications et traductions.

Nous décrivons les modèles dans la section suivante.

## 2.2 Modèles

Nous organisons les modèles selon leur caractère initial, transitoire ou final.

### 2.2.1 Modèles initiaux

#### 2.2.1.1 Concepts

Le modèle des concepts est un enrichissement des modèles objet classiques. Il s'apparente à une structure d'objets augmentée, pour la plasticité, de la spécification des domaines de variation des différents attributs. Par exemple, dans le cadre du gestionnaire d'énergie, *nomZone* est une chaîne prenant comme valeur "*Salon*", "*Chambre*" ou "*Salle de bain*".

Cette spécification, utile en contexte graphique, sert, en phase de génération, à optimiser la présentation et l'organisation des objets à l'écran. Elle sert notamment à choisir les objets d'interaction incarnant les concepts à l'écran. Par exemple, *nomZone* peut être incarnée par un onglet, un menu déroulant ou tout interacteur métier dédié à la représentation d'une pièce de la maison (une icône "lit" par exemple).

En pratique, ce modèle du domaine est capitalisé. Des adaptations peuvent s'avérer nécessaires pour certains contextes d'interaction. En termes d'architecture logicielle, les adaptations sont alors gérées dans un adaptateur du noyau fonctionnel. Par exemple, en système, la date est maintenue comme un entier *t* calculé par rapport à une référence *t0*. Dans certains contextes, le concepteur peut vouloir la modéliser comme :

- trois entiers représentant le jour, le mois, l'année (10 / 08 / 2000)
- ou un entier, une chaîne, un entier représentant le jour, le mois, l'année (10 Août 2000).

Ces adaptations sont des réparations sémantiques à usage humain.

#### 2.2.1.2 Tâches

Le modèle des tâches est empreint de la mise en œuvre, sur support informatique, des tâches utilisateur. Le modèle s'apparente à un arbre structuré en nœuds et feuilles :

- les nœuds hébergent les **tâches abstraites** ;
- les feuilles regroupent les **tâches** dites **d'interaction** ;
- les branches expriment des relations logiques et temporelles entre tâches.

A la différence d'une tâche abstraite, une tâche d'interaction requiert une interaction utilisateur : elle implique l'utilisateur dans une interaction homme-machine. C'est typiquement une tâche de sélection ou de spécification offertes par les interacteurs d'utilité publique tels que radio boutons, cases à cocher, champs texte, etc. Nous en proposons un lexique extensible :

- Consulter
- Spécifier
- Sélectionner
- Activer
- Autre : tâche métier (par exemple mettre-hors-gel).

Par opposition aux tâches d'interaction, une tâche abstraite se veut structurante. Elle s'apparente au parenthésage mathématique :

- elle fédère des tâches dont l'union fait sens ;
- ou factorise un ensemble de tâches dont l'agencement est récurrent.

Comme en architecture logicielle, l'arbre des tâches exprime ce qui est important. Sa décomposition s'arrête là où le concepteur juge opportun. Ce niveau peut être apprécié de différentes façons :

- le concepteur peut consulter sa boîte à outils (interacteurs disponibles) et suspendre la décomposition dès lors qu'un interacteur offre la tâche concernée ;
- le concepteur peut, au regard du domaine, s'arrêter sur des tâches faisant sens dans ce domaine. Il exprime alors des exigences en termes d'interacteurs. Par exemple, la mise à disposition d'un interacteur permettant la commutation hors-gel ;
- le concepteur peut aussi naturellement décomposer, puis prendre du recul et proposer des tâches et interacteurs métier permettant de factoriser des agencements récurrents.

#### 2.2.1.3 Plate-forme

Le modèle de la plate-forme décrit le matériel ciblé en termes de dispositifs d'interaction, de performances de calcul et de communication. Pour l'objet de cette étude, il comprend la spécification de la surface d'affichage en :

- nombre de pixels (largeur x hauteur) ;
- profondeur.

#### 2.2.1.4 Environnement

Le modèle de l'environnement décrit le contexte d'utilisation. Cette notion de contexte est encore hésitante dans la littérature. L'annexe A en propose une définition et caractérisation.

#### 2.2.1.5 Interacteurs

Le modèle des interacteurs spécifie les objets d'interaction ou widgets disponibles dans l'environnement de développement adopté. Un interacteur peut être vu comme une modalité et donc spécifié par un couple $m=<r, d>$ où $r$ et $d$ dénotent respectivement un système représentationnel et un dispositif d'interaction. Dans le contexte de la plasticité, nous affinons cette modélisation par l'expression d'un pouvoir représentationnel, d'une capacité interactionnelle et d'un coût d'exploitation.

**Pouvoir représentationnel**

Le pouvoir représentationnel d'un interacteur exprime le type des données que peut représenter cet interacteur. Par exemple :

- Un champ texte rend observable une chaîne quelconque (Figure 2.11a) ;
- Un menu déroulant rend observable une chaîne définie par extension, c'est-à-dire une chaîne contrainte à évoluer dans un ensemble prédéfini (Figure 2.11b) ;
- Un interacteur dédié rend observable un concept du domaine avec sa sémantique métier (Figure 2.11c).

(a) Champ texte : chaîne quelconque


(b) Menu déroulant : chaîne définie par extension


(c) Interacteur dédié : les mois de l'année représentés par des symboles spécifiques (Halloween pour octobre, bonhomme de neige pour l'hiver, etc.)

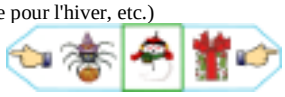


***Figure 2.11 :*** Pouvoir représentationnel des interacteurs.

Mais "*qui peut le plus, peut le moins*", aussi :

- La chaîne définie par extension est-elle représentable par une chaîne quelconque ;
- Le concept du domaine est-il représentable par une chaîne s'il offre une méthode permettant de l'écrire sous cette forme (typiquement *asString*).

Ainsi, en supposant une organisation des types selon une relation de généricité (Figure 2.12), le pouvoir représentationnel d'un interacteur exprime le type le plus précis (spécifique) que peut représenter cet interacteur.

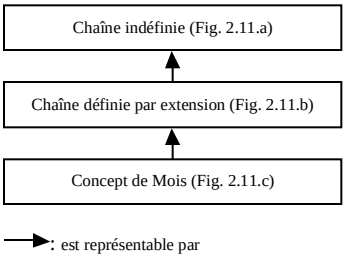



***Figure 2.12:*** Organisation des types selon une relation de généricité.

Le coût d'exploitation des interacteurs guidera le concepteur dans son choix d'interacteurs.

**Capacité interactionnelle**

La capacité interactionnelle d'un interacteur exprime la tâche d'interaction déployable sur celui-ci par l'utilisateur. Cette tâche est une tâche d'utilité publique (cf lexique) ou une tâche métier telle que « *commuter hors gel*».

**Coût d'exploitation**

Le coût d'exploitation d'un interacteur mesure son coût de mise en œuvre. Nous l'évaluons d'un point de vue système et utilisateur.

- Dans une perspective système, nous quantifions les ressources matérielles nécessaires à l'exploitation de l'interacteur. Cette mesure évalue les requis physique et computationnel. Aujourd'hui, seul le niveau physique du système représentationnel est appréhendé : nous modélisons l'encombrement spatial d'un interacteur graphique et spécifions les dispositifs d'interaction requis. Il convient, à terme, de quantifier les requis en bande passante, réseaux, etc.

3. Dans une perspective utilisateur, le coût d'exploitation représente la difficulté de mise en œuvre de la tâche. Il se mesure aux niveaux physique et cognitif. Le coût physique relève du niveau opératoire : il intègre (a) l'indice de difficulté des actions physiques, (b) le surcoût des actions articulatoires et (c) la longueur de la

trajectoire d'interaction. Le coût cognitif représente la difficulté liée à la formulation d'intentions. Nous l'incombons, en partie, à la non proactivité ou non réactivité de l'interacteur. Par définition, la proactivité dénote une saine intention de bien-faire : elle préserve l'utilisateur d'actions inopportunes. La réactivité laisse, au contraire, l'utilisateur libre d'actions mais en vérifie, a posteriori, la légitimité. Un menu déroulant est ainsi, par essence, proactif : il ne propose à l'utilisateur que des options valides. Par opposition, un champ texte est non proactif : il laisse l'utilisateur saisir n'importe quelle chaîne. Il deviendra faiblement proactif s'il est associé à d'autres interacteurs tels qu'un libellé explicitant le domaine de variation autorisé. Une faible proactivité peut être contrée par une réactivité : le système vérifie alors, a posteriori, la validité des informations spécifiées par l'utilisateur. Cette vérification peut être menée de façon locale à l'interacteur (elle serait typiquement déclenchée à la perte de focus de l'interacteur - on pourrait alors parler de contrôle *formatif*) ou globale à l'espace de travail contenant l'interacteur (on pourrait alors parler de contrôle *sommatif*).

L'intérêt de la mesure du coût d'exploitation est double : aider, en phase de génération, à identifier l'interacteur le plus approprié et quantifier toute dégradation de l'utilisabilité liée à un changement d'interacteur.

Nous notons ainsi une différence d'usage entre les trois facettes d'un interacteur :

- Le pouvoir représentationnel et la capacité interactionnelle sont d'un usage *absolu* : ils statuent quant à l'adéquation d'un interacteur pour représenter un concept et offrir une tâche donnés ;
- Le coût d'exploitation est, par opposition, d'un usage *relatif* : il permet une comparaison entre interacteurs.

Nous proposons une caractérisation *absolue* des interacteurs selon leurs pouvoir représentationnel et capacité interactionnelle (Figure 2.13). Cette caractérisation exprime pour chaque interacteur :

- Le statut métier / public des concepts et tâches offerts par l'interacteur ;
- Le statut non décomposé / non décomposable des concepts et tâches offerts par l'interacteur. Cette information servira lors d'un changement de plate-forme pour éventuellement décomposer un concept ou une tâche décomposables.

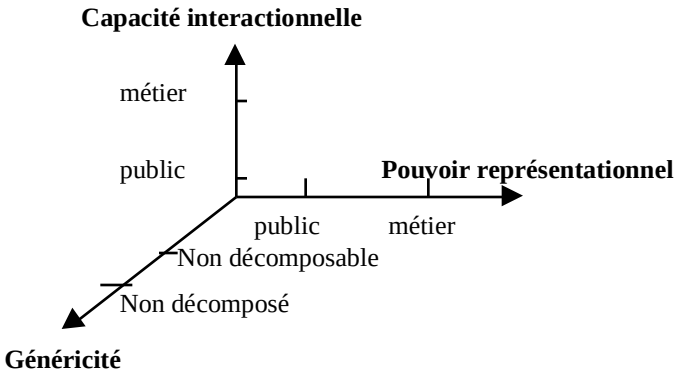



***Figure 2.13 :*** Caractérisation des interacteurs.

Les interacteurs sont aujourd'hui gérés par plate-forme. Nous envisageons, à terme, une organisation objet explicitant les spécificités par plate-forme selon un arbre d'héritage (Figure 2.14).

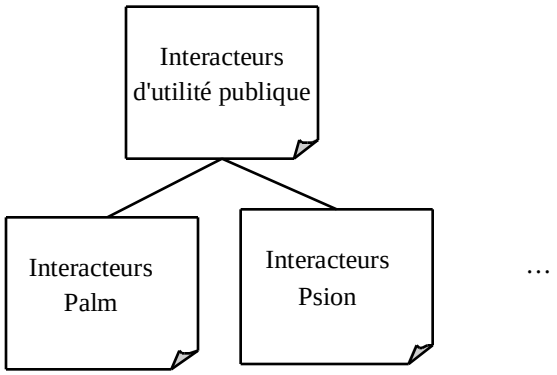



***Figure 2.14 :*** Organisation hiérarchique des interacteurs selon la plate-forme ciblée.

#### 2.2.1.6 Evolution

Le modèle d'évolution (introduit dans [4]) spécifie la réaction à mettre en œuvre en cas de changement de contexte. Pour chaque déclencheur (combinaison d'entrées et de sorties de contextes), le concepteur y spécifie un prologue, une réaction, un épilogue. Par exemple :

- en cas de connexion au réseau, commuter automatiquement en mode "bureau" (réaction) ;
- en cas de mémoire insuffisante, sauvegarder les applications courantes (prologue), rebooter (réaction) puis réouvrir les documents de travail (épilogue).

Ces spécifications sont facultatives :

- Si aucun prologue n'est spécifié, aucune tâche pré-réaction ne sera exécutée ;
- Si aucun épilogue n'est spécifié, aucune tâche post-réaction ne sera exécutée ;
- Si la réaction n'est pas spécifiée, le système la calculera.

Considérons, par exemple, un changement de contexte $C1 \rightarrow C2$ pour lequel la réaction n'est pas spécifiée dans le modèle d'évolution. Quatre cas se présentent (Figure 2.15) :

- Cas a : Les contextes C1 et C2 sont couverts par la même interface. L'interface courante reste valide.
- Cas b et c : Le contexte C2 est situé au-delà du seuil de plasticité de l'interface courante (couvrant C1). Un changement d'interface s'impose. Il s'agit de recenser les interfaces couvrant C2 : une seule dans le cas b) ; plusieurs, dont potentiellement l'interface courante, dans le cas c) ;
- Cas d : Le contexte C2 n'est couvert par aucune interface. L'activité de l'utilisateur est dégradée par une perte de l'utilisabilité, voire condamnée.

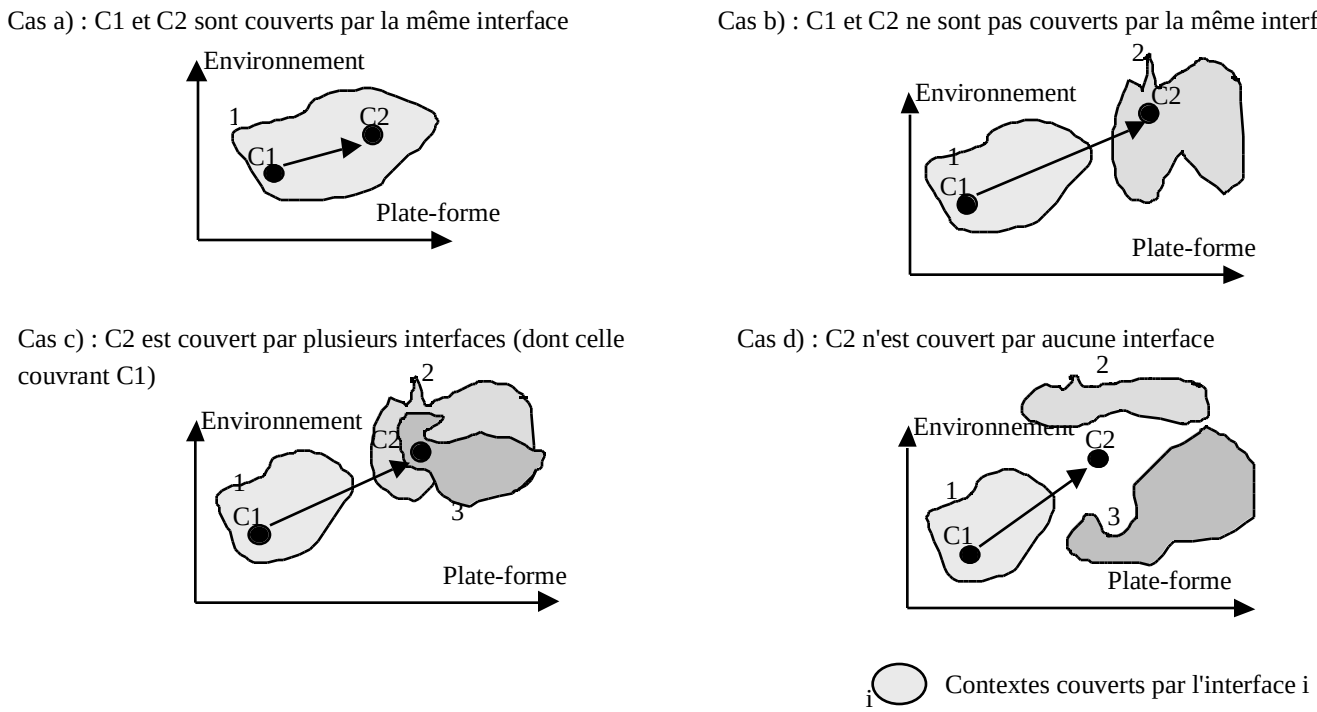



***Figure 2.15 :*** Degré de plasticité et changement de contexte.

Dans le cas c), un critère de choix s'impose. Les coûts de migration d'un contexte à l'autre peuvent être considérés.

En pratique, le modèle d'évolution n'est pas forcément un modèle centralisé. Il peut être distribué dans les différents modèles.

### 2.2.2 Modèles transitoires

#### 2.2.2.1 Spécifications orientées-tâches

Les spécifications orientées-tâches établissent des liens entre les tâches et le système. Ces liens s'expriment en termes de concepts et de fonctions.

**Concepts**

Chaque tâche manipule potentiellement des concepts. Par exemple, la tâche « *sélectionner zone* » manipule le concept de *zone*. Chaque concept est décoré d'une propriété de centralité reflétant son importance dans la tâche. La centralité est une information numérique : un concept sera dit de Nème classe, N=1 dénotant une information centrale. Cette propriété servira, en phase de génération à optimiser le rendu des informations à l'écran. Par exemple, un concept de Nème classe, N≥2, pourra devenir accessible au prix d'une tâche de navigation si la surface d'affichage se restreint.

La centralité d'un concept peut être exprimée :

- Au niveau du concept ou d'un de ses attributs ;
- En absolu (tel concept est toujours central dans telle tâche) ou en relatif (tel concept est toujours plus central que tel autre dans telle tâche).

Le sens IN/OUT des concepts n'est pas exprimé. Il est implicite, déductible de la sémantique de la tâche.

**Fonctions**

Chaque tâche est assortie d'un prologue et d'un épilogue. Prologues et épilogues sont des appels de fonction système respectivement pré et post exécution de la tâche.

- Le prologue permet typiquement de récupérer un concept du noyau fonctionnel manipulé dans la tâche. Par exemple, le premier programme hebdomadaire ;
- Inversement, l'épilogue permet de mettre à jour le noyau fonctionnel après modification du concept manipulé dans la tâche.

Plus généralement, prologues et épilogues permettent l'activation automatique de fonctions système (Figure 2.16).

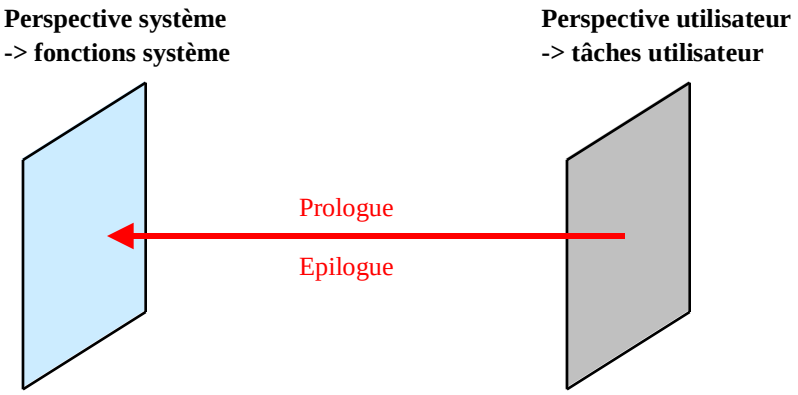



***Figure 2.16 :*** Prologues et épilogues définissent des passerelles entre deux perspectives : la perspective utilisateur en termes de tâches utilisateur et la perspective système en termes de fonctions offertes par le système.

En termes d'architecture logicielle ARCH, les spécifications orientées-tâches établissent des liens explicites entre le dialogue d'une part (contrôleur de dialogue, techniques de présentation et d'interaction) et le noyau fonctionnel d'autre part (noyau fonctionnel, adaptateur de noyau fonctionnel) (Figure 2.17).

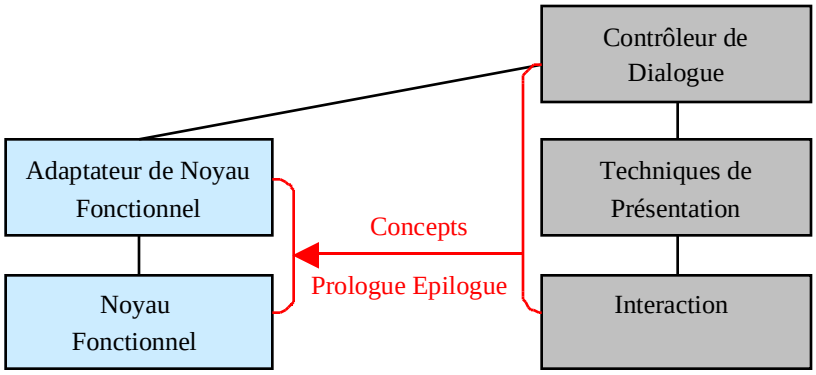



***Figure 2.17 :*** Les spécifications orientées-tâches établissent des liens explicites entre les deux branches de l'arche : dialogue / noyau fonctionnel.

#### 2.2.2.2 Interface abstraite

L'interface abstraite structure l'interface en espaces de travail et définit l'enchaînement entre espaces. Les solutions sont nombreuses et justifient l'adoption d'heuristiques de génération. La figure 2.18, par exemple, représente trois interfaces abstraites envisageables pour la tâche « Programmer son confort ». Cette tâche consiste à définir son rythme de vie (RdV) et programmer son confort par zone. Trois zones sont identifiées : le salon, la chambre et la salle de bain.

3. En (a), les quatre tâches d'interaction sont hébergées dans un même espace de travail ;
4. En (b), éditer rythme de vie est isolée dans un espace dédié ;

· En (c), un espace de travail est accordé à chaque tâche d'interaction.

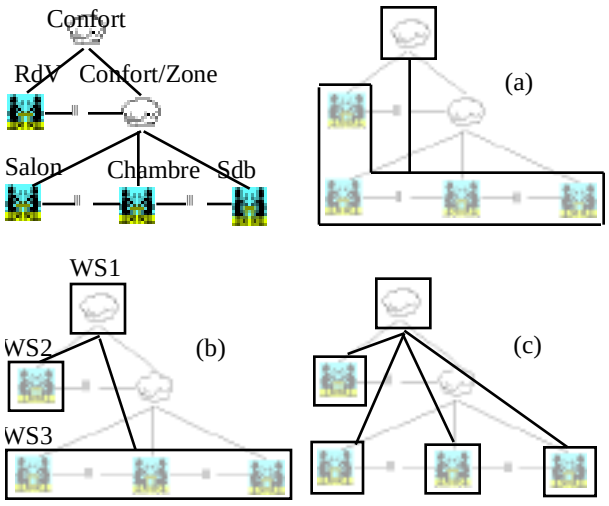



***Figure 2.18 :*** Non unicité de l'interface abstraite d'où la nécessité d'heuristiques de génération.

#### 2.2.2.3 Interface concrète

L'interface concrète matérialise :

· les espaces de travail en fenêtres ou canvas ;
· les concepts et systèmes de navigation en interacteurs.

### 2.2.3 Modèles finaux

L'application finale est obtenue par simple compilation.

# 3 OUTIL

Dans cette partie, nous présentons ArtStudio, un assistant logiciel pour la production d'interfaces plastiques. ArtStudio se décline en deux outils :

- PlasticityApp, un générateur automatique, utilisable en ligne de commande. PlasticityApp n'offre aucun environnement d'édition des modèles. Les modèles peuvent être édités manuellement, hors PlasticityApp, via un éditeur de texte, par exemple.
- ArtStudioApp, un environnement de gestion, d'édition et de génération des modèles impliqués dans la production d'interfaces plastiques. ArtStudioApp permet de spécifier les modèles des tâches, concepts et liens entre tâches et concepts (spécification orientée tâches), de visualiser le modèle de l'IHM abstraite et d'éditer le modèle de l'IHM concrète.

PlasticityApp et ArtStudioApp adoptent une même logique de fonctionnement. Ils diffèrent par leurs logiques d'utilisation. Nous en décrivons ici la logique de fonctionnement puis les logiques d'utilisation.

## 3.1 Logique de fonctionnement

Après avoir présenté le processus de développement implémenté dans ArtStudio, nous décrivons ici les modèles impliqués ainsi que leurs heuristiques de génération.

### 3.1.1 Processus de développement

Le processus de génération implémenté dans ArtStudio est une instance du processus de référence décrit en partie II (Figure 3.1) :

- Le modèle des tâches n'est pas explicite. C'est un modèle Tâches-Concepts (spécification orientée tâches) référençant déjà les concepts manipulés dans les tâches ;
- Les modèles de l'environnement et d'évolution ne sont pas intégrés ;
- Les modèles de la plate-forme et des interacteurs sont référencés à partir de l'interface abstraite pour la génération de l'interface concrète.

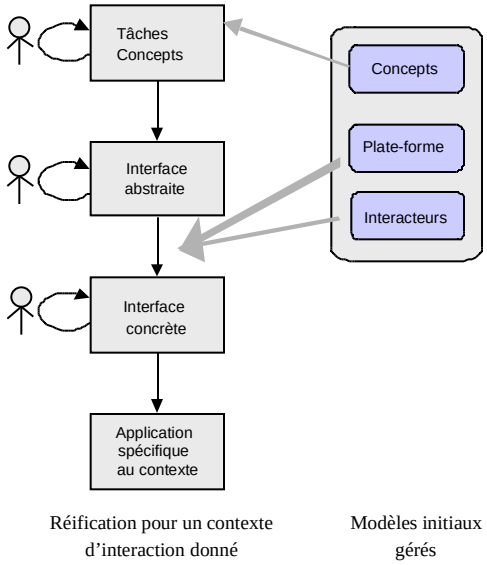



***Figure 3.1 :*** Processus de développement implémenté dans ArtStudio.

D'un point de vue mutli-plates-formes, deux mécanismes sont implémentés (Figure 3.2) :

- Une réification avec point de divergence entre les IHMs abstraite et concrète. C'est le mécanisme mis en œuvre pour la production d'interfaces graphiques Mac/PC et Pilot, dont les interacteurs différent (d'où l'ouverture de la fermeture éclair) ;
- Une traduction par simple copie. C'est le mécanisme mis en œuvre pour créer un brin de réification propre aux plates-formes WAP.

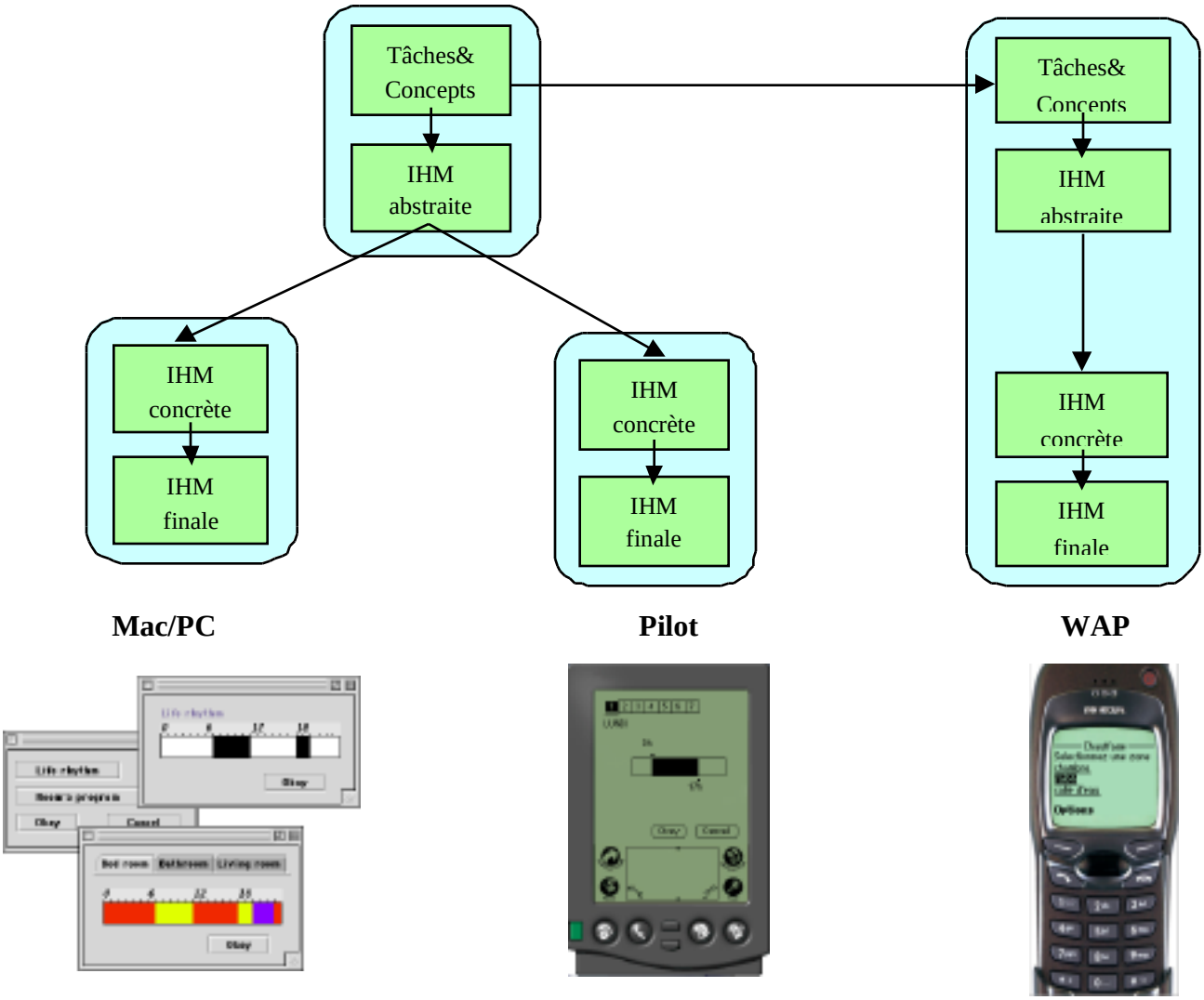



***Figure 3.2 :*** Processus de développement implémenté dans ArtStudio pour la production d'interfaces multi-plates-formes.

PlasticityApp et ArtStudioApp respectent cette instance du processus (Figures 3.1 et 3.2). ArtStudioApp l'organise autour de la notion de projet.

### 3.1.2 Notion de projet

Un projet est une structure qui consigne les fichiers nécessaires à la génération. La figure 3.3 en montre un exemple. Il s'agit d'un extrait du fichier «projets/empty/empty.x ».

```
<projectModels>
  <TaskModel>
   <name>empty</name>
  </TaskModel>
  <ConceptModel>
   <name></name>
  </ConceptModel>
  <AbstractUIModel>
```

***Figure 3.3 :*** Un exemple de projet.

Un projet complet comporte neufs fichiers :

- Le fichier du projet « name.x » ;
- Le fichier de l'arbre de tâches « name.xmx » ;
- Le fichier des concepts « name.xmi » ;
- Le fichier de l'IHM abstraite « name.xaui » ;
- Le fichier de l'IHM concrète « name.xmu » ;
- Le fichier de description de la plate-forme « name.xpf » ;
- Le fichier de description des interacteurs « name.xim » ;
- Ainsi que deux fichiers « name.xmp » et « name.xauip » respectivement nécessaires à ArtStudio pour la présentation graphique du modèle de tâches et de l'IHM abstraite.

A sa création, un projet doit pointer vers un modèle des tâches comportant au moins une tâche. Les autres modèles sont rajoutés au fil de la réification.

Chaque projet est sauvé en un fichier texte au format XML.

### 3.1.3 Modèles et heuristiques

Tous les modèles sont des fichiers texte au format XML. Les DTD sont accessibles par Internet sur le site suivant : http://iihm.imag.fr/thevenin/ProjetX/dtd/v1/. Elles sont également disponible dans le répertoire /dtd/.

#### 3.1.3.1 Concepts

Le modèle des concepts est une spécification UML au format XML. La spécification doit être faite avec ArgoUML (http://argouml.tigris.org/). ArtStudio génère plusieurs fichiers dont un fichier « name.xmi ». C'est ce fichier qui contient la spécification des concepts.

#### 3.1.3.2 Tâches et concepts

Le modèle Tâches-Concepts se base sur la notation ConcurrTaskTree. C'est un modèle des tâches augmenté des concepts manipulés dans les tâches. Les contraintes (conventions) sont les suivantes :

- Un seul concept manipulé par tâche ;
- Tout concept est référencé par son instance (et non sa classe);
- Toute tâche est identifiée par son nom (les tâches doivent donc avoir des noms différents, servant d'identifiant unique).

Le modèle Tâches-Concepts est sauvé dans le fichier « name.xmx ».

**Définition d'une tâche.**
Une tâche est définie par :
· Un nom ;
· Un type (abstraction / interaction) ;
· Le type de l'interaction dans le cas d'une tâche d'interaction (spécification / sélection / …) ;
· Un identificateur (déterminé automatiquement dans ArtStudio) ;
· L'identificateur de la tâche mère (défini automatiquement dans ArtStudio).

On peut éventuellement y rajouter :
· Un prologue ;
· Un épilogue ;
· Le concept manipulé dans la tâche.

C'est dans le prologue et l'épilogue que sont définies les nouvelles instances des concepts et spécifiés les appels au noyau fonctionnel. Les prologues et épilogues sont de la forme :
1 #NomClasse NomInstance# ;
2 #NomClasse NomInstance# = *du code* ;
3 *du code* ;
Une nouvelle instance de concept est déclarée entre '#' '#'. La figure 3.4 en montre un exemple.

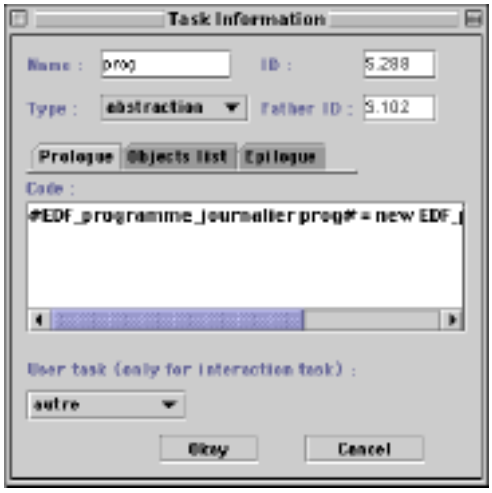



***Figure 3.4 :*** Interface de spécification du modèle Tâches-Concepts.

**Portée des concepts.**
Un concept défini dans une tâche T est visible de toutes les tâches filles de la tâche T. Il peut aussi être visible des tâches sœurs de droite si un opérateur d'enchaînement avec passage de paramètres lie la tâche T à sa sœur de droite. La figure 3.5 schématise cette portée des concepts.

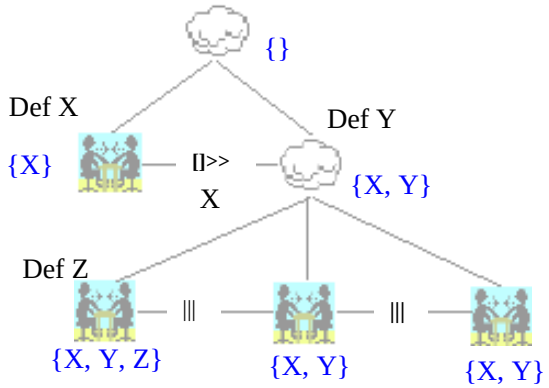



***Figure 3.5 :*** Portée des concepts.

**Enchaînement de tâches.**

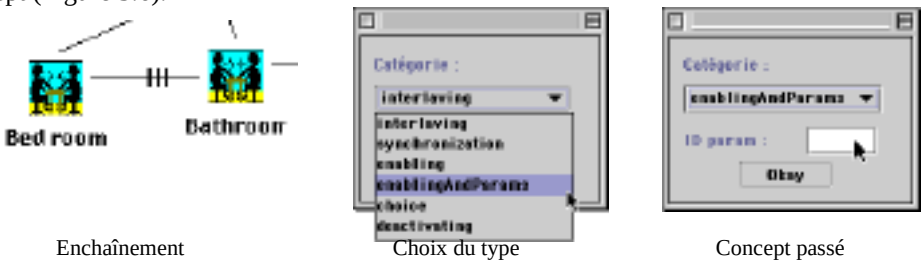
Les tâches sont reliées entre elles par des liens de parenté. Ces liens, de type TaskLink, sont des opérateurs Lotos qui décrivent le type d'enchaînement. Il existe différents types d'enchaînement, dont l'activation avec passage de paramètre qui requiert la spécification du concept ainsi transmis en paramètre. Pour chaque lien, sont à spécifier :

- L'identificateur de la tâche de départ ;
- L'identificateur de la tâche d'arrivée ;
- Le type du lien (opérateur lotos : choix, activation, activation avec paramètres, …) ;
- Et éventuellement l'instance du concept passée en paramètre.

Le lien définit le lien de parenté entre deux tâches soeur. On a évidemment :

- Une tâche T et sa mère ne peuvent être sœurs ;
- Deux tâches qui n'ont pas la même mère ne peuvent pas être sœurs ;
- Si A est sœur de B et B est sœur de C alors A ne peut pas être sœur de C.

Il est donc impossible de créer un lien TaskLink dans ces cas. ArtStudio prévient de ces erreurs en ne créant les liens que lorsque nécessaire et en ne proposant que la spécification de l'opérateur d'enchaînement et de l'instance de l'éventuel concept (Figure 3.6).




Enchaînement

Choix du type D'enchaînement

Concept passé en paramètre

***Figure 3.6 :*** Spécification de l'enchaînement entre tâches..

La figure 3.7 présente l'interface de création et de modification du modèle Tâches-Concepts.

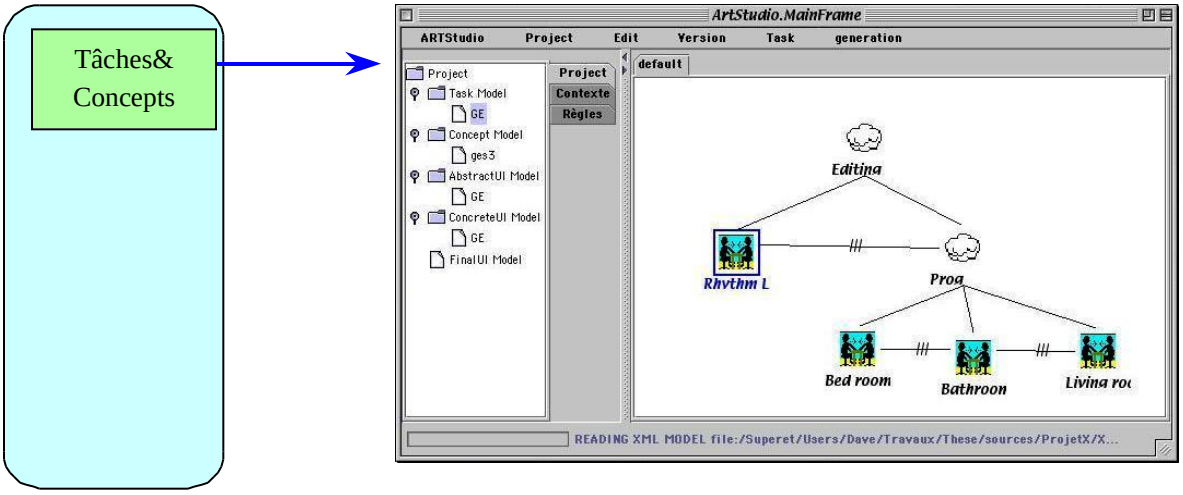



***Figure 3.7 :*** Modèle Tâches-Concepts.

### 3.1.3.3 IHM abstraite

L'IHM abstraite structure l'IHM en espaces de travail et enchaînements entre espaces. Elle est obtenue par réification selon les heuristiques suivantes :

· Association d'un espace de travail final à chaque tâche d'interaction ;

· Association d'un espace de travail à chaque tâche abstraite.

Les opérateurs d'enchaînement ne sont pas encore utilisés à ce stade de la réification.

Ces heuristiques sont appliquées selon un processus en deux étapes (Figure 3.8) :

4. Association des espaces de travail finaux;
2. Construction de la structure des espaces de travail en sous espace de travail.

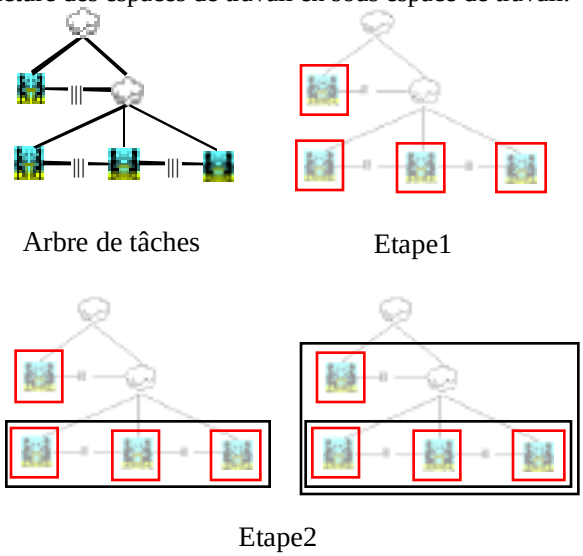



***Figure 3.8 :*** Etapes et heuristiques de génération de l'IHM abstraite.

La figure 3.9 présente l'interface de production de l'IHM abstraite.

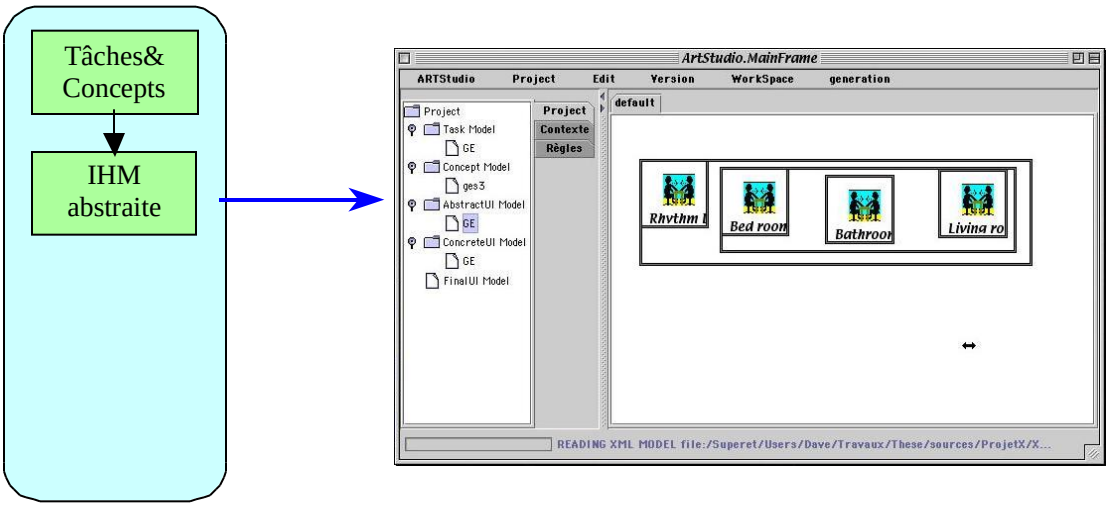



***Figure 3.9 :*** Interface de production de l'IHM abstraite.

#### 3.1.3.4 IHM concrète

L'IHM concrète instancie l'IHM abstraite en fenêtres/canvas et interacteurs. Les heuristiques sont les suivantes :

- Les interacteurs sont choisis en fonction des concepts à représenter et des tâches à réaliser (cf 2.2.1.5). L'algorithme élit le premier interacteur convenable, compte tenu du nombre important d'interfaces possibles ;
- Par défaut le générateur propose une navigation par bouton. Ce choix permet de minimiser (a) le nombre d'interfaces possibles à la génération (un seul type de navigation) et (b) la taille des fenêtres (tout espace de travail étant instancié en fenêtre, il n'y a pas d'imbrication d'espaces de travail). Mais bien entendu, ce choix non réaliste contraint le concepteur à de nombreux remaniements ;
- Le choix entre fenêtres ou canvas découle du choix de l'interacteur de navigation.

L'expérience prouve la nécessité d'un solveur de contraintes puissant. Le solveur actuel est insuffisant : trop sous-contraint (il ne gère pas tous les types de contraintes – en particulier les contraintes géométriques et ergonomiques), il calcule des milliers d'interfaces et rend, par conséquent, nécessaire l'adoption de critères d'arrêt arbitraires tels que "s'arrêter à la première solution".

En pratique :

- Le modèle de l'IHM concrète se décompose en deux sous modèles. Le premier est indépendant du langage de programmation final. Il est utilisé par un compilateur pour produire le deuxième modèle. Ce deuxième modèle est sauvé sur le disque dur, sous la forme d'un fichier texte écrit en Java ou tout autre langage de programmation géré par ArtStudio. Aujourd'hui : Java et Waba ;
- Seules les interactions graphiques sont considérées.

La figure 3.10 montre l'IHM de production de l'IHM concrète.

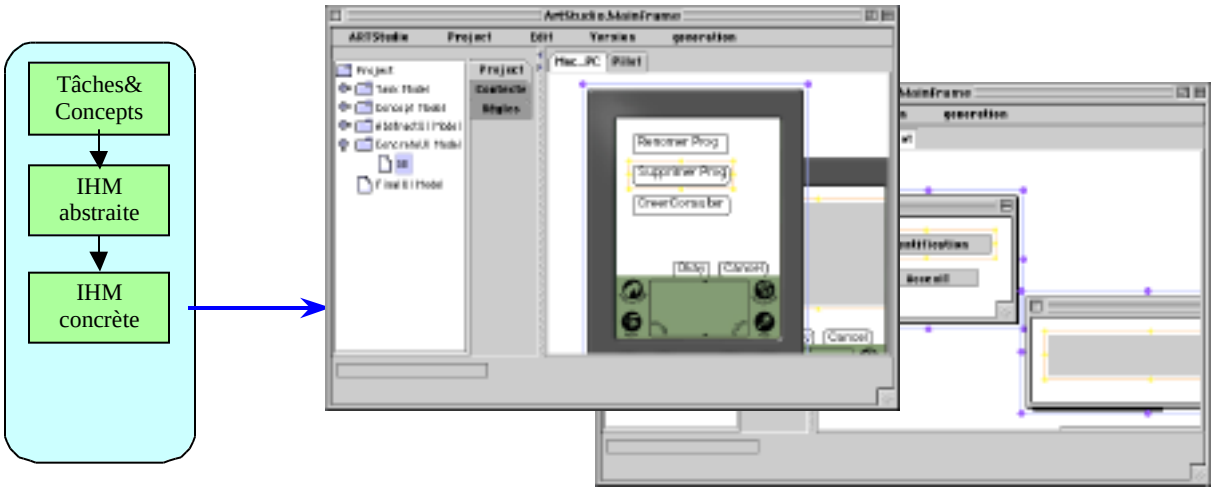



***Figure 3.10 :*** Interface de production de l'IHM concrète.

### 3.1.3.5 Interface finale

Nous présentons ici l'architecture logicielle de l'application puis son implémentation pour la génération du code.

**Architecture logicielle.**

L'architecture logicielle est de type PAC-Amodeus. Chaque agent PAC :

- A une vue sur l'Adaptateur de Noyau Fonctionnel (ANF) à travers les instances des concepts qu'il manipule ;
- A une présentation ;
- A un système de communication entre son père et lui, et l'ANF.

***ANF<->agent***

La communication entre un agent et l'ANF a deux rôles :

- Modifier le Noyau Fonctionnel (NF) ou lancer une tâche système suite à une interaction utilisateur (agent -> ANF) ;
- Changer l'état de l'interface suite à un changement du NF (ANF -> Agent).

Dans le premier cas, l'agent agit sur l'ANF soit par l'appel des méthodes spécifiées dans le prologue ou l'épilogue, soit en modifiant directement les objets manipulés par l'agent.

Dans le deuxième cas, l'ANF communique avec un agent en utilisant la méthode notifyMethode. En paramètre de cette méthode est passé l'objet du NF dont l'état a été modifié. Pour qu'un agent puisse être appelé par l'ANF, il est nécessaire qu'il s'abonne à ce type d'événement (méthode « registry » présente dans l'objet TObject dont tout concept hérite).

***Agent->Agent***

Un agent peut communiquer avec son père pour l'informer qu'une action vient d'être réalisée. Ces actions sont du type – Validation, Annulation, …– ou non typée. L'agent implémente pour cela une méthode « actionPerformed » qui reçoit en paramètre un identifiant de l'agent et un message.

**Production du code.**

Un agent est réifié en une classe qui est une spécialisation d'une fenêtre ou d'un canva.

- En Java, une fenêtre est un « Interactors.XframeAndValidation » et un canvas un « Interactors.Xpanel ».
- En Waba, une fenêtre est une « WInteractors.XframeAndValidation » et un canvas un « WInteractors.Xpanel ».

Pour chaque classe générée :

- Deux constructeurs sont crées :
  1- Le premier avec comme seul argument, l'instance de l'agent père.
  2- Le second avec comme arguments toutes les instances des concepts externes dont la portée touche cet agent (cf spécification du modèle des tâches/Concepts).
- Des méthodes sont définies :
  1. les méthodes définissant les instances des concepts externes ;
  2. les méthodes de communication « notifyMethode » et « actionPerformed » ;
  3. les prologue et épilogue.

Lors de la génération de l'interface finale, sont produits autant de fichiers que de plates-formes cibles.

#### 3.1.3.6 Interacteurs

Seuls les interacteurs graphiques sont aujourd'hui traités. Pour chaque interacteur sont spécifiés :

- Un nom ;
- Une surface d'affichage ;
- Les concepts qu'il est capable de représenter ;
- Les tâches qu'il implémente ;
- Son type *visualisation* ou *navigation*.

La figure 3.11 en présente un exemple :

```
<GraficInteractor>
   <name>XComboBox</name>
   <width value="null"/>
   <height value="20"/>
   <data list = "{TString,TStringFromTVector}"/>
   <elementaryTask list = "{selectionner}"/>
   <type value="visualisation"/>
 </GraficInteractor>
```

***Figure 3.11 :*** Spécification d'un interacteur.

#### 3.1.3.7 Plate-forme

Aujourd'hui seuls le langage d'interaction cible et la surface d'affichage sont spécifiés. La figure 3.12 en présente un exemple.

```
<Container id="S.103">
   <versionName value="Pilot"/>
   <predVersionID id="S.101"/>
   <subVersionIDs listID="{}"/>
     <containerElement>
       <Language value="Waba"/>
       <Ressources>
         <screenWidth value="160"/>
         <screenHeight value="160"/>
         <screenResolution value="1"/>
    </Ressources>
  </containerElement>
 </Container>
```

***Figure 3.12 :*** Spécification d'une plate-forme.

## 3.2 Logique d'utilisation

Nous décrivons ici les logiques d'utilisation de PlasticityApp et ArtStudioApp.

### 3.2.1 PlasticityApp

La commande d'exécution de l'application est la suivante :
java –cp Classes/ArgoUML.jar ;Classes/Concepts.jar ;Classes/Jess6.jar ;Classes/xml4j-2.jar plasticityGen projectName.x -acf –s

où projectName peut prendre comme valeurs :
· /ptittet/Users/thevenin/projets/GE/GE.x
· ou GE/GE.x, etc.

Les options sont les suivantes :
· -a : génération de l'IHM abstraite ;
· -c : génération de l'IHM concrète;
· -f : génération de l'IHM finale ;
· -ac : génération des IHM abstraite et concrète ;
· -cf : génération des IHM concrète et finale ;
· -acf : génération des IHM abstraite, concrète et finale ;
· -s : sauvegarde le résultat (attention efface alors l'ancien projet).

### 3.2.2 ArtStudioApp

La commande d'exécution de l'application est la suivante :
java –cp Classes/ArtStudio.jar ;Classes/Plasticity.jar;Classes/ArgoUML.jar;Classes/Jess6.jar;Classes/xml4j-2.jar;Classes/Concepts.jar;Classes/ConceptsEDFJDK.jar;Classes/ConceptsEDFWaba.jar;Classes/Interactors.jar;Classes/InteractorsWaba.jar;Classes/EDFInteractors.jar;Classes/EDFInteractorsWaba.jar;Classes/WabaBiz.jar ArtStudioApp

## 4 CONCLUSION

L'approche basée modèles que nous proposons sert de cadre de pensée au concepteur pour la conception d'interfaces multicontextes :

- elle identifie des modèles-clé ;
- et discute de leur occurrence dans le processus de développement.

Un méta-processus, donnant lieu à différentes instanciations, synthétise ce cadre de pensée. Parmi elles, la Fermeture Eclair qui émerge pour la capitalisation des connaissances qu'elle favorise. Cette capitalisation des connaissances est intéressante pour EDF dont :

- le métier est clairement détouré et stable ;
- la sous-traitance logicielle fréquente ;
- la diversité des plates-formes croissante.

Cette capitalisation est stratégique à plusieurs titres :

- pour la qualité logicielle qu'elle favorise (cohérence inter-versions, fiabilité, etc.) ;
- pour la réutilisabilité qu'elle procure ;
- et, en conséquence, pour les coûts de développement et de maintenance réduits.

D'un point de vue outillage, ArtStudio est intéressant pour l'identification des espaces de travail et enchaînements entre espaces. Son utilisation par EDF requiert, par contre, la mise en œuvre d'un convertisseur traduisant une spécification Prospect en spécification ConcurrTaskTree. L'assistance pour le choix des interacteurs est de la même façon intéressante. Mais, la qualité des maquettes attendues renforcera l'aspect manuel de la génération lors de la production d'interfaces concrètes.

## 5 BIBLIOGRAPHIE


1. Abowd, Gregory D., Atkeson, Christopher G., Hong, Jason, Long, Sue, Kooper, Rob, Pinkerton, Mike Cyberguide: A Mobile Context-Aware Tour Guide, In *GVU Technical Report GIT-GVU-96-27*, December 1996.
2. Bérard, F., Coutaz, J., Crowley, J.L. Le tableau Magique : un outil pour l'activité de réflexion, *Actes de la conférence ErgoIHM'2000* (3-6 octobre 2000, Biarritz, France), pp. 33-40.
3. Calvary, G. Proactivité et réactivité : de l'Assignation à la Complémentarité en Conception et Evaluation d'Interfaces Homme-Machine, *Thèse de l'Université Joseph-Fourier-Grenoble I, Spécialité Informatique*, Octobre 1998, 250 pages.
4. Calvary, G., Coutaz, J., Thevenin, D. A Unifying Reference Framework for the Development of Plastic User Interfaces, A paraître dans *EHCI'2001*.
5. Clarkson, Brian, Sawhney, Nitin, Pentland, Alex Auditory Context Awareness via Wearable Computing, (1998) *Proceedings of the Perceptual User Interfaces Workshop*, San Francisco, CA.
6. Crowley, J.L., Coutaz, J., Bérard F. Things That See, *Communication of the ACM (CACM)* Vol 43 (3), March 2000, pp. 54-64.
7. Dey, Anind K., Salber, Daniel, Futakawa, Masayasu, Abowd, Gregory D. An Architecture To Support Context-Aware Applications, *GVU Technical Report GIT-GVU-99-23*. June 1999.
8. Dey, Anind K., Abowd, Gregory D. Towards a Better Understanding of Context and Context-Awareness, *Proceedings of the CHI 2000 Workshop on The What, Who, Where, When, and How of Context-Awareness*, The Hague, Netherlands, April 1-6, 2000.
9. Gram, C., Cockton, G. Ed., *Design Principles for Interactive Software*. Chapman & Hall, 1996.
10. Harter, Andy, Hopper, Andy, Steggles, Pete, Ward, Andy, Webster, Paul The anatomy of a context-aware application (p. 59-68), *Proceedings of the fifth annual ACM/IEEE international conference on Mobile computing and networking* , August 15 - 19, 1999, Seattle, WA USA.
11. Lee, Jay, Su, Victor, Ren, Sandia, Ishii, Hiroshi HandSCAPE: A Vectorizing Tape Measure for On-Site Measuring Applications, *Videos of CHI'2000*, April 2000.
12. Anabuki, Mahoro, Kabuka, Hiroyuki, Yamamoto, Hiroyuki, Tamura, Hideyuki Welbo : An embodied Conversational Agent Living in Mixed Reality Space, *Videos of CHI'2000.*
13. Orr, Robert J., Abowd, Gregory D. The Smart Floor: A Mechanism for Natural User Identification and Tracking. *GVU Technical Report GIT-GVU-00-02*. January 2000.
14. Rekimoto, Jun Multiple Computer User Interfaces : "Beyond the desktop", Direct Manipulation Environments, *Videos of CHI'2000*, April 2000.
15. Salber, D., Abowd, Gregory D. The Design and Use of a Generic Context Server, In the *Proceedings of the Perceptual User Interfaces Workshop (PUI '98)*, San Francisco, CA, November 5-6, 1998. pp. 63-66.
16. Salber, D., Dey, A.D., Abowd, Gregory D. The Context Toolkit: Aiding the Development of Context-Enabled Applications, In the Proceedings of the 1999 Conference on Human Factors in Computing Systems (CHI '99), Pittsburgh, PA, May 15-20, 1999, pp. 434-441.
17. Schilit, B.N., Adams, N.I., Want, R. Context-Aware Computing Applications. In *Proc. IEEE Wkshp. Mobile Computing Systems and Applications*, IEEE, Santa Cruz CA, December 1994.

18. Schmidt, Albrecht, Asante Aidoo, Kofi, Takaluoma, Antti, Tuomela, Urpo, Van Laerhoven, Kristof, Van de Velde, Walter Advanced Interaction in Context. *1th International Symposium on Handheld and Ubiquitous Computing (HUC99)*, Karlsruhe, Germany, 1999 & Lecture notes in computer science; Vol 1707, ISBN 3-540-66550-1; Springer, 1999, pp 89-101.

19. Schmidt, A. Implicit human-computer interaction through context. *2th Workshop on Human Computer Interaction with Mobile Devices*. Edinburgh, Scotland, 31 August 1999.

20. Thevenin, D., Coutaz, J. Plasticity of User Interfaces: Framework and Research Agenda. In *Proceedings of INTERACT'99*, 1999, pp. 110-117.

21. Yan, Hao, Selker, Ted Context-Aware Office Assistant, *IUI 2000*, New-Orleans LA USA, pp 276-279.

# 6 ANNEXES

## 6.1 Annexe A : Notion de contexte

### 6.1.1 Définition

Etant donnés un utilisateur U, une tâche T et un instant t, nous définissons la notion de contexte comme étant le cumul des situations rencontrées, entre les instants 0 et t, pour la réalisation de la tâche T, par l'utilisateur U.

Dans cette définition, la notion de situation est définie comme suit :
Etant donnés un utilisateur U, une tâche T et un instant t, la situation à l'instant t est l'ensemble des états des variables et relations entre variables qui sont périphériques à la tâche T mais qui peuvent l'influencer, à cet instant t ou plus tard.

Dans cette définition :

- les variables sont de deux ordres : environnement physique et environnement social. L'environnement physique comprend : les plates-formes d'interaction et leur voisinage. Ce voisinage est défini par un ensemble d'entités, non exhaustivement répertoriées. Par exemple :
  - Le lieu géographique ( (x, y, z), (latitude, longitude, altitude), symbolique (ex : bureau / domicile ce qui correspond à des classes de lieux));
  - Les objets physiques présents (tables, chaises, crayons, etc.);
  - L'ambiance (lumière, son, etc.).

  L'environnement social se réfère aux personnes présentes dans l'environnement physique.
- La notion de périphérie se réfère aux informations non centrales à la tâche, mais susceptibles d'en influencer la réalisation. L'influence peut consister en :
  - Une interruption (puis reprise);
  - Un abandon;
  - Une poursuite avec modification des propriétés nominales attendues. Par exemple : le temps de réalisation ou la qualité des résultats. La modification peut être une augmentation ou une diminution des valeurs escomptées.

  Aujourd'hui, typiquement, ces variables périphériques ne sont pas référencées dans le modèle de tâches. En plasticité, ce sont des informations de seconde classe.

Remarques : la définition du contexte est valide quelque soient

- l'épaisseur du temps (δt) ;
- la granularité de la tâche T ;
- le nombre d'utilisateurs.

### 6.1.2 Classification

On distingue deux types de situations, et par conséquence, deux types de contextes : les situations/contextes bruts et les situations/contextes nets.

- Le brut se réfère aux situations/contextes factuels, indépendamment de l'exploitation qui en est (ou en sera) faite par l'utilisateur et/ou le système;
- Par opposition, le net se restreint à la partie du brut exploitée ou exploitable par l'utilisateur et/ou le système.

Dans cette définition :

- L'exploité se réfère à la partie du brut effectivement exploitée à l'exécution par l'utilisateur et/ou le système ;
- l'exploitable se réfère aux informations identifiées lors de la conception comme étant potentiellement exploitables par l'utilisateur et/ou le système.

On distingue le net du point de vue système, utilisateur et couple système/utilisateur :

- Le net du point de vue système correspond à l'exploité/exploitable du brut par le système. On le dira capté / captable ;
- Le net du point de vue utilisateur correspond à l'exploité/exploitable du brut par l'utilisateur. On le dira utilisé / utilisable;
- Le net du point de vue utilisateur et système correspond à l'exploité/exploitable du brut par l'utilisateur et le système. On le dira partagé / partageable. En notation ensembliste (cf figure 1), ce net utilisateur/système est l'intersection entre le net système et le net utilisateur.

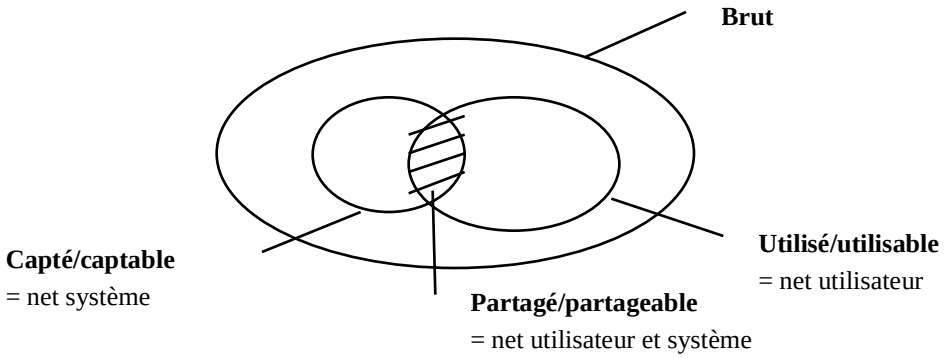



Figure 1 : Brut, net système, net utilisateur, net système et utilisateur.

Cette caractérisation est appliquable :

- A la conception et à l'exécution de l'application. En conception, on parlera de *captable*, *utilisable* et *partageable*. A l'exécution, on parlera de *capté*, *utilisé* et *partagé*. Les relations d'inclusion sont à étudier selon ces deux dimensions.
- Aux situations et aux contextes.

### 6.1.3 Formalisation

Notons situation $_{U,T}$ (t), la situation rencontrée par l'utilisateur U, réalisant la tâche T, à l'instant t.
Notons contexte $_{U,T}$ (t), le contexte de l'utilisateur U, réalisant la tâche T, à l'instant t.

Indiçons par B les situations et contextes bruts.

Par définition, on a :

- contexte $_{U,T,B}$ (0) = situation $_{U,T,B}$ (0)
- t>0, contexte $_{U,T,B}$ (t) = situation $_{U,T,B}$ (t) + contexte $_{U,T,B}$ (t-1).

Dans cette équation, le '+' est à valeur ensembliste. Il correspond à une union entre la situation courante (une photo à l'instant t des variables et relations entre variables) et l'historique des situations précédentes (le contexte à l'instant t-1).

D'où :

$$t \geq 0,\ contexte_{U,T,B}(t) = situation_{U,T,B}(0) + \int_0^t situation_{U,T,B}(x).dx$$

L'utilisé/utilisable et le capté/captable peuvent être vus comme des filtres $f_U$ et $f_C$ appliqués à ces valeurs brutes, à un instant t. Ainsi :

- $t \geq 0,\ contexte_{U,T,U}(t) = f_U(contexte_{U,T,B}(t), t)$
- $t \geq 0,\ contexte_{U,T,C}(t) = f_C(contexte_{U,T,B}(t), t)$.

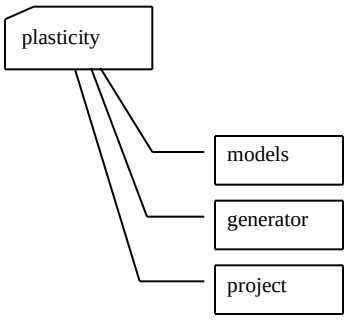
### 6.1.4 Changement de contexte

En plasticité, on aura changement de contexte si, entre deux instants t1 et t2, la variation du contexte partagé/partageable est supérieure à un certain seuil. Ce seuil est fonction des variables considérées.

## 6.2 Annexe B : Compléments sur l'implémentation d'ArtStudio

### 6.2.1 Implémentation

Le package plasticity se décompose en trois sous-packages :




#### 6.2.1.1 Le package « models »

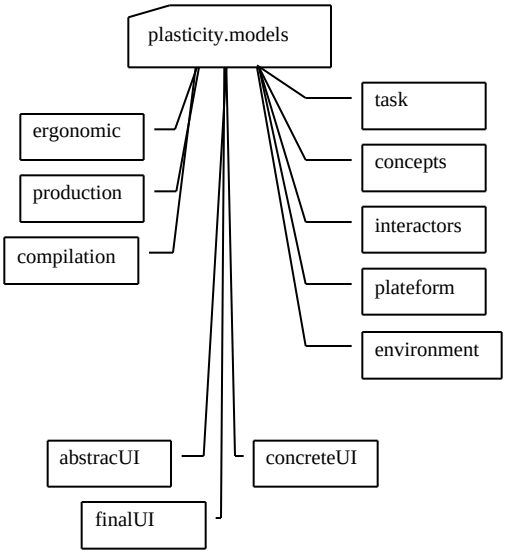
Les modèles implémentés dans ArtStudio sont au nombre de 11 :

1. le modèle des tâches (plasticity.models.task.TaskModel) ;
2. le modèle des concepts (plasticity.models.task.ConceptModel) ;
3. le modèle des interacteurs (plasticity.models.task.InteractorModel) ;
4. le modèle de la plate-forme (plasticity.models.task.InteractorModel) ;
5. le modèle de l'environnement (plasticity.models.task.EnvironmentModel) ;
6. le modèle des règles de production (plasticity.models.task.ProductionModel) ;
7. le modèle des règles ergonomiques (plasticity.models.task.ErgonomicModel) ;
8. le modèle des règles de compilation (plasticity.models.task.CompilationModel) ;
9. le modèle de l'IHM abstraite (plasticity.models.task.AbstractUIModel) ;
10. le modèle de l'IHM concrète (plasticity.models.task.ConcreteUIModel) ;
11. et le modèle de l'IHM finale (plasticity.models.task.FinalUIModel).

Un douzième modèle existe. C'est le modèle du projet. Il décrit un projet, c'est-à-dire qu'il fait le lien sur les onze modèles.

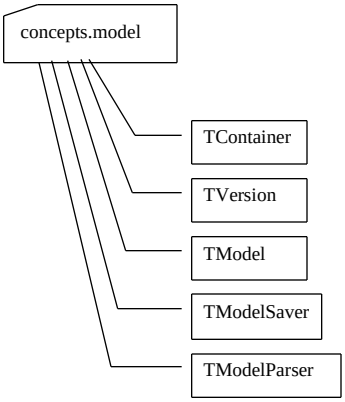
Il est à noter que les modèles 5 à 8 ne sont pas utilisés :

- L'environnement n'est en effet pas modélisé dans l'état actuel des travaux ;
- Les règles de production sont codés dans des fichiers à part (*.clp) qui sont utilisés par le système expert Jess. Actuellement nous avons deux fichiers :
  – jess/AbstractUIGen.clp
  – jess/ConcreteUIGen.clp
  Ils contiennent respectivement les règles permettant la production des IHM abstraite et concrète ;
- Aucune gestion de règles ergonomiques n'est pour le moment intégrée dans ArtStudio ;

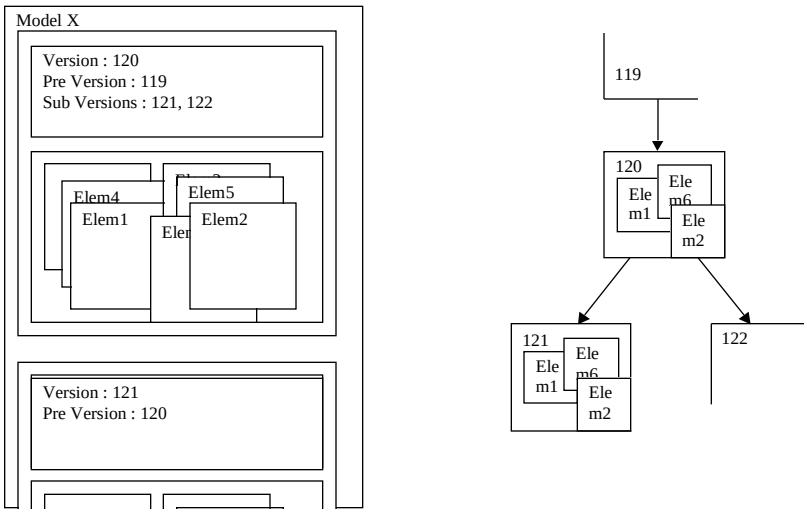



**Un modèle.**

Tout modèle ArtStudio est une spécialisation du modèle décrit dans le package concept.model.
Ce modèle est composé de cinq classes : TModel, TModelParser, TModelSaver, TContainer, TVersion.




Un modèle est intrinsèquement multiversions. Ainsi il se décompose en sous-modèles appelés « TContainer » . Chaque TContainer pointe sur une structure TVersion et sur une table de hachage qui contient les éléments décrivant le modèle.




Chaque modèle peut implémenter une classe de sauvegarde et/ou de lecture. Il hérite alors des classes TModelSaver et TModelParser, qui respectivement sauvegardent et lisent un modèle dans fichier au format TXT. Ce fichier est sauvé sous la forme d'un langage de type XML.
Nous montrons maintenant comment utiliser les classes TModel, TModelSaver et TModelParser pour implémenter un modèle pour la plasticité.

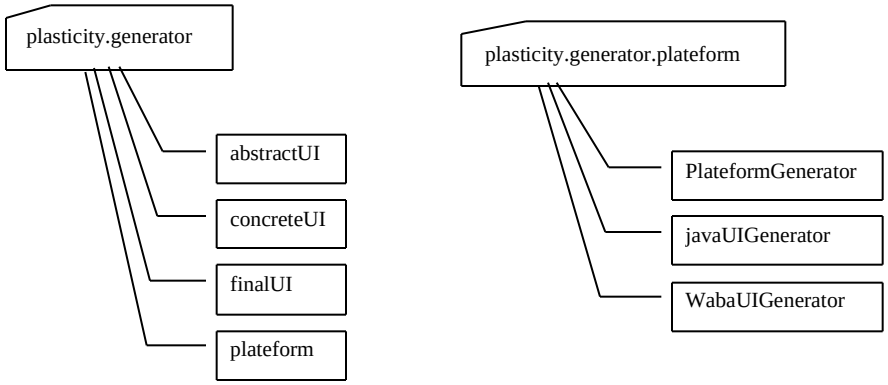
**TmodelSaver.**
TModelSaver est une classe abstraite, implémentant la sauvegarde d'un modèle ArtStudio (TModel). Le fichier est sauvé au format texte, en utilisant une structure de type XML.
Implémenter sa propre classe de sauvegarde revient à créer une classe MyModelSaver qui hérite de TModelSaver et qui implémente la méthode « writeData ». Cette méthode reçoit en paramètre le TContainer à sauver. Elle écrit directement dans le flux de sortie « **outStream ».** Ca peut être une socket, un fichier … (dans ArtStudio c'est un fichier). Voir en fin d'annexe un exemple d'utilisation de TModelSaver.

**TmodelParser.**
TModelParser est une classe abstraite, implémentant la lecture d'un modèle ArtStudio (TModel). Le fichier doit être au format texte, en utilisant une structure de type XML.
Implémenter sa propre classe de lecture revient à créer une classe MyModelParser qui hérite de TModelParser et qui implémente les méthodes « handleStartTag » et « handleEndTag ».Ces méthodes sont appelées respectivement au début et à la fin de chaque tag XML. Elles reçoivent en paramètre les éléments codés dans le tag.
Par exemple en écrivant : <MyElem id="120"> handleStartTag reçoit une structure contenant le nom du tag (MyElem), le non de l'attribut (id) et sa valeur (120). Voir en annexe un exemple d'utilisation de TModelParser.

**Tmodel.**
TModel est une classe abstraite. Implémenter un TModel nécessite la création d'une classe (MyModel) héritant de TModel et implémentant les méthodes getNewContainer, loadFromFile et saveToFile.
Voir en fin d'annexe un exemple d'utilisation de TModel.

#### 6.2.1.2 Le package « generator»

Le package des générateurs (plasticity.generator) se divise en trois sous packages. Respectivement pour la génération de l'IHM abstraite (plasticity.generator.abstractUI), la génération de l'IHM concrète (plasticity.generator.concreteUI) et de l'IHM finale (plasticity.generator.finalUI).




Le dernier package contient les compilateurs spécifiques à chaque plate-forme (plasticity.generator.plateform). Ils traduisent l'IHM finale en code source compilable.

Aujourd'hui les plates-formes Waba et Java/JFC sont supportées. C'est le « PlateformGenerator » qui utilise le générateur adéquat pour produire le code final.

Les générateurs représentent le cœur du moteur d'inférence pour la réification. Cette inférence est implémentée selon deux méthodes :

- Soit impérative : la réification est codée en java ;
- Soit déclarative : la réfication est codée sous forme de règles qui sont utilisées par le système expert Jess pour produire les modèles ;
- Soit hybride : en utilisant les deux méthodes précédente en même temps.

**Génération de l'IHM abstraite.**

Ce générateur est codé dans la classe « plasticity.generator.abstractUI.AbstractUIGenerator ».
C'est un générateur purement déclaratif. L'objet initialise le système expert puis lance l'inférence. Pour changer ou rajouter des règles de réification, il faut modifier le fichier « genAbstractUI.clp » qui se trouve dans le répertoire « jess » à la racine des exécutables.
La première partie du fichier « genAbstractUI.clp » déclare les classes manipulées par les règles de génération, les patrons des assertions et les fonctions transformant l'arbre de tâches en un ensemble d'assertions.
La deuxième partie du fichier déclare les règles de génération. Se reporter au fichier pour plus d'information.

**Génération de l'IHM concrète.**

Ce générateur est codé dans la classe « plasticity.generator.concreteUI. GenerateConcretUI». C'est un générateur hybride.

GenerateInstance :
Génère la table de toutes les instances de concepts dans l'arbres de tâches, et leur porté. Pour chaque tâche, elle détermine les concepts externes

**Génération de l'IHM finale.**

Ce générateur est codé dans la classe « plasticity.generator.finalUI. GenerateFinalUI». C'est un générateur purement impératif.

### 6.2.2 Exemple d'utilisation de TModelSaver

```
import concepts.*;
import concepts.model.*;

import java.util.*;
import java.io.OutputStream.*;

public class MyModelSaver extends TModelSaver
{
	public MyModelSaver (MyModel model)
	{
		super (model);
	}

	protected void writeData (TContainer container)
	{
		MyElem elem ;
		Enumeration enumElem ;

		// Récupération de la liste des éléments contenus dans le container
		Hashtable listElems = (MyModel)theModel).getElems (container);

		// Enumération des éléments
		for (enumElem  = listElems.getElements () ; enumElem.hasMoreElements () ;)
		{
			elem (MyElem) enumElem.nextElement ();

			// Ecriture d'un élément
			outStream.println ("<MyElem id=\"" + elem.getID () + "\">");
			outStream.println ("	<name>" + elem.getName () + "</name>");
			outStream.println ("</MyElem > ") ;
		}
	}
}
```

Instanciation et utilisation de la classe :

```
MyModelSaver saver = new MyModelSaver (myModel);
saver.save (pathDir);
```

### 6.2.3 Exemple d'utilisation de TModelParser

```
import concepts.*;
import concepts.model.*;

import java.util.*;
import com.ibm.xml.parser.*;

public class MyModelParser extends TModelParser
{
	MyElem currentElem;

	public MyModelParser ()
	{
		super ();
	}

	public MyModelParser (MyModel model)
	{
		super (model);
	}

	public void handleStartTag (TXElement e, boolean empty)
	{
		// Appel du handleStartTag pere.
		super.handleStartTag (e, empty);
		String n = e.getName();

		// Gestion des tags XML
		 if (n.equals ("MyElem")) MyElemHandelEnd (e);
	}

	public void handleEndTag (TXElement e, boolean empty)
	{
		// Appel du handleStartTag père.
		super.handleEndTag (e, empty);
		String n = e.getName();

		// Gestion des tags XML
		if (n.equals ("MyElem")) MyElemHandel (e);
	}

	 protected void MyElemHandel (TXElement e)
	{
		// Instanciation d'un MyElem
		 currentElem = new MyElem () ;
```

```
                // Initialisation de l'élément
                currentElem.setID (e. getAttribute ("id") ) ;
        }

        protected void MyElemHandelEnd (TXElement e)
        {
                // A la fin du tag, l'élément est ajouté au modèle
                ((MyModel) theModel).addElem (currentElem, m_currentContainer);
        }
}
```

Instanciation et utilisation de la classe :

```
MyModelParser parser = new MyModelParser (myModel);
parser.parse (pathDir);
```

### 6.2.4 Exemple d'utilisation de TModel

```
import concepts.*;
import concepts.model.*;

import java.util.*;

public class MyModel extends TModel
{
        public MyModel ()
        {
                super ();
        }

        public MyModel (TString name)
        {
                super (name);
        }

        public TContainer getNewContainer ()
        {
                // Création du container
                TContainer result = super.getNewContainer ();

                // Création des éléments contenus dans le container.
                // Dans cet exemple, le container contient une table de hachage
                // appelée « listElem »
```

```
		result.elements.put ("listElem", new Hashtable ());

		return result;
	}

	public boolean loadFromFile (TString pathDir)
	{
		// supression de tous les container dans le model
		removeAllContainer ();

		// Instanciation du parser
		MyModelParser loader = new MyModelParser (this);

		// Lecture
		loader.parse (pathDir);

		return true;
	}

	public boolean saveToFile (TString pathDir)
	{
		// Instanciation du sauveur
		MyModelSaver saver = new MyModelSaver (this);

		// Sauvegarde
		saver.save (pathDir);

		return true;
	}

	public void clear ()
	{
		super.clear ();
		removeAllContainer ();
	}

	public void addElem (MyElem elem, TContainer container)
	{
		// Ajout d'un élément à un container

		// Récuperation de la table de hachage contenant les éléments (« listElem »)
		Hashtable listElem = container.elements.get ("listElem") ;

		 // Ajout dans cette table de l'élément.
		// Chaque élément a une ID qui lui sert d'identifiant unique et
		// ainsi de clef dans les tables de hachage.
		listElem .put (elem.getID (),elem);
```

```
        }

        public void addElem (MyElem elem)
        {
                // Ajout d'un élément au container courrant
                return addElem (elem, currentContainer) ;
        }

        public Hashtable getElems (TContainer container)
        {
                // Retourne les  éléments contenus dans le container
                return container.elements.get ("listElem") ;
        }

        public Hashtable getElems ()
        {
                // retourne les éléments du container courrant
                return getElems (currentContainer) ;
        }
}
```

### 6.3 Annexe C : Liste des publications de l'équipe sur le sujet

Conférences internationales

Thevenin, D., Coutaz, J. Plasticity of User Interfaces: Framework and Research Agenda. In *Proceedings of INTERACT'99*, 1999, pp. 110-117.

Calvary, G., Coutaz, J., Thevenin, D. Embedding Plasticity in the Development Process of Interactive Systems, in the Proceedings of the 6th ERCIM Workshop on "User Interfaces for All", October, 2000

Calvary, G., Coutaz, J., Thevenin, D. A Unifying Reference Framework for the Development of Plastic User Interfaces. EHCI'01, IFIP WG2.7 (13.2) Working Conference, Toronto, May 2001.

Workshops internationaux à comité de sélection

Coutaz, J., Calvary, G. The Plasticity of User Interfaces, the Disappearing Computer and Situated Computing, Workshop on Research Directions in Situated Computing,, M. Beaudouin-Lafon, W. Mackay, Extended Abstracts, CHI2000, pp 369-369

Calvary, G., Coutaz, J., Thevenin, D. Embedding Plasticity in the Development Process of Interactive Systems, HUC (Handheld and Ubiquitous Computing) First workshop on Resource Sensitive Mobile HCI, Conference on Handheld and Ubiquitous Computing, HU2K, Bristol, September, 2000. Presented also at UI4All, Florence, 6$^{th}$ ERCIM workshop.

Coutaz, J., Lachenal, C., Calvary, G., Thevenin, D. Software Architecture Adaptivity for Multisurface Interaction and Plasticity. IFIP WG2.7 workshop on Software Architecture requirements for CSCW, CSCW2000 workshop, Philadelphia.

Thevenin, D., Calvary, G., Coutaz, J. A Development Process for Plastic User Interfaces, CHI'2001.

Workshops nationaux à comité de sélection

Thevenin, D., Calvary, G., Coutaz, J. Plasticité en Interaction Homme-Machine, Premières Rencontres Jeunes Chercheurs en IHM, RJC-IHM 2000, Vannes, 3-5 Mai 2000, pp. 71-74.

Thevenin, D., Calvary, G., Coutaz, J. La Multimodalité en Plasticité, Actes du colloque sur les interfaces multimodales, 9-10 Mai 2000, Grenoble, pp. 55-58