Construction d'interface de saisie d'un problème de dénombrement

Résumé

Ce rapport présente une implémentation d'une interface pour un environnement d'aide à l'apprentissage, dans le cadre du projet COMBIEN?. Cette interface permet à un élève de construire des représentations modélisées d'exercices de dénombrement, à partir d'énoncés donnés en langue naturelle. Les représentations sont fondées sur un modèle conceptuel informatique du domaine défini par le groupe Combien, élaboré aussi bien pour conduire des raisonnements mathématiques que pour communiquer avec l'élève sous une forme proche de ses formulations habituelles. Le rapport décrit l'interface et le modèle conceptuel et discute des choix et des problèmes d'implémentation.

Mots clés : interface, environnement d'apprentissage, dénombrement, EIAO.

Abstract

This report presents an implementation of an interface for a learning environment, in the context of the COMBIEN? Project. This interface allows a student to build modelled representations of combinatorics exercices, starting from texts in natural language. Representations are based on a conceptual model of the domain, which was defined by the Combien? group to execute mathematical reasoning as well as to communicate with the student in a form close to his usual formulations. The report describes the interface and the conceptual model and discusses implementation choices and problems.

Key words : interface, learning environment, combinatorics, ICAL.

Table des matières

1. Introduction

Ce rapport présente une implémentation d'une interface pour un environnement d'aide à l'apprentissage, dans le cadre du projet COMBIEN. Cette interface permet à un élève de construire des représentations modélisées d'exercices de dénombrement, à partir d'énoncés donnés en langue naturelle. Les représentations sont fondées sur un modèle conceptuel informatique du domaine défini par le groupe COMBIEN, élaboré aussi bien pour conduire des raisonnements mathématiques que pour communiquer avec l'élève sous une forme proche de ses formulations habituelles. Le modèle conceptuel est lui-même fondé sur une formalisation mathématique d'une méthode de résolution de problèmes de dénombrement, la méthode constructive [Tisseau & al 96]. Cette méthode consiste à énoncer une construction algorithmique non déterministe d'un ensemble qu'on veut dénombrer puis à raisonner sur cette construction pour effectuer le dénombrement. L'objectif de l'environnement d'apprentissage est de familiariser l'élève avec cette méthode.

L'implémentation a été effectuée par Maria Annell et Virginia Malthet pour leur projet de DESS IA à Paris 6 sous la direction de Hélène Giroire et de Gérard Tisseau. Le rapport décrit l'interface réalisée en s'appuyant d'abord sur un exemple puis en dégageant les principes pédagogiques et ergonomiques mis en œuvre. Il rappelle ensuite le modèle conceptuel utilisé et l'illustre avec des exemples, puis indique comment ce modèle a été implémenté en Smalltalk (version VisualWorks). Il décrit la réalisation de l'interface en discutant les choix effectués. Finalement, on analyse les extensions souhaitables et les problèmes qui restent à résoudre.

2. Interface

Dans ce chapitre, nous allons montrer un exemple de modélisation d'un exercice, en suivant l'enchaînement des fenêtres et sous-fenêtres. Evidemment, toutes les sous-fenêtres existantes ne seront pas montrées ici, seulement celles utilisées dans cet exercice type.

2.1. Exemple de parcours

2.1.1. Fonctionnalités globales

On démarre le programme de modélisation en cliquant sur l'icône suivante dans la barre des tâches de Smalltalk : , ce qui lance l'interface de la figure 1.

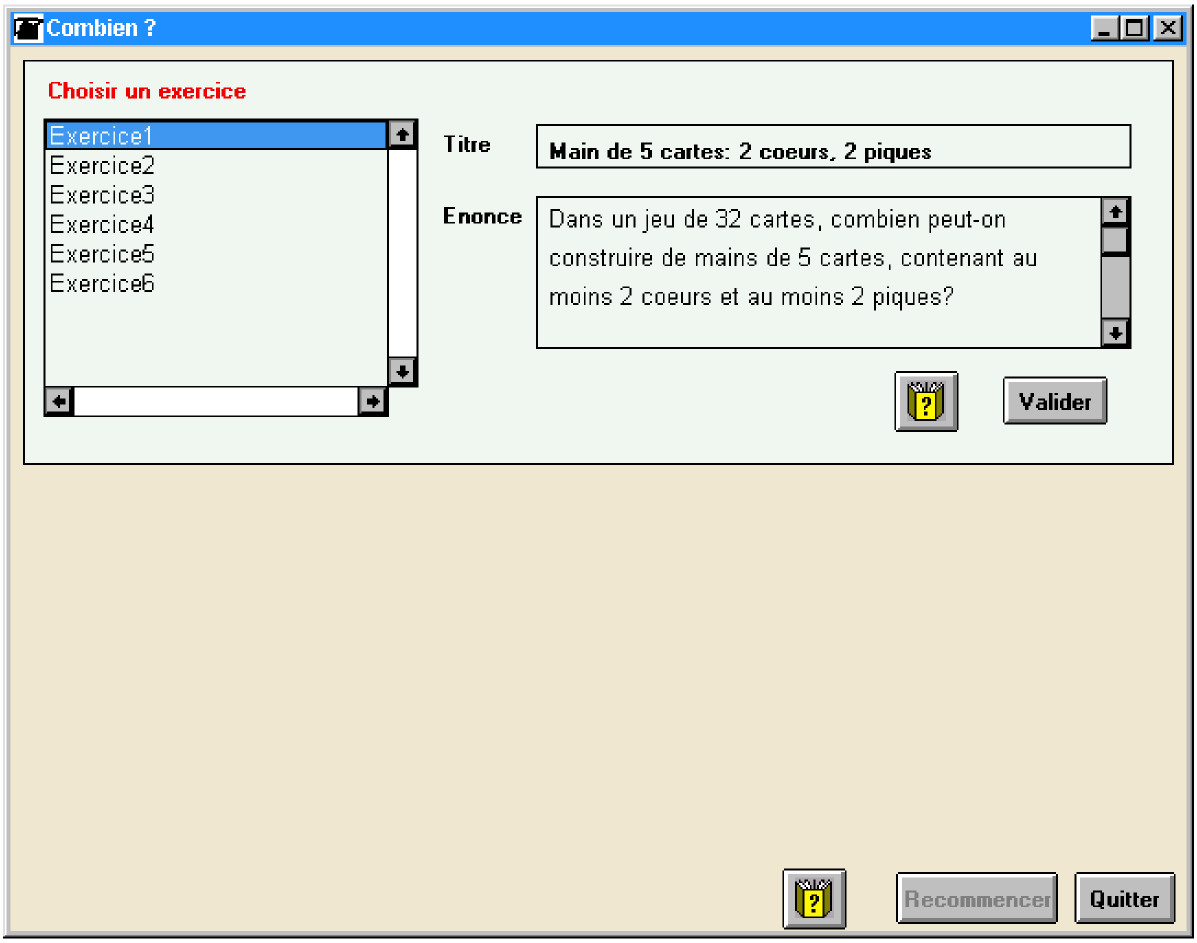

Figure 1 : Interface de l'application lors du lancement

L'interface est une fenêtre 640x480, qui gardera la même apparence tout au long de l'exercice. Des fenêtres apparaîtront et disparaîtront à l'intérieur de celle-ci, comme la sous-fenêtre de choix d'un exercice dans la figure 1.

En cliquant sur **Valider**, on vérifie que tous les champs obligatoires ont été cochés ou remplis et que les saisies sont du type demandé (entier positif par exemple) voire correctes. Quand ce n'est pas le cas, une boîte de dialogue en avertit l'utilisateur, qui doit corriger avant de pouvoir passer à la suite. Chaque fenêtre a son bouton **Valider**, qui lance toujours une vérification et passe à la fenêtre suivante.

Chaque fenêtre a son bouton **Annuler**, qui provoque la fermeture de la fenêtre active et ouvre la précécente avec les choix effectués auparavant mis en surbrillance. On peut ainsi revenir sur ses choix aussi loin qu'on le souhaite. Une limite : on perd les données déjà saisies dans la fenêtre à chaque annulation.

A tout moment, on peut quitter l'interface en cliquant sur **Quitter**. Un message demande confirmation. On peut ainsi revenir au programme en cas d'erreur.

Avec le bouton **Recommencer,** on a toujours la possibilité d'abandonner l'exercice en cours et d'en choisir un nouveau. Cette action doit aussi être confirmée.

Une **Aide** est disponible dans toutes les fenêtres, en cliquant sur l'icône : . Cette aide donne :

- Des informations contextuelles : une aide sur la marche à suivre dans cette fenêtre.
- Des informations conceptuelles : un mini-cours sur le vocabulaire ou le concept sous-jacent.
- Un exemple (différent des exercices proposés) sur ce que l'on peut faire.

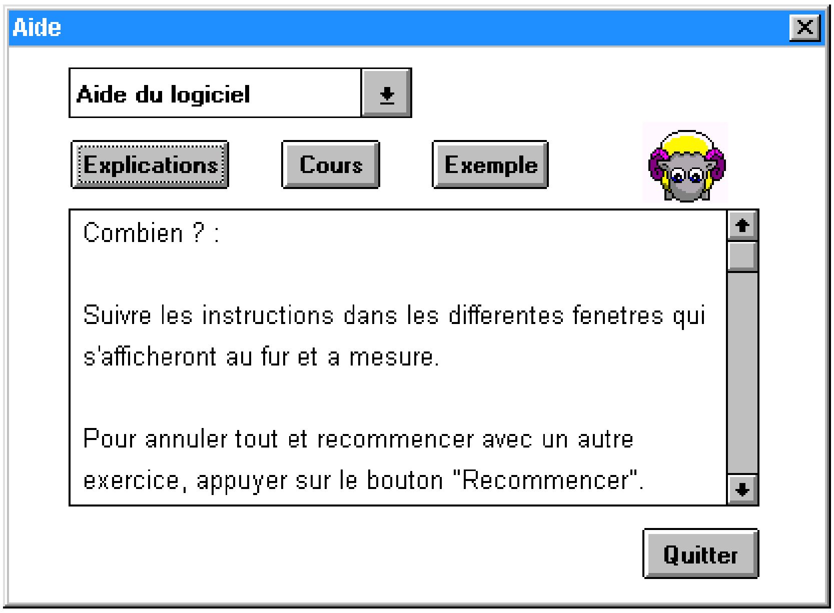

Figure 2 : Aspect de la fenêtre d'aide

On peut choisir le type d'aide en cliquant sur le bouton correspondant ou dans la liste déroulante. On quitte l'aide en cliquant sur **Quitter** et on revient à la fenêtre d'où l'on est parti.

2.1.2. Exercice

Une première sous-fenêtre demande à l'utilisateur de choisir un exercice. Lorsqu'il sélectionne un exercice dans la liste, le titre et l'énoncé correspondants apparaissent. Une fois le choix fait, il suffit de valider.

Si on valide sans avoir choisi d'exercice, un message d'alerte apparaît. Une fois correctement rempli et validé, la fenêtre disparaît. On rappelle l'énoncé de l'exercice en haut et la fenêtre de définition du support de l'exercice apparaît (cf. figure 4).

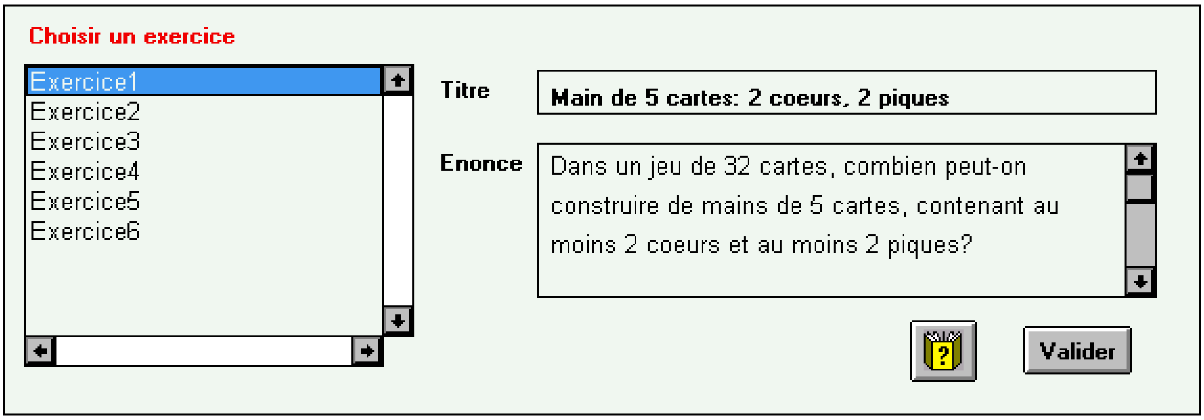

Figure 3 : Choix de l'exercice à modéliser

2.1.3. Référentiel

L'utilisateur choisit alors le support - appelé aussi monde, ou référentiel - dans la fenêtre suivante :

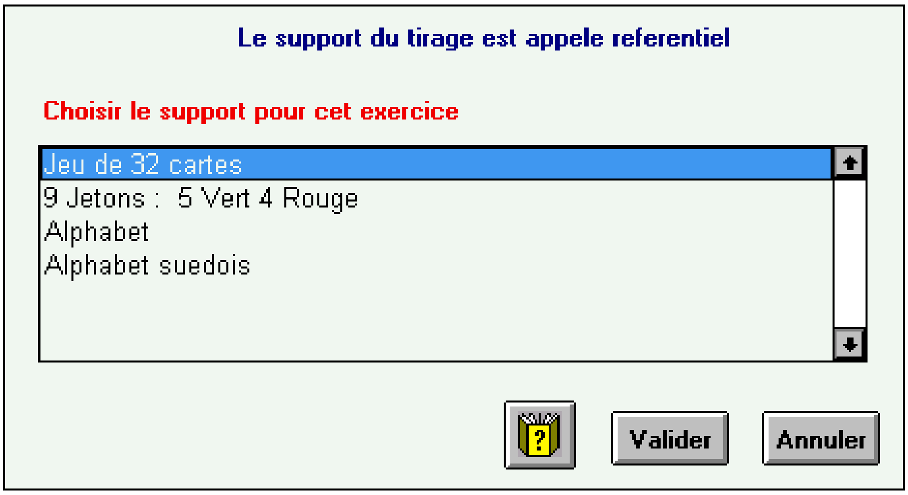

Figure 4 : Choix du référentiel de l'exercice

L'utilisateur choisit le référentiel en le sélectionnant dans la liste qui lui est proposée. S'il annule, il reviendra à la fenêtre précédente.

S'il y a validation sans que le support ait été choisi, un message l'en avertira. Sinon, la fenêtre disparaît. On rappelle le support choisi en dessous de l'énoncé et la fenêtre de définition de l'univers apparaît (cf. figure 5).

2.1.4. Univers

Le référentiel étant choisi, l'utilisateur définit l'ensemble des tirages possibles, appelé l'univers :

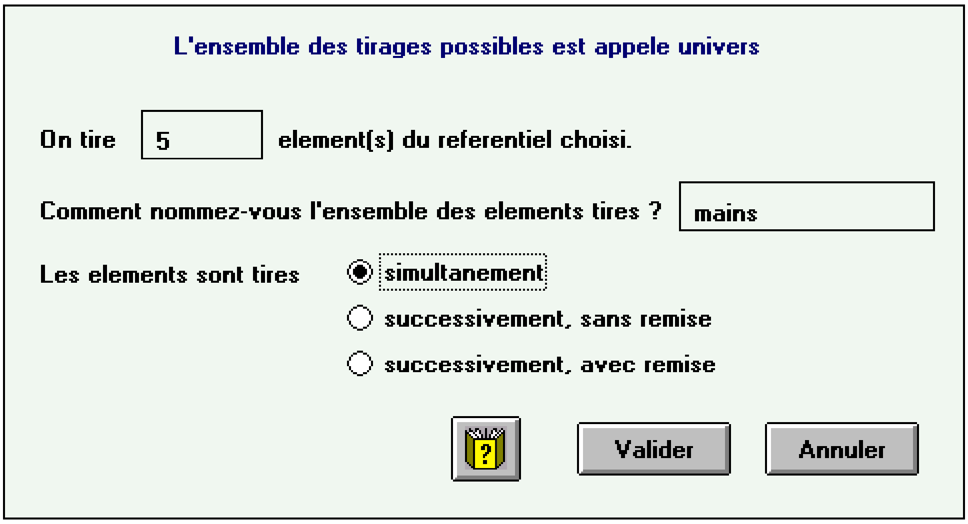

Figure 5 : Définition de l'univers de l'exercice

Il faut indiquer :

- le nombre d'éléments que l'on va tirer,
- éventuellement le nom de l'ensemble des éléments tirés (comme les mains, les mots...),
- le type de tirage : simultané ou successif avec ou sans remise.

La validation fait passer à l'étape suivante à condition, d'une part que le nombre d'éléments ait été correctement donné : un entier strictement positif, d'autre part que le type de tirage ait été indiqué. Dans ce cas, la fenêtre disparaît et est remplacée par un rappel de l'univers, et par la fenêtre de définition des conditions portant sur les éléments à tirer (cf. figure 6)

2.1.5. Contraintes

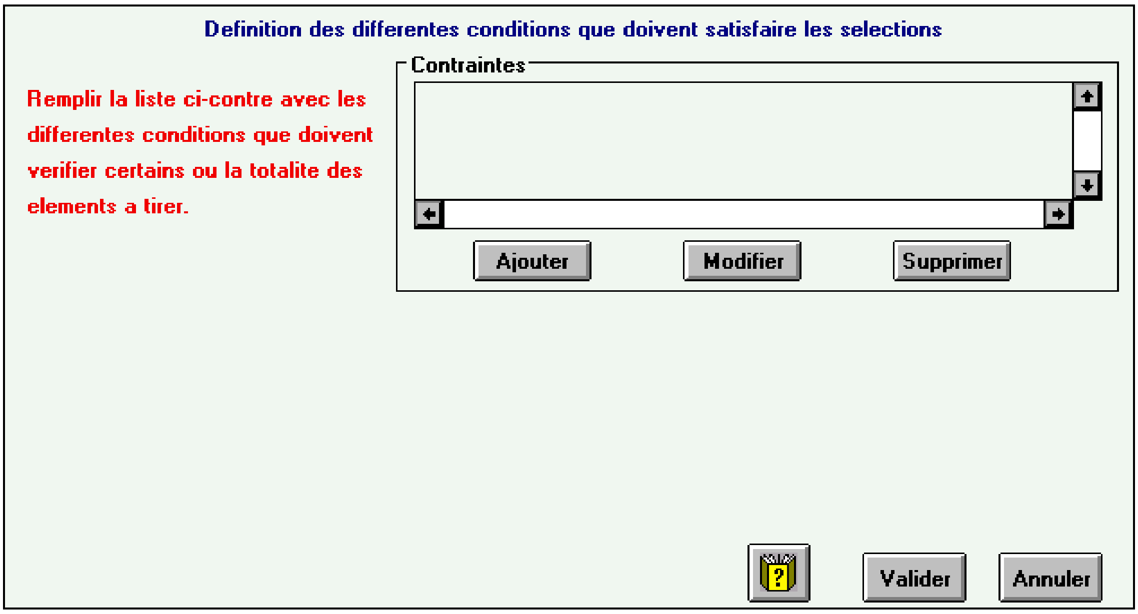

Figure 6 : Sous-fenêtre de définition des contraintes

Ayant déterminer l'univers, il faut définir une liste de conditions, que doivent vérifier ceux de ses éléments à dénombrer. Ceci se fait en cliquant sur **Ajouter**, ce qui fait apparaître une sous-fenêtre demandant d'indiquer le type de condition que l'on souhaite obtenir :

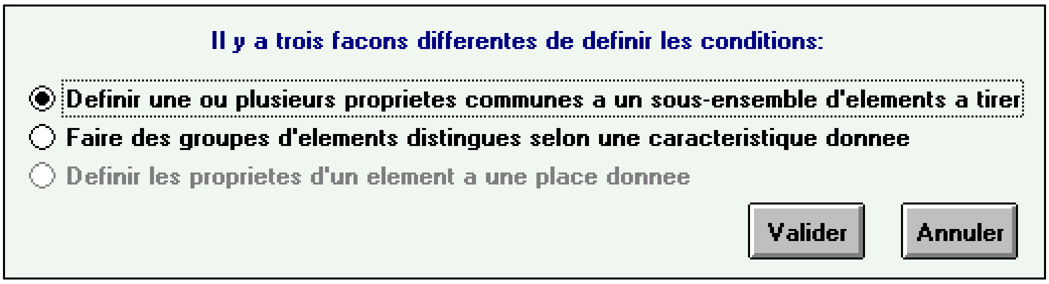

Figure 7 : Choix du type de contrainte

3 types de conditions sont possibles :

- Une partie des éléments à tirer vérifient la (les) même(s) propriété(s).
- Tous les éléments sont regroupés selon une caractéristique, chaque groupe prend une valeur distincte de celle des autres. Les valeurs ne sont pas arrêtées : on connaît juste leur répartition.
- Des éléments, situés à des places précises, vérifient une ou plusieurs propriétés.

Ce dernier type de condition n'est valable que dans le cas des tirages successifs, et est ici grisé. Dans notre exercice, on a les conditions suivantes : au moins 2 coeurs et au moins 2 piques. Coeur et Pique sont des propriétés communes à au moins 2 cartes. On choisit donc le premier type de condition et on valide. La sous-fenêtre disparaît et une autre sous-fenêtre s'ouvre (cf. figure 8).

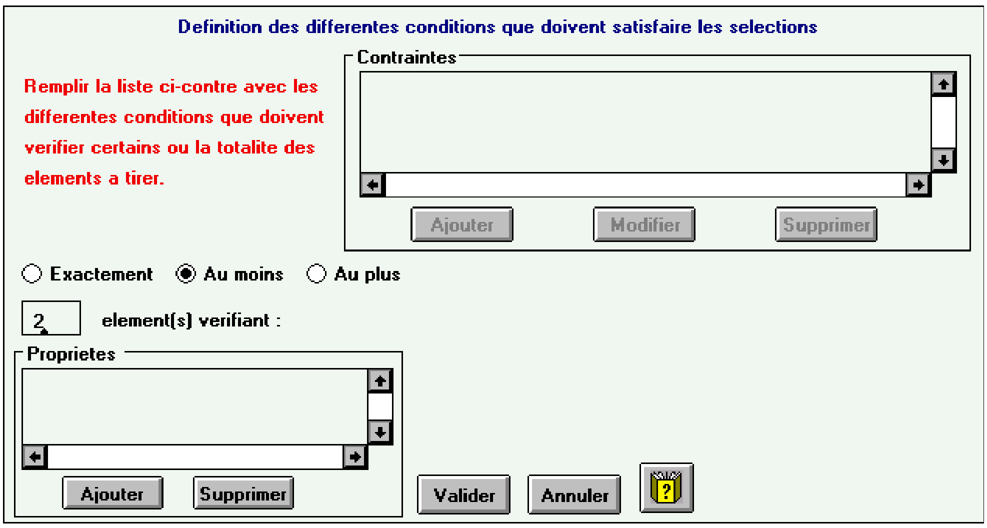

Figure 8 : Définition du nombre d'éléments concernés par la contrainte

Il suffit d'indiquer que l'on veut au moins 2 éléments et cliquer sur **Ajouter**. Un autre sous-panneau s'affiche demandant cette fois le type de propriété que l'on souhaite :

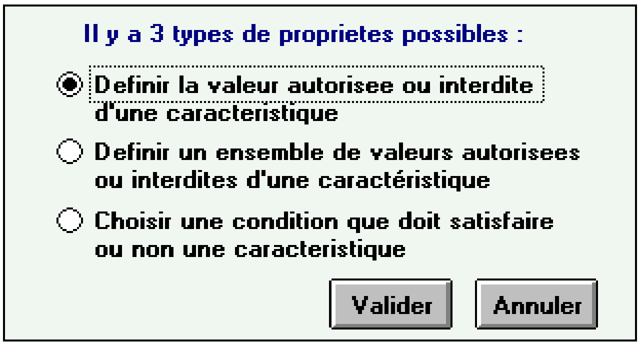

Figure 9 : Choix du type de propriété

Dans notre cas, on veut que les cartes soient du coeur, c'est une valeur autorisée (pour la couleur) ; on coche donc la première possibilité et on valide. Cette sous fenêtre disparaît et une sous-fenêtre supplémentaire s'affiche :

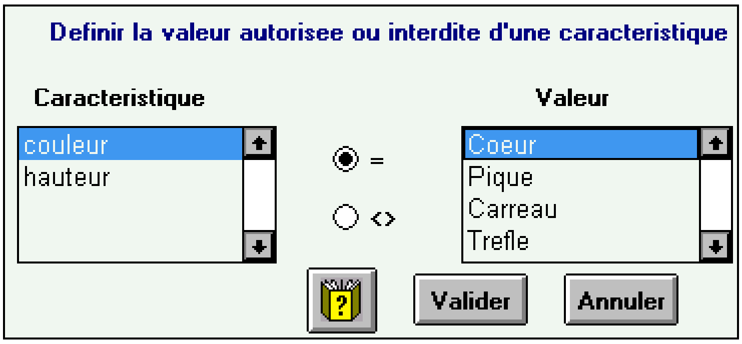

Figure 10 : Définition de la propriété

Il suffit simplement de sélectionner la caractéristique désirée : la couleur, ce qui fait apparaître la liste des couleurs possibles dans la liste des valeurs. On sélectionne = et on choisit Coeur. Si on choisit hauteur, la liste des hauteurs s'affichera (du 7 à l'as).

On valide et la sous-fenêtre disparaît. La liste des propriétés se remplit avec la contrainte couleur = coeur. La condition « avoir au moins deux coeurs » est maintenant définie. On peut donc valider.

Attention au piège, il ne faut pas ajouter une deuxième propriété : on obtiendrait la contrainte au moins deux cartes dont la couleur est coeur et dont la couleur est pique, ce qui revient à donner une contrainte impossible à réaliser ! Au moins 2 coeurs et au moins 2 piques sont deux contraintes distinctes, portant sur une même caractéristique : la couleur.

On remplit de la même manière la deuxième contrainte, et après validation, on obtient la liste des conditions remplie :

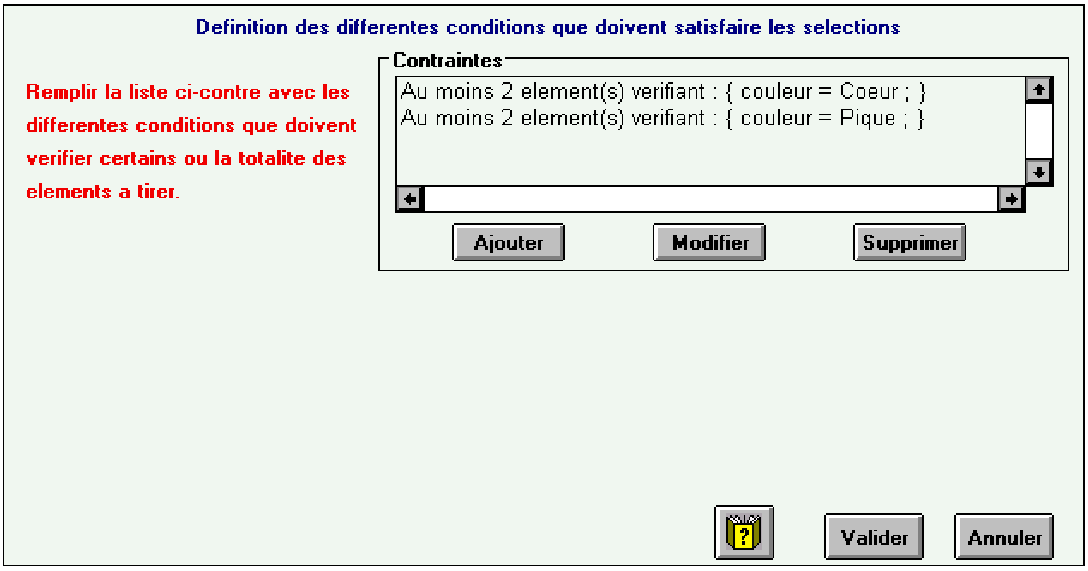

Figure 11 : Rappel de toutes les contraintes

Si à ce moment on est satisfait des contraintes définies, on peut valider, sinon, il y a possibilité de revenir sur une ou plusieurs d'entre elles en la sélectionnant dans la liste et en cliquant sur supprimer ou modifier, le bouton modifier rouvre la sous-fenêtre correspondante, où l'on peut ajouter ou supprimer une ou plusieurs propriétés.

On valide, la fenêtre disparaît, la liste des contraintes définies s'affiche dans la fenêtre principale, ainsi qu'une fenêtre indiquant que la modélisation est terminée (cf. figure 12).

2.1.6. Correction

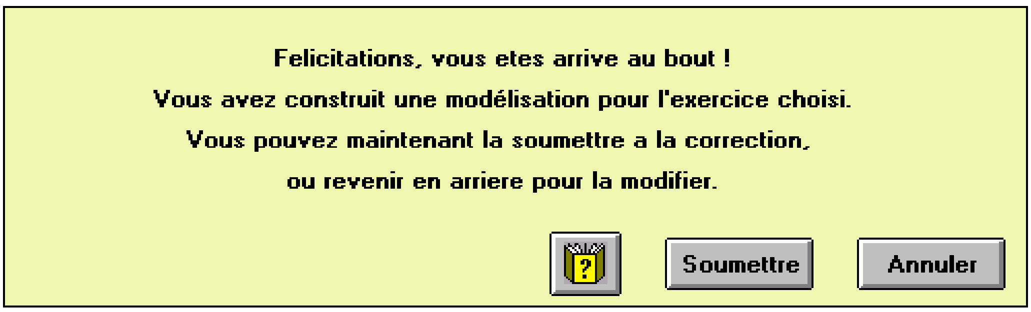

Figure 12 : Sous-fenêtre indiquant la fin de la modélisation

Dans cette fenêtre, on peut soit lancer la correction, soit revenir sur la définition des contraintes.

Si on clique sur **Soumettre**, une fenêtre s'ouvre en dehors de l'interface principale :

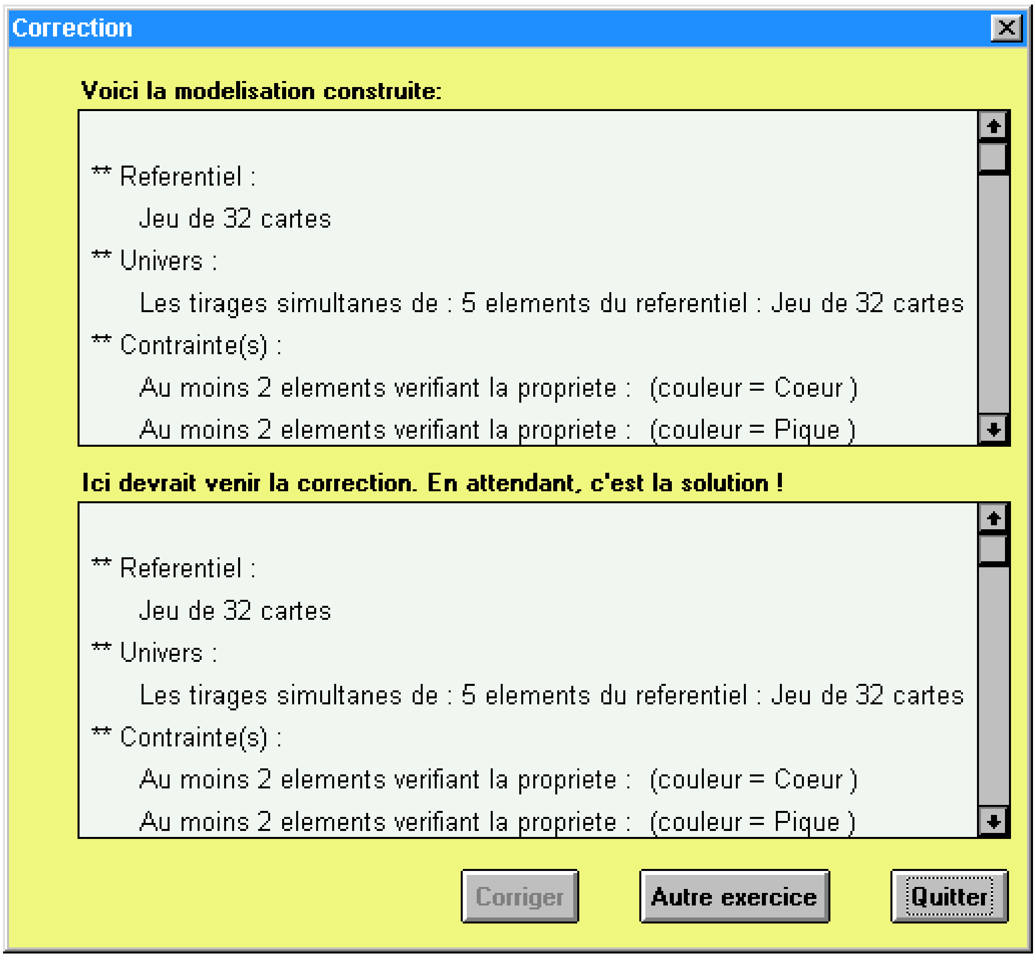

Figure 13 : Fenêtre de correction de la modélisation

La correction n'ayant pas été implémentée, cette fenêtre affiche la modélisation construite par l'élève et la solution proposée.

Quand la correction sera implémentée, le bouton **Corriger** permettra de revenir sur ses erreurs.

Dans tous les cas, on a la possibilité soit de refaire un nouvel exercice avec **Autre exercice**, soit de **Quitter** le programme.

2.2. Principes pédagogiques

Le vocabulaire utilisé au départ par le modèle conceptuel, est pratiquement incompréhensible pour un élève de terminale. Il a donc fallu l'adapter et trouver des phrases simples, susceptibles de faire passer des concepts plus compliqués.

2.2.1. Terminologie, vocabulaire

Nous avons eu quelques difficultés à construire des phrases simples. En effet, quand on est plongé dans certain niveau d'abstraction avec son « jargon », il est difficile de s'en éloigner. De plus, nous sommes limitées en espace dans les fenêtres, il est donc impératif de faire des phrases courtes. Difficile de faire passer un concept complexe en 4 ou 5 mots !

Nous avons fait tester l'interface par quelques personnes (qui ne sont plus en terminale depuis quelques années), et le message ne passe pas quand on parle le langage conceptuel, mais passe beaucoup mieux après les changements de vocabulaire.

Choix effectués :

Nous avons décidé de donner à la fois des instructions et des explications, notamment dans les fenêtres les plus compliquées. Dans tous les cas, une aide vient au secours de ceux qui ne comprendraient pas : des instructions, des explications et même des exemples sont disponibles à chaque étape, mais nous y reviendrons plus tard.

Les concepts de référentiel, d'univers et de contraintes sont plutôt simples, mais le vocabulaire est peu familier voire inconnu des élèves. Nous avons donc donné une phrase d'explication, mais les noms sont conservés et utilisés lors des rappels des choix de l'élève. Il peut ainsi acquérir en partie le vocabulaire spécifique au dénombrement, ce qui fait aussi partie des finalités du logiciel.

Par contre, les différentes contraintes et propriétés ont des noms bien trop complexes à notre avis, et les connaître n'a pas vraiment d'intérêt. Seul importe que les élèves comprennent les concepts sous-jacents. Ils ne sont donc pas donnés.

Notez que les noms d'origine des concepts ont été conservés dans l'implémentation, et que seul le vocabulaire de l'interface a été modifié. Cela semble préférable dans le sens où ceux qui travailleront sur la partie programmation seront des experts en dénombrement et auront accès au schéma conceptuel tel qu'il nous a été donné. On garde ainsi une cohérence entre la partie conceptuelle et l'implémentation. Dans les définitions des interfaces, on garde aussi les noms conceptuels et on y ajoute les noms simplifiés, c'est donc dans cette interface que l'on trouve le lien direct entre les deux types de vocabulaire.

Le **référentiel** est défini comme le support de l'exercice (la notion de monde est aussi présent dans l'aide). Le nom de référentiel est conservé dans l'interface. Nous avons estimé que la liste des référentiels que l'on voit parle d'elle même. Ce concept ne pose donc pas vraiment de problème.

L'**univers** est défini comme l'ensemble des tirages, mais garde son nom dans la suite. Par contre, le type d'univers est complètement remanié :

- l'**univers des parties** est défini comme un univers dans lequel les éléments sont tirés simultanément, et donc où il n'y a pas de notion d'ordre.
- l'**univers association** quant à lui, est défini par le tirage successif et séparé en deux parties suivant qu'il est **injectif** ou non : c'est la notion de remise qui remplace la

notion d'injectivité. Le **référentiel but** est le référentiel sélectionné auparavant et le **référentiel source** est construit à partir du nombre n d'éléments tirés : les places sont numérotées de 1 à n. L'élève n'a jamais connaissance directe de ce dernier.

Les **contraintes** sont définies comme des conditions que doivent vérifier certains éléments mais le concept garde son nom, notamment sur la liste des conditions définies par l'utilisateur. Le type de contrainte est lui aussi remanié :

- une **contrainte d'effectif** est définie comme un ensemble de propriétés que vérifient une partie des éléments tirés.
- une **contrainte de distribution** est définie comme une répartition de tous les éléments sur une de leurs caractéristiques. Les éléments sont séparés en groupes selon leur valeur et tous les éléments d'un groupe ont la même valeur (couleur, lettre...).
- une **contrainte de restriction**, qui n'est valable que dans le cas de l'univers association, est définie comme une condition portant sur un ou plusieurs éléments à une place donnée. Les éléments ainsi désignés doivent vérifier certaines propriétés.

Les **propriétés** à préciser dans le cas des contraintes d'effectifs, portent sur le sous-ensemble des éléments que définit l'élève, et ne gardent pas leur nom :

- une **propriété de comparaison** est définie comme la valeur que peut prendre ou ne pas prendre une caractéristique.
- une **propriété d'appartenance** est définie comme l'ensemble des valeurs autorisées ou interdites d'une caractéristique.
- une **propriété prédicat** est définie comme la condition que doit vérifier une caractéristique.

Les **attributs** d'un élément sont appelés caractéristiques, terme que nous avons jugé plus compréhensible. Une caractéristique donnée peut prendre différentes valeurs, et un élément peut en posséder plusieurs.

2.2.2. Le principe d'enchaînement

Dans l'interface, l'enchaînement est imposé par la modélisation, car certaines fenêtres dépendent des choix effectués auparavant. Par exemple, les possibilités de définir les contraintes dépendent du type d'univers. De même, le choix du type de propriété va influer sur la fenêtre suivante. A cause de cette séquentialité, nous avons choisi de ne pas utiliser de barre de menu.

Il est indispensable de valider une étape avant de pouvoir passer à la suivante.

2.2.3. Les rappels

Quand un concept est entièrement défini, la fenêtre correspondante disparaît. Il est donc impératif de rappeler ce que l'utilisateur a déjà fait, afin de faciliter sa compréhension et lui permettre éventuellement de se rendre plus facilement compte de ses erreurs et de leurs implications. Sont visibles (une fois définis, bien sûr) :

- l'énoncé que l'on a choisi,

• les données importantes qui sont rappelées : le référentiel, l'univers et la liste des contraintes.

2.2.4. La correction

Une fois terminée, la modélisation de l'élève doit être validée : il faut donc effectuer une correction. Un certain nombre de vérifications sont faites en cours de modélisation, elles concernent essentiellement les champs obligatoires et une contrainte d'intégrité : on ne peut pas définir une contrainte d'effectif sur plus d'éléments que ceux tirés.

L'intérêt de ne corriger qu'à la fin de l'exercice, par rapport à la correction pas à pas, est multiple :

• elle évite que les utilisateurs ne cliquent un peu partout et ne trouvent la solution par le biais des essais-erreurs,

• elle les incite à réfléchir sur le problème et sur les idées parfois implicites que contient l'énoncé (comme le « Exactement », pas toujours donné),

• elle permet les retours arrière et les corrections spontanées de l'élève.

Il y a cependant certaines contraintes d'intégrité qui n'ont pas été implantées, comme celle qui éviteraient de définir des contraintes irréalisables, comme de demander à avoir deux jetons à la fois de couleur rouge et de couleur verte. Sur une seule contrainte, ce serait réalisable, mais quand cela se passe sur une conjonctions de contraintes, cela devient vite ardu : comment vérifier que sur une main de 5 cartes, on ne peut pas avoir au moins 3 piques et exactement 4 trèfles ? De même pour les contraintes de distribution : on ne peut pas faire 5 groupes de couleurs différentes pour les cartes.

Dans l'ensemble, une grande liberté est laissée à l'élève : aucune valeur n'est donnée par défaut et peu de garde-fous sont disponibles. Un effort peut être fait dans ce sens, mais est-ce utile, ne vaut-il pas mieux le laisser se rendre compte de ses erreurs ? Il est vrai que l'on apprend mieux en faisant des erreurs, mais l'empêcher de faire les plus grosses serait peut être à étudier.

2.2.5. Cohérence avec la construction en cours

La construction se fait en fonction des choix précédents de l'élève, et non pas de la solution. Cela consiste par exemple à afficher les caractéristiques du référentiel qu'il a donné, même s'il est faux. L'idée est de ne pas lui donner d'indices, mais de lui faire assumer ses choix.

2.2.6. La correction

Telle qu'on l'imagine, la correction doit vérifier que le problème construit par l'élève, est **équivalent** à la solution proposée, en suivant le même cheminement que lui. On est amené à se poser le problème de l'équivalence à chaque étape de la construction du modèle du problème en comparant les objets choisis par l'élève aux objets correspondants dans la solution

Notion d'équivalence :

• **Problème** : deux problèmes sont équivalents si leurs référentiels, univers et contraintes sont équivalents entre eux.

• **Référentiel** : pas de difficulté, dans la mesure où l'élève choisit dans une liste de référentiels existants. Il suffit de vérifier l'égalité entre les deux référentiels.

• **Univers** : pas de difficulté majeure, il faut vérifier le nombre, le type d'univers et l'injectivité.

• **Contraintes** : les difficultés commencent ici. L'ordre des contraintes n'a pas d'importance. Deux contraintes sont équivalentes si :

◊ Elles sont identiques.

◊ Elles sont identiques aux permutations des propriétés près.

◊ Elles ont le quantificateur, le nombre et les propriétés opposés. Ex : « Sur une main de 5 cartes, au moins 2 cartes rouges » est équivalent à « Sur une main de 5 cartes, au plus 3 cartes noires »

◊ Leurs propriétés sont équivalentes. Ex : « couleur vérifie rouge » et « couleur appartient à [Coeur, Carreau] »

◊ L'une des contraintes est implicite. Ex : Dans une urne contenant des boules rouges et vertes, « tirer exactement 3 boules vertes et 2 boules rouges » et « tirer exactement 3 boules vertes (sur les 5 boules) »

◊...

La liste ci-dessus n'est pas exhaustive, mais nous avons déjà du mal à imaginer comment détecter ces types d'équivalence ! Une solution serait peut-être de calculer et comparer pour les deux problèmes (solution et construit) toutes les possibilités, ou toutes les restrictions causées par une ou plusieurs contraintes.

Retours sur la (les) erreur(s)

Si on détecte une ou plusieurs erreurs, il faut commencer à corriger la plus ancienne, car elle peut impliquer de sérieuses modifications dans la suite du problème. Il faut donc recommencer l'exercice à partir de ce point (?). Dans la limite du possible, on pourrait envisager de conserver les saisies des fenêtres qui suivent, même si elles sont devenues incohérentes.

La correction implémentée

Nous n'affichons pour l'instant qu'un bilan du problème construit, destiné à terme au professeur, et la solution proposée. L'idée est de donner un retour visible et d'implémenter un embryon de classe de correction.

2.3. Ergonomie

2.3.1. Aspects visuels

Les instructions sont données en rouge, tandis que les explications sont en bleu foncé ; le reste étant en noir.

Les sous-fenêtres sont en blanc, tandis que le fond de la fenêtre principale est saumon. C'est là qu'on note les rappels.

Les boutons **Aide**, **Valider** et **Annuler**, sont toujours dans cet ordre, en bas à droite, dans la fenêtre concernée.

Les fenêtres s'enchaînent de façon à simuler un parcours de gauche à droite et de haut en bas.

Il n'y a pas de barre de menus, à cause de la séquentialité inhérente à une partie de l'exercice.

La taille de la fenêtre est fixée à 640x480, pour permettre une exécution sur les écrans basse définition. Cela nous a compliqué la vie dans le sens où la fenêtre est encombrée pour toutes les sous-fenêtres. L'édition de la fenêtre principale est en effet complètement illisible à cause de tous les panneaux superposés. Heureusement, ils sont définis séparément, et ce n'est qu'une fois complets qu'ils sont intégrés dans la fenêtre principale.

2.3.2. Messages

Il y a 2 types de messages :

- les messages indiquant une erreur lors de la validation :

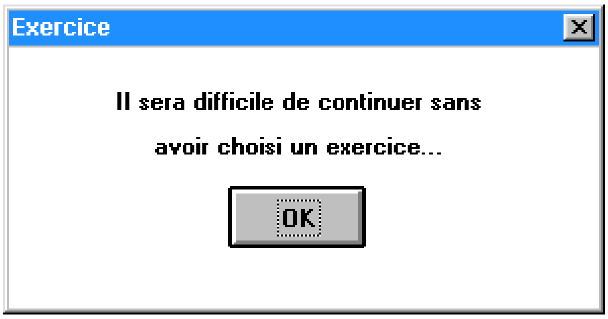

Figure 14 : Boîte de dialogue d'erreur

- les messages demandant confirmation, avant les actions « fatales » (quitter, recommencer, supprimer).

Figure 15 : Boîte de dialogue d'avertissement

3. Programmation Smalltalk

3.1. Modèle conceptuel

Tout ce programme repose sur le modèle conceptuel défini par le groupe « Combien? ». En effet, il est très important de respecter sa structure, puisqu'elle servira de trame de fond à la résolution. D'ailleurs, elle nous paraît solide et complète.

3.1.1. Schéma global

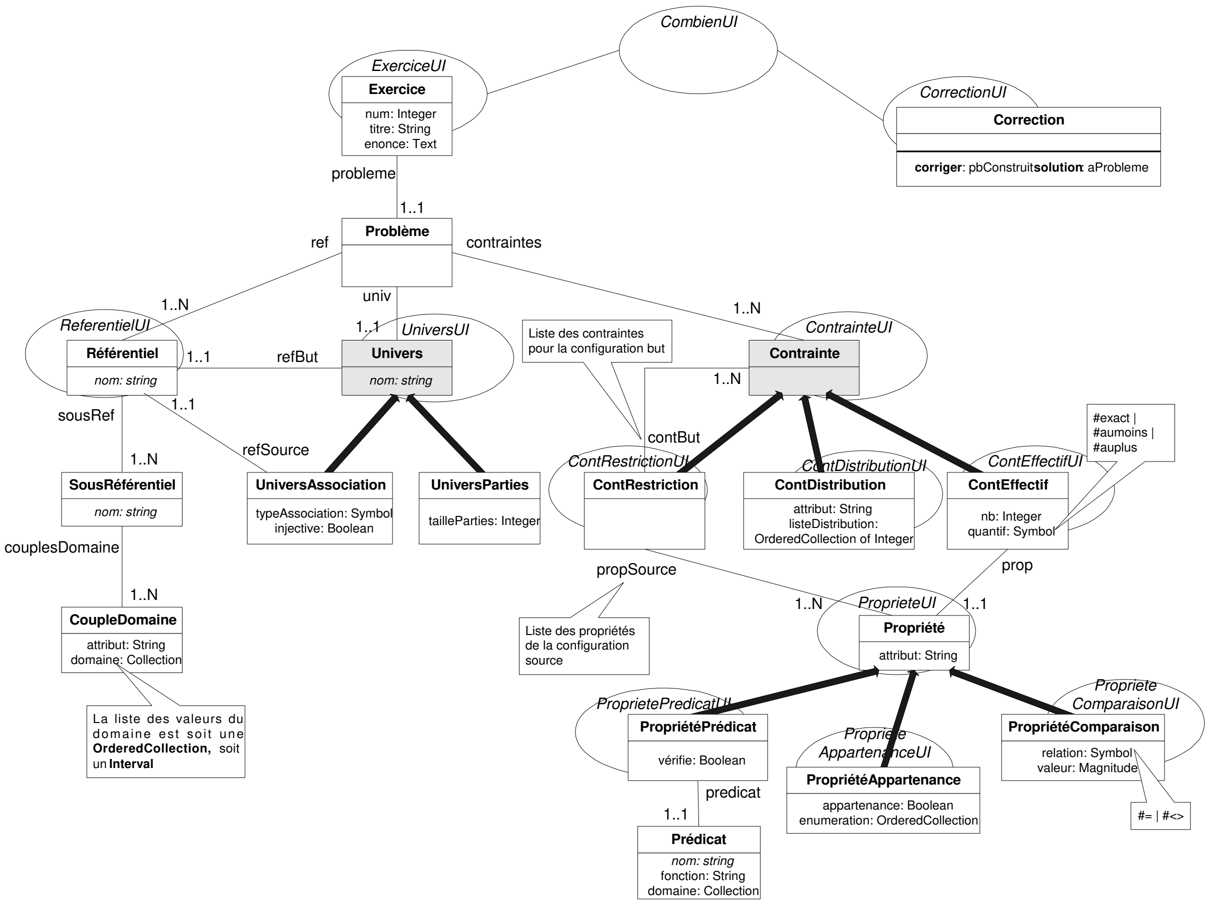

Figure 16 : Modèle conceptuel de l'application

Les ovales apparaissant en arrière-plan derrière certaines classes, ainsi que l'ovale CombienUI, matérialisent les classes d'interface..

3.1.2. Modifications du modèle conceptuel

• D'une part, nous avons ajouté une classe `ObjetNommé`, pour factoriser la propriété nom dans les classes : `Référentiel`, `SousRéférentiel`, `Univers` et `Prédicat`. Cette modification minime, n'a d'intérêt que pour l'implémentation, en effet, elle nous permet de faire toujours référence aux mêmes méthodes d'accès. Enfin, le nom est l'un des rares champs des objets à être visible par l'utilisateur.

• D'autre part, nous avons considéré qu'une contrainte de restriction ne peut contenir que des contraintes d'effectifs concernant la configuration but, ceci pour simplifier l'implémentation graphique de ce type de contrainte. Ce choix restreint l'ensemble des problèmes que l'on pourra proposer. Théoriquement, on devrait également accepter les contraintes de distribution, mais déjà avec cette modélisation, nous pensons couvrir bon nombre de cas.

3.1.3. Exemples existants

Nous avons travaillé sur un échantillon de 7 exercices, correspondant à des problèmes classiques de dénombrement.

Exemple1 : Les mains de 5 cartes contenant 2 coeurs et 2 piques :
Exemple2 : Les groupes d'1 jeton vert et 2 rouges parmi 9
Exemple3 : Les mots de 5 lettres contenant 1 a
Exemple4 : Les mains de 5 cartes (parmi 52) étant un « Full »
Exemple5 : Les mots de 5 lettres contenant 2 b et 2 voyelles
Exemple6 : Les mots en suédois de 5 lettres contenant 2 b et 2 voyelles
Exemple7 : Les mots de 6 lettres commençant par X et contenant 3 voyelles

Ils permettent d'illustrer un certain nombre de concepts présents dans le modèle :

un référentiel formé d'un sous-référentiel (exemple1 exemple3 exemple4 exemple5 exemple7), un réferentiel composé de deux sous-référentiels (exemple2 exemple6), plusieurs référentiels (exemple3 exemple5 exemple6 exemple7),

les deux sortes d'univers possibles : univers de partie (exemple1 exemple2 exemple4) , univers d'association (exemple3 exemple5 exemple 6 exemple7),

les différentes contraintes : contrainte d'effectifs (exemple1 exemple2 exemple3 exemple5 exemple6) contrainte de distribution (exemple4) contrainte de restriction (exemple7),

les différentes propriétés : propriétéPrédicat (exemple5 exemple6), propriétéComparaison (exemple1 exemple2 exemple3 exemple6 exemple7) .

Les sept pages suivantes présentent les schémas d'instances des différents exemples :

Les mains de 5 cartes contenant au moins 2 coeurs et au moins 2 piques :

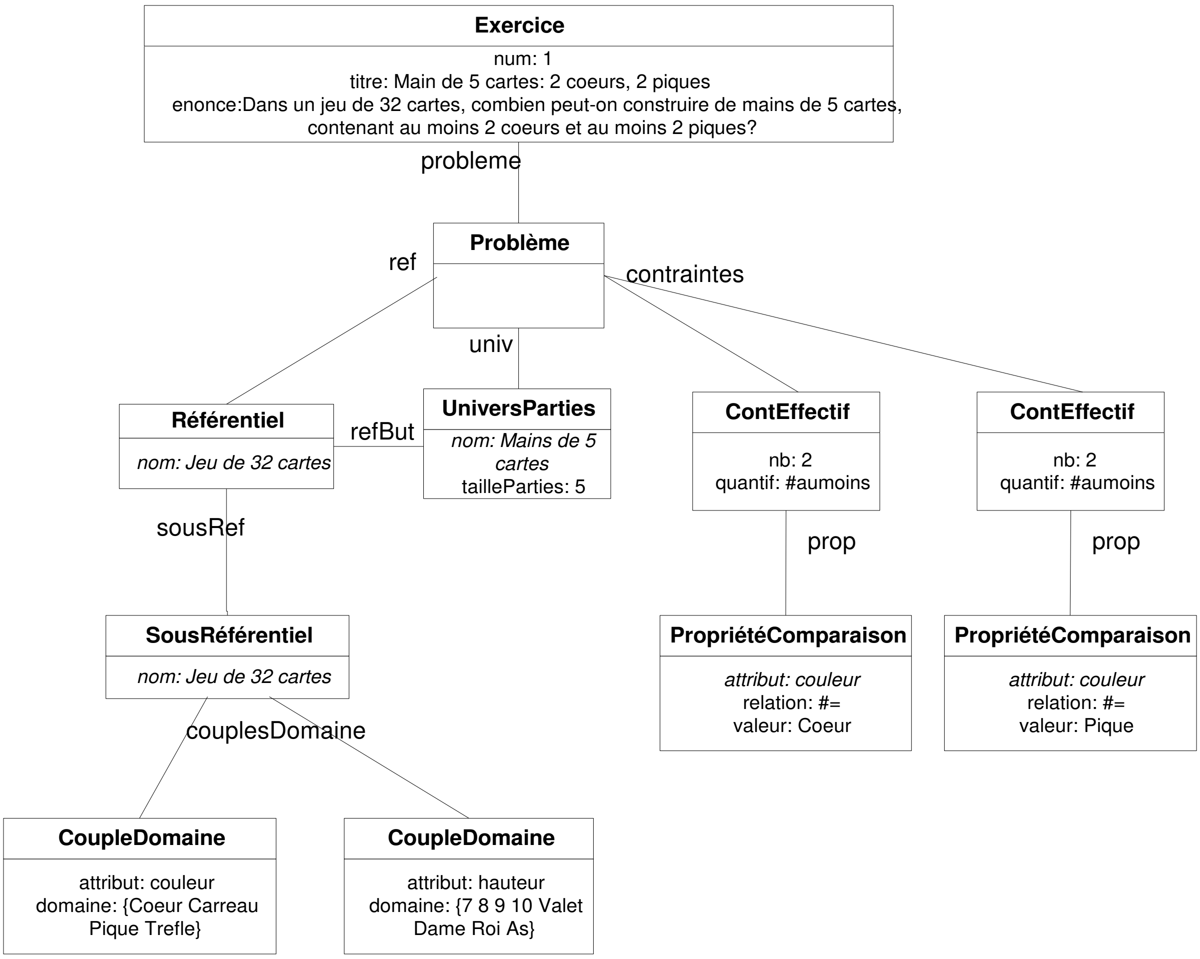

Figure 17 : Exemple 1

Cet exemple sur les **cartes** porte sur un **univers de parties** de taille 5 et utilise une conjonction de deux **contraintes d'effectifs** différentes : « au moins 2 coeurs » et « au moins 2 piques ». C'est une sorte de piège : il y a 2 contraintes différentes, et non pas 1 contrainte contenant 2 **propriétés de comparaison** : « couleur = coeur » et « couleur = pique ».

Les groupes d'1 jeton vert et 2 rouges parmi 9 :

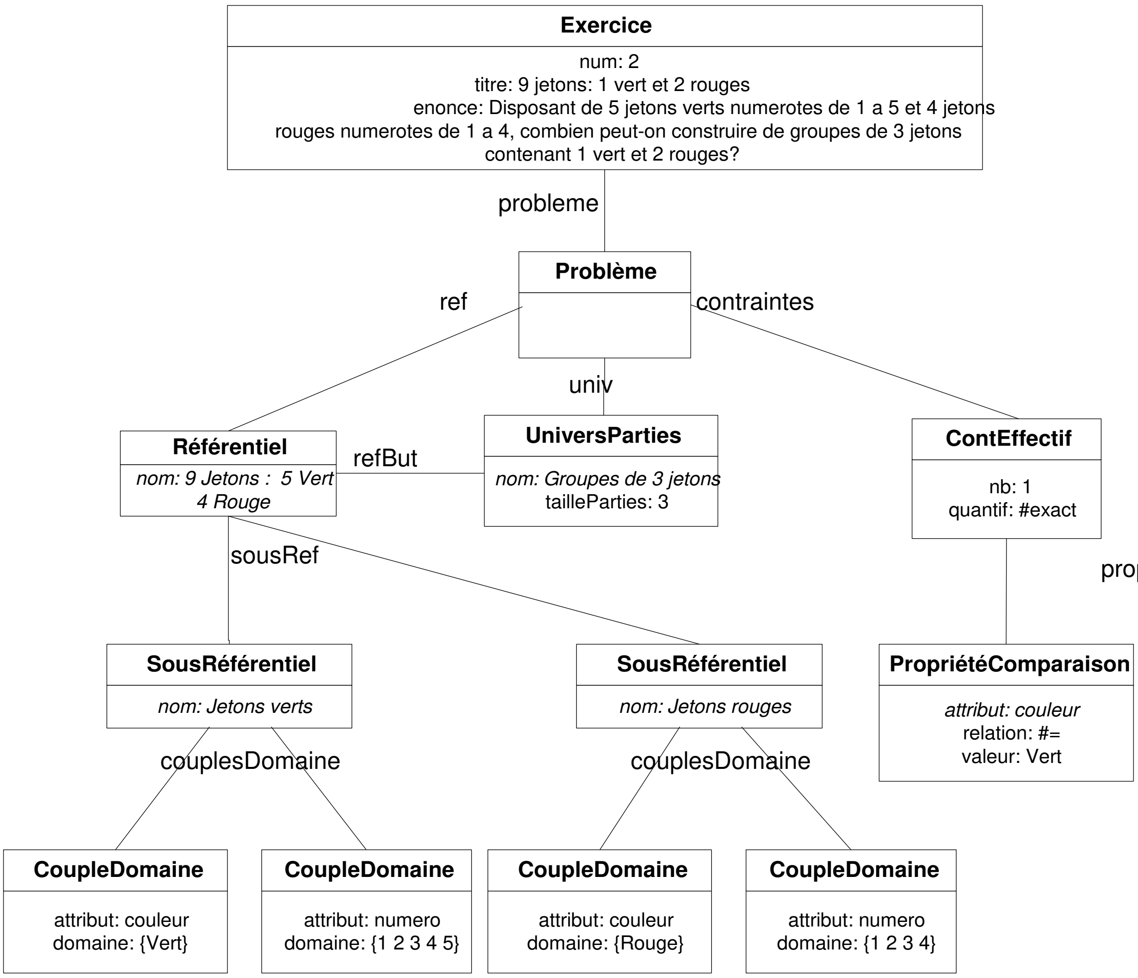

Figure 18 : Exemple 2

Cet exemple sur les **jetons** porte sur un **univers de parties** de taille 3 et contient une seule **contrainte d'effectifs**. C'est également un piège : d'après l'énoncé, on pourrait croire qu'il y 2 contraintes, mais une seule contrainte suffit à définir tout l'ensemble des possibilités. Ceci montre les **problèmes d'équivalence** qui surviendront à la correction.

Les mots de 5 lettres contenant 1 a :

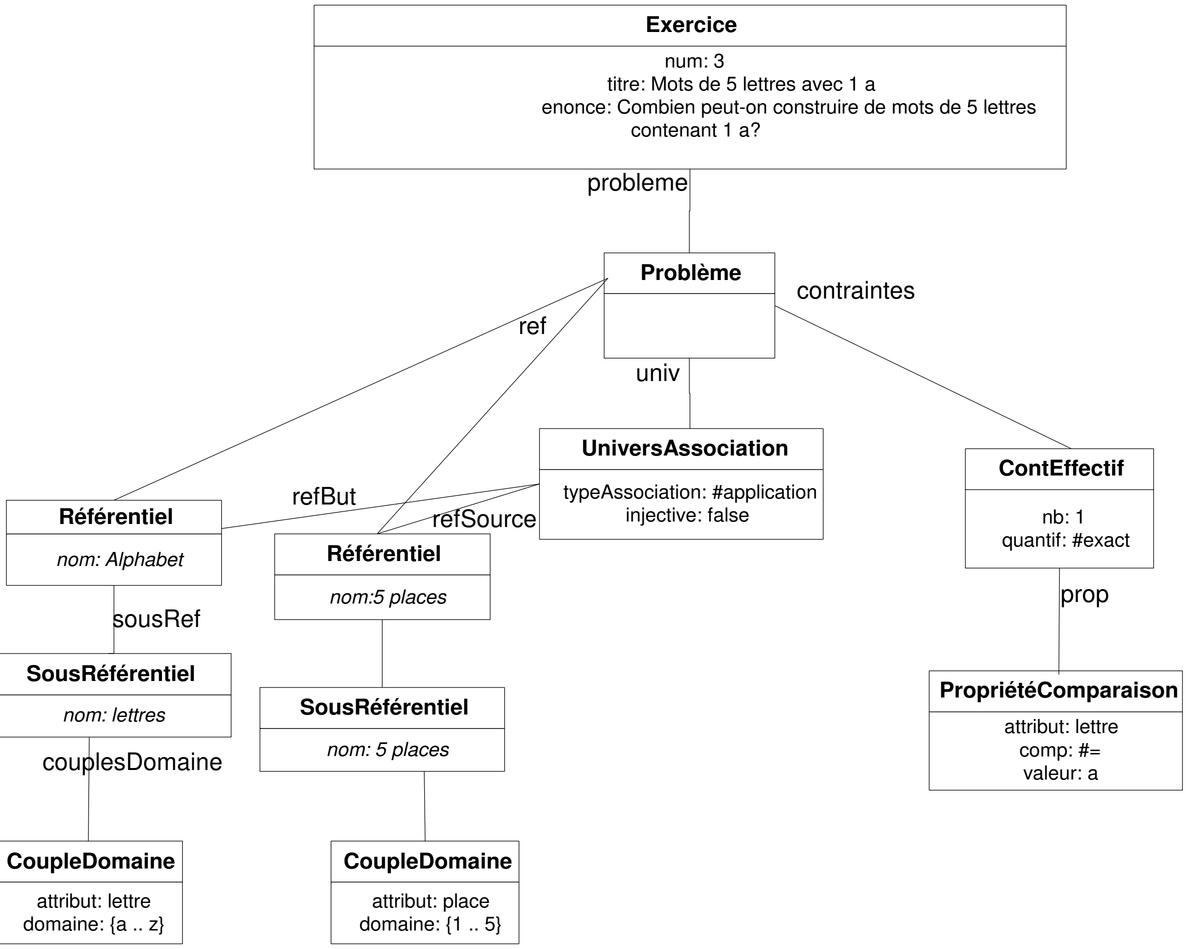

Figure 19 : Exemple 3

Cet exemple sur les **mots** de **l'alphabet** utilise **un univers de type association**, il s'agit donc d'un tirage successif avec remise. Le problème comporte **une contrainte d'effectifs**, « exactement 1 a », et une **propriété de comparaison**, « lettre = a ».

Les mains de 5 cartes (parmi 52) étant un <<Full>> :

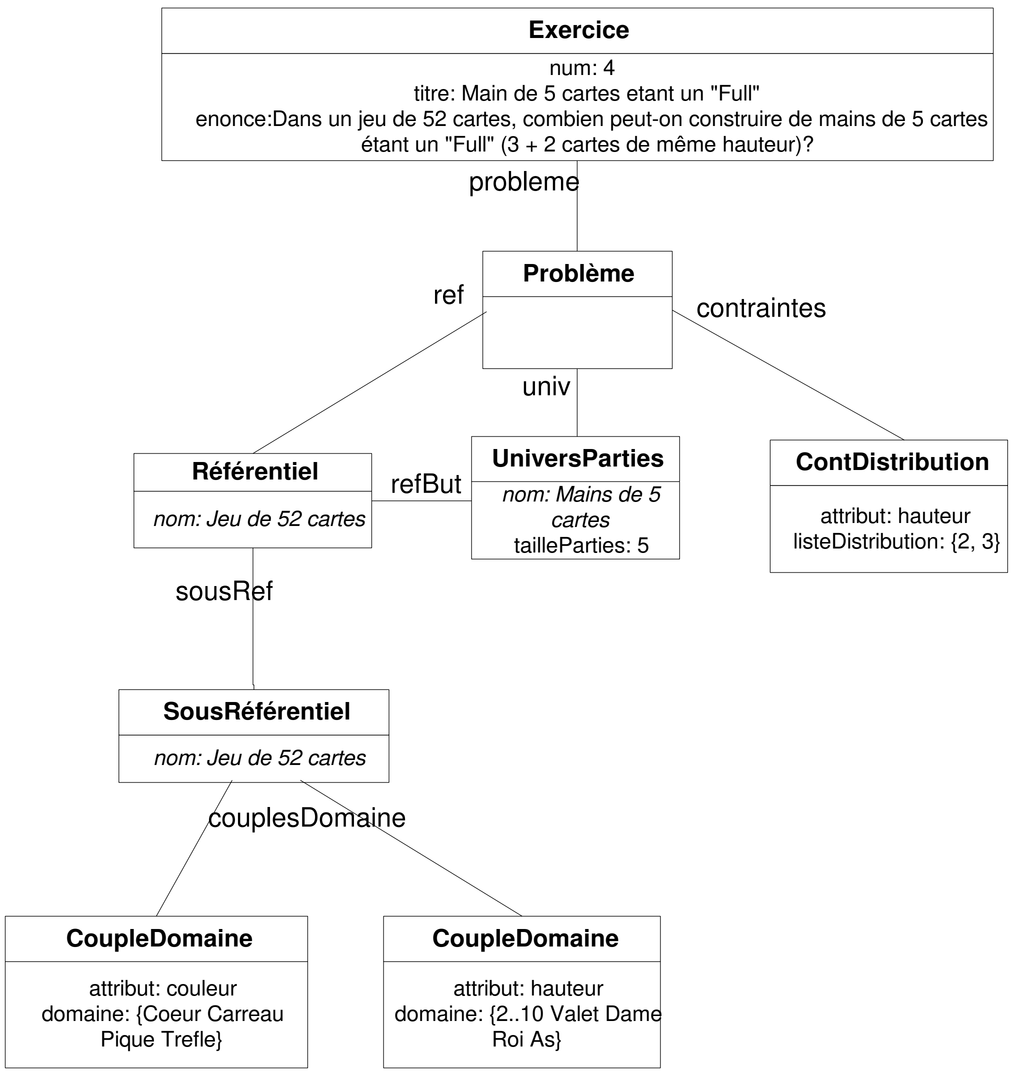

Figure 20 : Exemple 4

Cet exemple sur les mains de 5 **cartes** tirées d'un **jeu de 52**, utilise une **contrainte de distribution** : 2 cartes d'une hauteur, et 3 d'une autre.

Les mots de 5 lettres contenant 2 b et 2 voyelles :

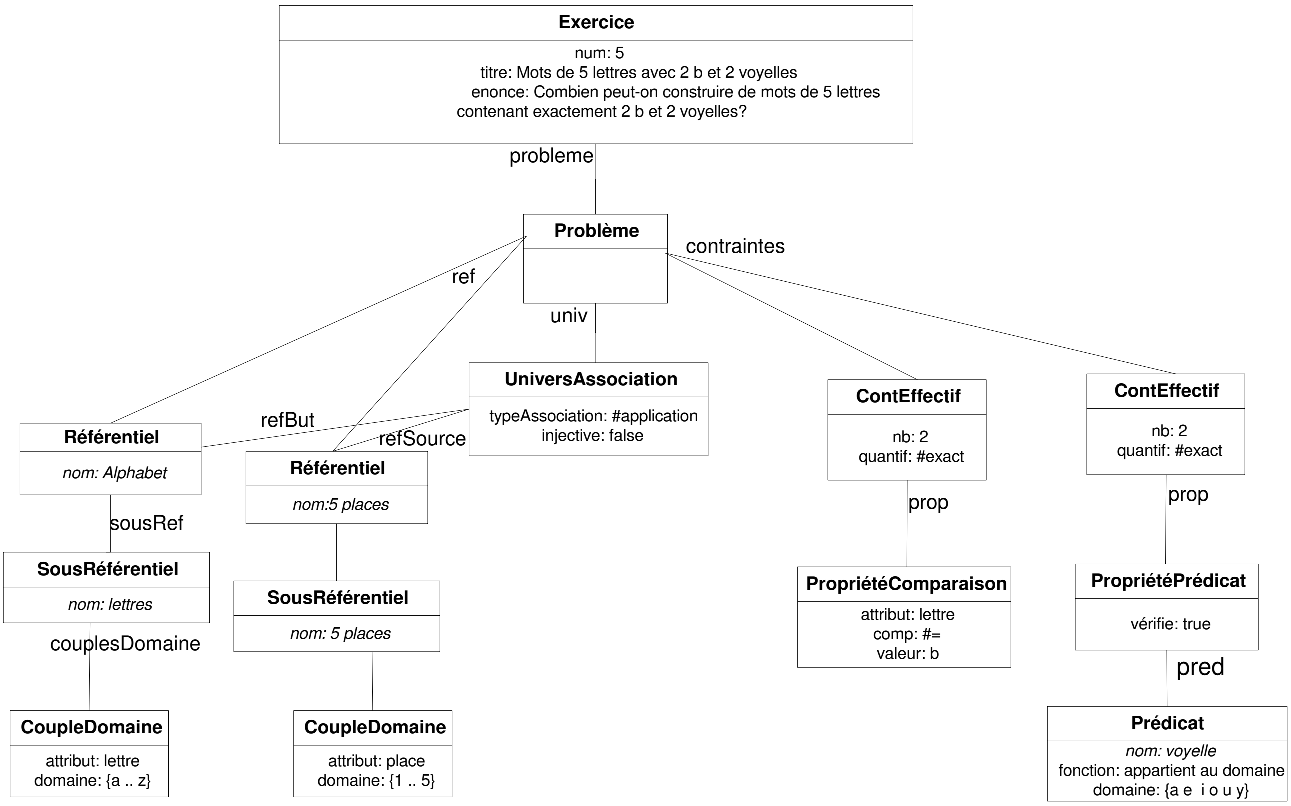

Figure 21 : Exemple 5

Cet exemple sur les **mots** de **l'alphabet** utilise un **univers de type association**, avec deux **contraintes d'effectifs**, dont l'une contient une **propriété de comparaison**, et l'autre une **propriété prédicat**.

Les mots en suédois de 5 lettres contenant 2 b et 2 voyelles :

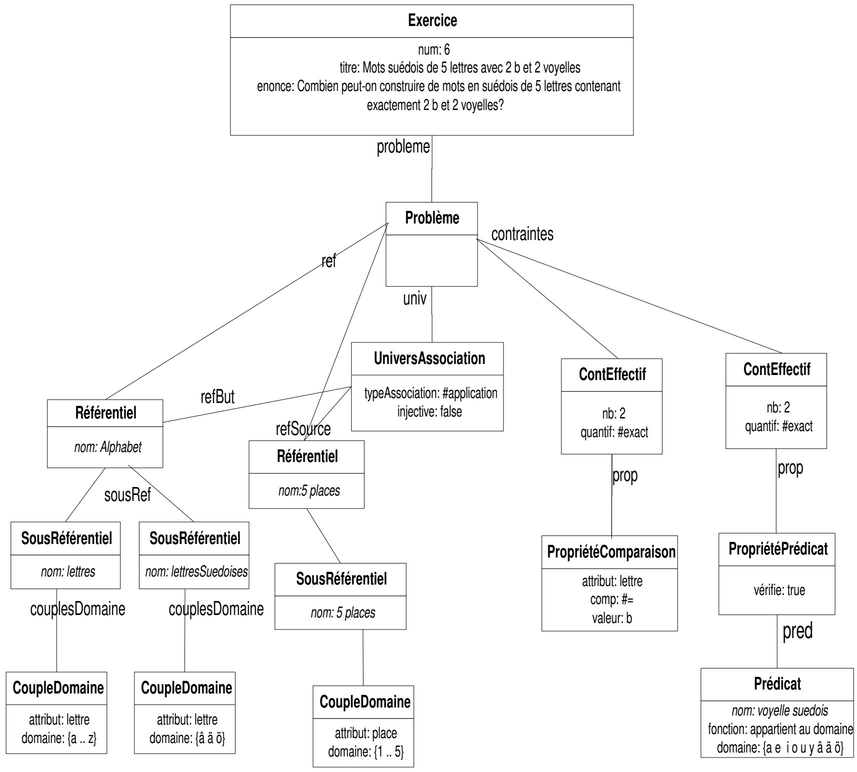

Figure 22 : Exemple 6

Cet exemple illustre les extensions d'un problème en **changeant** simplement **de référentiel**. **L'alphabet suédois** est un référentiel ayant pour sous-référentiels l'alphabet classique et les voyelles suédoises supplémentaires. Le problème utilise deux **contraintes d'effectifs**, portant respectivement sur une **propriété de comparaison** et de **prédicat**.

Les mots de 6 lettres commençant par X et contenant 3 voyelles :

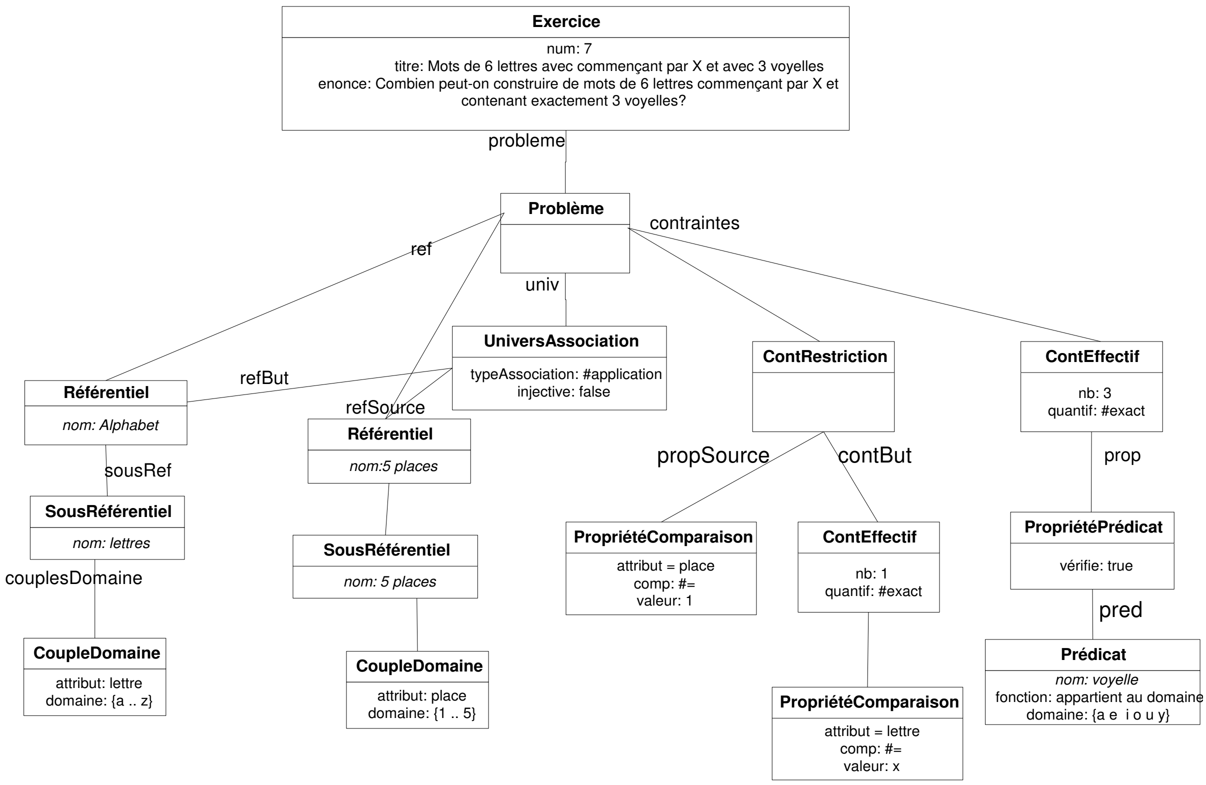

Figure 23 : Exemple 7

Cet exemple sur les **mots** de **l'alphabet** porte sur un **univers d'association** avec 2 contraintes de 2 types différents : une **contrainte d'effectifs** avec **une propriété prédicat**, et une **contrainte de restriction**, avec une **propriété de comparaison** pour les places et une **propriété de comparaison** pour les lettres correspondantes.

3.1.4. Choix des classes

Nous avons implémenté les classes du modèle conceptuel.

Méthodes d'instance

Ces classes n'ont que des méthodes d'accès et d'impression. Pour l'impression, nous avons distingué les méthodes donnant une description succincte de l'objet, utilisées pour l'affichage des objets dans l'interface, et celles décrivant l'objet de façon plus « bavarde », utilisées pour l'affichage du problème construit lors de la correction.

Méthodes de classe

Pour créer les instances des différentes classes, nous avons écrit des méthodes de classe. Par exemple, pour créer un `Référentiel` jeu de 32 cartes, il faut faire appel à la méthode de classe *`jeuNCartes: 32`* dans la classe `Référentiel`, qui va elle-même créer d'autres instances en faisant appel aux classes concernées.

Cette façon de faire est **modulaire**, et facilite la **réutilisation** : une même instance de `SousRéférentiel` peut être utilisée par différentes instances de `Référentiel`.

Les instances des objets nommés `Exercice`, `Référentiel` et `Prédicat` sont stockées dans une liste au niveau de chaque classe, afin de pouvoir donner tous les éléments existants dans des listes déroulantes de l'interface. L'ajout dans cette liste se fait lors de la création d'une instance de `ObjetNommé`.

Nous nous sommes rendu compte que ces listes comprenaient parfois des doublons. C'est pourquoi nous avons défini une méthode *`épurerListe`* dans `Application-ModelCombien`, qui les élimine.

Exemple d'extension: l'alphabet suédois

L'un des intérêts de la programmation objet étant **l'évolutivité**, nous nous sommes penchées sur le problème de l'extension du programme, dans le sens ajout d'objets semblables : on a déjà un jeu de 32 cartes, pourquoi pas un jeu de 52, par exemple.

Pour ce faire, nous avons choisi l'exemple de l'alphabet suédois, qui contient trois « nouvelles » lettres par rapport au français: å, ä et ö. (Au passage, remarquons qu'il ne s'agit pas d'accents, ce sont réellement des lettres à part entière. L'alphabet suédois contient 29 lettres au lieu de 26). De plus, il se trouve que ces trois lettres sont toutes des voyelles.

Voici comment nous avons procédé :

- Créer une nouvelle instance de `Référentiel`, constituée de l'union de deux `SousRéférentiel` : *`lettres`* et *`lettresSuédoises`*.
- Créer l'instance de `SousRéférentiel` *`lettresSuédoises`*, de valeur {å ä ö}. L'instance *`lettres`*, contenant les lettres habituelles de l'alphabet français, existe déjà, on ne fait que la réutiliser.
- Première contrainte : réutiliser l'instance de `ContEffectif` « 2 éléments tels que lettre = b ».
- Deuxième contrainte : créer une instance de `ContEffectif` « 2 éléments vérifiant la condition voyelle (en suédois) ».
- Créer une instance de `Prédicat` pour les voyelles en suédois, qui pourrait à son tour faire appel au `Prédicat` *`voyelle`*.

De façon analogue, on pourrait imaginer de créer un jeu de tarot à partir d'un jeu de 52 cartes. Il suffirait de rajouter deux `SousRéférentiel`, pour les cavaliers et les atouts, et de nouvelles contraintes, si elles portent sur les nouveaux éléments.

3.2. Implémentation Smalltalk

3.2.1. Généralités

La plupart des classes du domaine ont été dotées de la classe interface correspondante, portant le même nom plus le suffixe UI. Les classes abstraites et quelques classes sans intérêt pour l'utilisateur final font exception à cette règle.

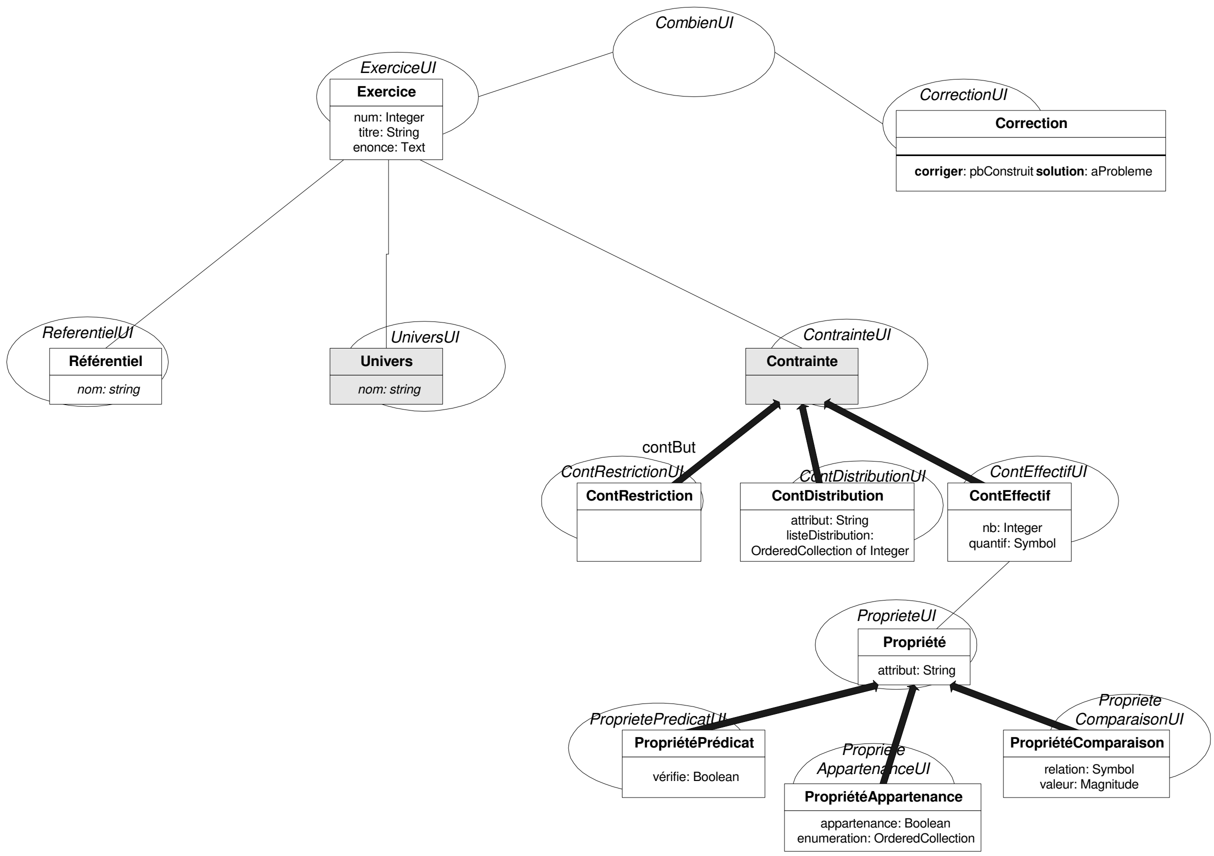

Figure 24 : Correspondance classes du domaine - classes de l'interface

Les différentes fenêtres de l'application sont reliées entre elles pour former une arborescence proche du modèle objet du domaine. Chaque fenêtre possède un « pointeur » sur sa fenêtre mère, pour pouvoir communiquer avec elle, principalement pour signaler que sa mission est terminée, mais aussi pour avoir accès à certaines données situées ailleurs dans la hiérarchie. Elle possède également une variable locale la reliant à sa classe domaine. Cette instance est reprise par la fenêtre mère lorsqu'un enfant termine son activité avec succès.

3.2.2. Initialisation de l'interface

Avant d'ouvrir la fenêtre principale, il faut initialiser le logiciel en créant les instances des exercices proposés. C'est une suite d'appels aux méthodes des classes concernées. La méthode correspondante est la méthode `initialize` dans la classe `CombienUI`.

3.2.3. Exercice

La liste de toutes les instances existantes `d'Exercice` est affichée dans la liste déroulante. Lorsque l'utilisateur appuie sur **Valider**, les vérifications d'intégrité sont lancées :

- Choix d'exercice obligatoire
  + OK : on fixe le choix d'exercice dans la variable correspondante. Le problème rattaché à cet exercice devient la solution proposée. On crée un nouveau problème, *`pbConstruit`*, qui contiendra les différents éléments à saisir tout le long du parcours dans l'application. A ce niveau, il est vide.
  - Aucun choix effectué : affichage d'un avertissement. On reste dans la fenêtre.

3.2.4. Référentiel

La liste de toutes les instances existantes de `Référentiel` est affichée dans la liste déroulante. Lorsque l'utilisateur appuie sur **Valider**, les vérifications d'intégrité sont lancées :

- Choix de référentiel obligatoire
  + OK : on fixe le choix de référentiel dans la variable correspondante, et il devient le premier référentiel du problème en cours de construction.
  - Aucun choix effectué : affichage d'un avertissement. On reste dans la fenêtre.

3.2.5. Univers

Cette fenêtre est proposée vide (mis à part l'intitulé d'une sélection, champ facultatif), pour que l'élève apprenne à bien la remplir. Lorsqu'il appuie sur **Valider**, les vérifications d'intégrité sont lancées :

- Nombre d'éléments tirés obligatoire
  + OK : on garde ce nombre pour s'en servir une fois que le type d'univers est fixé.
  - Aucun choix effectué : affichage d'un avertissement. On reste dans la fenêtre.
- Type de tirage obligatoire
  + OK : en fonction du type défini, on crée une instance `d'Univers` en conséquence:
    - ◊ tirage simultané → `UniversParties`
      Le nombre d'éléments est utilisé comme taille des parties.
    - ◊ tirage successif sans remise → `UniversAssociation`, *`injective`* = true
    - ◊ tirage successif avec remise → `UniversAssociation`, *`injective`* = false

Dans ces deux cas, le nombre d'éléments est utilisé pour créer un référentiel source d'autant de places.

Aucun choix effectué : affichage d'un avertissement. On reste dans la fenêtre.

3.2.6. Contraintes

Le panneau principal (`ContrainteUI`) contient la liste des contraintes définies, et offre la possibilité d'en ajouter et de modifier ou supprimer des contraintes existantes.

Lors d'un ajout, il faut choisir le type de contrainte. Une fois ce choix validé, le sous-panneau approprié s'affiche :

- **Contrainte d'effectif**

Ce sous-panneau contient la liste des propriétés à vérifier, et offre la possibilité d'en ajouter ou supprimer. Lors d'un ajout, il faut choisir le type de propriété. Une fois ce choix validé, on crée une instance vide correspondant, et le sous-panneau approprié s'affiche.

Quel que soit le type de propriété choisi, les actions effectuées dans le sous-panneau suivent le même principe :

◊ remplissage des champs par l'élève,
◊ validation,
◊ vérification des champs obligatoires,
◊ fermeture du panneau,
◊ mise à jour de la liste des propriétés.

Les trois sous-panneaux sont constitués de deux parties : un premier sous-panneau commun à tous, qui traite de l'attribut, et un deuxième spécifique à chaque type de propriété. Le bouton **Valider** est situé sur le panneau commun, et il appelle les validations spécifiques à la propriété concernée.

Lors de la validation de la contrainte d'effectif, la fenêtre se ferme, et la contrainte créée est affichée dans la liste des contraintes.

- **Contrainte de distribution**

Ce sous-panneau permet de définir la répartition des éléments selon un attribut. On peut noter que le fait de changer d'attribut réinitialise les deux autres listes. Lorsqu'il n'y a plus d'éléments à traiter, le bouton flèche devient inactif.

La validation entraine les mesures habituelles.

- **Contrainte de restriction**

Ce sous-panneau est le plus récent, et nous ne garantissons pas sa robustesse... En principe, il permet de définir les places sur lesquelles porte la contrainte, et les propriétés à vérifier pour les éléments à ces places.

Nous avons limité les contraintes possibles pour la configuration but à une contrainte d'effectif. Ce sous-panneau est donc très semblable à `ContEffectifUI`, il en fait d'ailleurs partie. Il faudrait éventuellement repenser l'architecture de ce panneau.

La saisie des propriétés se fait de la même façon que pour une contrainte d'effectif. La validation entraine les mesures habituelles.

Une fois toutes les contraintes nécessaires définies, la validation du sous-panneau fait apparaître la liste dans la fenêtre principale, et son contenu devient les contraintes du *`pbConstruit`*.

3.2.7. Correction

Un sous-panneau signale que la modélisation est terminée. On peut revenir sur la définition de contraintes, ou soumettre l'exercice à la correction.

La correction rudimentaire proposée ouvre une boîte de dialogue avec deux champs de texte : la modélisation construite par l'élève et la solution de l'exercice. Cet affichage utilise les méthodes *`printOnBavard`* de chaque classe concernée. Une classe `Correction` a cependant été créée, pour faciliter la suite du développement. La classe `CorrectionUI` regroupe ces deux fenêtres.

3.2.8. Principe de la validation

Chaque fenêtre est **responsable** de ses propres données : elle doit vérifier leur intégrité, et signaler à l'utilisateur si des champs obligatoires n'ont pas été correctement remplis au travers d'une boîte de dialogue.

Pendant la saisie dans une fenêtre, les différentes informations sont stockées dans une **variable temporaire**, instance de la classe domaine correspondant. Lorsque l'utilisateur appuie sur le bouton **Valider**, les tests d'existence et d'intégrité (lorsque cela est possible) sont effectués.

En cas de **succès**, la fenêtre se ferme, en signalant à sa fenêtre mère qu'elle a fini, et que son élément construit est disponible et correct. La fenêtre mère rattache l'instance qui vient d'être créée à la structure de son problème en construction.

En cas **d'échec**, la fenêtre reste ouverte pour que l'utilisateur puisse compléter ou corriger les données. Toutefois, il pourra toujours quitter la fenêtre en appuyant sur le bouton **Annuler**.

Remarque : à cause du choix d'utilisation des sous-panneaux, l'action effectuée par la fenêtre mère lors de la validation d'un enfant, consiste surtout à fermer le panneau devenu inutile. Les méthodes pour prendre en compte les changements dans les fenêtres filles sont regroupées dans le protocole `changing` dans chaque classe de l'interface.

3.2.9. Annulation

Lors de l'appui sur un bouton **Annuler**, la fenêtre doit se fermer, sans donner suite aux actions effectuées. Il faut demander à la fenêtre mère de fermer le sous-panneau annulé, sans prendre en compte l'instance créée.

Remarque : si on avait utilisé des boîtes de dialogue, cette méthode aurait été un simple *`#cancel`*, mais avec les sous-panneaux, il reste le problème de la fermeture de la fenêtre, qui ne peut être réalisée que par la fenêtre mère.

3.2.10. Réflexes

Nous avons installé trois types de réflexes :

- Sur la liste des **attributs** dans la fenêtre des **propriétés** : un changement d'attribut dans la liste recalcule la liste des valeurs que peut prendre cet attribut.
- Sur la liste des **attributs** dans la fenêtre de **contrainte de distribution** : un changement d'attribut vide la liste des tailles des groupes de la distribution
- Sur les listes de choix d'**exercice** et de **référentiel** : le fait de choisir un élément dans la liste met à jour les champs descriptifs à côté.

3.2.11. Tests sur les champs obligatoires

Pour alléger le code, les tests sur chaque champ ont été délégués à des méthodes spécialisées, qui se chargent de l'affichage des avertissements d'erreur. Ces méthodes sont généralement regroupées dans un protocole *`testing`*.

3.2.12. Widgets

A cause du principe des sous-panneaux, nous avons créé des groupes de « widgets » qui vont ensemble, pour pouvoir manipuler facilement le changement d'aspect visuel (invisibles ou grisés). Par exemple, lorsqu'un sous-panneau s'ouvre, il convient de cacher les boutons **Valider** et **Annuler** de la fenêtre précédente : ces deux boutons constitueront une liste de widgets.

Chaque classe de l'interface a ses propres listes de widgets, regroupées dans un protocole *`widgets`*, alors que les méthodes génériques *`cacher`*, *`afficher`*... sont implémentées dans la classe `ApplicationModelCombien`, mère de toutes les classes de l'interface.

3.2.13. Aide en ligne

Chaque fenêtre possède un bouton d'aide. Il existe trois types d'aide :
- **aide contextuelle**, sur l'utilisation du logiciel,
- **cours**, avec des rappels de définitions sur les concepts,
- **exemples**, qui concrétisent les concepts du cours.

Chaque classe a le choix d'implémenter un ou plusieurs types d'aide. Quant à la méthode qui affiche la fenêtre d'aide, elle est logée dans la classe `Application-ModelCombien`. Elle contient également trois messages, utilisés par les classes qui n'ont pas leurs propres messages. Nous n'avons pas implémenté la totalité des méthodes d'aide sur toutes les classes.

Cette façon de faire est **modulaire** et **évolutive**. La méthode derrière chaque bouton d'aide est la même partout, il n'y a que la méthode qui construit le message qui change d'une classe à l'autre. L'affichage de l'aide est automatiquement « normalisé », vu qu'il s'agit toujours d'une même fenêtre, dont seul le contenu change.

4. La suite...

4.1. Les améliorations

4.1.1. Tirage d'éléments

Rajouter la possibilité de tirage d'un élément de l'univers, avec ou sans respect des contraintes définies. Ceci rendrait plus concret les différents concepts, et montrerait bien l'évolution au cours de la construction. On pourrait éventuellement donner un contre-exemple, qui montre un tirage interdit.

4.1.2. La correction

La correction n'a pas été implémentée. Pour nos réflexions à ce sujet, voir § 2.2.6.

4.1.3. Création d'exercices

Au lieu de créer les exercices par méthodes de classe, on pourrait envisager de les lire à partir d'un fichier, voire de les créer directement depuis l'interface de l'application. Ces actions pourraient être regroupées dans une classe à part qui constituerait une sorte de `Livre d'exercices`, avec éventuellement pour chaque exercice des degrés de difficultés, les concepts traités etc.

4.2. Comment ajouter un nouvel exercice

Dans les classes suivantes, il faut créer des méthodes de classe :

◊ `Exercice` : titre, énoncé et problème attaché (la solution)
◊ `Problème` : référentiel, univers et contraintes
  ◊ `Référentiel` : sous-référentiels
    ◊ `SousRéférentiel` : couples-domaine
      ◊ `CoupleDomaine` : nom de l'attribut et son domaine de valeurs
  ◊ `Univers`
    ◊ `UniversParties` : référentiel but, taille parties
    ◊ `UniversAssociation` : référentiel but, référentiel source, injectivité, type d'association
  ◊ `Contrainte` :
    ◊ `ContEffectif` : nombre d'éléments, quantificateur et liste de propriétés
      ◊ `Propriété` :
        ◊ `PropriétéComparaison` : attribut, relation et valeur
        ◊ `PropriétéAppartenance` : attribut, appar-tenance et énumération
        ◊ `PropriétéPrédicat` : attribut, vérifie et prédicat
          ◊ `Prédicat` : fonction, domaine
    ◊ `ContDistribution` : attribut, liste de distribution
    ◊ `ContRestriction` : liste propriétés configuration source, liste contraintes (d 'effectif) configuration but

5. Glossaire

Interface	Implémentation	Explication, exemples
Exercice	**Exercice**	C'est un titre, un énoncé et un problème.
	Problème	C'est une modélisation du problème, contenant ses référentiels, son univers et ses contraintes.
Référentiel **Support** **Monde**	**Référentiel**	C'est le support du tirage : le jeu de cartes, les jetons, l'urne...
Univers **Sélections** **Tirages**	**Univers**	C'est l'ensemble des tirages possibles : toutes les mains de 5 cartes, tous les mots de 3 lettres...
Contrainte **Condition**	**Contrainte**	C'est une condition que doivent remplir des éléments.
Propriété	**Propriété**	C'est une propriété que doit vérifier un attribut d'un élément : couleur = vert, numéro pair...
Caractéristique	**Attribut**	C'est une caractéristique intrinsèque d'un élément : sa hauteur, sa couleur...
Condition sur les caractéristiques	**Prédicat**	C'est une condition portant sur un attribut, définie comme une sorte de fonction : voyelle, impair...

6. Conclusion

Le rapport a présenté la réalisation d'une interface de modélisation d'exercices de dénombrement pour un environnement d'aide à l'apprentissage. Ce travail illustre en particulier l'importance des modèles et principes théoriques sous-jacents pour obtenir une interface utilisable par un élève, et cela sur plusieurs niveaux : théorie mathématique (la méthode constructive de dénombrement), modèle conceptuel informatique et principes pédagogiques et ergonomiques. Il n'aurait pas été réaliste de présenter directement la méthode mathématique à l'interface bien qu'elle soit l'objet de l'apprentissage visé.

Cette implémentation est une première approche d'un type d'interface proposé à l'élève dans le système COMBIEN : les "machines à construire des configurations combinatoires" [Le Calvez & al 97] [Tisseau & al 98]. La réalisation effective de ces machines se poursuit, en utilisant un générateur d'interfaces permettant d'engendrer la majeure partie du code d'une interface à partir d'une spécification déclarative saisie interactivement à l'aide d'un "assistant".

7. Références

[Le Calvez & al 97] Le Calvez F., Urtasun M., Giroire H., Tisseau G., Duma J., Les machines à construire. Des modèles d'interaction pour apprendre une méthode constructive de dénombrement. EIAO'97, actes des cinquièmes journées EIAO de Cachan (Baron, M., Mendelsohn, P., Nicaud, J.-F., (eds), Hermès, 1997, p.49-60.

[Tisseau & al 96] Tisseau G., Giroire H., Le Calvez F., Urtasun M., Duma J., Une méthode "constructive" de résolution de problèmes de dénombrement et sa mise en oeuvre. Rapport interne Laforia 96/11, mai 1996.

[Tisseau & al 98] Tisseau G., Duma, Giroire H., Le Calvez F., Urtasun M., Les "machines à construire" : des interfaces dans le projet COMBIEN? Conception, spécification et réalisation. Actes du Colloque d'AI : Apprentissage et Acquisition de Connaissances, Berder 1998 Kornman S. . Rapport interne LIP6 98 n°7 p. 47-77.