TD 11 : Patrons de conception pour un éditeur WYSIWYG de documents

Génie Logiciel Avancé – M1 II

8–9 avril 2010

La conception d'un éditeur WYSIWYG ("What You See Is What You Get") de documents est l'étude de cas proposée par le GoF (Gamma, Helm, Johnson et Vlissides) dans leur livre "*Design Patterns*" pour illustrer l'utilisation concrète de certains patrons de conception.

Une représentation d'un document dans cet éditeur (appelé Lexi) occupe une grande zone centrale rectangulaire. Le document peut mélanger librement texte et graphiques dans toute une variété de formats (choix de polices pour le texte et format de images pour le graphiques). Le document sera entouré des menus déroulants et des ascenseurs de défilement habituels.

Exercice 1: *Structure du document*

Le choix d'une représentation interne du document a un impact sur tous les aspects de la conception. Toutes les opérations (d'édition, de formatage, d'affichage et d'analyse textuelle) nécessitent de parcourir la structure de représentation interne du document. En particulier, cette structure doit permettre :

- de maintenir la structure physique du document (organisation du texte et des graphiques en lignes, colonnes, tables, section, etc.),
- de créer et de présenter le document visuellement,
- d'établir un mécanisme de correspondance entre positions d'affichage et les éléments de la représentation interne (pour l'utilisation de la sourie),
- traiter uniformément le texte et le graphiques, les éléments composés et les éléments simples.

Pour cela, le GoF propose le patron *Composite* ayant comme racine une classe abstraite `Glyph`[1] avec les opérations ci-dessous (syntaxe Java). Les composants d'un glyphe sont indexés et accessible par un index entier.

[1] Un glyphe est un signe de l'écriture pictographique des Mayas.

d'aspect	`void Draw(Window)` `Rectangle Box()`
de positionnement	`bool Intersect(Point)`

de structure	`void Insert(Glyph,int)`
	`void Delete(Glyph)`
	`Glyph Son(i)`
	`Glyph Parent()`

Donner le diagramme de classe pour la classe **`Glyph`** *et des instances correspondant à un caractère, à un mot, à un espace, à une ligne, à un graphique rectangle et à un graphique image GIF. Pour chaque composant, esquisser le code Java des opérations de l'interface de* **`Glyph`**.

Exercice 2: *Formatage*

Pour représenter visuellement les éléments du document, il faut utiliser les paramètres de formatage de l'éditeur. De plus, plusieurs algorithmes de formatage sont possibles en fonction de la représentation visuelle : uniquement textuelle (ASCII) ou texte avec des attributs de couleurs et pagination, etc.

Le GoF propose le patron *Strategy* utilisé comme suit : les algorithmes de formatage sont abstractisés par une classe `Composer` (la *Stratégie*) qui publie deux méthodes :

- `void InstallComposition(Composition)` qui défini l'objet `Composition` sur lequel s'applique l'algorithme et
- `void Compose()` qui applique l'algorithme de formatage. La classe `Composition` est aussi un `Glyph` mais qui détient comme champ un `Composer`.

Dessiner le diagramme de classe du patron Strategy. *Utiliser ce diagramme pour notre cas d'étude.*

Exercice 3: *Décoration de GUI*

On s'intéresse maintenant à inclure la représentation visuelle du document dans une GUI simplifiée qui permet de disposer d'une bordure de page et d'un défilement à l'intérieur d'une page. Ces décorations doivent pouvoir être ajoutées ou supprimées dynamiquement.

Pour cela, le GoF propose le patron *Decorator* utilisé comme suit : les décorations d'une composition sont des glyphes spécifiques qui ont un seul composant (la composition). Ils constituent ainsi des "enveloppes transparents" de d'un glyphe.

Donner le diagramme de classe de deux décorateurs (bordure et défilement) comme une instance du patron Decorator *(vu en cours).*

Exercice 4: *Portabilité vers différents systèmes de fenêtrage*
Tout glyphe est affiché dans une fenêtre (classe abstraite `Window`), indépendamment du système de fenêtrage. Cette classe encapsule tous ce que les fenêtres réalisent dans les systèmes de fenêtrage : tracer les formes géométriques de base, rédimensionnement, minimisation/maximisation, passer en avant/arrière plan, etc. Les systèmes de fenêtrage cible offrent ces opérations avec diverses interfaces (paramètres). De plus, il est souhaitable de ne pas modifier le code du glyphe quand on change de système de fenêtrage et permettre des raffinements génériques de la fenêtre (dialogue, icône, etc.)

Pour cela, le GoF propose d'utiliser le patron de conception *Bridge* comme suit : la classe `Window` contient un membre qui fait référence à une classe abstraite `WindowImpl` ; cette dernière classe est une interface avec les systèmes de fenêtrage et doit être implémentée pour chaque système.

Donner le diagramme de classe pour l'utilisation des systèmes de fenêtrage X11, Mac et PM comme une instance du patron Bridge *(vu en cours).*

Exercice 5: *Opérations utilisateur*
Les menus de Lexi sont des glyphes qui ont associée une action (sur le document en cours d'édition ou sur les paramètres internes de l'éditeur) qui se déclenche à sa sélection (`onClic`).

Pour cela, le GoF propose d'utiliser le patron de conception *Command* comme suit : le glyphe `ElemMenu` contient un membre de la classe abstraite `Command` sur lequel il appelle l'opération `Execute` lors de son activation. C'est à l'application elle même de gérer la correspondance entre un élément de menu et la commande qui lui est associée. Lors de ce paramétrage, l'application indique les entrées et les sorties de chaque commande.

Donner le diagramme de classe pour le patron Command *et puis l'utiliser pour notre exemple afin d'obtenir des menus pour les commandes de sauvegarde, changement de police d'un glyphe sélectionné, de copier et de coller.*