*David Mas, 06/04/2001*

# Médiateur du cours JAVA

Cette étude est une tentative d'application de VDL (View Design Language) à l'EIAO (Enseignement Interactif Assisté par Ordinateur). Le principe est de modéliser un médiateur du cours JAVA en ligne de l'IUT de Montreuil ( ) et de tester la possibilité d'implémentation de ce médiateur en VDL.

Pour cela on va tout d'abord modéliser le cours en lui même, afin d'en avoir une représentation interne. Puis il faudra déterminer les usages possible du médiateur, c'est à dire lister et catégoriser les actes de langage que ce dernier devra traiter. On pourra alors chercher quelles représentations internes permettront l'implémentation de ces usages avec notre représentation du cours.

Nous avons choisi de décrire nos représentations en XML, tout au long du document seront donc fournies les DTD permetant d'implémenter les représentations spécifiées dans ce document.

## 1 Représentation du cours

### 1.1 Principe

On va chercher à segmenter le cours en "entités" de base qui représentent des unités d'information du cours.

Ces "entités" se différencient par:

1. Leur apparence à l'écran : sous forme de texte simple, de tableau, voire d'applet java.
2. le thème traité : les classes ou les opérateurs par exemple.
3. le point de vue adopté sur le thème et/ou la justification pédagogique du contenu : définition, exemple, etc...

Avec ces critères, l'étude du cours en ligne a fait apparaître 8 sortes d'entités :

1. Les concepts
2. Les entités JAVA
3. Les remarques
4. Les exemples
5. Les algorithmes
6. Les schémas et illustrations
7. Les Tables
8. Les Tables d'étude de boucle

Nous allons maintenant voir plus en détail ce que sont ces entités.

## 1.2 Liste des entités de base

Chaque entité sera présentée en trois points :

- Description de l'entité et de ses caractéristiques : son apparence et le type de sujets abordés et le point du vue sur le sujet notamment. (Remarque : comme ces entités ont été construites à partir du cours en ligne, l'apparence est une partie integrante de chaque entité, on pourrait très bien l'en enlever pour permettre une production dynamique adaptée à l'utilisateur...)
- Un exemple représentatif tiré du cours en ligne.
- La définition de la DTD associé à l'entité.

Quelques mots à propos de l'affichage : notre médiateur ne considère pas les entités dans leur ensemble mais juste quelques caractéristiques (theme, apparence, approche, ...) et les relations entre les entités. Le contenu de l'entité peut donc être indifféremment recopié dans le tag "affichage" ou simplement être représenté par son URL. Pour des raisons de commodité, et puisque cela ne change en rien notre vision du médiateur, nous représenterons toujours le contenu par le mot clé HTML.

### 1.2.1 Concept de programmation Objet

#### 1.2.1.1 description

Ceci correspond aux définitions de concepts génériques de programmation (indépendants de l'implémentation en JAVA).

L'apparence d'un concept est du texte HTML simple.

Les informations qui nous intéressent sont :

- le nom du concept, pour pouvoir s'y référer.
- les pré-requis, c'est à dire les notions sur lesquelles il s'appuie et qui sont nécessaires à sa compréhension, c'est à dire en pratique les entités qui contiennent ces notions.
- la difficulté du concept, définie sur une échelle (facile, moyen ou difficile) qui évalue le niveau des élèves auquel s'adresse ce concept.

#### 1.2.1.2 définition

<!ELEMENT **concept** (affichage, infos)>

<!ELEMENT **affichage** (#PCDATA)>

<!ELEMENT **infos** (nom, prerequis*, difficulte)>

<!ELEMENT **nom** (#PCDATA)>

<!ELEMENT **prerequis** (#PCDATA)>

<!ELEMENT **difficulte** (#PCDATA)>

#### 1.2.1.3 exemple

**Principe (simplifié) de la programmation orientée objet**

La programmation *orientée objet* peut se définir comme l'art de décomposer une application en un certain nombre d'objets qui ont des caractéristiques communes. Le but est de réduire la difficulté de la tâche à accomplir en la divisant en un grand nombre de petits problèmes qui sont plus simples à comprendre et à résoudre.

Le regroupement des variables concernant un même objet permet de gérer plus facilement un grand nombre d'objets ayant chacun un grand nombre de variables.

La programmation *orientée objet* peut être utilisée dans de nombreux domaines :

- Gérer les contrats d'assurance voiture d'une société d'assurance
- Gérer les prêts dans une bibliothèque
- Gérer les achats d'un magasin
- Gérer la paye du personnel d'une entreprise
- Développer un logiciel de dessin
- ...

exemple de définition de concept

<**concept**>
<**affichage**>HTML</**affichage**>
<**infos**>
<**nom**>Principe (simplifié) de la programmation objet</**nom**>
<**prerequis**>programmation fonctionnelle</**prerequis**>
<**difficulte**>simple</**difficulte**>
</**infos**>
</**concept**>

### 1.2.2 Notion JAVA

#### 1.2.2.1 description

Ce sont toutes les définition des notions de JAVA (variables, classes, opérateurs...).

Elles sont présentées sous forme de tableau à trois entrées maximum : but, syntaxe, exemple. Déjà ici on peut se poser des questions sur la granularité de notre représentation : doit-on considérer cette entité dans son ensemble, ou bien comme un agrégat d'entité plus petites, i.e. but, syntaxe, et exemple. Tout dépend des besoins, dans ce document on recherche a poser les bases d'un médiateur, on se contentera donc d'un grain assez gros.

Les informations intéressantes sont :

- le nom de l'entité
- son code JAVA
- les pré-requis (c.f 1.2.1)
- sa difficulté (c.f 1.2.1)

### 1.2.2.2 définition

```
<!ELEMENT java (affichage, infos)>
<!ELEMENT affichage (#PCDATA)>
<!ELEMENT infos (nom, nomJava, prerequis+, difficulte)>
    <!ELEMENT nom (#PCDATA)>
    <!ELEMENT nomJava (#PCDATA)>
    <!ELEMENT prerequis (#PCDATA)>
    <!ELEMENT difficulte (#PCDATA)>
```

### 1.2.2.3 exemple

**Les opérateurs relationnels**

| BUT | Comparer deux valeurs et déterminer leur relation. |
|---|---|
| SYNTAXE | opérande1 opérateur opérande2 |
| EXEMPLE | ... if (score != SCORE_MAX) ... |

Exemple de définition d'une entité JAVA

```
<java>
  <affichage>HTML</affichage>
  <infos>
    <nom>Opérateur relationnel</nom>
    <nomJAVA><, >, <=, >=, ==, !=</nomJAVA>
    <prerequis>variable</prerequis>
    <prerequis>constantes</prerequis>
```

```
        <difficulte>moyen</difficulte>
    </infos>
</java>
```

## 1.2.3 Remarque

### 1.2.3.1 description

Tout texte du cours qui n'appartient pas à une des deux catégories ci-dessus, notamment les avertissements

L'apparence peut être du texte HTML simple précédé d'un titre ou comme ici un avertissement sous forme de tableau.

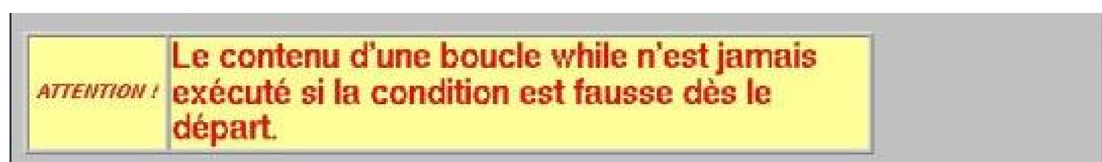
Les informations attachées sont :

- le type de la remarque : avertissement, truc/astuce, etc..
- la cible, c'est à dire l'entité du cours à laquelle elle se rapporte.
- la difficulté (c.f. 1.2.1)

### 1.2.3.2 définition

```
<!ELEMENT remarque (affichage, infos)>
<!ELEMENT affichage (#PCDATA)>
<!ELEMENT infos (type, cible, difficulte)>
    <!ELEMENT type (#PCDATA)>
    <!ELEMENT cible (#PCDATA)>
    <!ELEMENT difficulte (#PCDATA)>
```

### 1.2.3.3 exemple




exemple de remarque

```
<remarque>
    <affichage>HTML</affichage>
```

```
    <infos>
        <type>avertissement</type>
        <cible>Notion de boucle while en JAVA</cible>
        <difficulte>simple</difficulte>
    </infos>
</remarque>
```

## 1.2.4 Exemple

### 1.2.4.1 description

Un exemple de programme en JAVA.

Présenté par un titre suivi du code du programme et des images d'écrans de compilation et d'exécution. Il serait plus intéressant d'avoir juste le code en JAVA et de pouvoir l'exécuter vraiment à l'aide d'un Applet qui réaliserait la compilation et l'exécution, ce que représente la requête "exécuter".

Les informations intéressantes sont :

- le nom de l'exemple
- la cible de l'exemple (c.f 1.2.3)
- la difficulté (c.f 1.2.1)

### 1.2.4.2 définition

```
<!ELEMENT exemple (affichage, infos)>

<!ELEMENT affichage (#PCDATA)>

<!ELEMENT requetes (executer)>
    <!ELEMENT executer (#PCDATA)>

<!ELEMENT infos (nom, cible, difficulte)>
    <!ELEMENT nom (#PCDATA)>
    <!ELEMENT cible (#PCDATA)>
    <!ELEMENT difficulte (#PCDATA)>
```

### 1.2.4.3 exemple

**Exemple :**

```
/* Boucle.java : Ce programme affiche les nombres de 1 à 5*/
class Boucle {
    public static void main (String arg[]) {
        int nb ;
        nb = 1 ;
        while (nb != 5)  {
                System.out.println (nb);
                nb = nb + 1 ;
        }
        System.out.println ("Fin du programme");
    }
}
```

exemple d'exemple

<**remarque**>
<**affichage**>HTML</**affichage**>
<**requetes**>
<**executer**>exécution du programme boucle.java dans une applet, avec visualisation du résultat et des logs.</**executer**>
</**requetes**>
<**infos**>
<**nom**>Boucle.java</**nom**>
<**cible**>Notion de boucle while en JAVA.</**cible**>
<**difficulte**>simple</**difficulte**>
</**infos**>
</**remarque**>

## 1.2.5 Schéma / illustration

### 1.2.5.1 description

Tous les schémas, dessins ou autres illustration du cours.

C'est en général une image (gif ou jpeg) affichée dans du HTML

Informations utiles:

- le titre du schéma
- la légende du schéma

- la cible du schéma (c.f. 1.2.3)
- la difficulté (c.f 1.2.1)

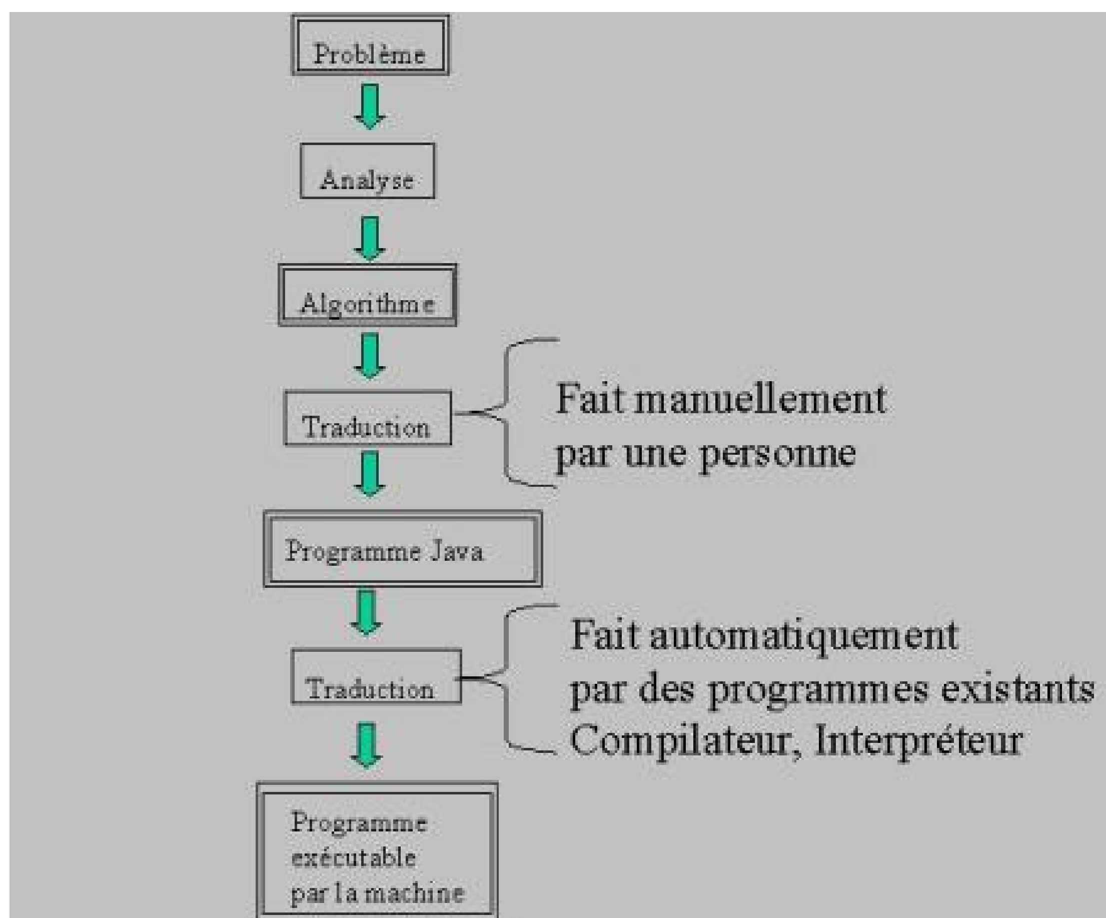
### 1.2.5.2 définition

<!ELEMENT **schema** (affichage, infos)>

<!ELEMENT **affichage** (#PCDATA)>

<!ELEMENT **infos** (titre, legende, cible, difficulte)>
<!ELEMENT **titre** (#PCDATA)>
<!ELEMENT **legende** (#PCDATA)>
<!ELEMENT **cible** (#PCDATA)>
<!ELEMENT **difficulte** (#PCDATA)>

### 1.2.5.3 exemple




exemple de schéma

<**schema**>

```
        <affichage>Image</affichage>
        <infos>
            <titre>Étapes de la programmation</titre>
            <legende></legende>
            <cible>La programmation comprend plusieurs étapes</cible>
            <difficulte>simple</difficulte>
        </infos>
    </schema>
```

## 1.2.6 Algorithme

### 1.2.6.1 description

Ce sont les algorithmes donnés en pseudo Language et donc non exécutables.

Généralement le listing est en texte HTML simple.

Les informations intéressantes sont :

- le nom de l'algorithme
- la difficulté (c.f. 1.2.1)

### 1.2.6.2 description

```
<!ELEMENT algorithme (affichage, infos)>
<!ELEMENT affichage (#PCDATA)>
<!ELEMENT infos (nom, difficulte)>
    <!ELEMENT nom (#PCDATA)>
    <!ELEMENT difficulte (#PCDATA)>
```

### 1.2.6.3 exemple

# Exemple sans sous-programme

## Algorithme

Exemple

```
Algorithme Repas
( Saisit le prix de deux plats et affiche le prix maximum ;
saisit le prix de deux boissons et affiche le prix maximum )
Variable

    prixPlatViande, prixPlatPoisson : réel   ( prix des plats )
    prixBoissonJus, prixBoissonVin : réel    ( prix des boissons )

Début

    ( Saisir le prix des plats )
    afficher ("prix du plat Viande : ")
    saisir (prixPlatViande)
    afficher ("Prix du plat Poisson : ")
    saisir (prixPlatPoisson)
    afficher ("Le plat le plus cher vaut : ")

    ( calcul du max des deux prix )
    Si prixPlatViande >= prixPlatPoisson Alors
        afficher (prixPlatViande)
    Sinon
        afficher (prixPlatPoisson)
    FinSi

    ( Saisir le prix des boissons )
    afficher ("prix de la boisson jus de fruits : ")
    saisir (prixBoissonJus)
    afficher ("prix de la boisson alcoolisee : ")
    saisir (prixBoissonVin)
    afficher ("La boisson la plus chère vaut : ")

   ( calcul du max des deux prix )
    Si prixBoissonJus >= prixBoissonVin Alors
        afficher (prixBoissonJus)
    Sinon
        afficher (prixBoissonVin)
    FinSi
FinAlgorithme
```

exemple d'algorithme

```
<algorithme>
   <affichage>HTML</affichage>
   <infos>
      <nom>Repas</nom>
      <difficulte>simple</difficulte>
   </infos>
</algorithme>
```

## 1.2.7 Table

### 1.2.7.1 description

Les Tables listant des éléments de JAVA. (ex: tables des types de variables).

Affichées sous forme de tableau HTML.

Les Informations intéressantes sont :

- le nom de la table
- les pré-requis (c.f 1.2.1)
- la difficulté (c.f 1.2.1)

Les tables posent un problème de granularité : on pourrait vouloir les représenter plus finement pour pouvoir accéder à chaque case de la table indépendamment, ou bien avoir la notion de ligne et de colonne pour faire des recherche au sein de la table, etc ... Notre modélisation n'est pas contraignante à ce sujet, si l'on souhaite un granularité plus fine il suffit donc de l'implémenter. Il faut juste se poser la question de l'utilité d'un grain fin pour les tables.

Le côté intéressant de cette question est le principe d'une modélisation à grain variable, c'est à dire avec des possibilités de zoom sur certain points. Cela permet d'adapter la granularité aux besoin du médiateur sans avoir à implémenter le même niveau de détails partout, mais augmente bien sûr le travail de modélisation.

#### 1.2.7.2 définition

<!ELEMENT **table** (affichage, infos)>

<!ELEMENT **affichage** (#PCDATA)>

<!ELEMENT **infos** (nom, difficulte)>
<!ELEMENT **nom** (#PCDATA)>
<!ELEMENT **difficulte** (#PCDATA)>

#### 1.2.7.3 exemple

**Opérateurs relationnels existants :**

| Operateur | Utilisation | Rend true si : |
|---|---|---|
| > | op1 > op2 | op1 est strictement supérieur à op2 |
| >= | op1 >= op2 | op1 est supérieur ou égal à op2 |
| < | op1 < op2 | op1 est strictement inférieur à op2 |
| <= | op1 <= op2 | op1 est inférieur ou égal à op2 |
| == | op1 == op2 | op1 et op2 sont égaux |
| != | op1 != op2 | op1 et op2 ne sont pas égaux |

exemple de table

```
<table>
   <affichage>HTML</affichage>
   <infos>
      <nom>Opérateurs relationnels existants</nom>
      <difficulte>simple</difficulte>
   </infos>
</table>
```

## 1.2.8 Table d'étude de boucle

### 1.2.8.1 description

Ce sont les tables de questions permettant la conception d'une boucle.

Présentées sous forme d'une table avec les questions posées en colonne 1, les réponses en colonne 2.

Informations intéressantes sont :

- le titre de la table
- la difficulté : au choix facile, moyen ou difficile

Cette entité est une entité ad-hoc  pour ce cours. On pourra en créer d'autres pour tout autre outil de conception similaire défini plus tard dans le cours (pas encore en ligne), voire pour d'autres besoins spécifiques du cours.

### 1.2.8.2 définition

<!ELEMENT **tableEtudeBoucle** (affichage, infos)>

<!ELEMENT **affichage** (#PCDATA)>

<!ELEMENT **infos** (titre, difficulte)>
<!ELEMENT **titre** (#PCDATA)>
<!ELEMENT **difficulte** (#PCDATA)>

### 1.2.8.3 exemple

**Etude du problème**

| | |
|---|---|
| Exemples de ce que l'on veut obtenir | Saisie du nombre 2<br>Affichage de **<br><br>Saisie du nombre 5<br>Affichage de *****<br><br>Saisie du nombre 10<br>Affichage de ********** |
| Est-ce qu'il y a un traitement répétitif ?<br>Si oui, lequel ? | Oui.<br>On répète l'affichage du caractère * |
| Qu'est-ce qui varie d'une répétition à une autre ? | Le nombre d'étoiles déjà affiché |
| Qu'est-ce qui ne varie pas d'une répétition à une autre ? | On affiche toujours une seule étoile à chaque fois |
| Quelle est la condition d'arrêt de la répétition ? | On a affiché n * où n est le nombre saisi par l'utilisateur |
| Quel est le traitement à faire avant les répétitions ? | Saisir le nombre n |
| Quelles sont les informations qu'il faut mémoriser ? | Le nombre saisi correspondant au nombre d'étoiles à afficher.<br>Le nombre d'étoiles affichées jusqu'à maintenant. |

exemple de table d'étude de boucle

<**tableEtudeBoucle**>
<**affichage**>HTML</**affichage**>
<**infos**>
<**titre**>afficher une ligne d'étoiles : étude</**titre**>
<**difficulte**>simple</**difficulte**>
</**infos**>
</**tableEtudeBoucle**>

# 2 Usages du médiateur

Les usages du médiateur sont sa raison d'être, dans notre cas on ne s'intéresse qu'aux actes de langages d'interaction avec le médiateur.

Pour étudier ces actes de langages nous allons utiliser un modèle de requête dont voici la DTD :

```
<!ELEMENT requete EMPTY>
  <!ATTLIST requete
    type (executer|demander)
    operateur CDATA #REQUIRED
    arguments CDATA #IMPLIED
  >
```

On distingue deux types de requêtes différentes: celles demandant implicitement ou explicitement l'affichage d'une nouvelle partie du cours (executer), et celles demandant des informations directement sur l'endroit du cours ou l'on se trouve (demander).

Les premières représentent différent moyens de navigation au sein du cours, les secondes permettent à l'utilisateur de se situer dans la représentation associée à l'usage et donc de s'orienter.

Les opérateurs peuvent être utilisés dans des sens différents suivant le type de la requête. Ils marquent en fait ce qui est pris en compte pour répondre à la requête. Le type et le nombre des arguments dépendant directement des opérateurs.

Un même argument peut être de différent type : soit vide, il représente alors l'acceptation standard, soit un mot-clé représentant un élément du médiateur, soit une chaine permettant de retrouver un élément.

Les usages que nous proposons sont :

1. Navigation suivant le plan du cours
2. Navigation au sein d'un thème
3. Navigation sémantique
4. Accès aux pré-requis
5. Progression dans la difficulté /la précision
6. Historique

## 2.1 Navigation dans le cours

C'est l'usage courant d'une personne qui lit un cours. On peut l'assimiller à la lecture d'un livre. L'utilisateur se déplace dans une structure qu'il connaît (l'organisation en chapitre puis section etc..) et qu'il manipule explicitement ou implicitement.

### 2.1.1 exemples de requêtes

- "voir la suite" <**requete** *type*="executer" *operateur*="suivant" *arguments*="page" />
- "revoir le point précédent" <**requete** *type*="executer" *operateur*="precedent" *arguments*="page" />
- "aller au chapitre suivant" <**requete** *type*="executer" *operateur*="suivant" *arguments*="chapitre" />
- "aller au debut du chapitre" <**requete** *type*="executer" *operateur*="debut" *arguments*="chapitre" />
- "aller au chapitre 3" <**requete** *type*="executer" *operateur*="nieme" *arguments*="chapitre, 3" />
- "aller au chapitre 'opérateurs binaires'" <**requete** *type*="executer" *operateur*="titre" *arguments*="chapitre, 'opérateurs binaires'" />
- "aller à la fin du cours" <**requete** *type*="executer" *operateur*="fin" *arguments*="cours" />
- "quel est le titre du chapitre courant" <**requete** *type*="demander" *operateur*="titre" *arguments*="chapitre" />
- "quel est le sujet du chapitre suivant" <**requete** *type*="demander" *operateur*="sujet" *arguments*="chapitre, suivant" />
- "quel est le sujet du chapitre 3" <**requete** *type*="demander" *operateur*="sujet" *arguments*="chapitre, 3" />

## 2.2 Navigation au sein d'un même thème

Souvent dans un cours il existe plusieurs entités qui traitent du même sujet mais d'un point de vue différent (ex: une définition d'une entité java et un exemple) et que l'utilisateur peut être amené à vouloir passer de l'un à l'autre indépendamment de l'ordre imposé par le cours.

### 2.2.1 exemples de requêtes

- "puis-je voir un exemple ?" <**requete** *type*="executer" *operateur*="entitéLiée" *arguments*="exemple" />

- "puis-je voir un exemple de déclaration de classe abstraite ?" <**requete** *type*="executer" *operateur*="entitéLiéé" *arguments*="exemple, déclaration de classe abstraite" />

- "y'a t'il un exemple de déclaration de classe abstraite ?" <**requete** *type*="demander" *operateur*="entitéLiée" *arguments*="exemple, 'classe abstraite'" />

- "y'a t'il d'autres exemples ?" <**requete** *type*="demander" *operateur*="entitéLiée" *arguments*="exemple" />

- "je peux revoir la définition ?" <**requete** *type*="executer" *operateur*="entitéLiée" *arguments*="notionJava" />

## 2.3 Recherche sémantique

C'est un usage qui considère non plus le cours en lui même mais l'objet du cours, c'est à dire dans notre cas le langage JAVA et la programmation objet.

Cet usage se manifeste dans deux type d'interaction : la recherche directe d'une information dont on sait ou on présume qu'elle est présente dans le cours, ou la demande d'information complémentaire sur un thème présenté par l'entité consultée.

### 2.3.1 exemple de requêtes

- "je voudrais en savoir plus..." <**requete** *type*="executer" *operateur*="sème" />

- "qu'y a t'il de disponible sur les pointeurs" <**requete** *type*="demander" *operateur*="sème" *arguments*="pointeurs" />

- "je voudrais en savoir plus sur les opérateurs relationnels" <**requete** *type*="executer" *operateur*="sème" *arguments*="opérateurs, relationnels" />

- "qu'est ce qu'une classe ?" <**requete** *type*="executer" *operateur*="sème" *arguments*="classe" />

## 2.4 Accès aux pré-requis

Cet usage est destiné a l'élève qui n'aurait pas bien assimilé une notion nécessaire à la compréhension du sujet qui lui est présenté ou à celui qui suit un parcours personnalisé et qui aurait raté un principe important.

### 2.4.1 exemples de requêtes

- "quelles sont les notions nécessaires à la compréhension de ce passage ?" <**requete** *type*="demander" *operateur*="prerequis" />
- "je ne comprends pas..." <**requete** *type*="executer" *operateur*="prerequis" />
- "c'est quoi un opérateur binaire ?" <**requete** *type*="demander" *operateur*="prerequis" *arguments*="opérateur binaire" />
- "hum, je n'aurais pas raté quelque chose d'important ?" <**requete** *type*="demander" *operateur*="prerequis" />

## 2.5 Progression dans la difficulté/la précision

Encore un usage qui permet une utilisation personnalisée du cours. Celui-ci permet d'accéder à des parties du cours plus complexes mais sur le même sujet que celui consulté.

on pourrait considérer que la difficulté et la précision sont deux concepts différents. Mais il me semble que l'on progresse dans la difficulté en s'intéressant aux détails et qu'il y a en quelque sorte un parallélisme entre les deux notions qu'il faut conserver dans le médiateur si l'on veut s'y retrouver dans les actes de langages.

### 2.5.1 exemples de requêtes

- "bien compris, je pourrais voir ça plus en détail?" <**requete** *type*="executer" *operateur*="difficulté" *arguments*="plus" />
- "existe-t'il une version plus précise ?" <**requete** *type*="demander" *operateur*="difficulte" *arguments*="plus" />

- "pourrais-je voir une version plus simple ?" <**requete** *type*="executer" *operateur*="difficulte" *arguments*="moins" />
- "puis-je voir une version plus complexe ?" <**requete** *type*="executer" *operateur*="difficulte" *arguments*="plus" />
- "quelle est la difficulté de ce passage ?" <**requete** *type*="demander" *operateur*="difficulte" *arguments*="courant" />

## 2.6 Historique

C'est la gestion du passé de l'utilisateur, qui peut vouloir revenir sur ces pas ou savoir ce qu'il a déjà fait et ce qui lui reste a faire.

### 2.6.1 exemple de requêtes

- "revenir en arrière" <**requete** *type*="executer" *operateur*="passe" />
- "retouner en avant" <**requete** *type*="executer" *operateur*="futur" />
- "ai-je déjà vu ce chapitre ?" <**requete** *type*="demander" *operateur*="passe" *arguments*="chapitre,courant" />

# 3 Les relations

Chaque usage met en évidence une relations entre les entités de base du cours. Ces relations peuvent être soit externes, i.e. dépendantes d'informations non portées par les entités, ou internes ou encore un combinaison des deux.

On distingue donc :

1. Des relations externes, qui seront représentée par des marco-stuctures, i.e. on va rajouter des entités dans notre modèle pour les représenter, c'est typiquement le cas d'un arbre ou d'un graphe :

- Le plan du cours
- Les fagots, qui regroupent des entités autour d'un même thème
- l'ontologie des notions Java
- Historique

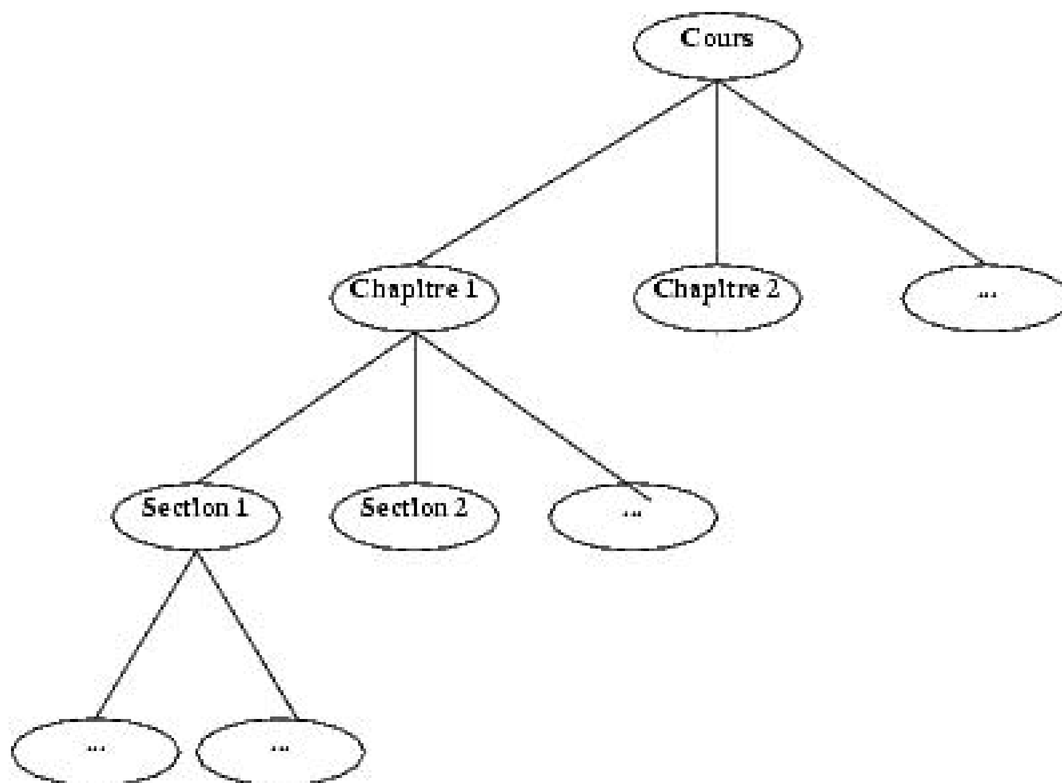
2. Des relations internes, qui sont calculées à partir des attributs des entités :
   - Relation de pré-requis

3. Des relation mixtes, calculées sur les macro-structures  à partir des attributs des entités :
   - Progression dans la difficulté/la précision

L'usage "historique" impose une prise en compte de l'utilisateur qui est un cas à part qui sera traité dans le chapitre suivant.

## 3.1 Les macro-structures

### 3.1.1 Plan du cours




Plan du cours

Cette relation est typiquement externe. Sa représentation la plus naturelle est un

arbre équilibré, comportant plusieurs niveaux de hierachies et se terminant par les entités.

Le problème posé par cette relation serait plutôt au niveau de la nomination des noeuds et du nombres de niveaux hiérarchiques que l'on permet dans la modélisation. Je vais donc décider arbitrairement de ces deux points, cela pouvant être remis en cause suivant les sensibilités personnelles de chacun.

J.C Martin souhaitait que le cours puisse s'adresser à plusieurs niveaux d'étude, rien n'empêche de faire une structure de cours par niveau.

Les opérateurs à réaliser pour cette relation sont de simples opérateurs de parcours d'arbre en pronfondeur. Ainsi suivant et précédent se comprend simplement dans la relation d'ordre sur un arbre étiqueté par un parcours en profondeur se limitant au niveau hiearchique précisé par l'argument. L'accès au attributs des noeuds n'est pas plus difficile, il suffit de localiser le noeud comme précédemment et de lire l'attribut cherché.

#### 3.1.1.1 modélisation

**Plan du cours**

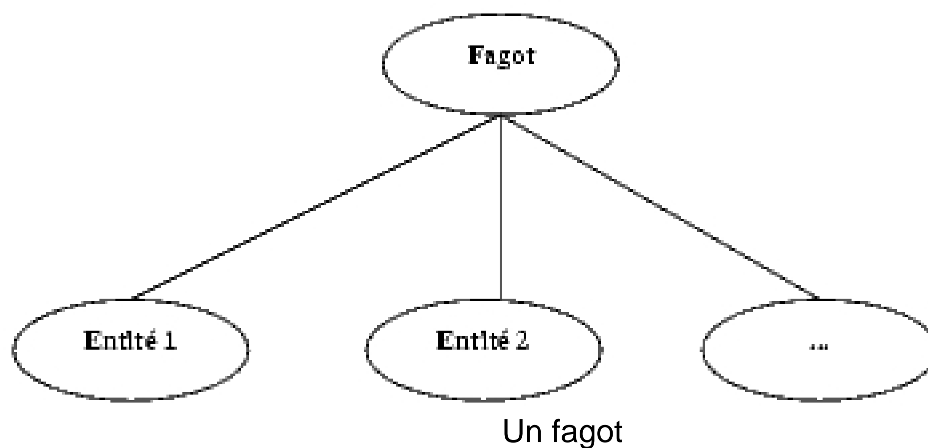
```
<!ELEMENT cours (infos, (chapitre)+)>
    <!ELEMENT infos (titre, niveau)>
        <!ELEMENT titre (#PCDATA)>
        <!ELEMENT niveau (#PCDATA)>
<!ELEMENT chapitre (infos, (section|entite)+)>
    <!ELEMENT infos (titre)>
        <!ELEMENT titre (#PCDATA)>
<!ELEMENT section (infos, (sous-section|entite)+) >
    <!ELEMENT infos (titre)>
        <!ELEMENT titre (#PCDATA)>
<!ELEMENT sous-section  (infos, (entite)+)>
    <!ELEMENT infos (titre)>
        <!ELEMENT titre (#PCDATA)>
<!ELEMENT entite (infos, ()+)>
    <!ELEMENT infos (type, reference)>
        <!ELEMENT type (#PCDATA)>
        <!ELEMENT reference (#PCDATA)>
```

#### 3.1.1.2 exemple

```
<cours>
    <infos>
```

```
        <titre>Cours JAVA -  IUT</titre>
        <niveau>1ère année</niveau>
    </infos>
    <chapitre>
        <infos>
            <titre>Introduction Générale</titre>
        </infos>
        <section>
            <infos>
                <titre>Principes généraux de la programmation</titre>
            </infos>
            <entite>
                <infos>
                    <type>concept</type>
                    <reference>lien vers la définition du concept : "les
                    différentes étapes de programmation"</reference>
                </infos>
            </entite>
            <entite>
                <infos>
                    <type>schema</type>
                    <reference>url du schéma présentant les différentes étapes
                    du la programmation</reference>
                </infos>
            </entite>
            ...
        </section>
        <section>
            <infos>
                <titre>Un premier exemple de programme Java</titre>
            </infos>
            ...
        </section>
    </chapitre>
    <chapitre>
        <infos>
            <titre>Introduction au langage Java</titre>
        </infos>
        ...
    </chapitre>
    ...
</cours>
```

### 3.1.2 Les fagots




Un fagot

Pour implémenter l'idée de navigation dans un même thème, je propose des fagots portant le thème et pointant vers chaque élément qui s'y rapporte. Le cas le plus classique de fagot est une définition de notion Java avec un ou plusieurs exemples et une ou plusieurs remarques.

Les opérateur de cette relation est très simple, il suffit juste d'être capable d'accéder à chaque entité du fagot pour répondre à toutes les requêtes de l'utilisateur.

#### 3.1.2.1 modélisation

**boucles locales**

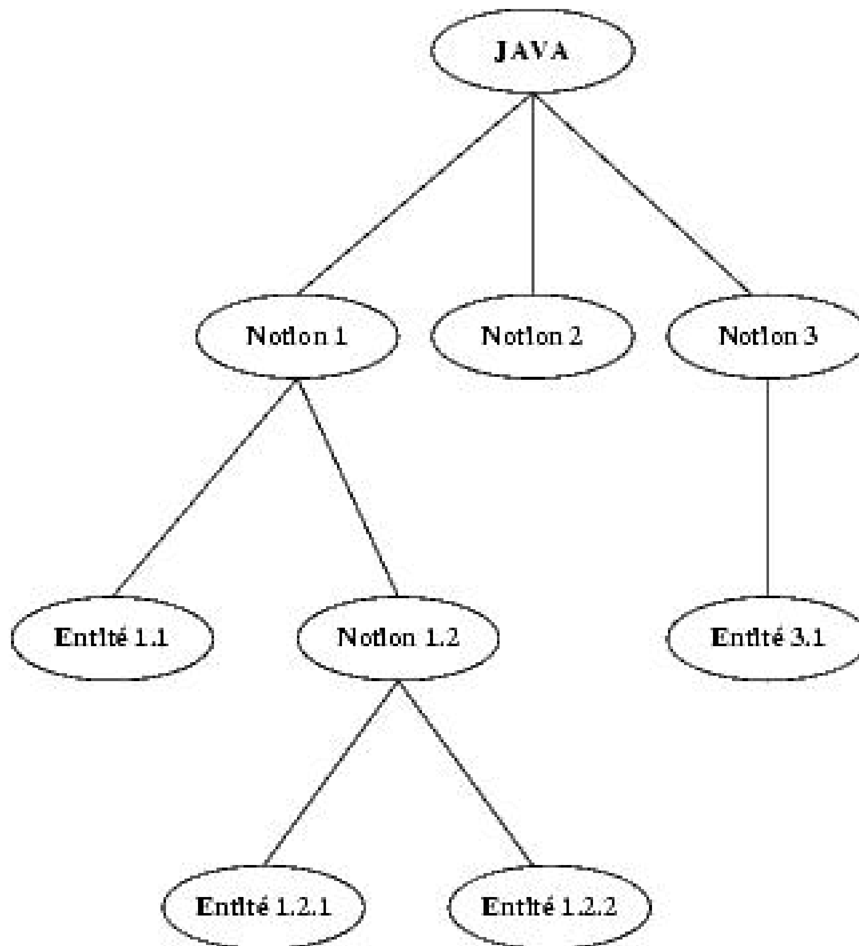
```
<!ELEMENT fagot (infos, (entite)+)>
   <!ELEMENT infos (theme)>
      <!ELEMENT theme (#PCDATA)>
<!ELEMENT entite (infos, ()+)>
   <!ELEMENT infos (type, reference)>
      <!ELEMENT type (#PCDATA)>
      <!ELEMENT reference (#PCDATA)>
```

#### 3.1.2.2 exemple

```
<fagot>
   <infos>
      <theme>les variables</theme>
   </infos>
   <entite>
      <infos>
         <type>java</type>
         <reference>lien vers la notion java "déclaration d'une
         variable"</reference>
      </infos>
```

```xml
        </entite>
        <entite>
            <infos>
                <type>commentaire</type>
                <reference>lien vers une remarque sur les conventions de
                nomnage des variables</reference>
            </infos>
        </entite>
        <entite>
            <infos>
                <type>table</type>
                <reference>lien vers la table des types de variables en
                JAVA</reference>
            </infos>
        </entite>
        <entite>
            <infos>
                <type>exemple</type>
                <reference>lien vers l'exemple "jeu.java" qui affiche le nombre de
                joueurs choisis</reference>
            </infos>
        </entite>
    </fagot>
    ...
```

### 3.1.3 L'ontologie des notions Java




Ontologie des notions Java

L'ontologie des notions Java se différencie structurellement du plan du cours par sa profondeur variable. De plus les usages qui y sont associés sont plus complexes.

Ainsi les opérateurs vus en 2.3.1 impliquent le parcours systématique de l'arbre pour y retrouver le concept cherché. L'usage particulier de l'opérateur sans arguments correspond à une généralisation c'est à dire le listage de tous les concepts ayant le même père que l'entité courante.

### 3.1.3.1 modélisation

**Arbre sémantique**

<!ELEMENT **java** (infos, (noeud|entite)+)>

<!ELEMENT **infos** (seme)>
<!ELEMENT **seme** (#PCDATA)>

<!ELEMENT **noeud** (infos, (noeud|entite)+)>

<!ELEMENT **infos** (seme)>
<!ELEMENT **seme** (#PCDATA)>

<!ELEMENT **entite** (infos, ()+)>

```
<!ELEMENT infos (type, reference)>
   <!ELEMENT type (#PCDATA)>
   <!ELEMENT reference (#PCDATA)>
```

**3.1.3.2 exemple**

```
<java>
   <infos>
      <seme>java</seme>
   </infos>
   <noeud>
      <infos>
         <seme>variables</seme>
      </infos>
      <noeud>
         <infos>
            <seme>déclaration</seme>
         </infos>
         <entite>
            <infos>
               <type>java</type>
               <reference>lien vers la notion java "déclaration des
               variables"</reference>
            </infos>
         </entite>
         <entite>
            <infos>
               <type>exemple</type>
               <reference>lien vers l'exemple "jeu.java"</reference>
            </infos>
         </entite>
      </noeud>
      ...
   </noeud>
   <noeud>
      <infos>
         <seme>classes</seme>
      </infos>
      ...
   </noeud>
   ...
</java>
```

## 3.2 Les relations calculées

### 3.2.1 Pré-requis

Voici le un cas où l'usage rend le calcul de la relation intéressant. En effet la plupart des requêtes (voire même toutes) vont s'effectuer à partir de l'élément courant vers ceux dont il dépend directement. Donc si l'on ajoute à tous les éléments des champs de pré-requis  on simplifie la saisie sans pénaliser la rapidité d'accès.

Cette relation est donc entièrement contenu dans son opérateur, qui ne fait que renvoyer suivant le type de la requête la liste des pré-requis  ou directement la page correspondant au pré-requis  demandé.

#### 3.2.1.1 exemple

```
<java>
   <affichage>HTML</affichage>
   <infos>
      <nom>Opérateur relationnel</nom>
      <nomJAVA><, >, <=, >=, ==, !=</nomJAVA>
      <prerequis>tests</prerequis>
      <prerequis>variable</prerequis>
      <prerequis>constantes</prerequis>
      <difficulte>moyen</difficulte>
   </infos>
</java>
```

## 3.3 Les relations mixtes

### 3.3.1 Progression dans la difficulté/le détail

Cette relation est mixte car je la considère comme le calcul d'un ordre local sur des entités reliées par le même thème, c'est à dire contenues dans le même fagot (c.f. 3.1.2). Ainsi on récupère tout d'abord le fagot comprenant l'entité courante avant de pouvoir trouver les éléments demandés.

L'implémentation de cette relation suppose que l'on possède pour chaque thème des informations destinées à plusieurs niveaux d'étudiants. L'opérateur faisant appel à la fois a une macro-structure  et à du calcul, il sera sûrement très

intéressant de garder le résultat de la première requête dans une mémoire cache tant que l'on reste dans le même fagot.

#### 3.3.1.1 exemple

```
<concept>
   <affichage>HTML</affichage>
   <infos>
      <nom>Principe (simplifié) de la programmation objet</nom>
      <prerequis>programmation fonctionnelle</prerequis>
      <difficulte>simple</difficulte>
      <detail>général</detail>
   </infos>
</concept>
```

# 4 Prise en compte de l'utilisateur

Avoir une représentation interne de l'utilisateur permet d'adapter le parcours non seulement à ce que celui-ci désire mais aussi suivant son évolution et son niveau. L'usage "historique" implique de conserver dans la modélisation la liste chronologique des entités consultées par l'utilisateur.

La modélisation que je vais proposer ici est très simple mais peut facilement être étendue.

```
<!ELEMENT etudiant (nom, prenom, niveau, historique)>
   <!ELEMENT nom (#PCDATA)>
   <!ELEMENT prenom (#PCDATA)>
   <!ELEMENT niveau (#PCDATA)>
   <!ELEMENT historique (etape+)>
      <!ELEMENT etape (#PCDATA)>
         <!ATTLIST etape
            date CDATA #REQUIRED
            entite CDATA #REQUIRED
         >
```

Cette structure permet de stocker les informations connues sur l'étudiant. Elles sont initialisées au début a l'aide d'un formulaire, puis recalculées au fur et à mesure de la consultation.

Avec cette modélisation l'usage "historique" s'implémente simplement en ajoutant une étape au profil à chaque nouvelle consultation, et en les parcourant ensuite.

De plus cette modélisation permet de modifier les algorithmes des relation calculées ou mixtes. Par exemple en prenant en compte le niveau de l'élève pour lui choisir l'exemple lui correspondant le mieux. Cela implique aussi de recalculer dynamiquement les variables d'état du profil (la seule crée ici est le niveau de l'élève) en se basant sur le niveau des pages consultées par exemple.

On peut aussi voir plus loin et imaginer un médiateur proposant un mode adaptatif qui choisirait la page suivante en fonction des parcours précédemment privilégiés par l'étudiant. Voire aussi lister l'historique des requêtes pour identifié ce que recherche en priorité l'étudiant et mieux cibler les réponses aux requêtes en cas de réponses multiples possibles.

La prise en compte de l'étudiant est donc potentiellement très intéressante pour augmenter la précision du médiateur.

# 5 Conclusion

La démarche de conception adoptée, partir de la représentation de l'objet médié et des usages pour concevoir les représentations internes du médiateur, nous a permis de concevoir un médiateur répondant bien à l'idée que l'on s'en fait.

Bien que l'étude faite ici reste assez générale elle permet je pense de faire le tour des différents aspects du médiateurs et de les décrire assez précisément. Il serait possible d'affiner ce modèle en rajoutant des entités de bases des usages qui seraient facilement modélisable en se basant sur ceux présentés ci-dessus.