UIMA

UIMA (***Unstructured Information Management Architecture***) est un framework de développement et de déploiement d'applications de traitement des données non structurées. Lancé par IBM, et maintenant en incubation à la fondation Apache, UIMA est en cours de normalisation à l'OASIS (*Organization for the Advancement of Structured Information Standards*). L'objectif de ce framework est de proposer une architecture supportant les différentes étapes du traitement d'un document non structuré (texte, image, vidéo, etc.) en vue d'en extraire de façon automatique des informations structurées. UIMA ne décrit ni comment ces informations doivent être extraites du texte, ni la façon de s'en servir. Par contre, cette plate-forme de développement prend en charge de nombreuses problématiques de façon native comme la réutilisation de composants, la montée en charge et le déploiement distribué, la prise en compte des erreurs, la mise en œuvre de services web, etc.

INTRODUCTION À UIMA

- Qu'est-ce que UIMA ?
- Principes d'une chaîne de traitements UIMA
- Les annotations

DESCRIPTION DES ÉLÉMENTS CLEFS D'UIMA

- Le schéma d'annotation commun (*Common Analysis Structure* ou *CAS*)
- Le système de types (*Type System* ou *TS*)
- Composant d'annotation (Analysis Engine)
- Annotateur (Annotator)
- Descripteur XML de composant (*Component Descriptors*)
- La chaîne de traitement UIMA
- CAS vs. JCAS

INSTALLATION D'UIMA

- Installation complète de l'environnement de développement UIMA sous Linux
- Installation d'UIMA sous Linux (… Suite)

PREMIERS PAS AVEC UIMA

- Tutoriel UIMA I : création d'un composant d'annotation (*Analysis Engine*) en utilisant le JCas
- Tutoriel UIMA I (variante) : code de l'annotateur utilisant directement le CAS plutôt que le JCas
- Tutoriel UIMA II : paramétrage d'un composant d'annotation
- Tutoriel UIMA III : enchaîner des composants d'annotation
- Tutoriel UIMA IV : Parcourir des annotations en utilisant le JCas
- Tutoriel UIMA V : Parcourir des annotations en utilisant le CAS

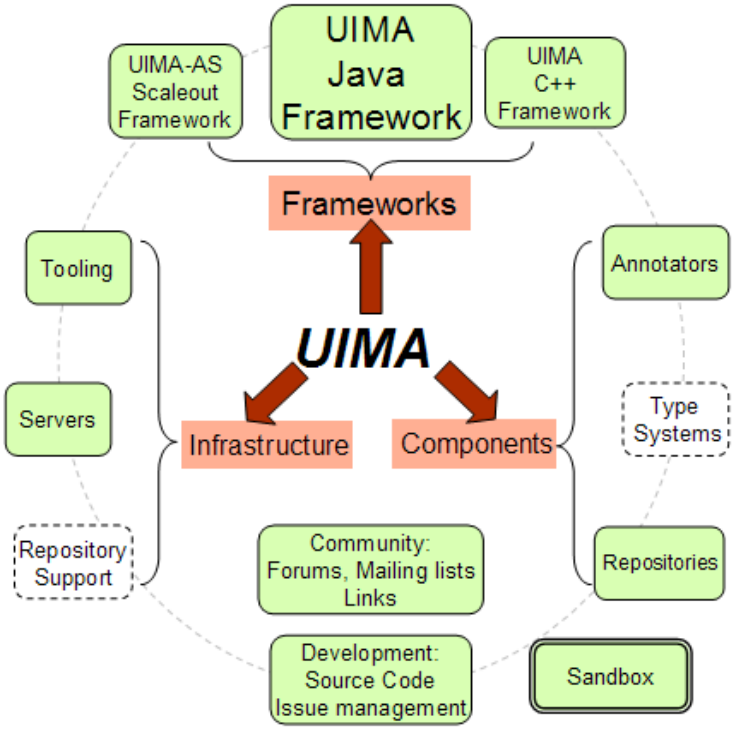
AUTRES INFORMATIONS

- Tutoriel (et environnement) d'introduction à UIMA proposé par le LINA lors de l'édition 2009 des Rencontres Mondiales du Logiciel Libre (RMLL)
- Quelques liens pertinents concernant UIMA
- Diapositives d'une présentation d'UIMA faite dans le cadre du projet DaFOE
- Documentation du SDK UIMA
- Tous les billets concernant UIMA sur ce blog

Qu’est-ce que UIMA ?

UIMA **(*Unstructured Information Management Architecture*)** est un framework de développement et de déploiement d'applications de traitement des données non structurées. Lancé par IBM, et maintenant en incubation à la fondation Apache, UIMA est en cours de normalisation à l'OASIS (*Organization for the Advancement of Structured Information Standards*). L'objectif de ce framework est de proposer une architecture supportant les différentes étapes du traitement d'un document non structuré (texte, image, vidéo, etc.) en vue d'en extraire de façon automatique des informations structurées. UIMA ne décrit ni comment ces informations doivent être extraites du texte, ni la façon de s'en servir. Par contre, cette plate-forme de développement prend en charge de nombreuses problématiques de façon native comme la réutilisation de composants, la montée en charge et le déploiement distribué, la prise en compte des erreurs, la mise en œuvre de services web, etc.

Le site Apache UIMA propose un schéma illustrant bien les différentes facettes d’UIMA.



Les composants (*Components*) sont les briques de base permettant de bâtir une application UIMA complète. Les composants d’annotation (*Annotators* sur le schéma) sont les composants qui supportent la responsabilité la plus importante qui est d’effectuer des traitements (un traitement par composant en principe). Le système de types permet de définir la hiérarchie de types (type au sens informatique du terme) qui supportera les annotations (*i.e.* c’est le diagramme de classes des annotations).

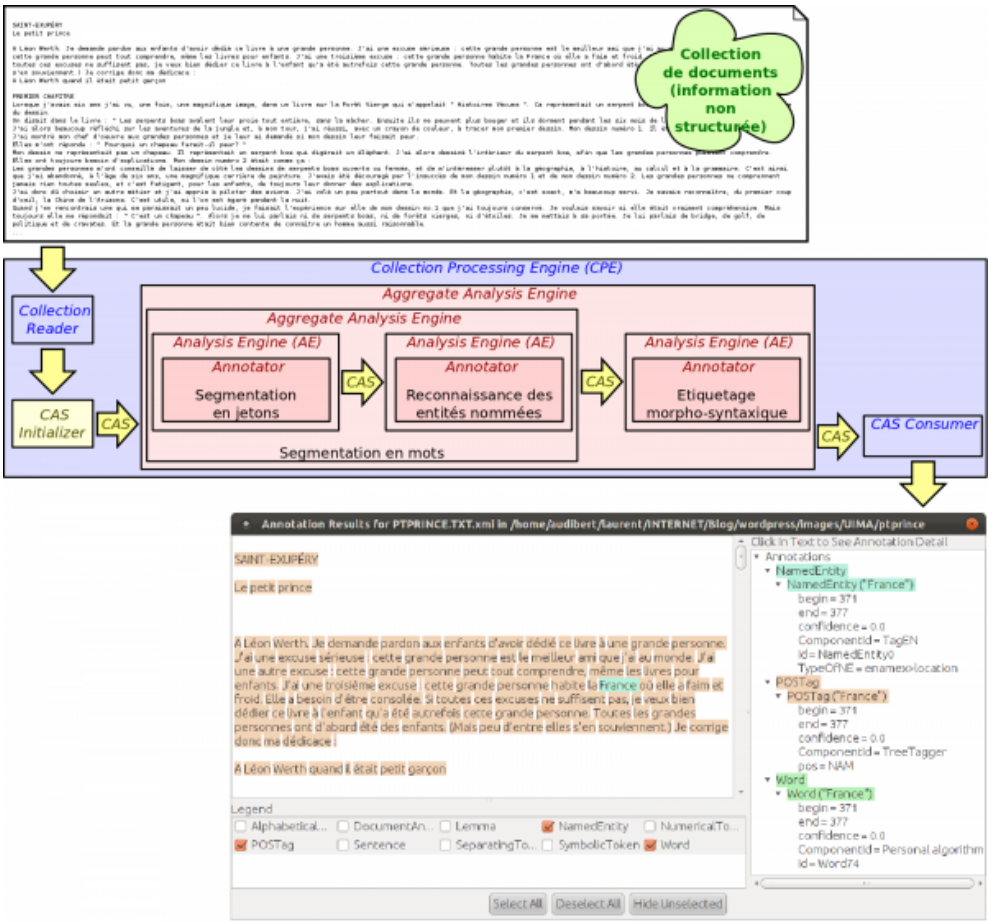
Les *Frameworks* sont l’ensemble des bibliothèques et outils permettant de développer et d’exécuter des applications UIMA.

Introduction au TALN, aux plateformes d’annotations linguistiques et à UIMA
Sommaire UIMA sur ce blog

Principes d’une chaîne de traitements UIMA

L’objectif d’une chaîne de traitements UIMA est d’extraire de façon automatique des informations structurées à partir d’un document non structuré (texte, image, vidéo, etc.). Une chaîne de traitements UIMA respecte généralement les étapes suivantes :

1. Charger un document non structuré (comme le texte du Petit Prince de Saint-Exupéry dans l’illustration ci-dessous).
2. Générer le schéma d’annotation commun (*Common Analysis Structure* ou CAS) qui encapsulera toutes les annotations concernant le document analysé, ainsi que le document lui-même.
3. Enchaîner les composants d’annotation (un segmenteur, un analyseur d’entités nommées et un annalyseur morpho-syntaxique dans l’illustration ci-dessous).
4. Générer une sortie (généralement un fichier xmi) à partir des annotations contenues dans le CAS (le fichier xmi généré est visualisé dans l’outil UIMA *Document Viewer* dans l’illustration ci-dessous).



Introduction au TALN, aux plateformes d’annotations linguistiques et à UIMA
Sommaire UIMA sur ce blog

UIMA : les annotations

Les annotations permettent d'apporter des informations sur des éléments. Dans le cas de corpus textuels, ces éléments sont généralement des segments de textes. Les annotations peuvent être intégrées au document ou, au contraire, être déportées dans un fichier ou une zone séparés. UIMA a fait le choix de représenter les annotations d'une manière déportée. Ce choix présente deux avantages principaux :

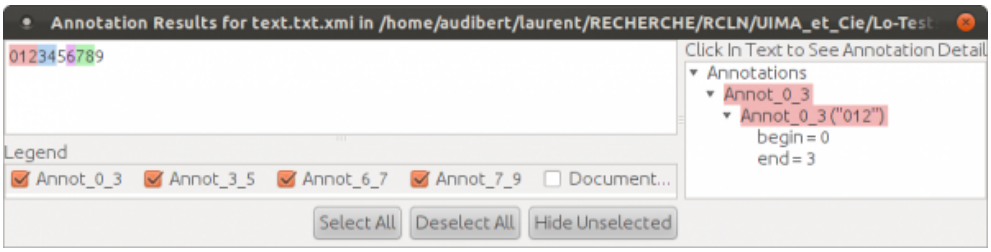
1. Le document source n'est pas altéré par l'insertion d'annotations.
2. Déporter les annotations peut autoriser la représentation d'annotations concurrentes ou enchâssées.

Bien entendu, le recours à des annotations déportées nécessite la mise en place d'un système d'adressage pour référencer des segments du texte source. Dans UIMA, une annotation hérite généralement du type `uima.tcas.annotation` qui définit deux attributs `begin` et `end`. C'est deux attributs permettent d'identifier le début et la fin du segment annoté en terme de nombre de caractères depuis le début du texte analysé.

Il existe plusieurs façons de réaliser une telle indexation. La figure ci-dessus en illustre deux. Le choix fait par UIMA correspond à l'indexation inférieure qui pointe entre les caractères (ce qui est cohérent avec la philosophie des itérateurs java).

- Le mot **JULES** est indexé de la manière suivante : `begin = 0` et `end = 5`
- Le mot **VERNE** est indexé de la manière suivante : `begin = 6` et `end = 11`

La capture d'écran ci-dessous montre un exemple d'annotations visualisées dans l'outil *Document Viewer* d'UIMA. Le texte annoté est « 0123456789 ».



Sommaire UIMA sur ce blog

UIMA : Le schéma d’annotation commun (Common Analysis Structure ou CAS)

Le schéma d’annotation commun (*Common Analysis Structure* ou *CAS*) est une notion centrale dans UIMA. Le *CAS* encapsule toutes les annotations concernant le document analysé ainsi que le document lui-même. Un *CAS* comporte deux types d’éléments :

1. Le documents, appelé *Sofa* pour *Subject of Analysis* dans UIMA.
2. Des annotations générées par des composants d’annotation.

Le *CAS* est une structure orientée objets permettant aux différents composants d’une chaîne d’annotation UIMA de représenter et d’échanger leurs annotations respectives. UIMA propose quelques types de base et permet de les étendre grâce au concept de système de types (*Type System*).

L’interface Java permettant de manipuler le CAS depuis du code Java se nomme JCas (cf. Tutoriel IV), mais le CAS reste tout de même accessible en Java sans passer par cette interface (cf. Tutoriel V).

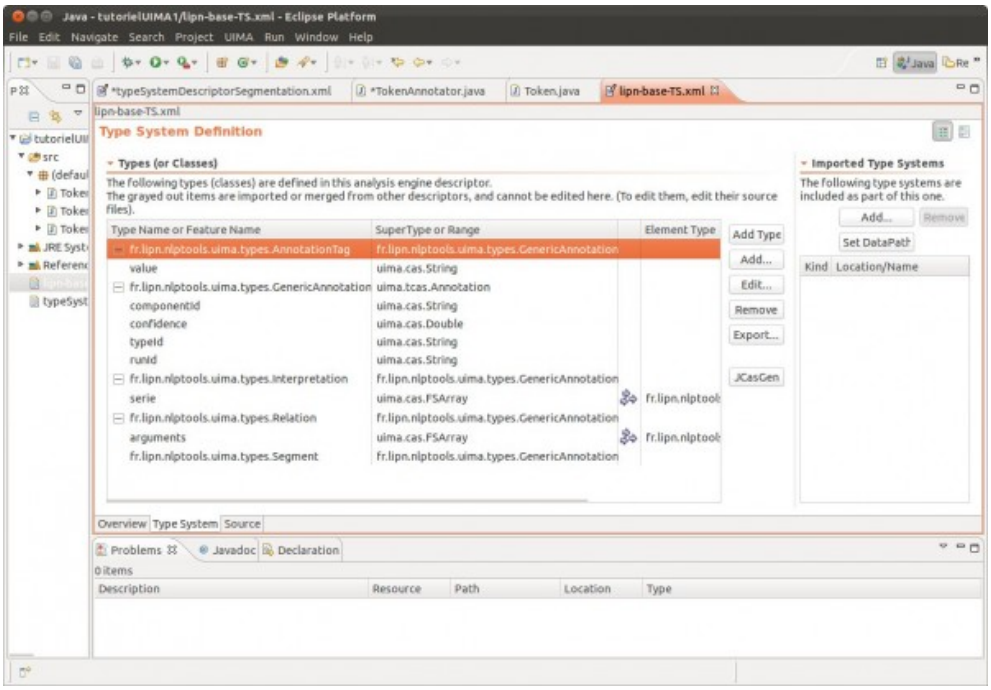
Sommaire UIMA sur ce blog

UIMA : Le système de types (Type System ou TS)

Pour composer un *CAS*, *UIMA* propose quelques types de base et permet de les étendre grâce au concept de système de types (*Type System* ou *TS*). Un système de types est une hiérarchie de types qui défini l'ensemble des types qu'il est possible d'utiliser dans le *CAS*.

Chaque composant *UIMA* (*i.e.* chaque *Analysis Engines*) défini son propre système de types. Pour assurer la cohérence d'une chaîne d'annotation *UIMA*, il est préférable que chacun des systèmes de types des composants de la chaîne soit un sous-ensemble d'un système de types plus général décrivant tous les types d'annotations que la chaîne peut produire.

Un système de types est décrit dans un fichier *XML* qui peut être édité sous *Eclipse* par le *Component Descriptor Editor* :



Sommaire UIMA sur ce blog

Composant d’annotation (Analysis Engine)

Le **composant d’annotation** (*Analysis Engine* ou AE) est le composant élémentaire d’une chaîne de traitements UIMA. Son rôle est d’effectuer un traitement (comme l’étiquetage morpho-syntaxique) en utilisant éventuellement des annotations contenues dans le CAS (comme une segmentation en mots) et en produisant de nouvelles annotations dans le CAS (comme les étiquettes morpho-syntaxique de chacun des mots).

Un composant d’annotation UIMA (*Analysis Engine*) est constitué de deux parties distinctes :

- Un descripteur de composant (*componant descriptor*) qui constitue la partie déclarative. Il contient des méta-données qui décrivent le composant (types de données en entrée et en sortie, identité …) et est stocké dans un fichier XML.
- Un annotateur (***Annotator***) qui est en fait le code (en Java par exemple) effectuant le traitement proprement dit du composant.

En résumé :

Composant d’annotation = Descripteur de composant + Annotateur
Analysis Engine = Componant Descriptor + Annotator

Tutoriel UIMA I : création d’un composant d’annotation (Analysis Engine)
Sommaire UIMA sur ce blog

Descripteur XML de composant (Component Descriptors)

Les composants UIMA sont composés de deux parties distinctes :

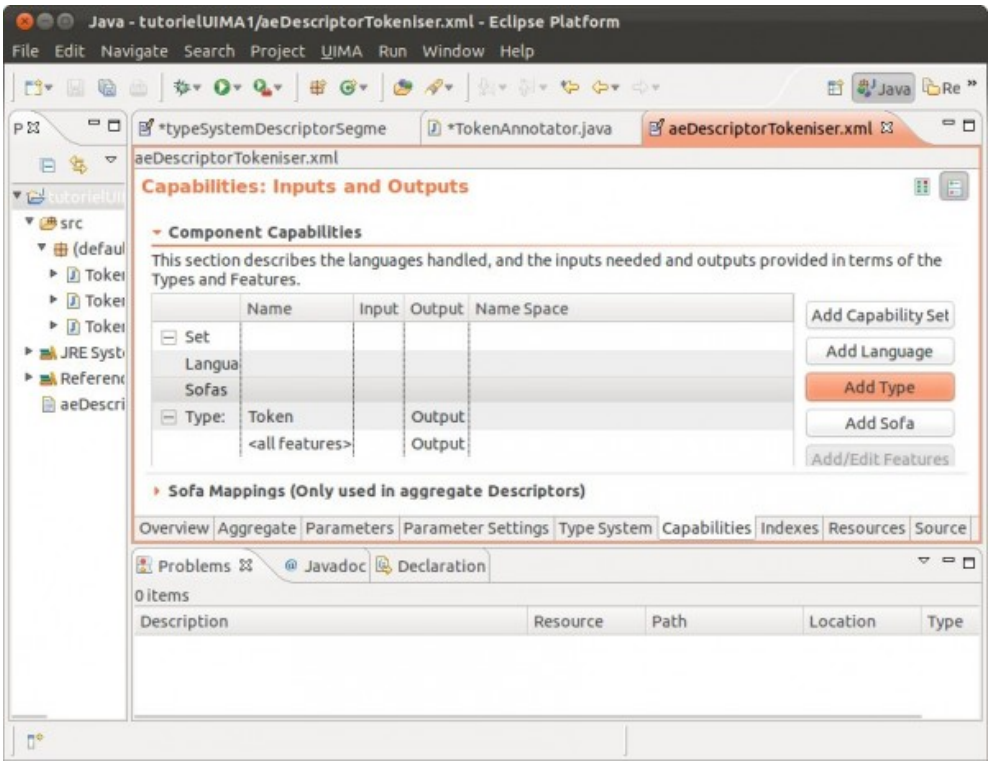
1. une partie déclarative (le descripteur de composant)
2. un code (en Java par exemple)

La partie déclarative est représentée dans un fichier XML et est appelée descripteur de composant (*Component Descriptor*). Le descripteur de composant contient des méta-données qui décrivent le composant. C'est le moyen qu'UIMA utilise pour mettre en œuvre les mécanismes de découverte, de réutilisation et de composition de composants.

Le descripteur de composant permet de renseigner, entre autres, les informations suivantes :

- nom, version, auteur, description et surtout le nom de la classe implémentant le composant (onglet *Overview*)
- Système de types utilisé (onglet *Type System*)
- Types utilisés (*i.e.* requis) dans le *CAS* par le composant (onglet *Capabilities*)
- Types renseignés dans le *CAS* par le composant (onglet *Capabilities*)

Sous Eclipse, un tel descripteur peut être édité avec le *Component Descriptor Editor*.



Sommaire UIMA sur ce blog

UIMA : La chaîne de traitement

Une chaîne de traitement UIMA comporte trois types d'éléments :

1. Le **Collection Reader** : généralement le premier élément de la chaîne, il permet de créer des *CAS* autour de chaque ressource.
2. Les **Analysis Engines** : raison d'être de la chaîne, ils effectuent les traitements en utilisant les annotations contenues dans le *CAS* et en produisant de nouvelles annotations dans le *CAS*.
3. Le **Workflow Controller** : c'est lui qui orchestre l'exécution des *Analysis Engines*.

On peut distinguer deux types de *Workflow Controller* : - le *CPM* (*Collection Processing Manager*) et les service Vinci (*Vinci Services*) ; - et le tout nouveau *AS* (*Asynchronous Scaleout*) bien plus flexible.

L'interface graphique *Collection Processing Managment*, instancie un *Workflow Controller* du type *CPM* appelé *FixedFlow* qui consiste simplement à enchaîner séquentiellement un *Collection Reader*, puis un certain nombre d'*Analysis Engine* et enfin un *CAS Consumer*. Un *CAS Consumer* n'étant rien d'autre qu'un *Analysis Engine* particulier qui n'est pas destiné à enrichir un *CAS* mais qui génère une sortie (généralement un fichier) à partir des annotations contenues dans le CAS.

Collection Processing Engine
Apache UIMA par la pratique
Sommaire UIMA sur ce blog

Installation complète de l'environnement de développement UIMA sous Linux

Ce billet décrit comment installer complètement l'environnement de développement UIMA sous une station de travail Linux sans avoir besoin des droits d'administrateur. Par installation complète il faut comprendre l'installation de tout ce qui est nécessaire : le SDK Java, Eclipse, les plugins nécessaires et le SDK UIMA.

INSTALLATION DU SDK JAVA

La première étape consiste, si ce n'est déjà fait, à installer la dernière version du kit de développement de logiciels (SDK) Java. Commencer par télécharger le fichier d'installation (*jdk-6u4-linux-i586.bin* dans mon cas) dans le répertoire de son choix. Saisir ensuite, dans un shell, en étant dans le répertoire où le fichier a été téléchargé et où l'installation doit se faire :

```
$ chmod +x jdk-6u4-linux-i586.bin
$ ./jdk-6u4-linux-i586.bin
```

(Il vous faut répondre *yes* aux termes du contrat pour réaliser l'installation)

INSTALLATION DE L'ENVIRONNEMENT DE DÉVELOPPEMENT INTÉGRÉ (IDE) ECLIPSE

Si ce n'est déjà fait, installer la dernière version d'Eclipse (version 3.3.1.1, build id M20071023-1652 au moment de l'écriture de ce billet). L'installation se résume à décompresser à l'endroit de son choix le fichier téléchargé.

Pour exécuter la version d'Eclipse qui vient d'être d'installée avec la version de Java installée précédemment, il faut saisir dans un shell une ligne de commande du style (penser à créer un raccourci) :

```
$ ~/eclipse/eclipse -vm ~/jdk1.6.0_04/bin/
```

Ensuite, dans Eclipse, il faut préciser la version de Java qui sera utilisée. Aller dans :

> *Window → Preferences… → Java → Installed JREs → Add…*

Puis préciser :

> *JRE name : Java jdk1.6.0_04*
> *JRE home directory : ~/jdk1.6.0_04*

Valider en cliquant sur *Ok*, puis penser à cocher la nouvelle version du JRE ajoutée avant de valider de nouveau en cliquant sur *Ok*.

Vérifier également, la cas échéant, que la variable d'environnement *ECLIPSE_HOME* est correcte (*i.e.* désigne bien le répertoire *~/eclipse*) dans :

> *Window → Preferences… → Java → Build Path → Classpath Variables*

INSTALLATION DU PLUGIN ECLIPSE EMF

Il faut ensuite installer le plugin EMF utilisé par UIMA. En fait, avec la dernière version d'Eclipse, *Eclipse Modeling Framework (EMF) Updates* apparait automatiquement dans la liste des sites d'installation

automatique de plugins (sinon, se référer à *EMF Update Manager Site*). Une fois Eclipse lancé, il suffit donc de faire :

> *Help → Software Updates → Find and Install… → Search for new features to install → Next*

Puis sélectionner *Eclipse Modeling Framework (EMF) Updates*, *Ignore features not applicable to this environment* et *Automatically select mirrors* puis *Finish.*
Ensuite déployer *Eclipse Modeling Framework (EMF) Updates* pour sélectionner *EMF SDK 2.3.1* puis :

> *Next → I accept the terms in the license agreements → Next → Finish → Install All → Yes*

INSTALLATION DU PLUGIN ECLIPSE UIMA

On peut alors passer à l'installation du plugin Eclipse d'UIMA :

> *Help → Software Updates → Find and Install… → Search for new features to install → Next → New Remote Site…*

Saisir :

> *Name : Apache UIMA*
> *URL : http://www.apache.org/dist/uima/eclipse-update-site*

Puis *Ok.*
Sélectionner *Apache UIMA*, *Ignore features not applicable to this environment* et *Automatically select mirrors* puis *Finish*.
Sélectionner *Apache UIMA* puis :

> *Next → I accept the terms in the license agreements → Next → Finish → Install All → Yes*

INSTALLATION DU SDK UIMA

Il faut enfin installer le kit de développement de logiciels (SDK) UIMA. Télécharger la dernière version du SDK d'UIMA, puis décompresser à l'endroit de votre choix le fichier téléchargé.
Éditez votre fichier *.bashrc* pour définir les variables d'environnement *UIMA_HOME* et *JAVA_HOME* et modifier le *PATH* en ajoutant les lignes suivante à la fin du fichier :

```
export UIMA_HOME="~/apache-uima"
export JAVA_HOME="~/jdk1.6.0_04"
export PATH=${PATH}:${UIMA_HOME}/bin
```

Puis saisir la commande suivante dans un shell pour mettre à jours les chemins des exemples du SDK :

```
$ adjustExamplePaths.sh
```

Penser à lire le fichier *README* dans le répertoire *apache-uima* pour plus d'information sur l'installation du SDK UIMA ainsi que pour une procédure permettant de tester votre installation (également décrite ici).

Sous Eclipse, vérifier également que la variable d'environnement *UIMA_HOME* est correcte (*i.e.* désigne bien

le répertoire *~/apache-uima*) dans :

Window → Preferences… → Java → Build Path → Classpath Variables

Si ce n'est pas le cas, il faut la modifier (bouton *Edit…*) ou la définir (bouton *New…*).

Si tout s'est bien passé, tous les fichiers concernant l'installation complète de l'environnement de développement UIMA doivent être répartis dans les répertoires *apache-uima* pour le SDK UIMA, *eclipse* pour l'environnement de développement intégré Eclipse et les plugins installés et *jdk1.6.0_04* pour le JDK Java.

UIMA Manuals and Guides
Sommaire UIMA sur ce blog

Installation d’UIMA sous Linux (… Suite)

1. **Verifying Your Installation** (extrait du fichier *README* dans le répertoire *apache-uima*)

   To test the installation, run the *documentAnalyzer.sh*:

   ```
   > documentAnalyzer.sh
   ```

   This should pop up a *Document Analyzer* window. Set the values displayed in this GUI to as follows:

   - Input Directory: `UIMA_HOME/examples/data`
   - Output Directory: `UIMA_HOME/examples/data/processed`
   - Location of Analysis Engine XML Descriptor:

     `UIMA_HOME/examples/descriptors/analysis_engine/PersonTitleAnnotator.xml`

   Replace *UIMA_HOME* above with the path of your Apache UIMA installation.

   Next, click the *Run* button, which should, after a brief pause, pop up an *Analyzed Results* window.

   Double-click on one of the documents to display the analysis results for that document.

2. **View the examples code with Eclipse** (extrait du document *UIMA Overview & SDK Setup* section *3.2. Setting up Eclipse to view Example Code*)

   You can create a special project in Eclipse to hold the examples. Here’s how:

   - In Eclipse, if the Java perspective is not already open, switch to it by going to: *Window → Open Perspective → Java*
   - Select the *File → Import* menu option.
   - Select *General/Existing Project into Workspace* and click the *Next* button.
   - Click *Browse* and browse to the *UIMA_HOME* directory.
   - Click *Finish*. This will create a new project called *uimaj-examples* in your Eclipse workspace. There should be no compilation errors.

   To verify that you have set up the project correctly, check that there are no error messages in the *Problems* view.

Sommaire UIMA sur ce blog

Tutoriel UIMA I : création d'un composant d'annotation (Analysis Engine) en utilisant le JCas

Ce billet décrit comment créer un composant d'annotation (*Analysis Engine*) UIMA simple qui segmente un fichier texte en mots.

1. **Préparation du projet dans Eclipse**

   Lancer *Eclipse* puis : *File → New → Other… → Java → Java Project → Next*. Saisir un nom dans le champs *Project name:* puis cliquer sur *Finish*.

   Il faut ensuite attacher la librairie UIMA au projet : *Project → Properties → Java Build Path → Libraries → Add External JARs…* ; sélectionner le fichier *uima-core.jar* dans le répertoire *apache-uima/lib*.

2. **Définir un type dans le schéma d'annotation**

   Le schéma d'annotation commun (*Common Analysis Structure* ou *CAS* dans UIMA) encapsule toutes les annotations concernant le document analysé ainsi que le document lui-même. Nous aurons besoin du type *Token* dans cette structure, nous allons donc le déclarer dans un système de types : *File → New → Others → UIMA → Type System Descriptor File* ; préciser le nom dans le champs *File name:* puis *Finish*.

   Faire ensuite un clic droit sur le fichier *.xml* ainsi créé puis *Open With → Component Descriptor Editor*. Dans l'onglet *Type System* cliquer sur *Add Type* et saisir le nom dans le champs *Type Name:* (ex : Token) et préciser *uima.tacs.Annotation* dans le champs *Supertype:*.

   L'action d'enregistrer vos modifications doit générer automatiquement les fichiers associés (ex : Token_Type.java et Token.java).

3. **Développer le code de l'annotateur**

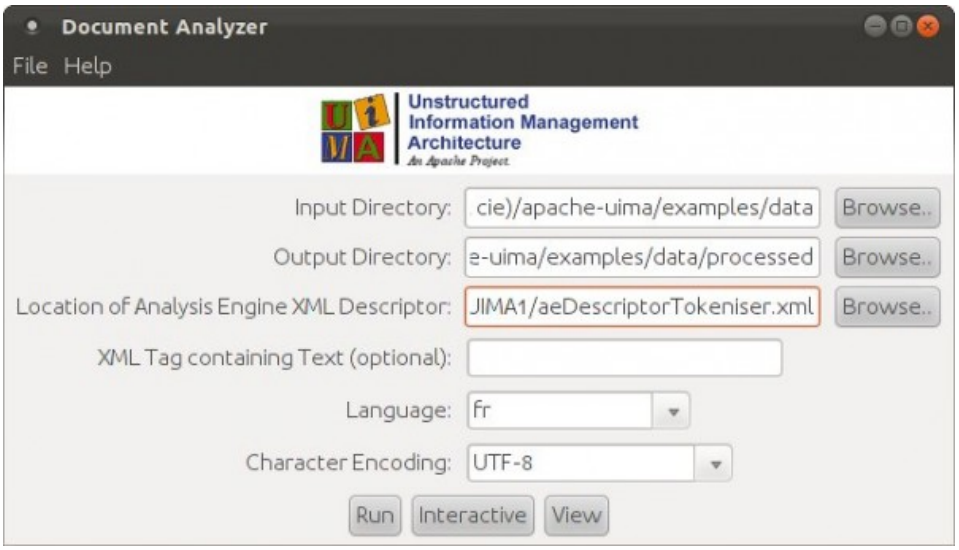
   Il faut créer une classe héritant de la classe *JTextAnnotator_ImplBase* correspondant à l'implémentation par défaut d'un annotateur de données textuelles : *File → New → Class* ; saisir le nom de la classe dans le champs *Name: TokenAnnotator* ; préciser la classe *org.apache.uima.analysis_component.JCasAnnotator_ImplBase* dans le champs *Superclass:* ; puis cliquer sur *Finish*.

   Il ne reste plus qu'à écrire le code proprement dit :

```
import org.apache.uima.analysis_component.JCasAnnotator_ImplBase;
import org.apache.uima.analysis_engine.AnalysisEngineProcessException;
import org.apache.uima.jcas.JCas;
import java.text.BreakIterator;
public class TokenAnnotator extends JCasAnnotator_ImplBase {
  public void process(JCas aJCas) throws AnalysisEngineProcessException {
    BreakIterator iterToken = BreakIterator.getWordInstance();
    iterToken.setText(aJCas.getDocumentText());
    int deb = iterToken.first();
    int fin = iterToken.next();
    while (fin != BreakIterator.DONE) {
      Token token = new Token(aJCas);
      token.setBegin(deb);
      token.setEnd(fin);
      token.addToIndexes();
      deb = fin;
      fin = iterToken.next();
    }
  }
}
```

   Le code ci-dessus passe par l'interface Java JCas pour injecter les annotations dans le CAS. Le recours à cette interface n'est pas indispensable.

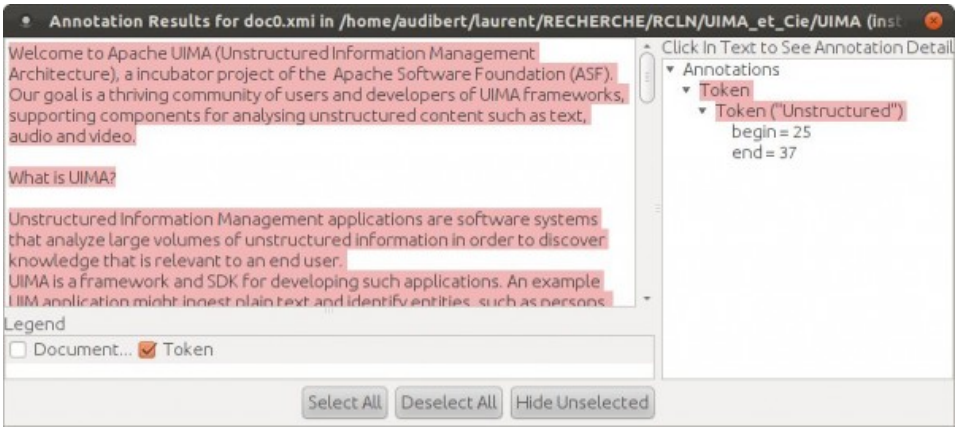
4. **Descripteur du composant d'annotation (*Analysis Engine Descriptor*)**
   L'architecture UIMA impose qu'un annotateur soit décrit par un fichier XML appelé *Analysis Engine Descriptor* : *File → New → Others → UIMA → Analysis Engine Descriptor File* ; préciser le nom dans le champs *File name:* puis *Finish*.
   Faire ensuite un clic droit sur le fichier *.xml* ainsi créé puis *Open With → Component Descriptor Editor*. Indiquer le fichier java implémentant l'annotateur dans le champs *Name of the java class file* de l'onglet *Overview*.
   Dans l'onglet *Type System*, il faut énumérer les types que l'annotateur utilise. Cliquer d'abord sur *Set DataPath* dans la colonne *Imported Type Systems* pour indiquer le chemin du fichier XML contenant schéma d'annotation commun (*Type System Descriptor*). Dans la même colonne, cliquer ensuite sur *Add…* pour sélectionner ce fichier.
   Enfin, dans l'onglet *Capabilities*, il faut préciser, parmi les types énumérés dans l'onglet *Type System*, lesquels sont utilisés et s'ils le sont en entrée ou en sortie. Cliquer sur *Add Type*, sélectionner la ligne correspondant à votre token et préciser que le type est utilisé en sortie uniquement (*Out*) puis valider (*Ok*).
5. **Exécuter le *Document Analyzer* pour tester le composant d'annotation**
   Il faut commencer par mettre à jour la variable d'environnement *CLASSPATH* pour y ajouter le chemin vers votre répertoire *bin*. Dans un shell, saisir :
   ```
   > export CLASSPATH=$CLASSPATH:chemin/vers/votre/bin
   ```
   Vous pouvez alors exécuter le script *documentAnalyzer.sh* qui vous permettra de tester votre composant d'annotation.
6. **Exécuter sous Eclipse le *Document Analyzer* pour tester le composant d'annotation**
   Commencer par importer le projet exemple d'UIMA : *File → Import… → General → Existing Project into Workspace → Next* ; puis dans *Select root directory* sélectionner le répertoire racine d'UIMA ; puis *Finish*.
   Si ce n'est pas déjà fait, il faut ajouter votre projet dans le *ClassPath* de la configuration du *Document Analyzer* : *Run → Run Configurations… → UIMA Document Analyzer* → Onglet *ClassPath*
   Il est maintenant possible d'exécuter directement le *CPE GUI* pour tester le composant : *Run → Run Configurations… → UIMA Document Analyzer → Run*
7. **Tester le composant d'annotation avec le *Document Analyzer***



   Il faut remplir les champs du *Document Analyzer* de la manière suivante :
   - Imput Directory: `UIMA_HOME/examples/data`
   - Output Directory: `UIMA_HOME/examples/data/processed`

- Location of Analysis Engine XML Descriptor: `chemin/du/descripteur/de/composant`

Puis cliquer sur *Run*.



Tutoriel et guide du développeur UIMA (puis cliquer sur *Tutorials and Users' Guides*)

Sommaire UIMA sur ce blog

Tutoriel UIMA I (variante) : code de l’annotateur utilisant directement le CAS plutôt que le JCas

La partie **3. Développer le code de l’annotateur** du Tutoriel UIMA I : création d’un composant d’annotation propose le code Java d’un annotateur utilisant l’interface Java JCas du CAS pour injecter les annotations dans le CAS. Voici le code réalisant le même travail sans utiliser le JCas :

```
import org.apache.uima.analysis_component.CasAnnotator_ImplBase;
import org.apache.uima.analysis_engine.AnalysisEngineProcessException;
import org.apache.uima.cas.CAS;
import org.apache.uima.cas.FeatureStructure;
import org.apache.uima.cas.Type;
import java.text.BreakIterator;
public class TokenCASAnnotator extends CasAnnotator_ImplBase {
  public void process(CAS aCAS) throws AnalysisEngineProcessException {
    BreakIterator iterToken = BreakIterator.getWordInstance();
    iterToken.setText(aCAS.getDocumentText());
    int deb = iterToken.first();
    int fin = iterToken.next();
    while (fin != BreakIterator.DONE) {
      Type TokenCAS = aCAS.getTypeSystem().getType("Token");
      FeatureStructure fs = aCAS.createAnnotation(TokenCAS, deb, fin);v
      aCAS.getIndexRepository().addFS(fs);
      deb = fin;
      fin = iterToken.next();
    }
  }
}
```

Sommaire UIMA sur ce blog

Tutoriel UIMA II : paramétrage d'un composant d'annotation

1. **Déclarer des paramètres**
   Pour déclarer des paramètres pour un composant, il suffit de se rendre dans l'onglet *Parameters* du descripteur XML de l'annotateur (clic droit sur le fichier *.xml* puis *Open With → Component Descriptor Editor*). L'onglet *Parameter Settings* permet de donner une valeur par défaut aux paramètres déclarés.
2. **Accéder aux paramètres**
   Supposons que le paramètre *maChaine* de type *String* ait été préalablement déclarée au niveau du descripteur XML de l'annotateur (cf. ci-dessus). On peut accéder à ce paramètre dans le code Java de l'annotateur de la manière suivante :
   ```
   String monParametre = (String) this.getContext().getConfigParameterValue("maChaine");
   ```
   La méthode *getContext()* permet de récupérer un objet du type *UimaContext* qui centralise l'accès aux différentes facettes du contexte de l'annotateur comme les paramètres ou l'historique des événements (*logging*).
3. **Accéder aux paramètres depuis la méthode *initialize()***
   Un moyen plus propre d'accéder aux paramètres est de surcharger la méthode *initialize(UimaContext)*, appelée automatiquement par le framework lors de l'instanciation de l'annotateur, qui reçoit justement comme paramètre un objet du type *UimaContext* :
   ```
   private String monParametre;
   public void initialize(UimaContext aContext) throws ResourceInitializationException {
      super.initialize(aContext);
      monParametre = (String) aContext.getConfigParameterValue("maChaine");
   }
   ```

Tutoriel et guide du développeur UIMA (puis cliquer sur *Tutorials and Users' Guides*)
Sommaire UIMA sur ce blog

Tutoriel UIMA III : enchaîner des composants d'annotation

Apache UIMA utilise le même type de fichier de description (*Component Descriptor*) pour désigner un composant ou une chaîne de traitement. Le fichier descripteur est :

- soit configuré comme primitif dans le cas où il décrit un traitement élémentaire (comme dans le Tutoriel UIMA I),
- soit configuré comme agrégé (*aggregate*) dans le cas où il fait référence à d'autres descripteurs de fichiers primitifs ou agrégés.

L'opération consistant à enchaîner plusieurs composants est extrêmement simple et ne requiert aucune écriture de code.

1. Créer un *Aggregate Analysis Engine Descriptor* :
   File → New → Others → UIMA → Analysis Engine Descriptor File ; préciser le nom dans le champs *File name:* puis *Finish*. Faire ensuite un clic droit sur le fichier *.xml* ainsi créé puis *Open With → Component Descriptor Editor*. Dans l'onglet *Overview*, cocher *Aggrerate* comme *Engine Type.*
2. Dans l'onglet *Aggregate*, ajouter tous vos annotateur en cliquant sur *Add…*
3. Enfin, dans l'onglet *Capabilities*, il faut préciser les types utilisés, et s'ils le sont en entrée ou en sortie, en cliquant sur *Add Type*.
4. Il ne reste plus qu'à tester l'*Aggregate Analysis Engine Descriptor* en utilisant le *Document Analyzer* comme décrit à la fin du Tutoriel UIMA I.

Tutoriel et guide du développeur UIMA (puis cliquer sur *Tutorials and Users' Guides*)
Construire une chaîne de traitement UIMA à partir de composants existants (chez Nicolas Hernandez)
Exécuter un traitement ou une chaîne de traitement sous UIMA (chez Nicolas Hernandez)
Sommaire UIMA sur ce blog

Tutoriel UIMA IV : Parcourir des annotations en utilisant le JCas

Le CAS maintient un index pour chacun des types d'annotations. Il est possible de récupérer un iterateur permettant de parcourir les annotations d'un type donné. Le plus simple étant d'utiliser les classes Java automatiquement créées par le JCasGen. Ce tutoriel montre comment parcourir, en utilisant le JCas, les annotation du Tutoriel UIMA I (*Token*) pour afficher dans la console le texte qu'elles annotent.

1. Il faut commencer par **créer un nouveau descripteur de composant** : *File → New → Others → UIMA → Analysis Engine Descriptor File* (cf. Tutoriel UIMA I).
2. Il faut ensuite **développer le code de l'annotateur** associé en créant une classe héritant de la classe *JTextAnnotator_ImplBase* (cf. Tutoriel UIMA I), puis écrire le code proprement dit :

```
import org.apache.uima.analysis_component.JCasAnnotator_ImplBase;
import org.apache.uima.jcas.JCas;
import org.apache.uima.cas.FSIndex;
import org.apache.uima.cas.FSIterator;
public class ParcoursJCasAnnotator extends JCasAnnotator_ImplBase {
  public void process(JCas aJCas) {
    FSIndex tokenIndex = aJCas.getAnnotationIndex(Token.type);
    FSIterator tokenIter = tokenIndex.iterator();
    while (tokenIter.isValid()) {
      System.out.println(((Token)tokenIter.next()).getCoveredText());
    }
  }
}
```

3. Il faut enfin **créer le composant d'annotation agrégé** (Tokenisation du Tutoriel UIMA I + parcours des annotations de ce tutoriel) comme décrit dans le Tutoriel UIMA III, puis tester ce composant d'annotation agrégé (*Aggregate Analysis Engine Descriptor*) en utilisant le *Document Analyzer* comme décrit à la fin du Tutoriel UIMA I.

Tutoriel et guide du développeur UIMA (puis cliquer sur *Tutorials and Users' Guides*)
Sommaire UIMA sur ce blog

Tutoriel UIMA V : Parcourir des annotations en utilisant le CAS

Le CAS maintient un index pour chacun des types d'annotations. Il est possible de récupérer un iterateur permettant de parcourir les annotations d'un type donné sans utiliser les classes Java automatiquement créées par JCasGen (comme dans le Tutoriel UIMA IV). Ce tutoriel montre comment parcourir, en utilisant le CAS (plutôt que le JCas), les annotations du Tutoriel UIMA I (*Token*) pour afficher dans la console le texte qu'elles annotent.

1. Il faut commencer par **créer un nouveau descripteur de composant** : *File → New → Others → UIMA → Analysis Engine Descriptor File* (cf. Tutoriel UIMA I).
2. Il faut ensuite **développer le code de l'annotateur** associé en créant une classe héritant de la classe *JTextAnnotator_ImplBase* (cf. Tutoriel UIMA I), puis écrire le code proprement dit :

```
import org.apache.uima.analysis_component.CasAnnotator_ImplBase;
import org.apache.uima.cas.CAS;
import org.apache.uima.cas.FSIndex;
import org.apache.uima.cas.Type;
import org.apache.uima.cas.FSIterator;
import org.apache.uima.cas.text.AnnotationFS;
public class ParcoursCasAnnotator extends CasAnnotator_ImplBase {
  public void process(CAS aCAS) {
    Type TokenCAS = aCAS.getTypeSystem().getType("Token");
    FSIndex tokenIndex = aCAS.getAnnotationIndex(TokenCAS);
    FSIterator tokenIter = tokenIndex.iterator();
    while (tokenIter.isValid()) {
      AnnotationFS annot = (AnnotationFS) tokenIter.get();
      System.out.println(annot.getCoveredText());
      tokenIter.moveToNext();
    }
  }
}
```

3. Il faut enfin **créer le composant d'annotation agrégé** (Tokenisation du Tutoriel UIMA I + parcours des annotations de ce tutoriel) comme décrit dans le Tutoriel UIMA III, puis tester ce composant d'annotation agrégé (*Aggregate Analysis Engine Descriptor*) en utilisant le *Document Analyzer* comme décrit à la fin du Tutoriel UIMA I.

Tutoriel et guide du développeur UIMA (puis cliquer sur *Tutorials and Users' Guides*)
Sommaire UIMA sur ce blog