# Rapport de stage
## Conception d'un éditeur de méthodes graphique


Auteur : **Hubert MONIN**

Tuteurs : Pierre Crescenzo et Philippe Lahire


Date 31/01/2003 Nom/Taille fichier : RapportDeStageI3S.pdf/553Ko

| **Historique** | | **Création : 31/01/03** |
|---|---|---|
| **Date** | **Version** | **Motif** |
| 31/01/03 | V 1.0 | Création |


### Synthèse

Le présent document à pour but de décrire le déroulement du stage. Il est composé de 2 parties principales : La description détaillée des techniques utilisées et la présentation de l'application développée au travers de ses différentes fonctions.

# SOMMAIRE

# 1 INTRODUCTION

## 1.1 Description du document

Ce document est composé de deux parties :

- Techniques et langage utilisés.

  Nous utiliserons Java pour la richesse des fonctionnalités de la plate-forme J2EE. Au passage, signalons que le pseudo-code employé dans notre application est très proche de la syntaxe Java.

  Nous verrons XML (Extensible Markup Language) qui nous servira dans ce projet doublement. Ce sera à la fois un outil de stockage de données relatives aux méthodes des utilisateurs, et à la fois un document prêt à subir une transformation pour offrir à l'utilisateur le code source qu'il attend.

  Le langage qui nous permettra cela s'appelle XSL. (Extensible Stylesheet Language) XSL est un langage pour décrire des feuilles de style. Il est composé de trois parties

  - XSL Transformations (XSLT) pour transformer des documents XML.
  - XML Path Language (Xpath) pour, par le biais d'XSLT, accéder aux données présentes dans le document XML.
  - XSL Formatting Objects pour décrire une présentation à l'aide d'un vocabulaire. (Que nous n'utiliserons pas dans ce projet)

  Et enfin, pour nos échanges entre notre application et les documents XML, nous aurons besoin d'utiliser des API Java de traitement XML. Nous décrirons les deux principales :

  - SAX (Simple API for XML) qui fut la première API XML en Java très largement adoptée par les développeurs.
  - DOM (Document Object Model) qui fournit un ensemble standard d'objets pour la représentation de documents HTML et XML, un modèle standard de la manière dont ces objets sont combinés entre eux, et une interface standard pour leur chargement et leur manipulation.

  L'API JAXP (Java API for XML Processing) regroupe DOM, SAX et XSLT. C'est donc elle que nous avons choisi pour ce projet. Elle nous offre en outre la possibilité de changer de processeur XML sans que nous ayons à modifier notre code.

- Description des différentes fonctionnalités de OOMethodEditor

## 1.2 Sujet

OOMethodEditor est une application évènementielle écrite en Java et utilisant la bibliothèque Swing pour les objets graphiques. Elle utilise le paradigme MVC (Model View Controler) à l'aide des interfaces Observer et Observable. Entièrement écrite à l'aide de JBuilder 8 Enterprise Edition, elle a été packagée à la main pour les puristes Java. (Pierre Crescenzo entre autres) Un soin tout particulier a aussi été apporté pour la rendre absolument portable.

L'idée principale est de fournir à l'utilisateur une interface graphique pour la création d'un modèle de méthode unique, qui nous servira pour la création de codes sources multiples. (C# et Java dans notre projet)

Les composants graphiques de notre application remplis par l'utilisateur viendront alimenter le document XML au moment de la sauvegarde du travail en cours. Ce document XML décrira alors la structure de notre modèle de méthode et sera alors prêt à subir la transformation en code source.

Pour cette transformation, une feuille de style XSLT est utilisée et correspond à un langage cible. Il y aura donc autant de feuilles de style que de codes sources finaux. Si l'on souhaite ajouter la possibilité de générer un nouveau type de code source, il nous suffira simplement d'ajouter une nouvelle feuille de style.

Pour le moment, l'utilisateur n'a pas la possibilité de créer les feuilles de styles depuis OOMethodEditor, mais les versions suivantes peuvent aller dans ce sens.

Le futur de ce programme est certainement aussi de rajouter à l'interface graphique des instructions lourdes qui feront appel à des fonctions utilisant des packages décrites dans la feuille de style.

## 1.3 Résumé

Nous ferons une description des techniques mises en œuvre pour OOMethodEditor, puis nous verrons un petit peu à la manière d'un manuel de l'utilisateur, les différentes fonctionnalités du programme.

## 1.4 Contexte

Le laboratoire I3S (Informatique, Signaux et Systèmes de Sophia-Antipolis) est une unité mixte de recherche (UMR6070) associant l'Université de Nice Sophia-Antipolis au CNRS.

Ses activités couvrent de nombreux domaines notamment celui de l'informatique. Le laboratoire I3S regroupe environ 150 personnes, dont l'équipe du projet OCL (Objets Composants Logiciels).

Ce stage fut basé sur le volontariat (aucune rémunération) avec pour objectifs :

- Mises en œuvre des techniques liées à Java et XML.
- Expérience professionnelle.

## 1.5 Conventions utilisées

| Police | Signification |
|---|---|
| `<?xml version="1.0" encoding="ISO-8859-1"?>` | Code java, xml |

# 2 TECHNIQUES UTILISEES

## 2.1 La plate-forme J2EE

J2EE définit une architecture logicielle orientée composants.

A cet effet elle définit :

- des conteneurs de composants :
  - Clients
    - autonomes (*standalone*)
    - applets
  - Serveur
    - Web (Servlets, JSP)
    - EJB
  - intégration
    - JCA
- des services offerts à ces composants via ces conteneurs :
  - transaction
  - distribution
  - persistance
  - sécurité
  - cycle de vie (pooling)

| J2EE | Version | 1 | | | | Commentaire | |
|---|---|---|---|---|---|---|---|
| | Release | 2 | | 3 | 4 | | |
| Couche | Technologie | 0 | 1 | 0 | | | |
| Plate-forme de base | J2SE | 1.2 | | | 1.3 | 1.4 | |
| Présentation Web | Servlet | 2.2 | | | 2.3 | 2.4 | |
| | JSP | 1.1 | | | 1.2 | 2.0 | |
| | JSF | Non | | | | | |
| API d'entreprise | EJB | 1.1 | | | 2.0 | 2.1 | |
| Intégration | JDBC extension | 2.0 | | | | | XARessource |
| | JNDI | 1.2 | | | J2SE | | CosNaming SPI |
| | RMI/IIOP | 1.0 | | | J2SE | | |
| | JTA | 1.0 | | | | | UserTransaction |
| | JavaMail | 1.1 | | | 1.2 | 1.3 | |
| | JAF | 1.0 | | | | | |
| | JMS | API seulement, pas d'implémentation | | | 1.0 | 1.1 | Queue, Topic |
| XML | JAXP | Non | | | 1.1 | J2 SE | |

## 2.2 XML

## 2.3 Introduction

### 2.3.1 Historique

Replaçons brièvement XML dans l'histoire des documents électroniques.

#### 2.3.1.1 La décennie 80 : SGML et les aides en ligne

SGML (Standard Generalized Markup Language, ou langage normalisé de balisage généralisé), adopté comme standard en 1986 (ISO 8879), a été la première tentative systématique de créer de réels documents électroniques, c'est-à-dire des documents qui n'étaient plus des documents papier sous forme électronique.

La principale idée sous-jacente étant de séparer le contenu (logique) d'un document de sa forme (matérielle/imprimée).

Mais l'intention finale était toujours principalement de produire des documents imprimés, quoique plus économiquement — un unique document (logique) étant transformé automatiquement en différents formats imprimés (par ex. des documents de maintenance pour des avions produits en différents formats selon les normes de présentation de différentes compagnies aériennes ou armées de l'air).

SGML a été une percée, mais il était si complexe que sa manipulation s'est trouvée restreinte aux spécialistes (rédacteurs techniques dans l'industrie ou spécialistes textuels dans les humanités).

Presque parallèle a été le développement de la documentation en ligne, interactive, la première forme de documentation à être purement électronique. Et avec elle est arrivée la popularisation des liens hypertextuels. Mais cette forme de documentation restait une "aide", accessoire à la documentation papier.

#### 2.3.1.2 Les années 90 : Le WWW et HTML

A partir de 1992, le WWW et HTML (Hypertext Markup Language, ou langage de balisage hypertexte, conçu vers 1990) sont devenus une réalité, et ont popularisé les documents électroniques hypertextuels sur une immense échelle. Depuis 1995, les moteurs de recherche ont démontré la stupéfiante capacité de recherche d'information rendue possible par le WWW.

Ainsi, avant l'apparition de XML, existaient :

* un langage de balisage normalisé, riche en sémantique mais relativement lourd à mettre en oeuvre et inadapté au Web : SGML
* un langage parfaitement adapté au Web (puisque développé uniquement pour cette application) mais dont les applications sont limitées par une bibliothèque de balises figée et réduite : HTML

Il convenait donc de définir un langage qui ait la facilité de mise en oeuvre de HTML tout en offrant la richesse sémantique de SGML. C'est la raison d'être de XML.

XML est un sous-ensemble au sens strict de SGML, dont il ne retient pas les aspects trop ciblés sur certains besoins. En cela il représente un "profil d'application" de la norme SGML.

#### 2.3.1.3 L'avenir prévisible de XML

Il est à prévoir que l'usage d'XML va déborder largement le WWW, en provoquant la convergence de deux mondes informatiques jusqu'ici séparés ; celui des documents et celui des données.
Il est très probable qu'il va de ce fait devenir très rapidement la lingua franca de l'informatique, parlée tout autant par les SGBD que par les outils de bureautique et de documentation, par les logiciels de gestion aussi bien que par les applications techniques et scientifiques.

Qu'il va rendre possible une automatisation des activités administratives et logistiques sans commune mesure avec ce que permettent les outils d'aujourd'hui. Et qu'il va considérablement simplifier l'Échanges de Données Informatisé (EDI).

On n'a donc pas fini d'entendre parler de XML !

### 2.3.2 Les possibilités du XML

Comme son nom l'indique, le langage XML se distingue par sa grande extensibilité. Cette capacité représente un atout très net par exemple par rapport au HTML : en effet, outre la possibilité de disposer d'un nombre d'éléments donné, le XML permet de définir des éléments en fonction de ses besoins.

XML va donc permettre aux humains :
- de saisir (ou mettre à jour) une seule fois un contenu (par ex. notre bibliographie) et un contenu pur, sans se soucier de la présentation ou des traitements futurs ; sans avoir à saisir des libellés tels que « auteur », « année de parution », et sans avoir à mettre les titres en italique - exactement, donc, à la manière dont on alimenterait une base de données.
- Et d'en générer ensuite automatiquement :
  - de multiples présentations (en tableau, en texte suivi…)
  - avec éventuellement tris, sélections, réorganisations, génération automatique de libellés, tables des matières, index, etc.
  - et ce sur de multiples médias (écran, papier, terminal Braille, etc.)

XML va également permettre aux logiciels de comprendre/exploiter au mieux le contenu de ces pages, rendu désormais explicite par un balisage spécifique, indépendant de toute application.

### 2.3.3 Les outils complémentaires au XML

Il existe un certain nombre d'outils et de compléments permettent de représenter les documents .xml sous la forme souhaitée ou de les convertir dans d'autres formats.
Nous verrons en particulier la DTD (définition de type de document) et le XSL, outil de transformation et de formatage.

## 2.4 XSL

## 2.5 Introduction : Les fonctions de XSL

Le langage XSL (extensible style langage) a été développé spécialement pour le XML. Il est formé de deux parties (Considérons que XSLT et XPath sont réunis) : un langage de transformation (xslt) et un langage de formatage (XSL-FO), tous deux étant des applications XML.
Le langage XSL a connu des modifications de fond ces dernières années ; les pages suivantes montrent l'état actuel d'avancement du XSL : www.w3.org/TR/2000/WD-xsl-20001011 et www.w3.org/TR/1999/REC-xslt-19991116

## 2.6 Les deux fonctions de XSL

- Le langage de formatage comporte des éléments chargés de définir de quelle manière les éléments XML doivent être affichés (cela concerne par exemple la police, la taille, la couleur, etc.)
- Le langage de transformation comporte des éléments chargés de définir de quelle manière un document .xml doit être transformé en un autre format de sortie ( .html, .rtf, .pdf, etc.). Dans ce cadre, le document .xml transformé peut utiliser la DTD initiale ou une nouvelle DTD. Il est possible également d'intégrer des objets de formatage supplémentaires destinés à l'affichage.

## 2.7 Transformation

### 2.7.1 Principe

Une transformation XSL peut être utilisée pour changer le format d'un document .xml en un document .html, .xml, .pdf, .rtf, .tex, etc.
Elle s'effectue à l'aide : - du document .xml à transformer bien sûr
- des règles définies dans une feuille de style XSL.

### 2.7.2 Modalités d'exécution de la transformation

La transformation peut être exécutée selon trois modalités :
- Le fichier .xml et .xsl est transféré vers le client, et c'est le navigateur de ce dernier qui se charge de la transformation puis de l'affichage du fichier .html. Ce n'est pas une bonne solution car la plupart des navigateurs n'effectuent pas correctement ces opérations.
- La transformation est exécutée en ligne, et le résultat est transféré vers le client. Cette procédure est particulièrement indiquée dans le cas où les fichiers .xml sont fréquemment modifiés ou générés à partir d'une base de données.
- La transformation est exécutée avant sa publication sur le serveur. Cette solution est indiquée pour des pages XML statiques dont la fréquence de modification est faible.

### 2.7.3 Les règles de la transformation

*a) Généralités*

Le code xml bien formé (c'est-à-dire qui respecte un certain nombre de règles imposées) représente une structure arborescente.
Exemple :

```
< method>
        <sequence>
                <statement>
                <affectation>
                …
        </sequence>
</method>
```

Un arbre est une structure de données constituée de nœuds et de liens, et dont la racine est le nœud racine (ou élément racine).
Chaque nœud peut comporter un nombre quelconque de sous-nœuds (ou feuilles). Les nœuds d'un arbre XML sont les éléments et leur contenu.

*b) Règles de transformation*

Une transformation XSL prend en entrée une arborescence XML (ou un fragment de document .xml) et génère en sortie une nouvelle arborescence XML.
Le processeur XSL, au cours de la transformation, traite de la même manière les attributs, les espaces de noms, les instructions et les commentaires comme des nœuds.
Par conséquent, il utilise sept types différents de nœuds :

- root
- éléments
- texte
- attributs
- espace de noms
- instructions
- commentaires

Les DTD et la déclaration de type de document sont en revanche ignorées par le processeur XSL, sauf si on associe d'autres documents .xml dans la DTD, par exemple par l'intermédiaire d'une entité.

Le XSL utilise le XML afin d'établir les règles définissant les modalités de la transformation. Le document lui-même est un élément xsl:stylesheet. L'élément xsl y est utilisé comme espace de noms.

L'URL correspondant à l'espace de noms est la suivante : http://www.w3.org/1999/XSL/transform ou http://www.w3.org/1999/XSL/Format (ces deux URL sont équivalentes).

Les règles de transformation permettent de :

- définir une règle s'appliquant à tous les nœuds ou éléments portant le nom xxx
- lier tous les sous nœuds
- d'adresser l 'élément racine sans connaître son nom
- d'appliquer des feuilles de style
- de copier le contenu d'un nœud précis du document d'entrée vers le document de sortie.

### 2.7.4 XPATH : travail sur les nœuds

*a) Introduction*

Le langage XPATH définit des motifs, des fonctions et des expressions, l'objectif étant d'identifier des éléments, des groupes d'éléments ou des attributs avec une meilleure précision.

Cette fonctionnalité a été retirée du langage XSL pour être placée dans un domaine à part, soit XPATH, permettent ainsi aux programmeurs de disposer d'un standard unifié pour toutes les applications XML.

*b) Sélection des nœuds*

L'attribut « select » détermine quel nœud doit être traité. Il est possible à cette occasion d'utiliser, pour la sélection, des éléments XPATH tels que des axes, des types de nœuds et des types d'expression.
- les axes
- abréviations
- les types de nœuds :

L'indication de l'axe (la suite des deux points : : ) peut être suivie non seulement d'un nom de nœud ou d'un marque place mais il est également possible d'utiliser les types de nœuds suivants :
  - comment ( ) : sélection d'un nœud commentaire
  - text ( ) : sélection d'un texte
  - processing-instruction ( ) : sélection d'une instruction PI
  - node ( ) : sélection d'un des types de nœuds précédents
- Les types d'expression :
  XPATH comporte plusieurs types d'expressions permettent de spécifier le résultat avec une bonne précision/
  - les ensembles de nœuds : ce sont des listes de nœuds du document d'entrée
  - les expressions booléennes : le XSL permet de transformer n'importe quel type de données en une expression booléenne. Cette opération est fréquemment réalisée de manière automatique lorsqu'une expression booléenne est attendue mais qu'une chaîne de caractères ou un nombre est passé en paramètre.
  - les nombres : dans XML et XSL, les nombres sont toujours des nombres 64 bits à virgule flottante, conformément au standard IEEE 754 (nombres à précision double).
  - chaînes de caractères : En XSL, toutes les chaînes de caractères (string) sont représentées sous forme de caractères unicode. D'autres types de données peuvent être transformés en chaîne de caractères à l'aide de la fonction string ( ).

*c) Utilisation des nœuds*

Il est possible d'utiliser les nœuds pour réaliser de nombreuses autres opérations :
- copie de nœuds : cette fonctionnalité est principalement utilisée afin de générer un document de sortie équivalent
- comptage de nœuds : l'élément xsl :number permet de compter les éléments et de les insérer dans le document de sortie
- tri de nœuds : cette fonctionnalité permet de copier dans le document de sortie des nœuds qui ont été triés. Des tris croissants ou décroissants peuvent être effectués.

### 2.7.5 Génération de la sortie

Il arrive souvent que la sortie générée dépende de certains éléments de l'entrée. Dans ces conditions, il est nécessaire dans un premier temps de lire l'entrée intégralement avant de pouvoir générer la sortie.
Les opérations possibles sont :
- utilisation de modèles de valeurs d'attributs : les modèles d'attributs copient les valeurs d'éléments dans les valeurs d'attributs de la sortie
- insertion d'un élément dans le document de sortie.
- Insertion des attributs dans le document de sortie.
- Insertion des instructions de traitement dans le document de sortie.
- Insertion de textes dans le document de sortie
- Insertion de commentaires dans le document de sortie

### 2.7.6 Formatage

#### *2.7.6.1 Introduction*

*a) Mise en page*

Les objets de formatage XSL (abrégé fo) prennent en charge un modèle de mise en page visuelle puissant, qui surpasse ce que le HTML propose à l'aide des feuilles de style en cascade (css).

*b) Description du modèle de formatage*

- On doit déclarer l 'espace de noms xsl pour la transformation et l'espace de noms xsl pour les objets de formatage.
- Le modèle de formatage XSL est basé sur des zones rectangulaires (des boîtes) appelées «areas ».
- Les éléments XSL prennent en charge de nombreuses propriétés qui sont définies par des attributs.
- Les fichiers fo sont traduits à l'aide de la bibliothèque FOP du projet Apache.

#### *2.7.6.2 Mise en page*

Elle consiste à définir les propriétés de la page (c'est-à-dire la largeur, la hauteur, les marges, etc.), la zone d'impression, l'unité de mesure (centimètre, millimètre, pouce, point, pixel ou picas).
Il est possible de définir également des mises en pages différenciées.

Par ailleurs, nous disposons de divers objets de formatage destinés au contenu, c'est-à-dire permettent la création de points, de lignes et de bordures.

Nous pouvons également inclure dans les documents finaux des images et des graphiques (en particulier les images SVG).

Le programme fo offre de nombreuses possibilités pour la mise en forme de texte. (Police de caractères, couleurs, décorations de texte)

## 2.8 SAX

Cette API a été définie sur la liste *xml-dev* hébergée par xml.org. Elle a été la première API XML largement utilisée et était à l'origine dédiée au langage Java. Il existe maintenant nombre d'implémentations pour d'autres langages de programmation (dont C, C++, Perl, Python ou Lisp). Cette API est devenue un standard *de facto* largement adopté.

Les spécifications SAX sont dans le domaine public et l'on peut les télécharger à l'adresse http://www.saxproject.org/. Ces spécifications en sont à la version 2 qui est maintenant implémentée par les principaux parsers XML.

### 2.8.1 Principes généraux

Cette API est essentiellement *event driven*. Un parser SAX ne construit pas d'arbre du document en mémoire mais appelle (*callbacks*) les méthodes d'un *handler* lorsqu'il rencontre des évènements particuliers (comme l'ouverture ou la fermeture d'un élément). On peut comparer la programmation d'un tel parser à du développement d'interfaces graphiques où l'on gère des évènements générés par l'utilisateur.

L'architecture d'une application SAX est présentée dans la figure ci-dessous.

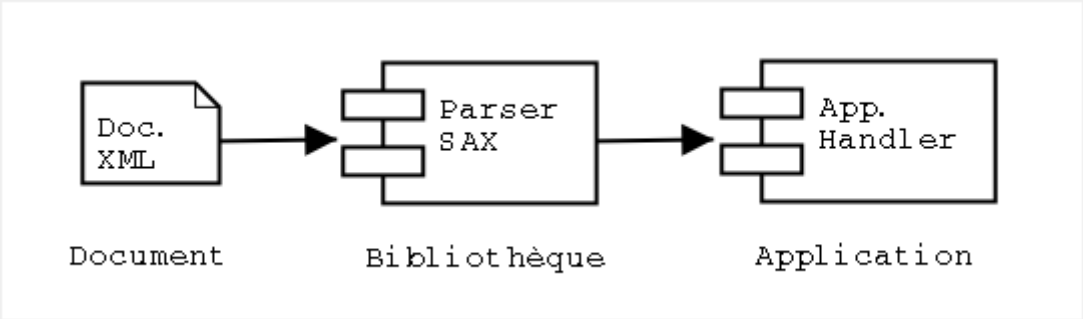



Figure 1: Architecture d'une application SAX

### 2.8.2 Exemple d'application SAX

Soit un document simple, comme celui du source ci-dessous :

*hello.xml*

```
<?xml version="1.0"?>
<doc>
 <para>Hello, world!</para>
</doc>
```

Lors du parsing SAX de ce document, il sera envoyé à l'application les événements suivants:

```
start document
start element: doc
start element: para
```

```
characters: Hello, world!
end element: para
end element: doc
end document
```

Le source d'un *handler* pour écrire la trace ci-dessus pourrait être le suivant :

*SaxParse.java*

```
public void startDocument() {
    System.out.println("start document");
}

public void endDocument() {
    System.out.println("end document");
}

public void startElement(String nameSpace,String name,
                         String qname,Attributes atts) {
    System.out.println("start element: "+name);
}

public void endElement(String nameSpace,String name,String qname) {
    System.out.println("end element: "+name);
}

public void characters(char[] ch,int start,int length) {
    String string=(new String(ch,start,length)).trim();
    if(string.length()>0)
        System.out.println("characters: "+string);
}
```

Lors du parsing de ce document, le parser SAX appelle les méthodes du *ContentHandler* qui affiche les événements sur la sortie standard.

### 2.8.3 Survol de l'API SAX

Après cette brève présentation de SAX, nous allons nous pencher en détail sur l'API SAX 2.0 :

#### *2.8.3.1 XMLReader*

Cette interface est implémentée par les parsers. Elle comporte en particulier une méthode pour parser un document, activer ou désactiver des features (comme la validation par exemple) ou des propriétés ainsi que des méthodes pour indiquer au parser les *handlers.* (pour gérer le contenu du document, les erreurs, etc.)

Les principales méthodes sont résumées ci-dessous :

| Méthode | Description |
|---|---|
| `parse(String file)` | Parse le fichier passé en paramètre. |
| `setFeature(String name,boolean flag)` | Active ou désactive (suivant la valeur du flag) une feature du parser. Les features du parser permettent de choisir si le document parsé sera validé (c'est à dire vérifié s'il est conforme à sa DTD) par exemple. Une liste des features standards se trouve à l'adresse http://www.saxproject.org/apidoc/org/xml/sax/package-summary.html. |
| `setContentHandler(ContentHandler handler)` | Indique au parser quelle classe appeler lorsqu'il rencontre des événements à notifier. |
| `setErrorHandler(ErrorHandler handler)` | Indique au parser quelle classe appeler pour la gestion des erreurs. |

Le ContentHandler étant une interface, on ne peut bien sur l'instancier et l'on doit faire appel à une *factory* qui se trouve dans le package org.xml.sax.helper. Le nom de la classe du parser SAX utilisée doit être indiqué dans la propriété système org.xml.sax.driver.

Par exemple, le code suivant instancie un parser Xerces:

```
if(System.getProperty("org.xml.sax.driver")==null)
    System.setProperty("org.xml.sax.driver",
        "org.apache.xerces.parsers.SAXParser");
parser=XMLReaderFactory.createXMLReader();
```

Il est important de tester la propriété système avant de l'assigner de manière à ce qu'il soit possible de changer de parser en renseignant la propriété sur la ligne de commande qui lance l'application. On pourra le faire avec la ligne suivante :

```
java -Dorg.xml.sax.parser=mon.parser.xml MonAppli
```

Le choix d'un parser est alors une simple question de configuration (il suffit de modifier le script de lancement de l'application et de mettre le fichier jar de la bibliothèque dans le CLASSPATH). On pourra ainsi changer de parser facilement pour choisir le plus adapté à ses besoins.

#### 2.8.3.2 ContentHandler

Cette interface doit être implémentée par la classe qui doit être notifiée des événements rencontrés lors du parsing. Comme nous l'avons vu ci-dessus, elle implémente des méthodes appelées lorsque le parser ouvre ou ferme le document ou un élément :

| Méthode | Description |
|---|---|
| `startDocument()` | Début du document. |
| `endDocument()` | Fin du document. |
| `startElement(String nsURI,String name,String qName,Attributes atts)` | Élément ouvrant. `nsURI` est l'URI de l'espace de nommage de l'élément, `name` son nom, `qName` son nom avec le préfixe de l'espace de nommage et `atts` la liste des attributs. |
| `endElement(String nsURI,String name,String qName)` | Élément fermant. Les attributs ont la même signification que pour l'élément ouvrant. |
| `characters(char[] ch,int start,int length)` | Texte passé sous forme d'un tableau de caractère, un indexe de début et la longueur du texte. |

On prendra garde à **épurer** le texte envoyé par le parser. En effet, celui comporte souvent des blancs sans signification (de simples retours de ligne par exemple). Pour ce faire, on pourra faire le test suivant à l'entrée de la méthode characters() :

```
String text=(new String(ch,start,end)).trim();
if(text.length()==0) return;
```

Il est souvent intéressant d'étendre DefaultHandler (du package org.xml.sax.helpers) plutôt que d'implémenter l'interface ContentHandler. En effet, cette classe abstraite implémente les interfaces EntityResolver, DTDHandler, ContentHandler et ErrorHandler avec des méthodes qui ont un comportement raisonnable. Il n'est plus alors nécessaire d'implémenter toutes les méthodes, mais seulement celles qui nous intéressent (par exemple startElement()).

#### 2.8.3.3 ErrorHandler

Cette interface doit être implémentée par les *handlers* de gestion des erreurs de parsing. Elle définit les méthodes suivantes :

| Méthode | Description |
|---|---|
| `warning(SAXParseException exception)` | Cette méthode est appelée lorsqu'un warning est levé. On peut parfois ignorer de tels warnings et continuer le parsing. |
| `error(SAXParseException exception)` | Cette méthode est appelée en cas d'erreur de parsing. |
| `fatalError(SAXParseException exception)` | Cette méthode est appelée en cas d'erreur fatale. De telles erreurs ne doivent être ignorées dans la mesure où le parsing ne peut plus se dérouler correctement. Quoi qu'il en soit, même si l'on ignore ces erreurs, le parsing est interrompu après une erreur fatale. |

En implémentant ces méthodes, on contrôle finement la gestion des erreurs. Il est ainsi possible d'ignorer les warnings et de continuer le parsing. Si l'on souhaite continuer le parsing, il suffit de ne pas lancer d'exception, dans le cas contraire, on relancera l'exception passée en paramètre. Il est cependant peu courant d'avoir à implémenter son propre ErrorHandler (on se contentera alors du comportement par défaut).

Il peut être utile d'écrire sa propre implémentation pour un validateur de document XML. Avec l'ErrorHandler suivant :

```
List errors=new ArrayList();

public void warning(SAXParseException exception) {
    errors.add(exception);
}

public void error(SAXParseException exception) {
    errors.add(exception);
}

public void fatalError(SAXParseException exception) {
    errors.add(exception);
    throw exception;
}
```

On empile les erreurs de parsing permettant ainsi de lister toutes les erreurs de syntaxe d'un document (par défaut, toute erreur interrompt le parsing).

#### *2.8.3.4 EntityResolver*

Ce *handler* permet une gestion fine des entités externes. Cette interface ne définit qu'une seule méthode :

| Méthode | Description |
|---|---|
| `InputSource resolveEntity(String publicId,String systemId)` | Cette méthode renvoie une `IputSource` à partir d'un identifiant *public* (nom symbolique désignant une ressource) ou *system* (URI d'un fichier). |

On peut alors définir la localisation d'un fichier dans un catalogue. Il existe plusieurs standards de catalogues XML, comme le standard OASIS ou le standard XML Catalog.

#### *2.8.3.5 Gestion des exceptions*

La gestion des exceptions par un parser SAX est assez particulières. L'API SAX définit deux exceptions importantes :

| Exception | Description |
|---|---|
| `SAXException` | Cette exception est lancée lorsqu'une erreur interne se produit dans le parser (y compris dans une méthode d'un *handler*). Elle contient une méthode `getException()` qui permet de récupérer l'exception à l'origine de cette erreur. |
| `SAXParseException` | Cette exception, qui étend `SAXException`, est lancée lorsque le document est *mal formé* ou *invalide*. Elle comporte les méthodes `getLineNumber()` (pour récupérer le numéro de ligne où se trouve l'erreur) et `getPublicId()` (pour récupérer le nom du fichier où se trouve l'erreur) qui permettent d'afficher un message clair pour les erreurs de parsing. |

Pour afficher correctement les Exceptions lancée par la méthode parse() du parser, on pourra s'inspirer du source suivant :

```
try {saxParse.parse(args);}
catch(SAXParseException e) {
    System.out.println("Erreur de syntaxe dans le fichier XML:");
    String message=e.getSystemId()+":"+e.getLineNumber()+": "+
        e.getMessage();
    System.out.println(message);
}
catch(SAXException e) {
    System.out.println("Erreur interne du parser:");
    e.printStackTrace();
}
catch(Exception e) {
    System.out.println("Erreur inconnue:");
```

```
    e.printStackTrace();
}
```

Il est essentiel d'afficher clairement les erreurs de parsing sans quoi l'utilisateur sera incapable de corriger l'erreur dans le fichier XML (surtout si celui-ci est très long).

## 2.9 DOM

### 2.9.1 Principes généraux

Cette API (dont le nom est l'acronyme de Document Object Model) a une approche radicalement différente de SAX : un parser DOM charge en mémoire **tout** le document et en fait une représentation sous forme d'un arbre d'objets. On pourrait donc schématiser le fonctionnement d'un parser DOM par la figure suivante :

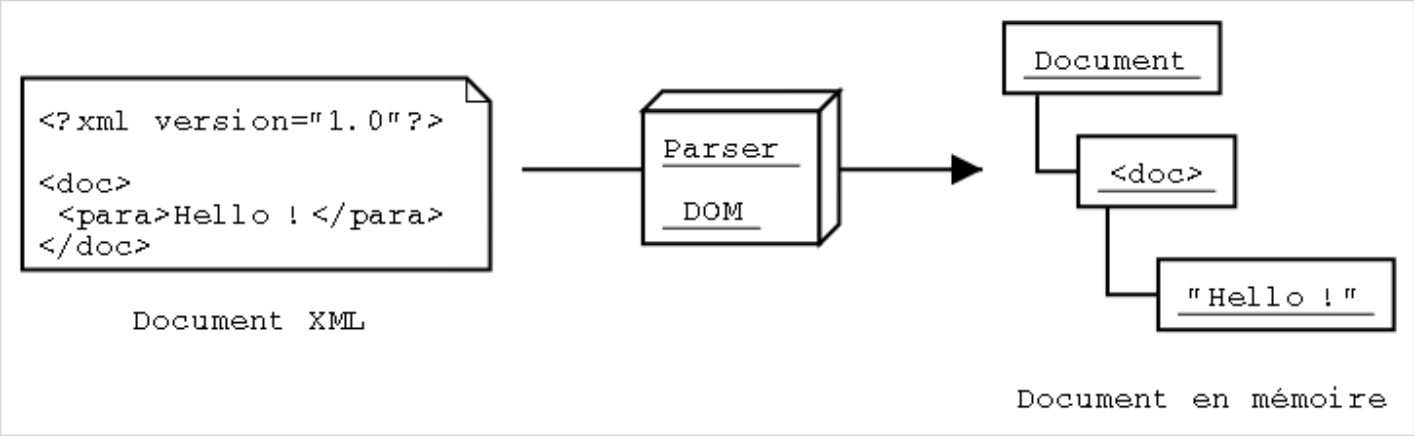



Figure 2: Fonctionnement d'un parser DOM

Le document étant entièrement en mémoire, il est possible de le parcourir (en parcourant l'arbre) et de le manipuler (en déplaçant, en créant ou en supprimant des noeuds).

#### 2.9.1.1 Exemple d'application DOM

Nous allons maintenant voir un exemple d'application DOM simple pour parcourir le document et écrire sa structure sur la sortie standard (donc l'équivalent en DOM de l'exemple SAX).

Pour commencer, il nous faut récupérer une instance du parser DOM (dans le cas ci-dessous, le parser DOM de Xerces). Nous devons ensuite récupérer le document (c'est à dire une référence vers un objet org.w3.dom.Document). Nous pouvons le faire de la manière suivante :

```
DOMParser parser=new DOMParser();
parser.setFeature("http://xml.org/sax/features/validation",true);
parser.parse("monDocument.xml");
Document doc=parser.getDocument();
```

Nous pouvons noter que :

- Le code est étroitement lié au parser utilisé (si l'on change de parser, il nous faudra modifier le code), aussi bien pour l'instanciation du parser que pour la récupération du document. Pour une solution à ce problème, voir la section JAXP.
- Il existe pour ce parser un système de paramétrage par *feature*, comme pour les parsers SAX. C'est logique dans la mesure où la plupart des parsers SAX reposent sur un parser SAX pour le parsing du document.

Nous devons ensuite écrire récursivement l'arbre du document sur la sortie standard :

```
void write(Node node,int depth) {
    // test for null nodes
    if (node==null) return;
    // get the type of the node
    switch (node.getNodeType()) {
        // The node is a document
        case Node.DOCUMENT_NODE: {
            writeTree(depth);
            System.out.println("Document");
            break;
        }
        // The node is an element
        case Node.ELEMENT_NODE: {
            Element element=(Element) node;
            writeTree(depth);
            System.out.println("Element: "+element.getTagName());
            NamedNodeMap atts=element.getAttributes();
            for(int i=0;i<atts.getLength();i++)
                write(atts.item(i),depth+1);
            break;
        }
        // The node is an attribute
        case Node.ATTRIBUTE_NODE: {
            Attr att=(Attr) node;
            writeTree(depth);
            System.out.println("Attribute: "+
                               att.getName()+"="+att.getValue());
            break;
        }
        // The node is text
        case Node.TEXT_NODE: {
            Text text=(Text) node;
            writeTree(depth);
            System.out.println("Text: "+text.getData().trim());
            break;
```

```
        }
        // The node is a comment
        case Node.COMMENT_NODE: {
            Comment comment=(Comment) node;
            writeTree(depth);
            System.out.println("Comment: "+
                               normalizeText(comment.getData().trim()));
            break;
        }
    }
    // write the child nodes (except for attributes)
    if(node.getNodeType()!=Node.ATTRIBUTE_NODE) {
        NodeList children=node.getChildNodes();
        sibling[depth+1]=children.getLength()>0;
        for(int i=0;i<children.getLength();i++) {
            if(i==children.getLength()-1) sibling[depth+1]=false;
            write(children.item(i),depth+1);
        }
    }
}
```

Cette méthode parse récursivement l'arbre du document et affiche la nature des noeuds de ce dernier. On notera au passage qu'il existe une interface Node qui est implémentée par les interfaces Element, Attr (pour Attribute), Comment ou Text (entre autres). Ces interfaces sont implémentées par des classes concrètes qui dépendent de l'implémentation. Ce design implique une gymnastique pour créer de nouveaux noeuds (en passant par des *factories* du document). Par exemple, pour créer un nouvel élément Foo et l'ajouter à bar, on écrira :

```
Node foo=doc.createElement("Foo");
bar.appendChild(foo);
```

Il existe des méthodes (dans l'interface Node) pour ajouter, supprimer ou déplacer des noeuds d'un document. C'est une fonctionnalité intéressante des parsers DOM : ils permettent la manipulation des documents (ce qui est très difficile de faire avec SAX).

D'autre part, cette API est générique (liée à aucun langage) et ne peut donc se reposer sur des APIs particulières (comme les collections du package java.util de Java) et doit fournir ses propres collections. C'est pourquoi, lorsqu'on demande la liste des attributs, ils ne sont pas renvoyés sous forme d'une collection Java (une HashMap par exemple), mais sous forme d'une NamedNodeMap.

## *2.10 JAXP*

### 2.10.1 Principe général

L'API JAXP (pour Java API for XML Parsing) est une initiative de Sun pour uniformiser les APIs de parsing des différents parsers XML et processeurs XSLT Java. On peut alors écrire du code complètement indépendant du parser utilisé (il suffit souvent de changer le nom de la classe du parser dans un fichier de configuration).

Cette API est particulièrement intéressante pour les parsers DOM et les processeurs XSLT. En effet, DOM définit l'interface des méthodes pour manipuler les documents, mais rien en ce qui concerne l'instanciation du parser ou la récupération du document. Pour ce qui est des processeurs XSLT, il n'existe pas (à part JAXP) d'API standard pour les utiliser dans du code Java. Par contre, l'API SAX définit une API indépendante du parser et l'on pourra sans regrets se passer de l'API de Sun.

Cette API est donc une surcouche aux parsers et elle est très légère (elle ne comporte en tout et pour tout que 7 classes ou interfaces).

### 2.10.2 Exemple de code JAXP

Du fait de sa nature, il n'est pas intéressant de présenter une application JAXP (qui serait pour l'essentiel conforme aux APIs SAX ou DOM). Je me contenterai donc ici de présenter des bouts de code pour instancier les parsers et processeurs.

Pour instancier un parser SAX, on pourra écrire :

```
SAXParserFactory spf=SAXParserFactory.newInstance();
spf.setValidating(true);
XMLReader xmlReader=spf.newSAXParser().getXMLReader();
```

À partir de ce moment, le reste du code doit se conformer à l'API SAX. On notera, comme indiqué plus haut, que l'on peut aussi écrire du code paramétrable avec l'API SAX.

Pour instancier un parser DOM et récupérer un document, on pourra écrire :

```
DocumentBuilderFactory dbf=
    DocumentBuilderFactory.newInstance();
dbf.setValidating(true);
DocumentBuilder db=dbf.newDocumentBuilder();
Document doc=db.parse(new File(filename));
```

On pourra ensuite manipuler le document récupéré avec l'API DOM.

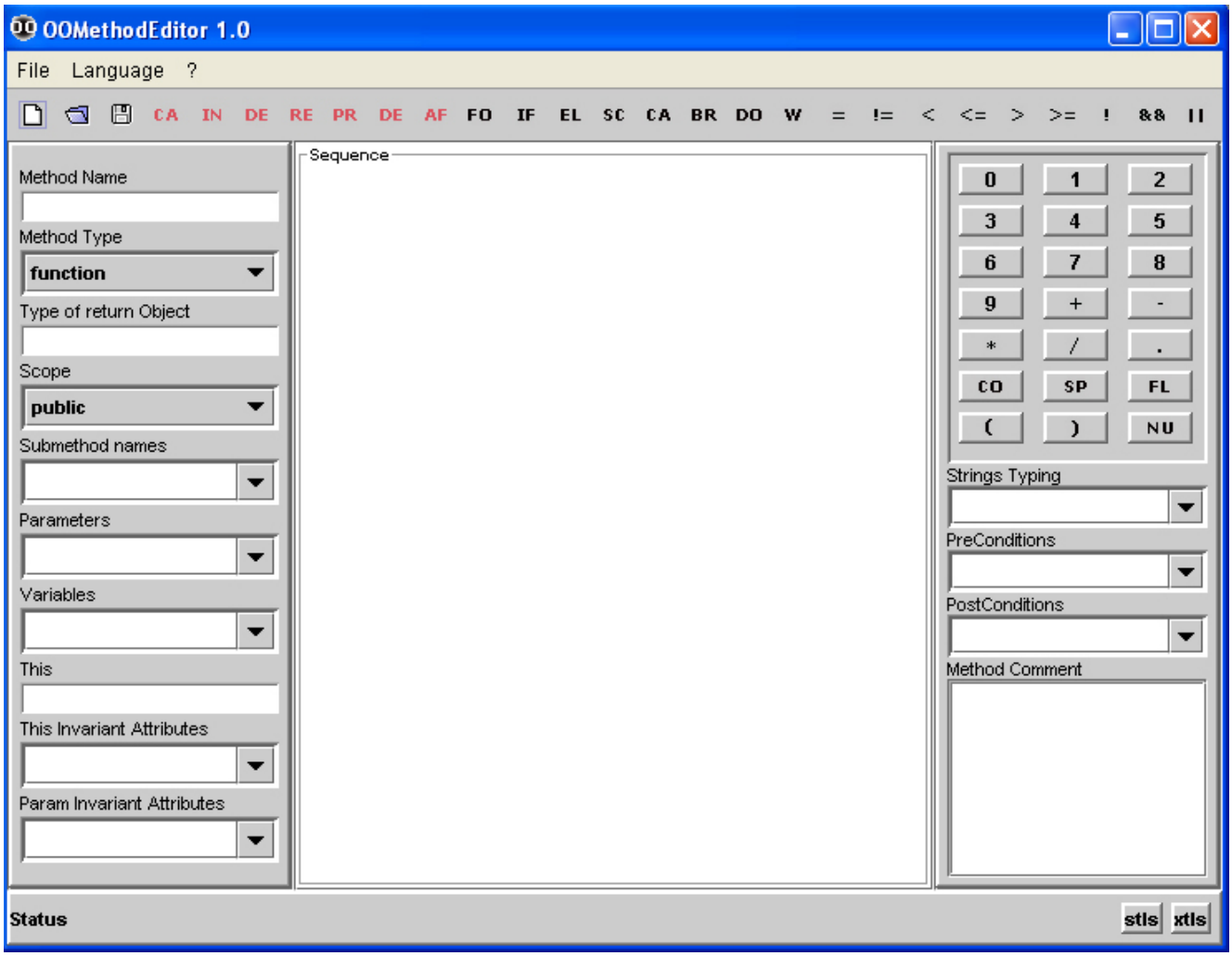
Pour finir, pour transformer un document XML en utilisant un processeur XSLT, on écrira :

```
TransformerFactory tf=TransformerFactory.newInstance();
Transformer transformer=tf.newTransformer(new
    StreamSource(new File("style.xsl")));
transformer.transform(new StreamSource(new File("source.xml")),
                      new StreamResult(new File("dest.html")));
```

Cette API est appelée TRAX (pour Transformation API for XML). C'est la seule API répandue pour la transformation XSLT et elle est implémentée par le processeur XSLT Apache (Xalan).

# 3 FONCTIONNALITES DU PROGRAMME

Interface de OOMethodEditor :




Le menu « Language » n'est pas utilisé pour choisir la langue de l'utilisateur, mais est utilisé pour choisir le langage cible.

La première chose intéressante est la barre d'outils avec ses deux principales commandes « insert » et « delete » utilisées pour la gestion des valeurs utiles stockées dans les parties gauches et droites.

Les panneaux gauche et droit destinés à recueillir les valeurs utiles et descriptives de la méthode (variables, paramètres, commentaires …)

L'éditeur au centre affichant la séquence de la méthode en cours de construction. Il n'est pas réactif aux frappes du clavier mais récupère les informations transmises par le biais de la souris qui a une fonction de gomme et d'indicateur de position d'éléments à insérer.

Enfin, tout en bas, un panneau contenant un indicateur d'état et deux boutons pour la barre d'outils.

## 3.1 Barres d'outils

OOMethodEditor possède deux barres d'outils :

- Outils standards
- Outils supplémentaires

### 3.1.1 Barre d'outils standard


Les trois premiers boutons sont généraux : « nouveau », « ouvrir » et « enregistrer ».

Les boutons rouges correspondent à des fonctions d'accès aux éléments à insérer ou représentent des instructions ayant des syntaxes différentes des principaux langages de programmation et à l'opérateur de Cast utilisé couramment. Dans l'ordre nous avons : « cast », « insert », « delete », « return », « print », « declaration » et « affectation ».

Les boutons noirs correspondants à des lettres sont des contrôles communs aux langages de programmation. Leurs syntaxes sont quasiment identiques : « for », « if », « else », « switch », « case », « break », « dowhile » et « while ».

Le reste ne nécessite aucune précision, sinon qu'ils sont absolument identiques aux opérateurs Java.

### 3.1.2 Barre d'outils supplémentaires

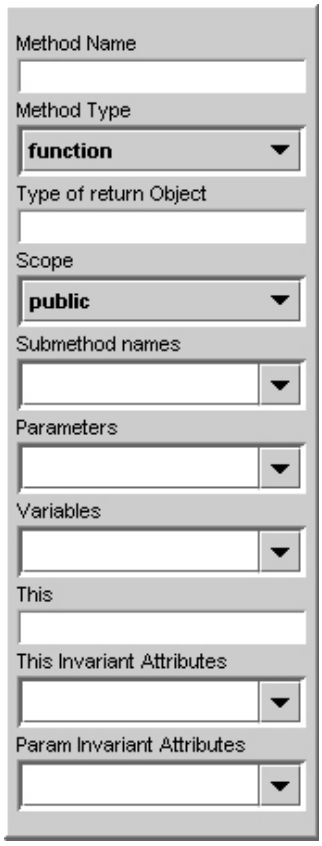

Vous reconnaîtrez certainement les premiers : « modulo », « et » et « ou ».

Quant aux suivants, les voici : « or exclusive », « shift left », « shift right », « shift right logical », « complement to one », « increase » et « decrease ».

Prenez garde à ne pas confondre avec « insert » et « delete ».

## 3.2 Partie gauche

Les descriptions suivantes pourront vous paraître confuses à ce stade mais le seront moins à la suite de la lecture de l'exemple donné plus bas.




Le nom de la méthode est modifié directement dans la zone de texte.

Le type de la méthode est sélectionné dans la liste déroulante non modifiable. (Il s'agit soit d'une procédure, soit d'une fonction)

Le type de l'objet retourné est le type qui sera renvoyé par la méthode dans le cas d'une fonction. Il est rendu non modifiable dans le cas d'une procédure.

La portée de la méthode est sélectionnée dans la liste déroulante non modifiable. (*Note* : Tout ce qui est au-dessus décrit la signature de la méthode)

Les noms des sous méthodes permettent de décrire les signatures des méthodes appelées par la principale méthode en cours de construction. La saisie est effectuée dans une liste dynamique.

Les paramètres de la méthode ainsi que leurs types sont stockés ici.

Les variables locales et globales sont stockées ici ainsi que leurs types.

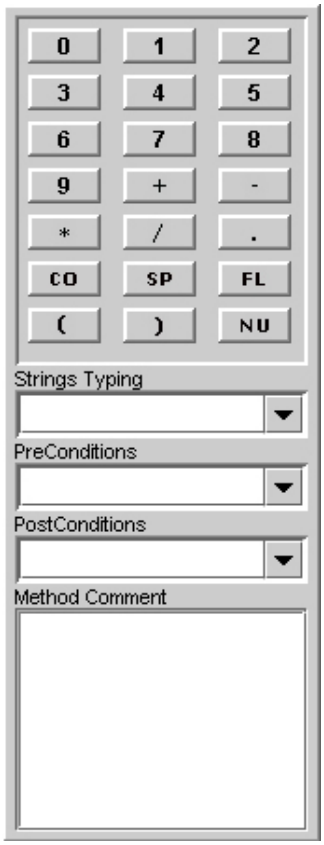
This représente l'objet qui appelle la méthode dans le cas d'une méthode déclarée non statique.

Les attributs « invariants » de this sont définis ici et automatiquement « collés » à this lors de leur insertion.

Les attributs « invariants » des paramètres sont quant à eux associés à leurs paramètres respectifs.

## 3.3 Partie droite

Même remarque que pour la partie gauche.




Le pavé numérique élargi à d'autres opérateurs que « + », « - », « / », « * » et « . ».

A savoir : « virgule (CO) », « espace (SP) », « saut de ligne (FL) », les parenthèses et « null ».

La saisie de chaînes permet de stocker des valeurs qui seront insérées entre guillemets.

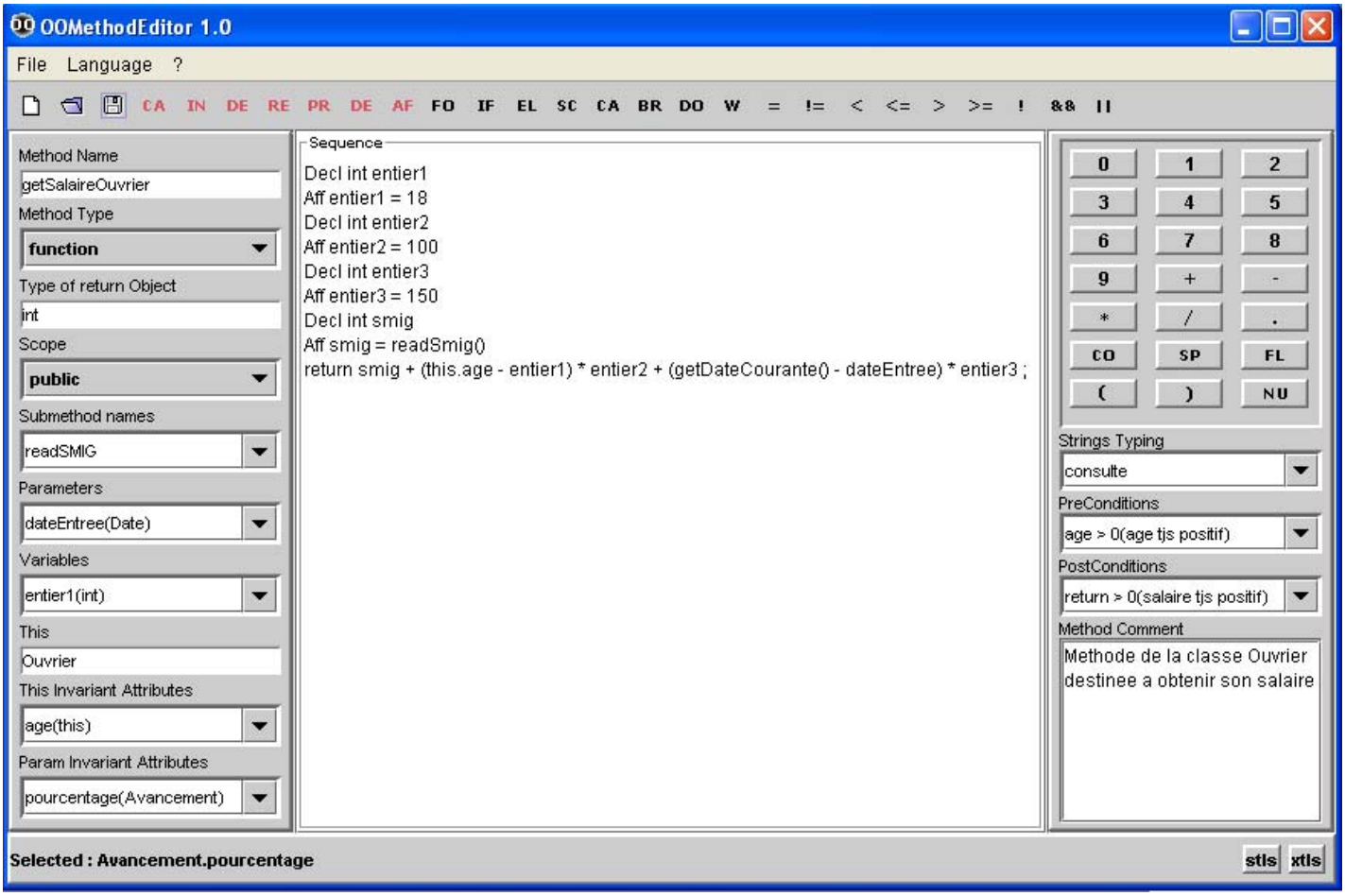
Les pré conditions sont définies ici en deux parties : d'abord le code à exécuter, puis le commentaire correspondant. (Pour ceux qui utilise Icontrat à la compilation)

Même chose pour les post conditions.

Commentaire général sur la méthode en cours de construction.

## 3.4 Exemple complet d'utilisation

### 3.4.1 L'interface complètement renseignée




Voici une simple méthode de calcul de salaire d'un ouvrier qui n'a, je l'espère, aucun rapport avec la vérité.

On remarque dans le champ Submethod names que readSMIG aurait pu être renseignée ici avec des parenthèses et pourquoi pas, avec des arguments. (Ex : readSMIG(int g, char c)) On constate que dans l'éditeur la méthode a été complétée à l'aide du pavé numérique du côté droit.

Pour les paramètres et les variables, notez que plusieurs champs ici non visibles, sont présents. Il est inutile pour moi de faire des copies d'écran de chaque liste puisque vous les verrez avec XML.

Notez l'objet appelant la méthode : Ouvrier. (Précisément sa classe) Ouvrier est remplacé par this dans l'éditeur à la dernière ligne de pseudo code. Un invariant de Ouvrier est âge par exemple qui est concerné par les pré et les post conditions. Un invariant de paramètres est pourcentage de l'avancement. Lui aussi est concerné par les pré et les post conditions.

Le premier champ à droite, n'a pas d'autre utilité que d'insérer des chaînes entre guillemets dans l'éditeur.

Nous cliquons sur le bouton « enregistrer », sélectionnons un emplacement et rajoutons l'extension .xml au fichier. (Ex : getSalaireOuvrier.xml)

### 3.4.2 Le document XML généré

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!--
   File name : getSalaireOuvrier.xml
  -->
- <METHOD>
  <METHOD_NAME>getSalaireOuvrier</METHOD_NAME>
  <METHOD_TYPE>function</METHOD_TYPE>
  <TYPE_OF_RETURN_METHOD>int</TYPE_OF_RETURN_METHOD>
  <SCOPE>public</SCOPE>
- <SUBMETHOD_NAMES>
  <NAME>readSMIG</NAME>
  <NAME>getDateCourante()</NAME>
    </SUBMETHOD_NAMES>
- <PARAMETERS>
- <PARAMETER>
  <NAME>dateEntree</NAME>
  <TYPE>Date</TYPE>
    </PARAMETER>
- <PARAMETER>
  <NAME>avancement</NAME>
  <TYPE>Avancement</TYPE>
    </PARAMETER>
    </PARAMETERS>
- <VARIABLES>
  <NAME>entier1</NAME>
  <TYPE>int</TYPE>
  <NAME>entier2</NAME>
  <TYPE>int</TYPE>
  <NAME>entier3</NAME>
  <TYPE>int</TYPE>
  <NAME>smig</NAME>
  <TYPE>int</TYPE>
    </VARIABLES>
  <THIS>Ouvrier</THIS>
- <THIS_INVARIANT_ATTRIBUTES>
  <NAME>age</NAME>
  <INVARIANT>this</INVARIANT>
    </THIS_INVARIANT_ATTRIBUTES>
- <PARAMETERS_INVARIANT_ATTRIBUTES>
  <NAME>pourcentage</NAME>
  <TYPE>Avancement</TYPE>
    </PARAMETERS_INVARIANT_ATTRIBUTES>
- <STRINGS_TYPING>
  <STRING>consulte</STRING>
  <STRING>non consulte</STRING>
    </STRINGS_TYPING>
- <PRECOND_COMMENTS>
- <PRECONDITION>
  - <NAME>
  - <![CDATA[
  age > 0
    ]]>
    </NAME>
  <DESCRIPTION>age tjs positif</DESCRIPTION>
    </PRECONDITION>
- <PRECONDITION>
  - <NAME>
  - <![CDATA[
  pourcentage >= 0
    ]]>
    </NAME>
  <DESCRIPTION>Pourcentage tjs positif</DESCRIPTION>
```

```
      </PRECONDITION>
      </PRECOND_COMMENTS>
- <POSTCOND_COMMENTS>
- <POSTCONDITION>
  - <NAME>
  - <![CDATA[
  return > 0
    ]]>
    </NAME>
  <DESCRIPTION>salaire tjs positif</DESCRIPTION>
    </POSTCONDITION>
- <POSTCONDITION>
  - <NAME>
  - <![CDATA[
  pourcentage <= 100
    ]]>
    </NAME>
  <DESCRIPTION>pourcentage tjs inferieur ou egal a 100</DESCRIPTION>
    </POSTCONDITION>
    </POSTCOND_COMMENTS>
  <METHOD_COMMENT>Methode de la classe Ouvrier destinee a obtenir son
      salaire</METHOD_COMMENT>
- <SEQUENCE>
- <INSTRUCTION type="declaration">
- <DECLARATION>
  <TYPE>int</TYPE>
  <IDENTIFIER>entier1</IDENTIFIER>
  <PRIM_TYPE />
    </DECLARATION>
    </INSTRUCTION>
- <INSTRUCTION type="aff">
- <AFF>
  <LEFT_EXPR>entier1</LEFT_EXPR>
  <RIGHT_EXPR>18</RIGHT_EXPR>
    </AFF>
    </INSTRUCTION>
- <INSTRUCTION type="declaration">
- <DECLARATION>
  <TYPE>int</TYPE>
  <IDENTIFIER>entier2</IDENTIFIER>
  <PRIM_TYPE />
    </DECLARATION>
    </INSTRUCTION>
- <INSTRUCTION type="aff">
- <AFF>
  <LEFT_EXPR>entier2</LEFT_EXPR>
  <RIGHT_EXPR>100</RIGHT_EXPR>
    </AFF>
    </INSTRUCTION>
- <INSTRUCTION type="declaration">
- <DECLARATION>
  <TYPE>int</TYPE>
  <IDENTIFIER>entier3</IDENTIFIER>
  <PRIM_TYPE />
    </DECLARATION>
    </INSTRUCTION>
- <INSTRUCTION type="aff">
- <AFF>
  <LEFT_EXPR>entier3</LEFT_EXPR>
  <RIGHT_EXPR>150</RIGHT_EXPR>
    </AFF>
    </INSTRUCTION>
- <INSTRUCTION type="declaration">
- <DECLARATION>
  <TYPE>int</TYPE>
```

```
    <IDENTIFIER>smig</IDENTIFIER>
    <PRIM_TYPE />
      </DECLARATION>
      </INSTRUCTION>
  - <INSTRUCTION type="aff">
  - <AFF>
    <LEFT_EXPR>smig</LEFT_EXPR>
    <RIGHT_EXPR>readSmig()</RIGHT_EXPR>
      </AFF>
      </INSTRUCTION>
  - <INSTRUCTION type="return">
  - <RETURN>
    <STRING_EXPR>smig + (this.age - entier1) * entier2 + (getDateCourante() - dateEntree) *
       entier3</STRING_EXPR>
      </RETURN>
      </INSTRUCTION>
      </SEQUENCE>
      </METHOD>
```

Notez le typage des instructions permettant d'optimiser le traitement.

### 3.4.3 Résultat (Code source Java)

```
/**
getSalaireOuvrier :
Methode de la classe Ouvrier
destinee a obtenir son salaire
        @pre
age > 0
 // age tjs positif
        @pre
pourcentage >= 0
 // Pourcentage tjs positif
        @post
return > 0
 // salaire tjs positif
        @post
pourcentage <= 100
 // pourcentage tjs inferieur ou egal a 100
*/

public int getSalaireOuvrier(Date dateEntree,Avancement avancement) {
entier1 = 18;
        entier2 = 100;
        entier3 = 150;
        smig = readSmig();

                return smig + (this.age - entier1) * entier2 + (getDateCourante() -
dateEntree) * entier3;

}
```

Une fois ouvert dans un éditeur Java, lorsque nous appliquons l'auto indentation, nous obtenons :

```
/**
getSalaireOuvrier : Methode de la classe Ouvrier destinee a obtenir son salaire
@pre age > 0 // age tjs positif
@pre pourcentage >= 0 // Pourcentage tjs positif
@post return > 0 // salaire tjs positif
@post pourcentage <= 100 // pourcentage tjs inferieur ou egal a 100
*/
   public int getSalaireOuvrier(Date dateEntree, Avancement avancement) {
     entier1 = 18;
     entier2 = 100;
     entier3 = 150;
     smig = readSmig();
     return smig + (this.age - entier1) * entier2 + (getDateCourante() - dateEntree) * entier3;
   }
```

### 3.4.4 Zoom sur la séquence d'une méthode

Alors vous allez me dire, « oui mais les instructions sont toutes données à la suite mais si on les imbrique, comment le programme réagit-il ?

Voyons cela :

Sequence

```
while( true ) {
print( "On fait un tour" )
break
}
for( i = 0 , j = 0 ; (i <= 10) && (j <= 10) ; i++ , j++ ) {
print( "Salut le Monde!" )
if( i == 5 ) {
print( "i est egal a 5" )
}
else {
print( "i n'est pas egal a 5" )
}
}
print( "C'est termine" )
```

C'est une séquence constituée de contrôles plus ou moins imbriqués. Notez les variables multiples du contrôle for. (Pour pouvez en déclarer une infinité)

Observons le document XML :

```
- <SEQUENCE>
- <INSTRUCTION type="print">
- <PRINT>
  <STRING_EXPR>"Salut le Monde!"</STRING_EXPR>
     </PRINT>
     </INSTRUCTION>
- <INSTRUCTION type="if">
- <IF>
  <BOOLEAN_EXPR_IF>i = 5</BOOLEAN_EXPR_IF>
- <SEQUENCE>
- <INSTRUCTION type="print">
- <PRINT>
  <STRING_EXPR>"i est egal a 5"</STRING_EXPR>
     </PRINT>
     </INSTRUCTION>
     </SEQUENCE>
     </IF>
     </INSTRUCTION>
- <INSTRUCTION type="else">
- <ELSE>
- <SEQUENCE>
- <INSTRUCTION type="print">
- <PRINT>
  <STRING_EXPR>"i n'est pas egal a 5"</STRING_EXPR>
     </PRINT>
     </INSTRUCTION>
     </SEQUENCE>
     </ELSE>
     </INSTRUCTION>
     </SEQUENCE>
     </FOR>
     </INSTRUCTION>
- <INSTRUCTION type="print">
- <PRINT>
  <STRING_EXPR>"C'est termine"</STRING_EXPR>
     </PRINT>
     </INSTRUCTION>
     </SEQUENCE>
```

L'indentation n'est malheureusement pas respectée dans ce copier/coller contrairement au document source. En revanche l'imbrication et l'ordre oui.

Vérifions :

```
while (true) {
    System.out.print("On fait un tour");
    break;
  }
  for (int i = 0, j = 0; (i <= 10) && (j <= 10); i++, j++) {
    System.out.print("Salut le Monde!");
    if (i == 5) {
      System.out.print("i est egal a 5");
    }
    else {
      System.out.print("i n'est pas egal a 5");
    }
  }
  System.out.print("C'est termine");
```

# 4 CONCLUSION

Manifestement XML à un bel avenir devant lui. Ce stage aura été pour moi l'occasion de m'immerger un peu plus dans son univers.

Qu'il s'agisse de bases de données ou de sites Web, il est impossible à l'heure actuelle de ne pas entendre parler d'XML.

Sans parler des services Web qui utilisent le protocole SOAP pour transporter du XML, les sites Web en font une grande consommation lorsqu'il s'agit de présenter son site selon plusieurs formats : WAP, iMode, HTML.

A ce titre Cocoon facilite grandement son utilisation et m'a permis par ailleurs de réaliser un moteur de recherche sans aucune ligne de code traditionnelle.

Dans le secteur des bases de données, on voit apparaître des bases de données XML et les éditeurs de BDD traditionnelles proposent dorénavant le support XML. (Oracle par exemple)

Les plus grands sites d'E-Commerce s'interfacent sur des GDS à travers des API XML. (Réservations d'avions, trains, hôtels et voitures)

Sans parler du géant de Redmond, Microsoft, qui donnera prochainement une part beaucoup plus importante à XML dans sa suite bureautique Office 11.

Je finis ce stage avec un bagage qui ne sera probablement pas inutile !