

Présentation de l'outil de reporting « JasperReports »



ATOL Conseils et Développements

3 bd Eiffel
21600 LONGVIC
contact@atolcd.com



www.atolcd.com

Tel : 03 80 68 81 68

Fax : 03 80 68 81 61

version 1.1

Présentation de Jasper Reports

Table des matières

1	BESOINS.....	3
1.1	Définitions.....	3
1.2	Panorama des outils de reporting.....	3
2	JASPERREPORTS.....	5
2.1	Présentation générale.....	5
2.2	Obtention du modèle XML.....	8
2.3	Construction du rapport.....	13
2.4	Remplissage des champs.....	14
2.5	Exportation du résultat.....	17
2.6	Génération d'un rapport sans modèle XML.....	18
2.7	JasperReports dans une architecture WEB.....	20
2.8	Exemples de rapports.....	21
3	PRÉSENTATION DE iREPORT.....	24
4	BIBLIOGRAPHIE.....	25

1 BESOINS

1.1 DÉFINITIONS

Les outils de reporting permettent aux analystes et aux décideurs de présenter facilement, pour eux-mêmes, leur direction ou l'extérieur, les données chiffrées de l'entreprise et leur évolution. Le besoin principal consiste à fusionner ces données afin de leur associer des modèles de documents déjà élaborés.

Les opérations à réaliser sont les suivantes :

- l'extraction sélective des informations à partir de tableurs, de bases ou de datawarehouses (structures informatiques dans lesquelles est centralisé un volume important de données consolidées à partir des différentes sources de renseignements d'une entreprise)
- les tris suivant des critères de progression judicieux
- la mise en forme, la plus claire et la plus agréable possible

1.2 PANORAMA DES OUTILS DE REPORTING

Depuis de nombreuses années, les sociétés les plus influentes du marché de l'informatique montrent un intérêt certain au monde du reporting. En effet, afin de concurrencer les leaders mondiaux dans ce domaine (en particulier, *Business Objects*, *Cognos* ou *Hyperion*), **Microsoft** et **Oracle** proposent dorénavant de véritables solutions venant en complément de leur « Système de Gestion de Bases de Données » (SGBD) respectif. Ainsi, ils ne se contentent plus de la répartition des rôles que l'on connaissait auparavant : aux éditeurs de bases de données les fonctions de stockage, aux éditeurs d'outils de reporting les fonctions d'interrogation. D'un côté, *Microsoft* a développé *Reporting Services*, une gamme de produits intégrés à *SQL Server* et basés sur les technologies acquises lors du rachat d'*Active Views*, de l'autre, *Oracle* a renforcé son offre de reporting dans la version *10g* de son célèbre SGBD. Face à ces deux offensives, les leaders du reporting contre-attaquent : **Business Objects** sort sa nouvelle plate-forme, *BusinessObjects XI*, qui reprend en grande partie les éléments de l'infrastructure de *Crystal Decisions* (racheté en 2003) et annonce un accord de partenariat avec *MySQL*, **Cognos** lance *ReportNet*, qui a remplacé l'ancien logiciel de reporting *Impromptu*.

A côté de ces « géants » de l'informatique, d'autres outils de reporting tentent de se faire connaître. Certains sont même développés en Open Source sous licence « Apache Software License », « GNU General Public License » (GPL) ou « GNU Lesser General Public License » (LGPL). Citons, par exemple, des logiciels comme *DataVision*, *JasperReports* ou *JfreeReport*.

Descriptif des différentes catégories d'outils ou d'utilitaires de reporting :

- Logiciels de reporting commerciaux (contiennent à la fois les librairies pour générer les rapports, un éditeur graphique de rapports et des options de déploiement) :
 - **Crystal Reports** (Business Objects : <http://www.france.businessobjects.com>)
 - **ReportNet** (Cognos : <http://www.cognos.com/fr>)
 - **Hyperion Reports** (Hyperion : <http://www.hyperion.com/fr>)
 - **Actuate** (Actuate : <http://www.actuate.com/fr>)
 - **KSL** (KSL, anciennement Kallisto Informatique : <http://www.ksl.fr>)
 - **Reporting Services**
(Microsoft : <http://www.microsoft.com/france/sql/technologies/reporting>)
 - **Oracle Business Intelligence 10g**
(Oracle : http://www.oracle.com/lang/fr/solutions/business_intelligence)
 - ...
- Logiciels de reporting en Open Source (tentent d'offrir les mêmes fonctionnalités que les logiciels de reporting propriétaires) :
 - **DataVision** (<http://datavision.sourceforge.net>) (licence *Apache Software license*)
 - **Agata Report** (<http://www.agata.org.br>) (licence *GPL*)
 - ...
- Librairies de génération de rapports en Open Source :
 - **JasperReports** (<http://jasperreports.sourceforge.net>) (licence *LGPL*)
Il doit être couplé avec un éditeur graphique (Graphical User Interface ou GUI) afin de faciliter la création des rapports. Il en existe un certain nombre que nous allons citer ci-dessous :
 - **JasperAssistant** (<http://www.jasperassistant.com>) : éditeur très complet mais propriétaire, peut s'intégrer dans l'environnement de développement Eclipse
 - **iReport** (<http://ireport.sourceforge.net>) : éditeur complet, pratique et gratuit (licence *GPL*)
 - **OpenReports Designer** (<http://opensourceoft.net>) : éditeur complet et gratuit (licence *GPL*)
 - ...
 - **JFreeReport** (<http://www.jfree.org/jfreereport>) (licence *LGPL*)
De même que JasperReports, il faut le coupler avec un éditeur graphique :
 - **JFreeDesigner** (<http://www.jfree.org/jfreedesigner>) : éditeur gratuit mais encore en cours de développement (licence *GPL*)

2 JASPERREPORTS

2.1 PRÉSENTATION GÉNÉRALE

JasperReports (version 1.1.0) est un outil (bibliothèque) Open Source puissant utilisé pour la génération d'états. Il permet de créer des rapports à partir de fichiers XML. Le résultat peut être affiché à l'écran, imprimé ou stocké dans des fichiers au format PDF, HTML, XLS, CSV ou XML.

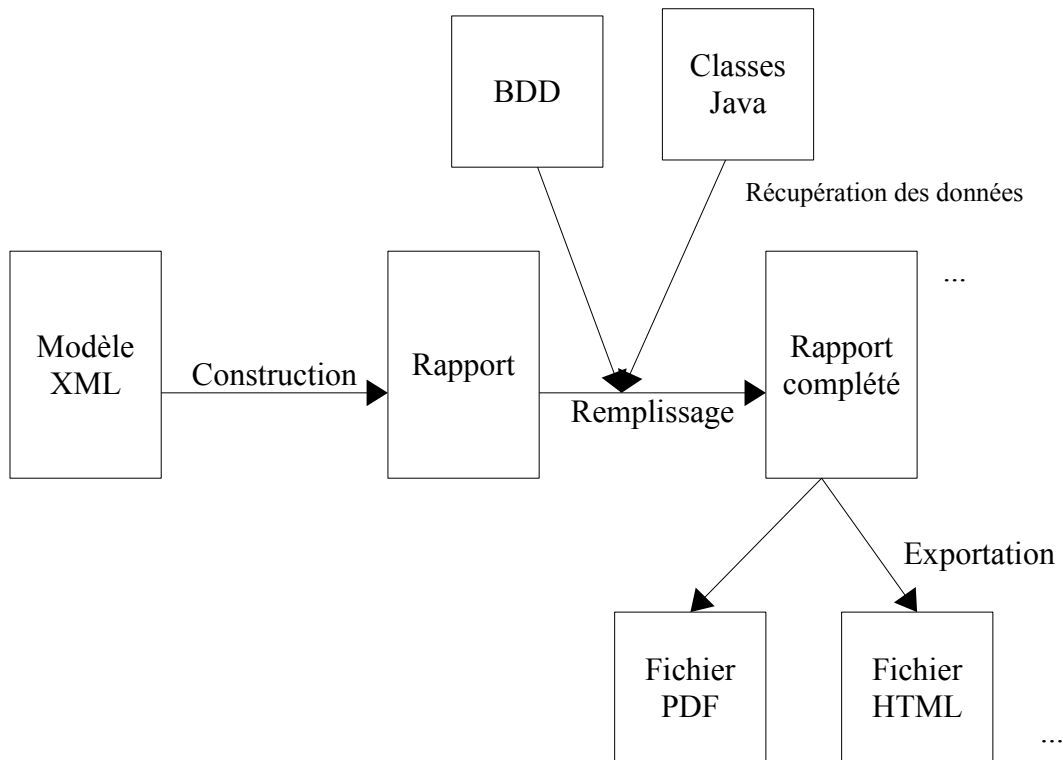
JasperReports est entièrement développé en Java et peut être intégré dans une gamme très variée d'applications Java (y compris les applications J2EE). Son objectif principal est de fournir un moyen simple et flexible pour la génération de documents.

La bibliothèque JasperReports a été conçue en 2001 par *Teodor Danciu*, qui a également participé à de nombreux autres projets Open Source (Hibernate, framework Avalon, ...). Il continue régulièrement à proposer de nouvelles évolutions pour JasperReports. Cependant, il n'imaginait pas un tel succès (plus de 300 000 téléchargements et 11 000 nouveaux téléchargements par mois). C'est pour cette raison qu'une nouvelle compagnie a vu le jour il y a peu de temps, **JasperSoft** (<http://www.jaspersoft.com>), qui a été formée afin d'investir dans le développement de JasperReports et afin d'offrir un support, des services et des produits commerciaux destinés à compléter JasperReports. JasperSoft est composée d'un certain nombre de développeurs qui vont pouvoir travailler à plein temps sur JasperReports, mais également sur un outil payant, **JasperDecisions**, qui va proposer des fonctionnalités supplémentaires à JasperReports. JasperDecisions est composé de deux produits principaux, *Scope Server* (solution de reporting) et *Scope Designer* (outil graphique destiné à produire les rapports).

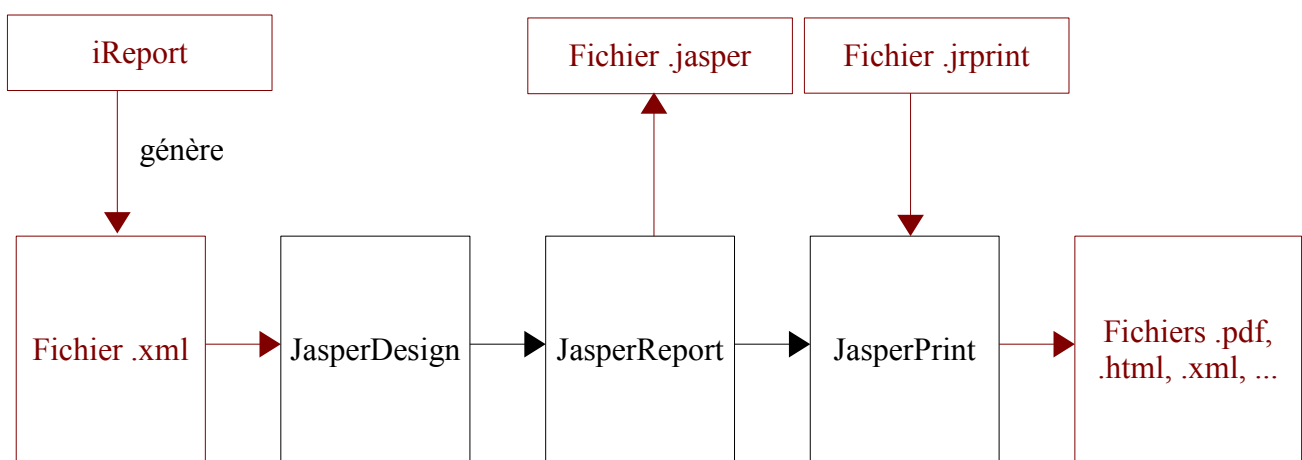
C'est donc une nouvelle étape importante dans le développement de JasperReports qui pourra ainsi prétendre à devenir l'une des meilleures solutions de reporting du marché. En outre, Teodor Danciu a récemment participé à deux conférences importantes, « MySQL Users Conference 2005 » (http://conferences.oreillynet.com/cs/mysqluc2005/view/e_sess/7076) et « JBoss World 2005 » (http://www.jboss.com/company/events/jbw_bestpractices), ceci afin de présenter son outil à un plus large public et afin de faire une démonstration des capacités d'intégration de JasperReports dans un environnement utilisant **JBoss** et **Hibernate**.

La création de rapports avec JasperReports se déroule généralement en 4 étapes :

- l'obtention d'un fichier modèle XML (à l'aide d'éditeurs graphiques comme iReport ou OpenReports Designer)
- la construction du rapport à partir du modèle
- le remplissage des différents champs du rapport avec les données en provenance de diverses sources (bases de données, classes Java, ...)
- l'exportation du résultat dans plusieurs formats possibles (PDF, HTML, ...)



Le fonctionnement de JasperReports est relativement simple. En effet, tous les concepts tournent autour du langage Java. Une fois le modèle XML (JasperDesign) compilé, il est chargé dans un objet Java (JasperReport) qui peut lui-même être sérialisé et stocké dans un fichier (avec l'extension `.jasper`). Cet objet sérialisé est alors utilisé lorsque l'application désire compléter le rapport avec des données. En fait, la définition du rapport nécessite la compilation de toutes les expressions Java déclarées dans le modèle XML. Le résultat obtenu après le processus de remplissage des champs est un nouvel objet Java (JasperPrint) qui représente le document final. Celui-ci peut être stocké sur disque pour un usage ultérieur (sous forme sérialisée et avec l'extension `.jrprint`), directement imprimé ou encore transformé dans un format lisible (PDF, HTML, ...).



Durant cette étude, certaines possibilités offertes par JasperReports ne seront pas abordées. Il faut savoir que JasperReports dispose d'atouts supplémentaires :

- associer plusieurs rapports afin d'obtenir un unique rapport à exporter
- utiliser des liens dans le rapport ou en direction d'Internet
- élaborer des graphiques complexes à l'intérieur d'un rapport (avec les bibliothèques **jCharts** pour la 2D et **jFreeChart** pour la 3D)
- protéger par mot de passe les fichiers au format PDF
- transmettre des sous-rapports à un rapport principal au moyen de l'élément *paramètre*
- employer un fichier XML comme source de données
- utiliser des *scriptlets* qui apportent des fonctionnalités nouvelles par rapport aux variables (manipulation des données pendant le remplissage des champs)

2.2 OBTENTION DU MODÈLE XML

Afin d'utiliser la librairie JasperReports, il est nécessaire de créer un fichier modèle XML. Celui-ci doit avoir une structure bien particulière. Cette structure est déclarée dans un fichier au format DTD (Document Type Definition) présent sur le site Internet de JasperReports.

Tous les fichiers XML doivent être définis de la façon suivante :

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">

<jasperReport name="nom_du_rapport" ...>
...
</jasperReport>
```

Les éléments principaux qu'il faut utiliser pour générer le modèle XML sont les suivants :

- les paramètres :

Les paramètres sont des références à des objets qui sont passés au rapport au moyen de méthodes de remplissage écrites en Java. Déclarer un paramètre est très simple et requiert uniquement un nom et un type :

```
<parameter name="Titre" class="java.lang.String"></parameter>
```

Les paramètres sont très utiles afin de transmettre des données qui n'apparaissent pas dans les sources de données. Par exemple, nous pouvons passer au rapport le nom de l'utilisateur qui l'a généré ou bien modifier dynamiquement le titre du rapport.

- les champs :

Les champs représentent la seule façon de récupérer les données à partir d'une source de données et de les transférer ensuite aux différentes routines gérant le rapport. Ces champs doivent avoir obligatoirement le même nom et le même type que les objets correspondants qui résultent de la requête réalisée sur la source de données.

Par exemple, nous désirons utiliser une table *Employes* qui possède la structure suivante :

Nom	Type	Longueur (octets)
Numero	integer	4
Nom	varchar	20
Prenom	varchar	10
Salaire	double	8
DateEmbauche	date	4

Nous pouvons alors déclarer les champs :

```
<field name="Numero" class="java.lang.Integer"></field>
<field name="Nom" class="java.lang.String"></field>
<field name="Prenom" class="java.lang.String"></field>
<field name="Salaire" class="java.lang.Double"></field>
<field name="DateEmbauche" class="java.util.Date"></field>
```

- les expressions :

Les expressions sont un dispositif très important de la librairie JasperReports. En effet, elles peuvent être employées pour déclarer des variables qui exécuteront divers calculs, pour spécifier le contenu des champs du rapport ou pour personnaliser l'apparence des différents objets constituant le rapport. Fondamentalement, toutes les expressions sont des expressions Java qui référencent des champs ou des variables du rapport.

Il existe plusieurs manières pour définir des expressions :

```
<variableExpression>, <initialValueExpression>, <groupExpression>,
<printWhenExpression>, <imageExpression>, <textFieldExpression>
```

Afin d'utiliser des champs dans une expression, le nom du champ doit être placé entre les deux séquences de caractères suivantes : **`$F{`** et **`}`**

Par exemple, nous désirons afficher sur le rapport la concaténation du nom et du prénom des employés :

```
<textFieldExpression>
  ${Nom} + " " + ${Prenom}
</textFieldExpression>
```

La syntaxe est la même pour les variables et les paramètres sauf qu'il faut les placer entre des séquences de caractères différentes : respectivement **\$V{** et **}**, **\$P{** et **}**

- les variables :

Les variables sont des objets spéciaux qui sont destinés à être utilisés exclusivement au sein des expressions. Elles peuvent être employées pour simplifier la conception du rapport en déclarant seulement une fois une expression qui sera utilisée à plusieurs reprises tout au long du modèle XML. Elles servent également à réaliser certains calculs dans les expressions où elles sont employées.

Dans une même expression, une variable peut référencer d'autres variables, mais seulement si celles-ci sont définies auparavant. L'ordre de déclaration des variables est donc primordial.

Comme cité précédemment, les variables peuvent exécuter plusieurs types de calculs différents : **count**, **sum**, **average**, **lowest**, **highest**, **variance**, ...

- Par exemple, nous désirons afficher sur le rapport la somme totale des salaires des employés :

```
<variable name="Somme" class="java.lang.Double" calculation="Sum">
  <variableExpression>${Salaire}</variableExpression>
</variable>
```

Il est possible également de spécifier le niveau d'initialisation des variables. Le niveau par défaut est *Report*, ce qui signifie que les variables sont initialisées une seule fois au début du rapport. Il existe d'autres niveaux d'initialisation (*Page*, *Column*, *Group*) qui permettent de réaliser les calculs pour chaque page, chaque colonne ou chaque groupe.

- les groupes :

Les groupes représentent une façon simple d'organiser les données à l'intérieur d'un rapport. Lorsqu'il remplit un rapport, le moteur de JasperReports teste toutes les définitions de groupes pour s'assurer qu'il n'y a pas de rupture de groupe et pour voir si les sections suivantes sont bien présentes pour chaque groupe :

```
<groupHeader>, <groupFooter>
```

Comme pour les variables, l'ordre de déclaration des groupes est important. En effet, un groupe contient toujours le groupe suivant et ainsi de suite.

Afin de déclarer un groupe, il faut procéder de la manière suivante :

```
<group name="Employes">
  <groupExpression>...</groupExpression>
  <groupHeader>
  ...
</groupHeader>
<groupFooter>
  ...
</groupFooter>
</group>
```

- les requêtes :

Lorsque la source de données est une base de données, il est possible d'effectuer une requête directement dans le fichier modèle XML.

Par exemple, nous désirons récupérer le nom et le prénom de tous les employés :

```
<queryString>
  SELECT Nom, Prenom FROM Employes
</queryString>
```

Quand on construit un modèle XML, il est nécessaire de respecter la disposition des différentes sections. La structure du fichier est divisée en 9 parties :

- background
- title
- pageHeader
- columnHeader
- detail
- columnHeader
- pageFooter
- lastPageFooter
- summary

Chaque section est une partie du rapport bien particulière qui possède une largeur et une hauteur spécifiques et qui peut contenir des objets comme des lignes, des rectangles, des images ou du texte. Pour spécifier le contenu et la disposition d'une section du rapport, il est indispensable d'utiliser l'élément générique suivant :

```
<band>
```

Les bandes peuvent inclure de multiples éléments, chacun d'eux étant identifié par une position, une taille et une valeur :

```
<staticText>, <textField>, <line>, <rectangle>, <image>
```

Par exemple, nous désirons afficher, en bas de chaque page du rapport, le numéro de la page correspondante :

```
<pageFooter>
  <band height="40">
    <line>
      <reportElement x="0" y="10" width="500" height="1"/>
      <graphicElement/>
    </line>
    <staticText>
      <reportElement x="0" y="20" width="50" height="15"/>
      <textElement textAlignment="Right">
        <font fontName="Arial" size="14" isItalic="true"/>
      </textElement>
      <text>Page : </text>
    </staticText>
    <textField>
      <reportElement x="50" y="20" width="30" height="15"/>
      <textElement>
        <font fontName="Arial" size="14"/>
      </textElement>
      <textFieldExpression class="java.lang.Integer">
        ${PAGE_NUMBER}
      </textFieldExpression>
    </textField>
  </band>
</pageFooter>
```

Afin de faciliter la conception du modèle XML, il existe quelques éditeurs graphiques qui permettent de générer des fichiers XML en utilisant la même syntaxe que celle proposée par JasperReports.

2.3 CONSTRUCTION DU RAPPORT

Avant d'employer JasperReports, il est nécessaire de détailler les différents objets Java qui entrent en jeu dans le processus de génération d'un rapport, de la conception de celui-ci à la production d'états :

- JasperDesign :

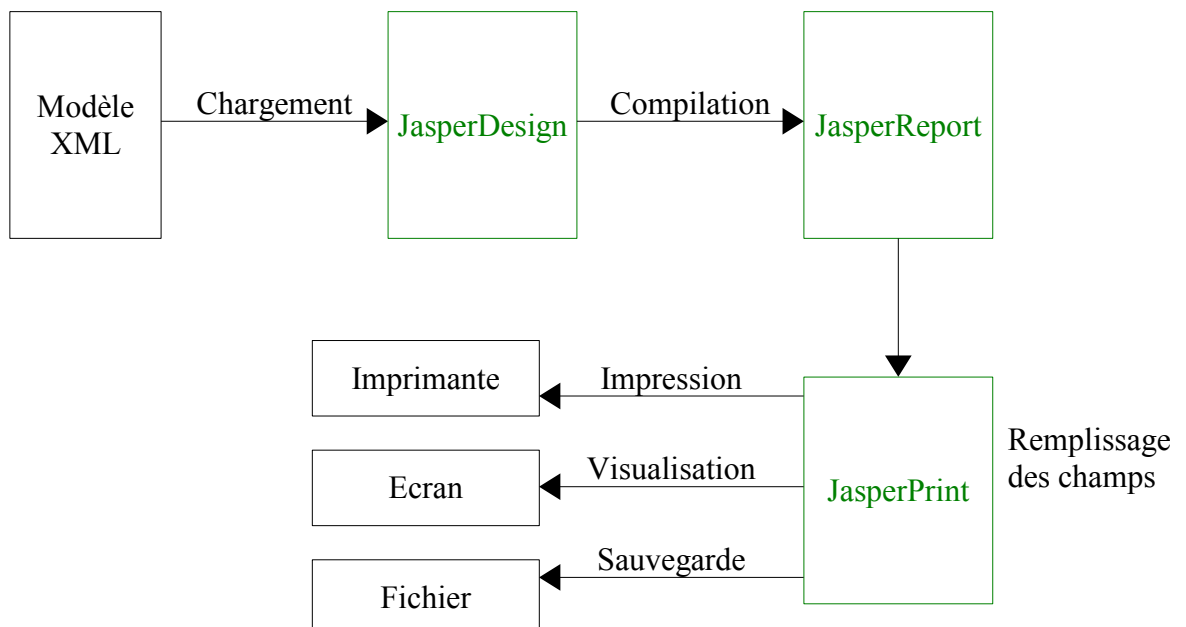
Cet objet représente la définition d'un rapport. Dans la plupart des cas, nous devons créer un objet JasperDesign à partir d'un fichier modèle XML bien qu'il soit également possible de générer ce modèle au moyen de code Java.

- JasperReport :

Cet objet représente un objet JasperDesign compilé. Le processus de compilation vérifie la structure du modèle XML, le compile et le stocke dans un objet JasperReport.

- JasperPrint :

Cet objet représente le rapport final. Un objet JasperPrint est élaboré à partir d'un objet JasperReport par un processus de remplissage qui consiste à insérer dans le rapport des données en provenance d'une source de données quelconque.



Afin de pouvoir employer correctement ces différents objets, nous devons utiliser un certain nombre de bibliothèques (.jar) indispensables au bon fonctionnement du programme Java :

- **jasperreports-1.0.2.jar** : cette bibliothèque possède toutes les fonctionnalités propres à JasperReports.
- **commons-digester.jar** : cette bibliothèque permet de transformer un fichier XML en objets Java.
- **commons-collections-3.1.jar** : cette bibliothèque introduit de nouvelles classes, de nouvelles interfaces et apporte des fonctionnalités supplémentaires aux classes de base (en particulier à l'interface *Map*).
- **commons-beanutils.jar** : cette bibliothèque apporte une aide à la génération d'objets Java.
- **commons-logging.jar** : cette bibliothèque fournit, de manière transparente, des fonctionnalités de log en utilisant n'importe lequel des frameworks de log existants.
- **jdt-compiler.jar** : cette bibliothèque est obligatoire pour développer avec Eclipse.
- **itext-1.3.jar** : cette bibliothèque permet de générer des fichiers PDF à la volée.
- **hsqldb.jar, ojdbc14.jar, ...** : ces bibliothèques permettent de communiquer avec la base de données.

Toutes les librairies **commons** sont issues du projet *Jakarta* d'Apache.

Nous allons maintenant détailler la phase de construction du rapport. Avant de remplir les différents champs du rapport, il est nécessaire de compiler le fichier modèle XML. Cette compilation se réalise de la façon suivante :

```
JasperDesign jasperDesign = JRXmlLoader.load("nom_modele_XML.xml");
JasperReport jasperReport = JasperCompileManager.compileReport(jasperDesign);
```

Dans un premier temps, il faut charger le modèle XML et le stocker dans un objet de type *JasperDesign*. Puis, celui-ci est compilé au moyen de la méthode *compileReport* de la classe *JasperCompileManager*. Cela permet de vérifier si la syntaxe du fichier XML est bien respectée.

2.4 REMPLISSAGE DES CHAMPS

En premier lieu, nous allons déjà expliquer comment sont créés les paramètres qui seront ensuite introduits dans les champs correspondants du rapport final. Pour cela, il est nécessaire de déclarer un objet de type *Map* dans lequel nous allons insérer les valeurs des paramètres :

```
Map parameters = new HashMap();
parameters.put("Titre", "Listing des employés");
```

Nous étudierons plus tard la façon de transmettre ces paramètres au rapport.

Un des nombreux avantages de JasperReports est la possibilité d'avoir recours à une grande quantité de sources de données différentes, ceci afin de remplir les champs du rapport :

- les bases de données :

JasperReports est capable de récupérer des données dans plusieurs bases de données (Oracle, hsqldb, ...). Dans tous les cas, le fonctionnement est toujours le même. En effet, il faut, au préalable, déclarer un élément *requête* dans le fichier modèle XML (comme expliqué dans la partie traitant du modèle XML). Ensuite, il est nécessaire, dans le code Java, de créer une nouvelle classe prenant en charge la connexion à la base de données.

Par exemple, nous désirons réaliser une connexion à une base de données *hsqldb* :

```
Class.forName("org.hsqldb.jdbcDriver");
Connection connection =
DriverManager.getConnection("jdbc:hsqldb:test", "login", "pwd");
```

Une fois l'objet Connection créé, il ne nous reste plus qu'à le passer à la méthode *fillReport* de la classe JasperFillManager de la manière suivante :

```
JasperPrint jasperPrint =
JasperFillManager.fillReport(jasperReport, parameters, connection);
```

A partir de ce moment, JasperReports se charge de remplir tous les champs du rapport (aussi bien les champs provenant de la requête que les paramètres). La méthode *fillReport* retourne un objet JasperPrint qui contient le rapport final. Le rapport est alors prêt à être imprimé ou exporté.

- les sources de données :

Dans ce cas précis, nous ne sommes pas contraints de déclarer un élément *requête* dans le fichier modèle XML. Par contre, il faut créer une nouvelle classe qui implémente l'interface JRDataSource. Cette classe devra obligatoirement redéfinir les deux méthodes suivantes :

```
public boolean next() throws JRException;
public Object getFieldValue(JRField field) throws JRException;
```

Il est nécessaire aussi de déclarer une variable, dans laquelle nous allons stocker les valeurs des différents champs (ex : un tableau à plusieurs dimensions contenant des objets de type Object, un objet de type List, ...), et un index, qui permettra de parcourir cette variable.

Voici un exemple de déclaration d'un tableau :

```
private Object [][] data = {
    {new Integer(1),
     "DUPONT", "Jean",
     new Double(1512.38),
     new Date()},
    ...
};
private int index = -1;
```

Afin que le processus de remplissage des champs du rapport se déroule correctement, il faut mettre en place, dans le fichier modèle XML, des éléments *champ*. Ceux-ci permettront de récupérer les valeurs envoyées par JasperReports.

La méthode *next* est seulement utilisée dans le but d'incrémenter l'index et de tester si celui-ci n'a pas dépassé la longueur du tableau. En ce qui concerne la méthode *getFieldValue*, son rôle est plus fondamental. En effet, elle permet de retourner la valeur qui correspond à la fois au champ passé en paramètre (objet de type JRField) et à l'index.

Par exemple, nous désirons obtenir le nom d'un employé :

```
Object value = null;
if (field.getName().equals("Nom"))
    value = data[index][1];
```

Pour pouvoir remplir les champs du rapport, il nous faut également faire appel à la méthode *fillReport* :

```
JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport,
    parameters, new Datasource());
```

Le dernier paramètre de la méthode est une nouvelle instance de la classe que nous avons préalablement créée (qui référence les méthodes *next* et *getFieldValue*). Ainsi, à chaque élément *band* du modèle XML, JasperReports appelle automatiquement la méthode *next* et, à chaque élément *textFieldExpression* de ce même modèle, il remplit le champ correspondant par la valeur retournée par la méthode *getFieldValue*.

- les tables :

Cette manière de procéder est à peu près identique à la précédente. Elle utilise des principes équivalents pour compléter le rapport. Il est nécessaire de déclarer une nouvelle classe qui, cette fois-ci, n'implémente plus l'interface JRDataSource mais hérite de la classe AbstractTableModel. Il faut aussi définir un certain nombre de fonctions permettant d'accéder aux différentes valeurs des champs (stockées également dans un tableau ou une liste).

L'appel à la méthode *fillReport* se fait alors de la façon suivante :

```
JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport,
    parameters, new JRTableModelDataSource(new Table()));
```

- les autres sources :

JasperReports est capable de récupérer des données de sources diverses :

- **Bean Array** (JRBeanArrayDataSource)
- **Bean Collection** (JRBeanCollectionDataSource)
- ...

Nous ne détaillerons pas ces autres méthodes car elles sont plus compliquées à mettre en place et n'apportent pas de réelles nouveautés (sauf peut-être la possibilité de créer directement un tableau contenant des classes que nous avons conçues).

2.5 EXPORTATION DU RÉSULTAT

Dés que le rapport est élaboré, nous pouvons alors l'imprimer, le visualiser (au moyen de JasperViewer) ou encore le sauvegarder en différents formats :

- Impression :

Pour imprimer directement le rapport, il suffit de faire appel à la méthode *printReport* de la classe JasperPrintManager. Il est également possible de paramétrer certaines options propres à l'imprimante (format de l'impression, ...).

- Visualisation :

Afin de visualiser le rapport après l'avoir généré, il suffit d'employer l'utilitaire JasperViewer fourni avec JasperReports. Pour cela, il faut utiliser la méthode *viewReport* de la classe JasperViewer de la manière suivante :

```
JasperViewer.viewReport(jasperPrint);
```

Nous pouvons aussi enregistrer le rapport et le lancer ensuite dans un navigateur Web.

- Sauvegarde :
JasperReports offre la possibilité de sauvegarder le rapport en plusieurs formats (PDF, HTML, XLS, CSV, XML). Nous allons en citer quelques uns :

Pour le format PDF :

```
JasperExportManager.  
exportReportToPdfFile(jasperPrint, "nom_rapport.pdf");
```

Pour le format HTML :

```
JasperExportManager.  
exportReportToHtmlFile(jasperPrint, "nom_rapport.html");
```

2.6 GÉNÉRATION D'UN RAPPORT SANS MODÈLE XML

Comme mentionné déjà précédemment, JasperReports autorise la génération d'un rapport sans disposer d'un fichier modèle XML. En effet, il est possible de construire le rapport directement à partir d'objets Java. Cela constitue un atout non négligeable si on ne dispose pas d'un éditeur graphique pour concevoir le modèle XML ou si on désire utiliser seulement le langage Java.

Les objets Java ont le même rôle que les éléments XML que nous avons détaillés préalablement. Il existe un objet Java pour chaque élément du modèle XML (ces objets Java sont ajoutés à l'objet JasperDesign) :

- **JRDesignParameter** pour les paramètres (*addParameter* pour ajouter à JasperDesign)
- **JRDesignField** pour les champs (*addField* pour ajouter à JasperDesign)
- **JRDesignExpression** pour les expressions (différent suivant les cas)
- **JRDesignVariable** pour les variables (*addVariable* pour ajouter à JasperDesign)
- **JRDesignGroup** pour les groupes (*addGroup* pour ajouter à JasperDesign)
- **JRDesignQuery** pour les requêtes (*setQuery* pour ajouter à JasperDesign)
- **JRDesignReportFont** pour les polices (*addFont* pour ajouter à JasperDesign)
- **JRDesignBand** pour les bandes

Les bandes peuvent contenir plusieurs autres objets (ajoutés au moyen de la méthode *addElement*) :

- **JRDesignStaticText** pour l'élément *staticText*
- **JRDesignTextField** pour l'élément *textField*
- **JRDesignLine** pour l'élément *line*
- **JRDesignRectangle** pour l'élément *rectangle*
- **JRDesignImage** pour l'élément *image*

Elles sont ensuite ajoutées aux différentes sections de l'objet JasperDesign à l'aide des méthodes suivantes :

- ***setBackground*** pour la section *background*
- ***setTitle*** pour la section *title*
- ***setPageHeader*** pour la section *pageHeader*
- ***setColumnHeader*** pour la section *columnHeader*
- ***setDetail*** pour la section *detail*
- ***setColumnFooter*** pour la section *columnFooter*
- ***setPageFooter*** pour la section *pageFooter*
- ***setLastPageFooter*** pour la section *lastPageFooter*
- ***setSummary*** pour la section *summary*

2.7 JASPERREPORTS DANS UNE ARCHITECTURE WEB

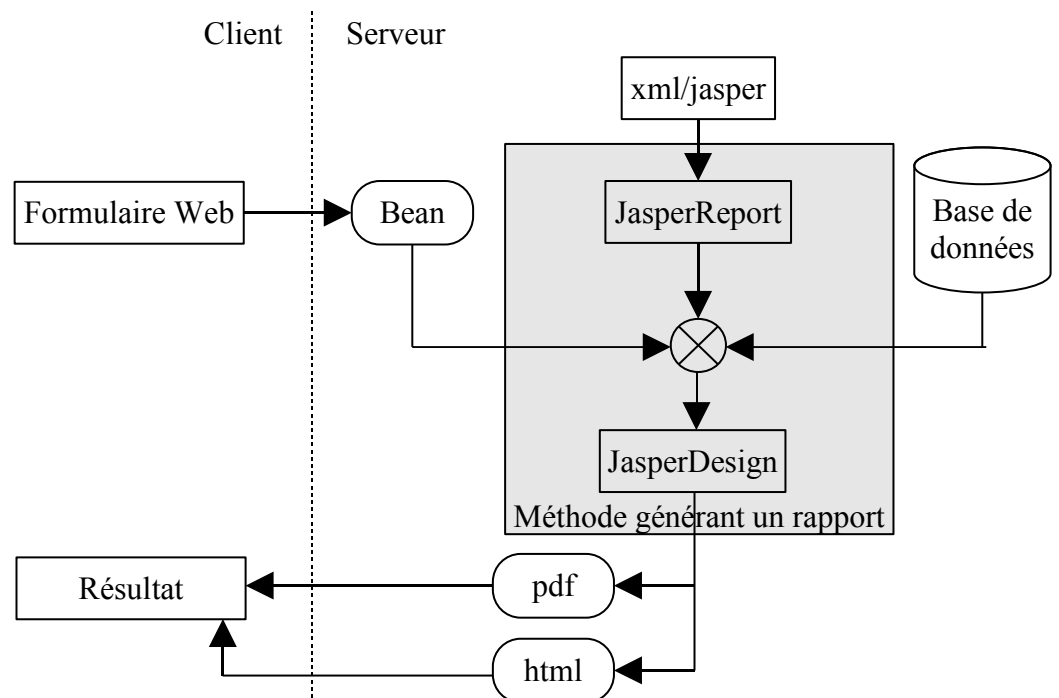
JasperReports peut être utilisé dans un environnement WEB J2EE. Les traitements sont effectués coté serveur et le résultat est renvoyé au client.

Le principe est le même, JasperReports a besoin d'un modèle de rapport, de paramètres et de données :

- le modèle de rapport est stocké sur le serveur sous forme de fichiers xml (JasperDesign) ou sous forme de fichiers jasper (JasperReport)
- les données sont disponibles depuis une base de données
- les paramètres sont récupérés depuis un formulaire web

Exemple de fonctionnement utilisant le Framework JSF [1] :

- l'utilisateur remplit un formulaire web contenant tous les paramètres nécessaires à la génération du rapport
- à la validation du formulaire, le framework JSF se charge de créer un Bean [2] contenant les paramètres renseignés par l'utilisateur, puis exécute une méthode pour générer le rapport. Cette méthode effectue les actions suivantes :
 - charge le rapport (le compile si besoin) pour avoir un objet JasperReport
 - utilise les paramètres enregistrés dans notre Bean et les données issues d'une base de données pour générer un rapport (objet JasperPrint)
 - génère le rapport sous divers formats (pdf, html, etc.), le format du rapport peut être indiqué par le formulaire web
 - le rapport est accessible à l'utilisateur depuis l'application web

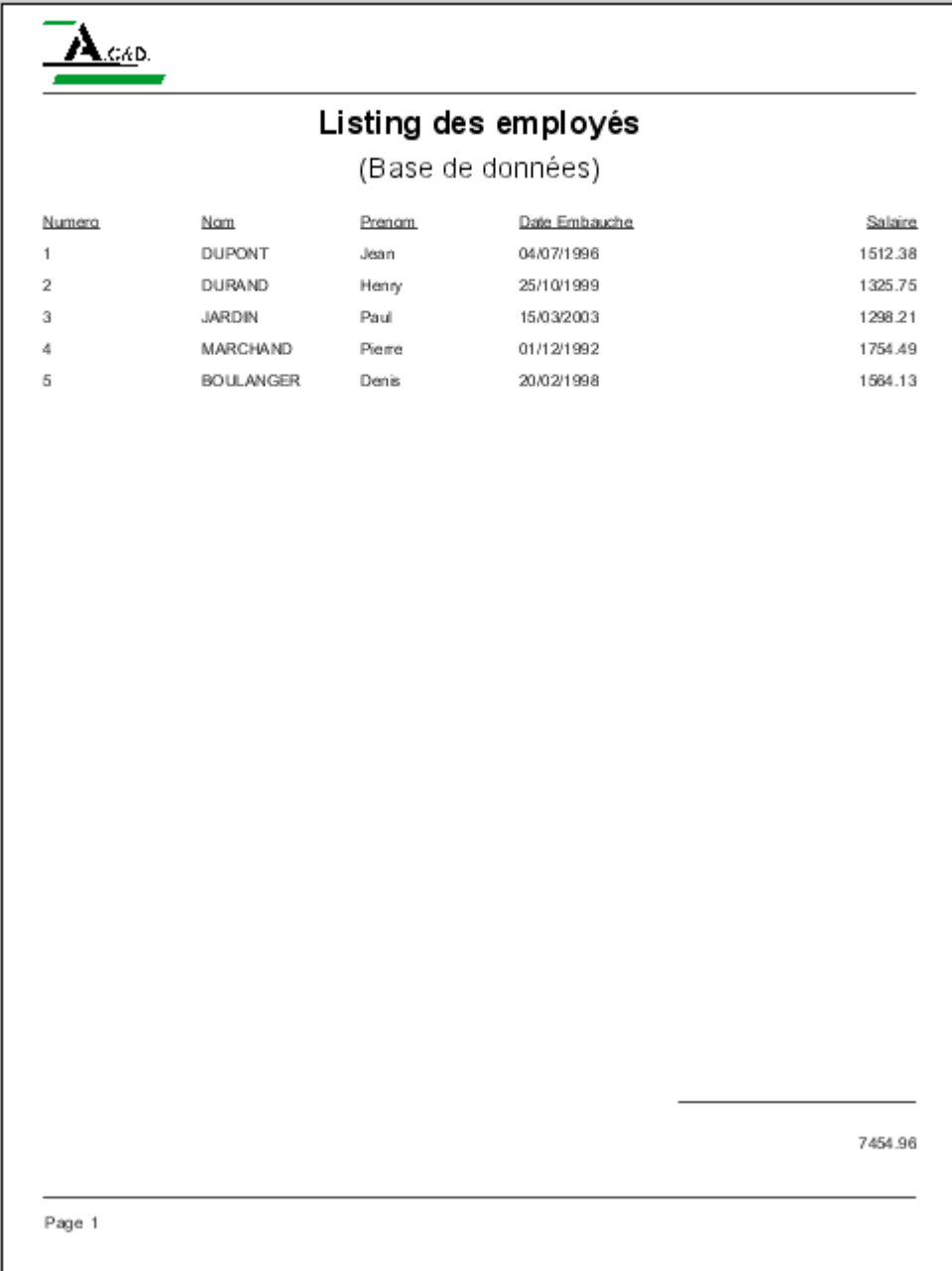


[1] <http://java.sun.com/j2ee/javaserverfaces/>

[2] <http://java.sun.com/products/javabeans/index.jsp>

2.8 EXEMPLES DE RAPPORTS

L'aperçu d'un rapport simplifié est présent à la page suivante. Les données sont récupérées dans une base de données de type *hsqldb* et affichées dans les différents champs du rapport. Le coût total des salaires et le numéro de page sont également inscrits en bas du rapport.



A.C.A.D.

Listing des employés (Base de données)

<u>Numero</u>	<u>Nom</u>	<u>Prenom</u>	<u>Date Embauche</u>	<u>Salaires</u>
1	DUPONT	Jean	04/07/1996	1512.38
2	DURAND	Henry	25/10/1999	1325.75
3	JARDIN	Paul	15/03/2003	1298.21
4	MARCHAND	Pierre	01/12/1992	1754.49
5	BOULANGER	Denis	20/02/1998	1564.13

7454.96

Page 1

Il est également possible de concevoir des rapports beaucoup plus élaborés, comme l'exemple suivant, qui construit un tableau dynamique (dont on ne connaît pas le nombre de colonnes et le nombre de lignes) sur plusieurs pages à partir de données issues d'une base de données *Oracle*.

10/05/05 10:22

Etapes/Traitements	1Aa MCO Hospitalisation	1C Urgences	1D MCO Consultations et actes externes	2A SAMU	2B EHPAD
<i>Affectation primaire</i>					
Affectation primaire	39 563 294,55 €	89 705,45 €	4 033 223,85 €	1 082 493,06 €	
<i>Calcul des charges nettes</i>					
Application règles d'Affectation	38 913 409,41 €	91 058,37 €	3 864 270,07 €	1 072 893,51 €	
Crédit non reconductible					
Régularisation manuelle					
Régularisation effectif				1 029 392,49 €	
<i>Ventilation logistique médicale</i>					
Ventilation logistique médicale					
<i>Ventilation hôtellerie</i>					
Ventilation restauration					
Ventilation blanchisserie					
<i>Ventilation médico technique</i>					
Ventilation Blocs					
Ventilation Laboratoire					
Ventilation Imagerie					
Ventilation Exploration fonctionnelles					
Ventilation Radiothérapie					
Ventilation SMUR Terrestre pour ordre					
Ventilation Prel d'organes pour ordre					
<i>Ventilation gestion générale</i>					
Ventilation gestion générale					
<i>Ventilation structure</i>					
Ventilation structure					

1/6

Afin de créer un tel rapport, il a été nécessaire de diviser le modèle XML en plusieurs sous-rapports (il n'est pas possible d'exécuter plusieurs requêtes dans un même fichier XML) qui prennent en charge différentes parties du rapport principal : un sous-rapport s'occupe de générer les colonnes, un autre sous-rapport s'occupe d'afficher les entêtes des lignes sur chaque page et le rapport principal s'occupe d'appeler les sous-rapports correspondants.

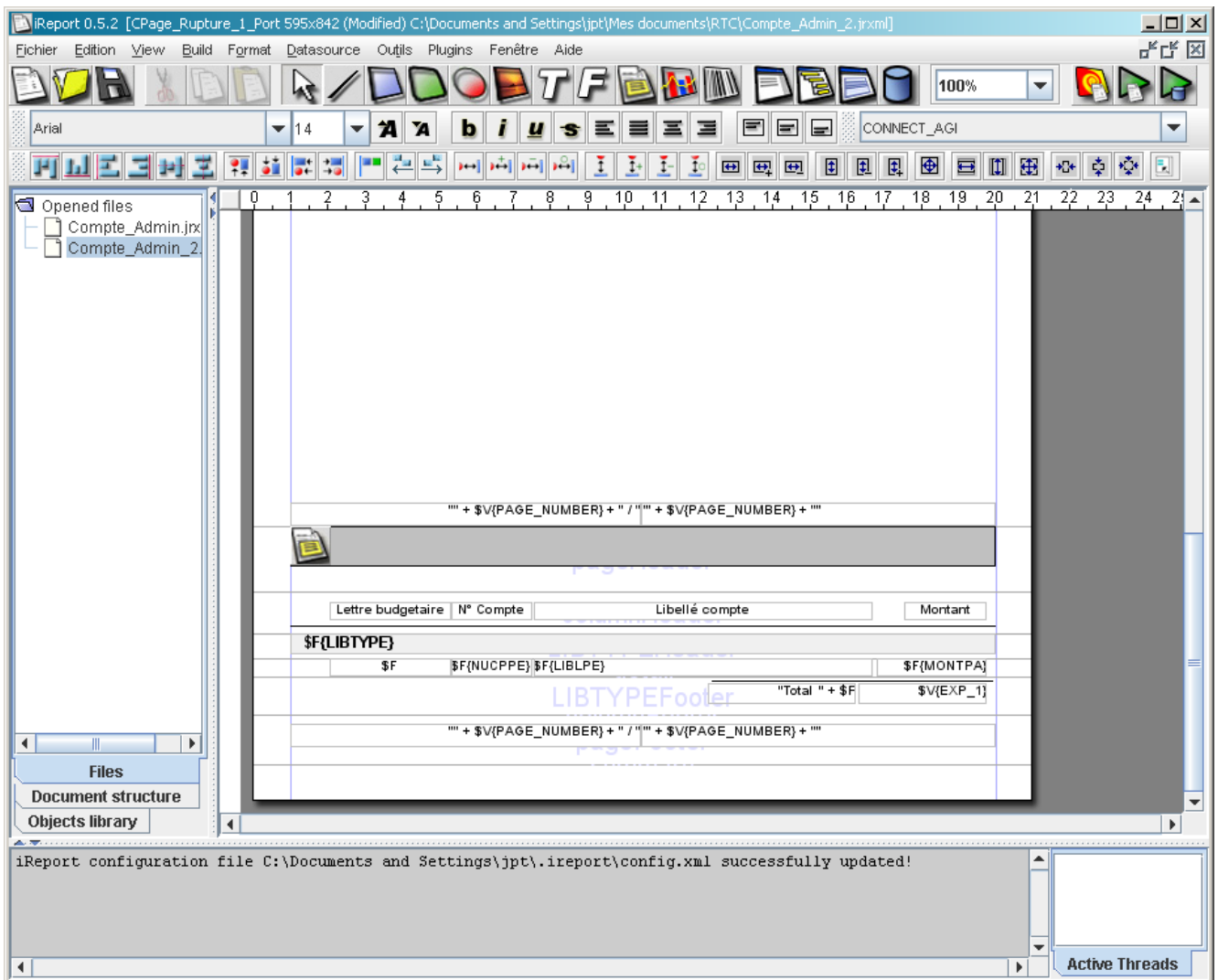
Nous pouvons aussi élaborer des rapports qui imbriquent un nombre important de groupes. C'est le cas de l'exemple suivant (3 groupes) :

SI	Groupe	Compte	Montant
2H - a déterminer			
Groupe 1			
Dépenses			
	61681	PRIMES D'ASSURANCES (MAL. MAT. ET A.T.)	2 945,00 €
	63111	TAXES SUR SALAIRES (PERS.NON MEDICAL)	-66,54 €
	63311	VERSEMENT DE TRANSPORT PERS NON MEDICAL	7 746,67 €
	63320	AIDE AU LOGEMENT PERS.NON.MEDICAL	765,76 €
	63330	COTISATIONS A N F H	9 125,97 €
	63331	COTISATIONS A.N.F.H.- 1%	2 198,00 €
	6336	FONDS EMPLOI HOSPITALIER	7 739,97 €
	641110	PERS TITUL ET STAG -SALAIRE DE BASE	774 617,71 €
	641130	PRIME DE SERVICE	38 938,00 €
	641150	PERS TITUL ET STAG-SUPPLEMENT FAMILIAL	13 781,64 €
	641180	PERS TITUL ET STAG-INDEMNITES	100 014,17 €
	641181	PREST.FAMILIALES	221,85 €
	645110	URSSAF : COT. PIPRESTATIONS FAMILIALES	41 791,35 €
	645111	URSSAF : COT. MALADIE VIEILLESSE	88 937,12 €
	645150	COTISATIONS C N R A C L - ATI	205 242,20 €
	645151	COTISATIONS A T I	3 747,66 €
	64784	OEUVRES SOCIALES	13 651,65 €
	64889	AUTRES CHARGES DE PERSONNEL	1 003,43 €
	672811	CHARGES DE PERSONNEL NON MEDICAL	370,40 €
TOTAL (Groupe 1) :			1 312 772,01 €
Groupe 3			
Dépenses			
	615681	MAINTENANCES AUTRES (SERVICES ECO)	90,12 €
	6161	PRIMES D'ASSURANCES (MULTIRISQUES)	1 204,70 €
	6165	PRIMES D'ASSURANCES (RESPONSABIL.CIVILE)	4 766,00 €
	62510	DEPLACEMENTS DU PERSONNEL NON MEDICAL	3 278,75 €
	6265	TELEPHONE	0,00 €
Recettes			
	758312	REMB. FEH/CFP/CFA	12 058,78 €
	75838810	REMBOURSEMENT FRAIS PAR E.F.S.	1 120 098,37 €
TOTAL (Groupe 3) :			1 141 496,72 €
Groupe 4			
Dépenses			
	6811235	DOT.AMORT. I.G.A.A.C.	270,53 €
	681125412	DOT. AMORT. MAT. ET OUT. MEDICAL	3 143,14 €
	6811284	DOT.AMORT.MOBILIER	291,75 €
TOTAL (Groupe 4) :			3 705,42 €
TOTAL (2H - a déterminer) :			2 457 974,15 €

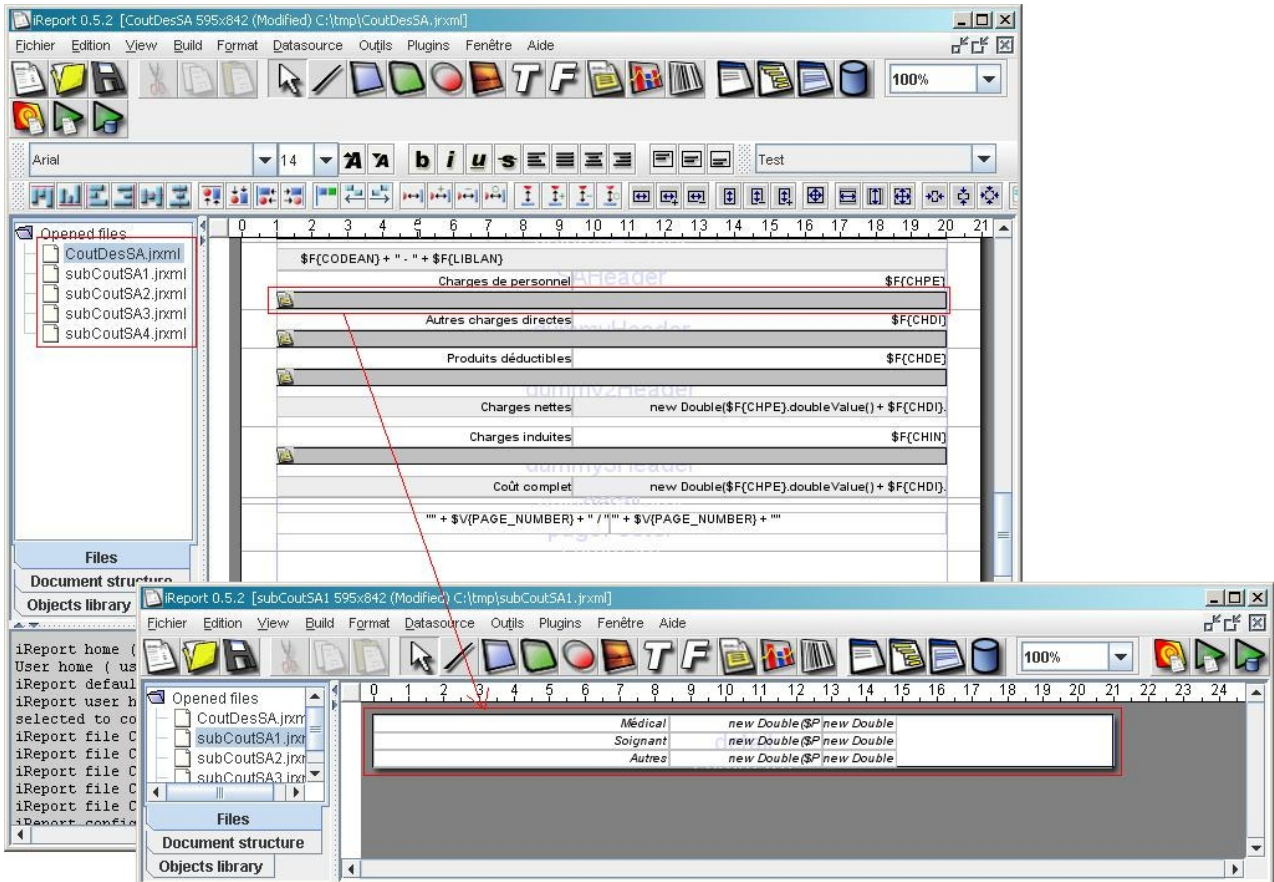
En conclusion, JasperReports offre un panorama de possibilités pour la création de rapports complexes. Il est intégrable dans de nombreuses architectures logicielles telles que les applications lourde (Java Swing), mais également dans les architectures WEB telles que le framework JSF ou les WEB Services.

3 PRÉSENTATION DE iREPORT

En ce qui concerne les outils graphiques permettant de concevoir les modèles, le plus performant est sans aucun doute iReport. C'est un logiciel Open Source entièrement écrit en Java et qui fonctionne à partir de la librairie JasperReports. Il permet de créer tous types de rapports et il inclut la plupart des possibilités offertes par JasperReports. On peut aussi visualiser directement le rendu des rapports sans avoir à programmer la moindre ligne de code Java. En outre, son interface est très intuitive et facile à prendre en main.



Comme on peut l'apercevoir sur la capture d'écran précédente, on retrouve exactement les mêmes concepts que ceux abordés au cours de la section sur la création du modèle XML (notions de bandes, de champs, de variables, ...). iReport permet également de gérer les sous-rapports dans un même projet.



4 BIBLIOGRAPHIE

Dynamic PDF Generation with JasperReports, Struts and a Database par ADC

(<http://www.adcworks.com/blog/index.php/archives/2004/11/16/dynamic-pdf-generation-with-jasperreports-with-struts-and-a-database>)

Using JasperReports with Hibernate

(<http://www.hibernate.org/79.html>)

Generating online reports using JasperReports and WebSphere Studio par Ricardo Olivieri

(http://www-106.ibm.com/developerworks/websphere/library/techarticles/0411_olivieri/0411_olivieri.html)

Reports made easy with JasperReports par Erik Swenson

(<http://www.javaworld.com/javaworld/jw-09-2002/jw-0920-opensourceprofile.html>)

Starting with JasperReports par Gregory Beumer

(<http://technology.amis.nl/blog/index.php?p=346>)