GENERACIÓ DE DOCUMENTS EN PDF A PARTIR D'UNA ESTRUCTURA XML

Lluís Jové Vila
EI

Àlex Alfonso Minguillón

18 de juny de 2004

A l'Eva, per la seva paciència i ajuda,
sinó no hagués estat possible.

RESUM:

La present memòria pretén mostrar que la tecnologia XML és la millor alternativa per afrontar el repte tecnològic existent en els sistemes d'extracció d'informació de les aplicacions de nova generació. Per una banda, han de garantir la seva independència dels esquemes de les bases de dades dels quals s'alimenten, i per l'altra, ser capaços de mostrar la informació en múltiples formats.

Per arribar a aquesta conclusió, hem realitzat un treball centrat en els següents tres punts:

1. Estudi de la tecnologia XML
2. Extracció d'informació amb XML
3. Processos de transformació

En el primer punt, hem realitzat un estudi dels estàndards de la família XML referents a la transformació i presentació de la informació (CSS, XSL, XSLT i XSL-FO) i aquells referents a la creació i manipulació de documents XML (SAX i DOM). Per cadascun d'ells mostrem les principals característiques. També hem fet un estudi dels principals processadors XSLT (Saxon, Xalan i JAXP) i processadors XSL-FO (FOP, XEP i XSLFormatter) que hi ha al mercat. S'analitzen les seves principals característiques en funció de les nostres necessitats.

A continuació, en el segon punt, hem analitzat les diferents tecnologies per generar documents XML a partir de consultes realitzades en una base de dades. Concretament, les APIs de transformació utilitzant SAX o DOM, i la utilitat XDK d'ORACLE.

Un cop hem estudiat les diferents tecnologies i eines que podem utilitzar per al desenvolupament del projecte, en el darrer punt, definim el procés sencer a realitzar per a obtenir el document PDF a partir d'informació extreta d'una base de dades. Aquest procés, el partim en dos fases: extracció i transformació.

La primer part es centra en el disseny i construcció d'un procés d'extracció d'informació genèric i independent de la base de dades, on l'estructura de la informació s'emmagatzema en la pròpia base de dades mitjançant taules que implementen plantilles.

La segona part, transformació, es centra en la construcció d'una plantilla XSLT i la utilització d'un processador XSLT dels que hem estudiat, per obtenir un nou document en format XSL-FO, que a la vegada, i mitjançant un processador XSL-FO, és transformat finalment al document PDF.

Finalment, i com a exemple, es mostra com es crea un nou informe amb la nostra eina sobre una aplicació ja existent.

Índex de continguts

Índex de figures

Índex de taules

1 Introducció

Cada cop més, ens trobem amb aplicacions desenvolupades amb el disseny anomenat de tres capes (interfície, lògica i dades), principalment en tecnologia J2EE. El seu creixement davant el disseny clàssic client/servidor es deu a una sèrie d'avantatges, com ara, la independència de plataforma, l'escalabilitat, la no necessitat d'instal·lació d'un aplicatiu a l'ordinador del client, etc.

Però amb aquestes aplicacions també ha sorgit un nou repte tecnològic en la part de llistats i informes. Cal que aquests, compleixin les mateixes característiques de la resta d'aplicació, en particular la independència de plataforma. Actualment, no serveix la utilització de sistemes d'impressió propietaris o l'extracció d'informació en formats ofimàtics "estàndards" com Microsoft Word o Microsoft Excel, donat que són estàndards per una plataforma en concret però inexistents en altres.

Els nous sistemes d'extracció d'informació han de permetre, per una banda, independitzar-se dels esquemes de les bases de dades dels quals s'alimenten, per tal que no els afectin canvis d'estructura si no comporten canvi d'informació, i per l'altra, ser capaços de mostrar la informació en formats més comuns o la possibilitat de mostrar-la en varis formats a petició de l'usuari, sense que això comporti la necessitat de crear diferents informes. També han de ser capaços de que l'usuari, sense coneixements informàtics, pugui donar-li un format adaptat a les seves necessitats. La resposta a aquestes necessitats la trobem a l'XML.

L'XML (eXtensible Markup Language) és un llenguatge extensible de marques que defineix un format estàndard per a l'estructuració de dades i informació desenvolupat pel W3C (*World Wide Web Consortium*). Una altra característica interessant de XML és que permet l'intercanvi de dades i informació de forma estàndard i totalment independent de les plataformes i aplicacions utilitzades.

Donada la flexibilitat i independència de les fonts de dades que proporciona XML, aquest és, avui en dia, el llenguatge més emprat en el traspàs d'informació entre diferents sistemes. La majoria de cops, aquests intercanvis d'informació es realitzen per poder emmagatzemar la mateixa informació en diferents sistemes d'informació. Altres cops, l'intercanvi és més aviat una extracció de la informació per al seu posterior tractament amb altres eines ofimàtiques o la generació de llistats o informes. Aquest segon punt és el que ens interessa per al nostre projecte.

Amb la implementació d'aquest projecte, utilitzant la tecnologia XML anteriorment comentada, aconseguirem poder definir els informes d'una aplicació amb les següents característiques:

- independència de plataforma
- elecció del format de sortida
- personalització
- estandarització (independència del model de dades).

Això vol dir, que disposarem d'una eina per incloure a qualsevol aplicació, que facilitarà la funcionalitat de generar informes amb aquestes característiques. El fet de canviar d'aplicació o de model de dades no implicarà canvis en l'eina, alhora que permetrà que cada aplicació o fins i tot cada usuari, es pugui personalitzar l'informe a les seves necessitats.

En resum, es tracta d'una eina genèrica i molt flexible, adaptable a qualsevol aplicació, que la dota d'una funcionalitat de generació d'informes molt potent i personalitzable, resolent un dels principals problemes, el d'impressió d'informes, que tenen les aplicacions i que tanta demanda genera per part dels usuaris de les aplicacions.

1.1 Descripció del Projecte

El que es pretén en una primera part d'aquest projecte és, a partir d'un esquema de base de dades relacional definit i ja construït, l'extracció de la informació en XML tenint en compte les necessitats definides per l'usuari.

En una segona fase del projecte es tractarà de generar documents en format PDF; l'usuari podrà definir a la base de dades l'estructura del document final, és a dir, podrà definir plantilles que caldrà combinar amb la informació extreta de la base de dades.

El sistema haurà d'estar dissenyat de manera que sigui independent de la base de dades d'origen (en aquest cas ORACLE) i del format final del document (en aquest cas PDF, però podria ser qualsevol altre, per exemple, en format Word).

Per conseguir-ho, s'utilitzarà el llenguatge de transformació XSLT i el llenguatge de formateig XSL-FO. XSLT i XSL-FO pertanyen a l'estàndard XSL.

XSL és l'estàndard definit per W3C que té per objectiu aplicar format als documents XML. Les fulles d'estil XSL afegeixen a un document XML una descripció de com s'ha de visualitzar el document i inclouen una certa capacitat de procés i transformació del contingut del document.

Mitjançant les fulles d'estil XSL el contingut d'un document font XML pot ser mostrat de diferents formes o adoptar diferents formats en funció del dispositiu de sortida (telèfon mòbil, dispositiu PDA, pàgina web, etc.).

XSL consta de dos parts:

- Un llenguatge de transformació: XSLT, mitjançant el qual un document XML és pot transformar en un altre document XML.

- Un llenguatge de formateig, XSL-FO, un vocabulari XML que defineix un conjunt d'elements anomenats objectes de formatatge i atributs.

En essència, el procediment per a generar un document, per exemple en PDF, a partir d'una informació extreta d'una base de dades en format XML, seria el següent:

1. Aplicar al document XML la plantilla XSLT per tal de generar un nou document en format XML-FO.
2. Transformar el document XML-FO a PDF o a altres formats.

1.2 Objectius

L'objectiu principal d'aquest projecte és la construcció d'una eina genèrica i flexible, que permeti la generació d'informes, en particular en format PDF, a partir d'un model de dades ja existent. Per tal de garantir aquesta fita, s'han definit objectius parcials que s'han d'anar completant al llarg del desenvolupament del projecte. Aquest objectius són els següents:

- Definició de la generació d'informació en XML a partir d'un esquema de bases de dades.
- Definir les transformacions necessàries a realitzar en els documents XML mitjançant XSLT.
- Definir la utilització del llenguatge de formateig XSL-FO.
- Construcció d'un sistema capaç de generar PDF a partir de la informació obtinguda en les plantilles dinàmiques i els fitxers XSL-FO.

En la figura 1.1 podem veure gràficament el procés que hem de realitzar per aconseguir els nostres objectius.

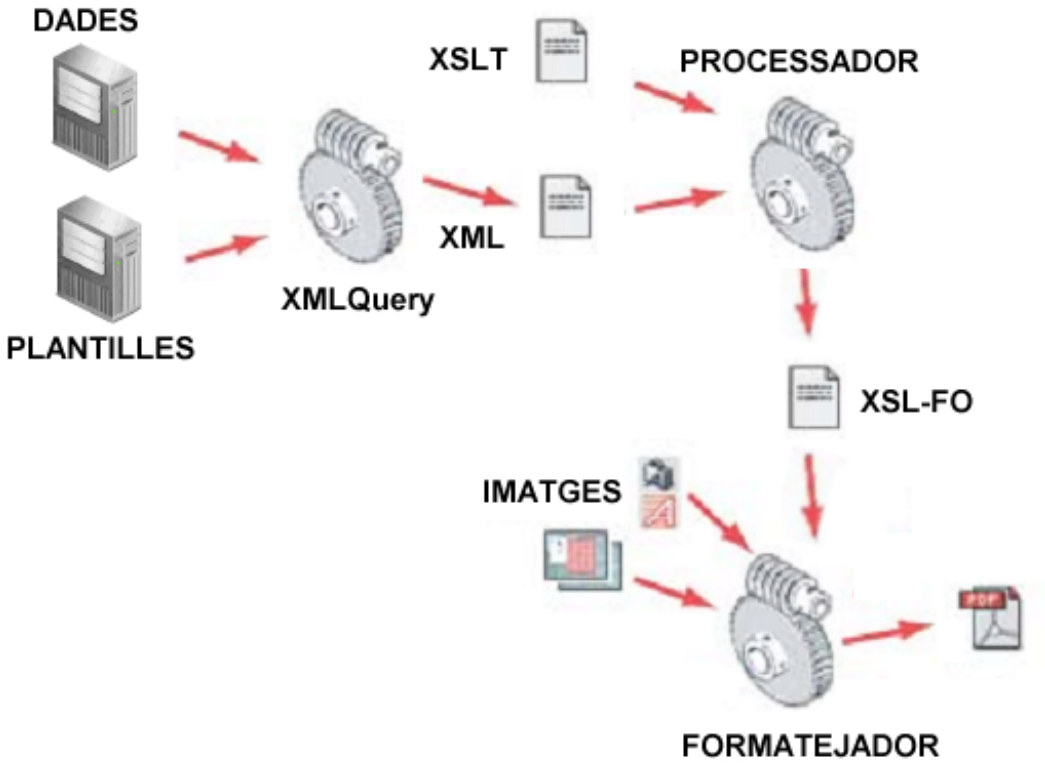

Figura 1.1: Esquema de funcionament del generador de PDFs.

1.3 Organització del projecte

Per a la realització del projecte, i per assolir els diferents objectius marcats, des del començament s'han definit les diferents activitats a realitzar, així com la seva planificació.

1.3.1 Relació d'activitats

Inici del projecte
Correspon a l'inici del projecte que es data a 1 de març de 2004.

Elaboració del pla de treball
Planificació de les diferents activitats, indicant una breu descripció y el cronograma temporal de realització de cada una d'elles. Com a resultat s'obtindrà el Pla de treball.

Durada esperada: 6 dies.

Estudi Tecnologia XML
Confecció d'un estudi dels diferents estàndards dins de la família XML. S'analitzaran les principals avantatges i les funcionalitats que ens poden aportar al nostre projecte, estudiant els estàndards XML, XSL (SXLT, XSL-FO), SAX i DOM i escollint aquella tecnologia que cobreixi millor les necessitats del projecte.

Durada esperada: 4 dies.

Processadors XSLT
S'estudiaran els diferents procesadors XSLT que existeixen al mercat per escollir el que més s'adapti a les nostres necessitats tenint en compte els següents punts:

- Independència de plataforma (preferiblement en Java)
- Lliure distribució / codi obert

Alternatives possibles són: Saxon XSLT Processor, Xalan-Java i JAXP. Com a resultat d'aquesta tasca es determinarà el processador XSLT que farem servir.

Durada esperada: 3 dies.

Processadors XSL-FO
S'estudiaran els diferents procesadors XSL-FO que existeixen al mercat per escollir el que més s'adapti a les nostres necessitats tenint en compte els següents punts:

- Independència de plataforma (preferiblement en Java)
- Lliure distribució / codi obert

Alternatives possibles són: FOP, XEP i XSL Formater. Com a resultat d'aquesta tasca es determinarà el processador XSL-FO que farem servir.

Durada esperada: 3 dies.

Generar BD proves
Creació d'una petita base de dades amb informació suficient per realitzar les proves del processos d'extracció d'XML i transformació a PDF. Contindrà tant el model de dades d'exemple, en el nostre cas d'una biblioteca, com el model de dades que emmagatzema l'estructura de les diferents plantilles.

Durada esperada: 2 dies.

Generació d'XML
Es construiran els processos de generació d'informació en XML a partir de l'esquema de bases de dades creada anteriorment. Correspon a la primera fase del projecte.

Durada esperada: 14 dies.

Creació de plantilles
Es crearan unes plantilles XSLT per als processos de transformació a XSL-FO, que s'encarregaran de definir com es transformaran les dades obtingudes per construir un document XSL-FO.

Durada esperada: 4 dies.

Generació de PDF
Definició dels processos de transformació dels XML amb les plantilles XSLT per tal de generar els XSL-FO. També es realitzaran els processos per recuperar l'estructura de la plantilla emmagatzemada a la BD. Posteriorment aquest document XSL-FO es transformarà a format PDF. Correspon a la segona fase del projecte.

Durada esperada: 20 dies.

Documentació
Al llarg de tot el projecte s'anirà completant la documentació que formarà la memòria a entregar al lliurament del projecte.

Durada esperada: 80 dies.

Lliurament
Correspon a la finalització i lliurament del projecte

1.3.2 Planificació

Totes les activitats a realitzar durant el projecte s'han ordenat en el temps. El calendari resultant és el següent:

	Nombre de tarea	Duración	Comienzo
1	Inici del projecte	0 días	lun 01/03/04
2	Pla de treball	6 días	lun 01/03/04
3	**Estudi tecnologia**	**10 días**	**mar 09/03/04**
4	XML	4 días	mar 09/03/04
5	Processadors XSLT	3 días	lun 15/03/04
6	Processadros XSL-FO	3 días	jue 18/03/04
7	Elecció processadors	0 días	lun 22/03/04
8	Generar BD Proves	2 días	mar 23/03/04
9	Creació Generació d'XML	14 días	jue 25/03/04
10	Generador XML	0 días	mar 13/04/04
11	Creació plantilles XSLT	4 días	mié 14/04/04
12	Creació Generació de PDF	20 días	mar 20/04/04
13	Generador de PDF	0 días	lun 17/05/04
14	Documentació	80 días	lun 01/03/04
15	Liurament	0 días	vie 18/06/04

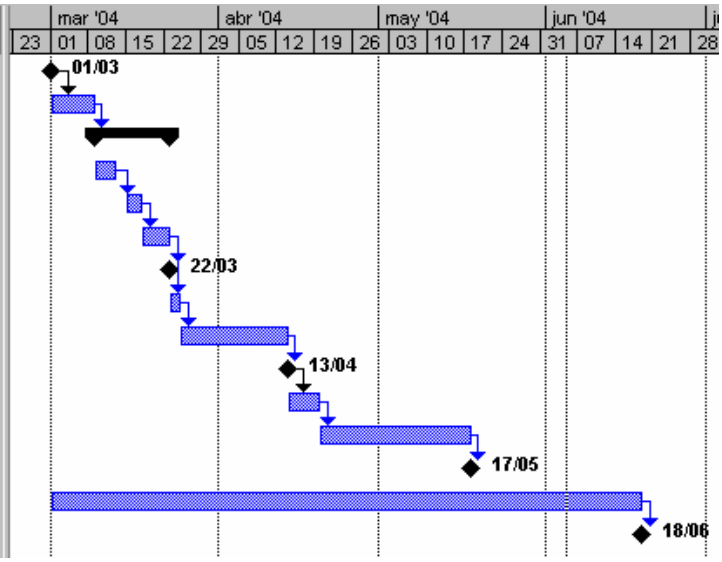

Figura 1.2: Planificació del projecte.

1.3.3 Fites Principals

D'aquesta planificació podem extreure les diferents fites temporals, que ens ajudaran a fer el seguiment y garantir la finalització del projecte. Les fites principals del projecte són:

Fita	Descripció	Data
Inici del projecte	Inici del projecte.	01/03/2004
Elecció processadors	Selecció dels processadors XSLT i XSL-FO que s'utilitzaran en el projecte.	22/03/2004
Generador XML	Finalització de la construcció dels processos d'extracció d'informació de la base de dades a XML.	13/04/2004
Generador PDF	Finalització dels processos de conversió de XML a PDF.	17/05/2004
Lliurament del projecte	Lliurament del projecte finalitzat.	18/06/2004

Taula 1.1: Fites principals del projecte.

1.4 Organització de la memòria

A continuació resumim el contingut de cadascun dels capítols i annexos de què es composa aquesta memòria:

Al capítol 2 presentem, en una primera part, un estudi dels estàndards de la família XML que ens seran de gran utilitat en el desenvolupament del projecte. Concretament aquells estàndards referents a la transformació i presentació de la informació (CSS, XSL, XSLT i XSL-FO) i aquells referents a la creació i manipulació de documents XML (SAX i DOM).

En la segona part del capítol mostrem un estudi realitzat dels principals processadors XSLT (Saxon, Xalan i JAXP) i processadors XSL-FO (FOP, XEP i XSLFormatter) que hi ha al mercat. S'analitzen les seves principals característiques, en funció de les nostres necessitats.

Al capítol 3 realitzem un estudi de les diferents tecnologies per generar documents XML a partir de consultes realitzades en una base de dades. Concretament, s'analitzen les APIs de transformació utilitzant SAX o DOM, i la utilitat XDK d'ORACLE.

Al capítol 4 mostrem com es realitza el procés de transformació de la informació des de que s'extreu de la base de dades fins que s'obté el document PDF corresponent.

El procés, el podem partir en dos fases: extracció i transformació. Per cadascuna d'elles es detallen el passos seguits per la seva implementació, així com la definició dels elements necessaris, com és el cas d'un model de dades per emmagatzemar l'estructura de les plantilles, una plantilla XSLT per realitzar la trasnformació o l'elecció dels processadors utilitzats.

Al capítol 5 presentem un cas pràctic. Un cop tenim desenvolupada la nostra eina, veiem pas a pas, que cal fer per implantar-la en una aplicació, fent servir com a exemple, la creació d'un informe que mostra les dades d'un llibre.

Al capítol 6 descrivim les conclusions que podem extreure del treball així com definir les futures línies de treball.

A l'annex 1 mostrem el codi font de l'eina que hem construït per a generar informes PDF a partir de l'extracció d'informació d'una base de dades en PDF.

A l'annex 2 mostrem el codi de la plantilla XSLT contruida per al procés de transformació del document XML a document XSL-FO.

2 Estudi de la Tecnologia XML

L'XML (eXtensible Markup Language) és un llenguatge extensible de marques que defineix un format estàndard per a l'estructuració de dades i informació desenvolupat pel W3C (*World Wide Web Consortium*). Una altra característica interessant de XML és que permet l'intercanvi de dades i informació de forma estàndard i totalment independent de les plataformes i aplicacions utilitzades.

Donat que no és objectiu principal d'aquest projecte l'estudi d'aquest llenguatge, donarem per assumits els principals conceptes fonamentals d'aquesta tecnologia, com pot ser la seva estructura o la verificació amb els DTD o amb els esquemes XML.

En aquest estudi ens centrarem en analitzar aquells estàndards de la família XML que ens seran de gran utilitat en el desenvolupament del projecte, concretament aquells estàndards referents a la transformació i presentació de la informació i aquells referents a la creació i manipulació de documents XML.

En la segona part del capítol mostrem un estudi realitzat dels principals processadors XSLT (Saxon, Xalan i JAXP) i processadors XSL-FO (FOP, XEP i XSLFormatter) que hi ha al mercat.

2.1 Tecnologies de transformació i presentació

Un document XML conté un conjunt d'informació, dades i estructura, però no defineix les propietats de presentació i visualització de les dades que conté. Precisament, això és l'objectiu principal que es busca amb l'XML, independitzar la part de visualització del contingut, és a dir, que es pugui modificar la forma de visualitzar les dades del document XML sense modificar la forma d'obtenir-les, o viceversa, obtenir de forma diferent les dades sense modificar la forma de mostrar-les. Però, per altra banda, ens caldrà algun altre mecanisme per poder mostrar la informació.

El principal mètode per definir la forma de visualització d'un document XML, és la utilització de fulles d'estil. Les fulles d'estil contenen la forma en que s'han de mostrar els elements d'un document XML. Existeixen principalment dos tipus de fulles d'estil per poder definir el format dels documents XML:

- CCS. Fulles d'estil en cascada.
- XSL. El llenguatge d'estil extensible.

2.1.1 CSS

Les fulles d'estil CSS són un estàndard desenvolupat pel W3C amb l'objectiu de proporcionar un mecanisme per modificar la visualització de les pàgines web, sense necessitat d'haver d'afegir a l'HTML noves etiquetes de format. Posteriorment, l'estàndard CSS també ha estat adaptat com a solució inicial per tal d'especificar la presentació dels documents XML.

Un fulla d'estil CSS consisteix en un conjunt de regles d'estil que s'apliquen als diferents elements del document HTML o XML que es vol visualitzar. Per cada element s'especifica una llista amb els diferents estils que es volen aplicar sobre l'element. Els estils són aplicats en cascada, és a dir, podem utilitzar múltiples estils i el navegador segueix les regles en cascada per determinar la prioritat i resoldre els conflictes. La definició d'una regla consisteix en:

- Un selector que indica l'element o elements als que s'aplicarà l'estil.

- Una declaració que descriu les propietats que conformen l'estil. Cada propietat s'identifica mitjançant el nom de la propietat i el valor que pren.

Així doncs, l'estructura bàsica d'una regla d'estil CSS és de la forma:

```
Etiqueta {
        nom_propietat1: valor1;
        nom_propietat2: valor2;
        ...
        nom_propietatn: valorn;
        }
```

Algunes de les propietats que es poden especificar dins dels estils són:

- de posició: definició de la posició dels elements, com s'ha de mostrar, etc.
- de tipus de lletra: definició de la font de text, de la mida, del seu estil, etc.
- de color i fons: definició del color del text i del color de fons.
- de text: alineació dels elements de text, espai entre caràcters, etc.
- d'enquadrament: definició dels marges, de mida de l'element, etc.

2.1.2 XSL

XSL és l'estàndard definit per W3C que té per objectiu aplicar format als documents XML. Les fulles d'estil XSL afegeixen a un document XML una descripció de com s'ha de visualitzar el document i inclouen una certa capacitat de procés i transformació del contingut del document.

Mitjançant les fulles d'estil XSL el contingut d'un document font XML pot ser mostrat de diferents formes o adoptar diferents formats en funció del dispositiu de sortida (telèfon mòbil, dispositiu PDA, pàgina web, etc.).

XSL consta de dos parts:

- Un llenguatge de transformació: XSLT, mitjançant el qual un document XML es pot transformar en un altre document XML.

- Un llenguatge de formateig, XSL-FO, un vocabulari XML que defineix un conjunt d'elements anomenats objectes de formatge i atributs.

2.1.3 XSLT

XSLT (*Extensible Stylesheet Language Transformations*) és un llenguatge de programació que permet especificar com un document XML és transformat en un altre document de text que pot ser o no, un altre document XML.

Un processador XSLT llegeix com a entrada un document XML i la plantilla XSLT que té associada (que es tracta també d'un document XML, ja que XSLT és una aplicació XML) i produeix un arbre resultant com a sortida. La clara separació entre el document font i l'arbre resultant compleix amb l'objectiu bàsic de XML, separar el contingut del document de la seva presentació, permetent que tant la gramàtica com l'estructura del document XML font romanguin independents de l'estructura i del llenguatge de presentació.

XSLT proporciona diferents elements que permeten definir les regles, patrons i accions que s'han d'aplicar sobre els elements del document XML origen. Cada element del document XML és comparat amb els patrons definits en la plantilla XSLT, i en el cas que coincideixi, és incorporat en l'arbre resultant d'acord amb l'acció associada al patró corresponent.

A continuació veurem un exemple. Primer mostrem el document XML que conté les dades a mostrar:

```
<?xml version = "1.0"?>
<ROWSET>
   <ROW num="1">
      <DNI>23388124E</DNI>
      <NOM>Josep</NOM>
      <COGNOM1>Gonzalvez</COGNOM1>
      <COGNOM2>Cornelles</COGNOM2>
      <DATAALTA>5/3/1979 0:0:0</DATAALTA>
   </ROW>
   <ROW num="2">
      <DNI>32011915D</DNI>
      <NOM>Josep</NOM>
      <COGNOM1>Fontanillas</COGNOM1>
      <COGNOM2>Aragon</COGNOM2>
      <DATAALTA>8/30/1976 0:0:0</DATAALTA>
   </ROW>
</ROWSET>
```

Exemple 2.1: Exemple de document XML.

Definim la plantilla XSLT, on indicarem com es mostra la informació:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="text" version="1.0" indent="yes" encoding="ISO-8859-1"/>

<xsl:template match="ROWSET">
  <xsl:apply-templates select="ROW"/>
</xsl:template>

<xsl:template match="ROW">
   DNI: <xsl:value-of select="DNI"/>
   NOM: <xsl:value-of select="NOM"/>
   1er. COGNOM: <xsl:value-of select="COGNOM1"/>
   2on. COGNOM: <xsl:value-of select="COGNOM2"/>
   DATA ALTA: <xsl:value-of select="DATAALTA"/>

   ------------------------------------------------

</xsl:template>
</xsl:stylesheet>
```

Exemple 2.2: Exemple de plantilla XSLT.

Utilitzant un processador XSLT, obtindríem el següent resultat:

```
   DNI: 23388124E
   NOM: Josep
   1er. COGNOM: Gonzalvez
   2on. COGNOM: Cornelles
   DATA ALTA: 5/3/1979 0:0:0

-------------------------------------------

   DNI: 32011915D
   NOM: Josep
   1er. COGNOM: Fontanillas
   2on. COGNOM: Aragon
   DATA ALTA: 8/30/1976 0:0:0

   -------------------------------------------
```

Exemple 2.3: Resultat d'aplicar la transformació a l'exemple 2.1 amb la plantilla XSLT de l'exemple 2.2.

En aquest cas es tracta de generar un document de text on simplement es llista la informació que conté el document XML.

2.1.4 XSL-FO

A diferencia de les fulles d'estil CSS, XSL-FO és una complexa aplicació XML que serveix per descriure de forma precisa l'aparença de les dades d'un document XML. Normalment no s'escriu XSL-FO directament, si no que s'utilitzen les plantilles XSLT per transformar un document d'entrada XML al format XSL-FO.

Un document XSL-FO descriu el comportament d'una sèrie de caixes o àrees aniuades que es van situant al llarg d'una pàgina i que contenen text i ocasionalment imatges o d'altres continguts externs. Existeixen diferents tipus d'àrees i mitjançant els elements proporcionats per XSL-FO es poden especificar les propietats d'aquestes àrees: la seva posició relativa o absoluta dins la pàgina, l'espaiat entre diferents àrees o regions de text, la seva mida, etc.

Mitjançant d'altres elements definits per XSL-FO, es poden especificar propietats referents al text, com per exemple, et tipus de font a utilitzar, el seu estil, la mida de la font, l'alineació del text, etc.

Al igual que en el cas de l'XSLT, mostrem un exemple complet. El document XML de l'exemple 2.4 conté les dades d'un usuari i el llibre que té en prèstec:

```
<?xml version = "1.0"?>
<ROWSET>
   <ROW num="1">
      <ISBN>84-7602-913-6</ISBN>
      <TITOL>EL BOSC, UN ECOSISTEMA I UN RECURS (12/16)</TITOL>
      <DNI>22761796F</DNI>
      <NOM>Sergio Raúl</NOM>
      <COGNOM1>Baixeras</COGNOM1>
      <DATAPRESTEC>2/12/2002 0:0:0</DATAPRESTEC>
   </ROW>
</ROWSET>
```

Exemple 2.4: Document XML que conté la informació d'un llibre i l'usuari que el té en prèstec.

Definim una plantilla XSLT de traducció a XSL-FO:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<xsl:stylesheet version="1.0" xmlns:fo="http://www.w3.org/1999/XSL/Format"
                      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" indent="yes" encoding="ISO-8859-1"/>

<xsl:template match="ROWSET">
  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>

      <fo:simple-page-master page-height="297mm" page-width="210mm"
          margin="5mm 25mm 5mm 25mm" master-name="PageMaster">
        <fo:region-body margin="20mm 0mm 20mm 0mm"/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <xsl:apply-templates select="ROW"/>
  </fo:root>
</xsl:template>

<xsl:template match="ROW">
  <fo:page-sequence master-reference="PageMaster">
    <fo:flow flow-name="xsl-region-body" >
      <fo:block>
        <xsl:apply-templates select="ISBN"/>
      </fo:block>
      <fo:block>
        <xsl:apply-templates select="TITOL"/>
      </fo:block>
      <fo:block>
        <xsl:apply-templates select="DNI"/>
      </fo:block>
      <fo:block>
        <xsl:apply-templates select="NOM"/>
      </fo:block>
      <fo:block>
        <xsl:apply-templates select="COGNOM1"/>
      </fo:block>
      <fo:block>
        <xsl:apply-templates select="DATAPRESTEC"/>
      </fo:block>
    </fo:flow>
  </fo:page-sequence>
</xsl:template>

<xsl:template match="ISBN">
  <fo:inline font-weight="bold">ISBN:</fo:inline>
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="TITOL">
  <fo:inline font-weight="bold">TITOL:</fo:inline>
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="DNI">
  <fo:inline font-weight="bold">DNI:</fo:inline>
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="NOM">
  <fo:inline font-weight="bold">NOM:</fo:inline>
  <xsl:apply-templates />
</xsl:template>
```

```
<xsl:template match="COGNOM1">
<fo:inline font-weight="bold">COGNOM:</fo:inline>
  <xsl:apply-templates />
</xsl:template>

<xsl:template match="DATAPRESTEC">
  <fo:inline font-weight="bold">DATA PRESTEC:</fo:inline>
  <xsl:apply-templates />
</xsl:template>

</xsl:stylesheet>
```

Exemple 2.5: Plantilla XSLT de transformació a XSL-FO.

Utilitzant un processador XSLT obtindríem el següent document:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
   <fo:layout-master-set>
      <fo:simple-page-master page-height="297mm" page-width="210mm"
                             margin="5mm 25mm 5mm 25mm" master-name="PageMaster">
         <fo:region-body margin="20mm 0mm 20mm 0mm"/>
      </fo:simple-page-master>
   </fo:layout-master-set>
   <fo:page-sequence master-reference="PageMaster">
      <fo:flow flow-name="xsl-region-body">
         <fo:block>
            <fo:inline font-weight="bold">ISBN:</fo:inline>84-7602-913-6</fo:block>
         <fo:block>
            <fo:inline font-weight="bold">TITOL:</fo:inline>EL BOSC, UN ECOSISTEMA
                                                  I UN RECURS (12/16)</fo:block>
         <fo:block>
            <fo:inline font-weight="bold">DNI:</fo:inline>22761796F</fo:block>
         <fo:block>
            <fo:inline font-weight="bold">NOM:</fo:inline>Sergio Raúl</fo:block>
         <fo:block>
            <fo:inline font-weight="bold">COGNOM:</fo:inline>Baixeras</fo:block>
         <fo:block>
            <fo:inline font-weight="bold">DATA PRESTEC:</fo:inline>
                                                        2/12/2002 0:0:0</fo:block>
      </fo:flow>
   </fo:page-sequence>
</fo:root>
```

Exemple 2.6: Document XSL-FO resultant de l'aplicació de la plantilla de l'exemple 2.5 sobre el document XML de l'exemple 2.4.

Aquest document es podria utilitzar amb un formatejador XSL-FO i generar un document PDF o qualsevol altre format que el formatejador XSL-FO sigui capaç de generar.

2.2 Tecnologies de suport per la creació i manipulació

A part de visualitzar o transformar documents XML a altres tipus de documents (PDF, HTML, etc.), també necessitarem poder construir documents XML a partir de zero, unir varis documents, transformar-los, etc. Per realitzar aquestes tasques, dins la tecnologia XML, trobem dos estàndards que veiem a continuació:

- SAX
- DOM

2.2.1 SAX

SAX (Simple API for XML) és un conjunt d'interfícies i classes (API), per al processament de documents XML, que està basat en events. Entenem com a event, per exemple, el fet de trobar l'etiqueta d'inici d'un element, el valor d'un atribut, etc.

Els events són disparats mentre s'analitza el document XML i estan aniuats de la mateixa forma que els elements dins el document, per tant, no es crea cap model de document intermedi. Quan analitzem un document XML amb SAX es genera un flux d'events.

Com que SAX no genera cap estructura intermitja, implica que la utilització de memòria és baixa, molt útil per al processament de documents molt extensos, però al no emmagatzemar en la memòria els events que succeeixen, no es pot retrocedir i per tant, el tractament dels events s'ha de realitzar en el moment en que tenen lloc. Això implica que el SAX no es pot utilitzar quan el document s'ha d'editar o processar diferents cops.

L'esquema de funcionament del SAX és el següent:

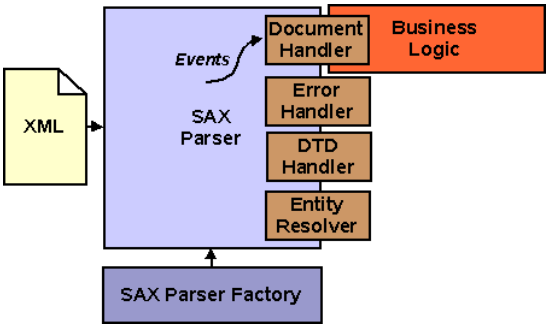

Figura 2.1: Esquema de funcionament del SAX.

SAX és independent de la plataforma i del llenguatge de programació. SAX és útil en aplicacions en les que s'ha de comptar les vegades que succeeix un element, en les que es vol obtenir el valor d'un element o atribut concret, per buscar una certa informació, etc., i en general, per aquelles aplicacions en les que no és necessari conèixer, navegar o modificar l'estructura d'un document XML.

2.2.2 DOM

DOM (Document Object Model) és una especificació W3C presentada com "una interfície neutral de llenguatge i la plataforma que permet a les aplicacions i *scripts* accedir dinàmicament i actualitzar el contingut, l'estructura i l'estil de documents XML". Essencialment és una estructura de dades en forma d'arbre i un conjunt de mètodes per accedir i editar aquesta estructura.

Com el DOM utilitza una estructura de dades en memòria, la seva utilització és molt més gran que en el SAX, però en canvi, pot accedir aleatòriament a un document o processar-lo diferents cops.

Les principals utilitats que proporciona DOM són:

- Construcció i modificació de nous documents: la construcció de documents es pot fer partint de zero o bé, a partir d'altres documents, ja que DOM permet incloure fragments d'un document dins d'un altre document.

- Navegació i modificació de la jerarquia del document: és possible extreure informació de les relacions entre objectes (quins són els elements fills d'un element, qui és el pare, etc.). També és possible modificar aquesta estructura, eliminant nodes de l'arbre, afegint-ne de nous, movent-los de lloc, etc.

- Gestió dels objectes: els objectes representen certes qualitats, com per exemple, el valor d'un atribut, que es poden consultar i modificar.

L'esquema de funcionament del DOM és el següent:

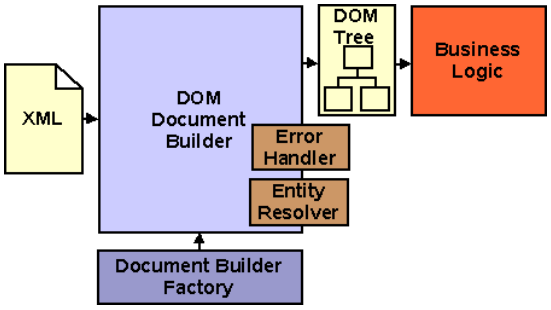

Figura 2.2: Esquema de funcionament del DOM.

2.2.3 Resum Comparatiu

La taula següent mostra les principals diferències entre el SAX i el DOM:

SAX	DOM
Model basat en events	Estructura de dades tipus arbre
Accés sèrie (flux d'events)	Accés aleatori (estructura de dades en memòria)
Baix ús de memòria (únicament es generen events)	Alt ús de memòria (es carrega tot el document en memòria
Per processar parts del document (capturar events importants)	Per editar el document (processar l'estructura de dades en memòria)
Per a processar el document un sol cop (flux d'events temporal)	Per a processar el document diferents cops (document carregat en memòria)

Taula 2.1: Diferències entre SAX i DOM.

En la figura 2.3 podem veure, mitjançant un ejemple, les diferències de representació d'un document XML en SAX i DOM.

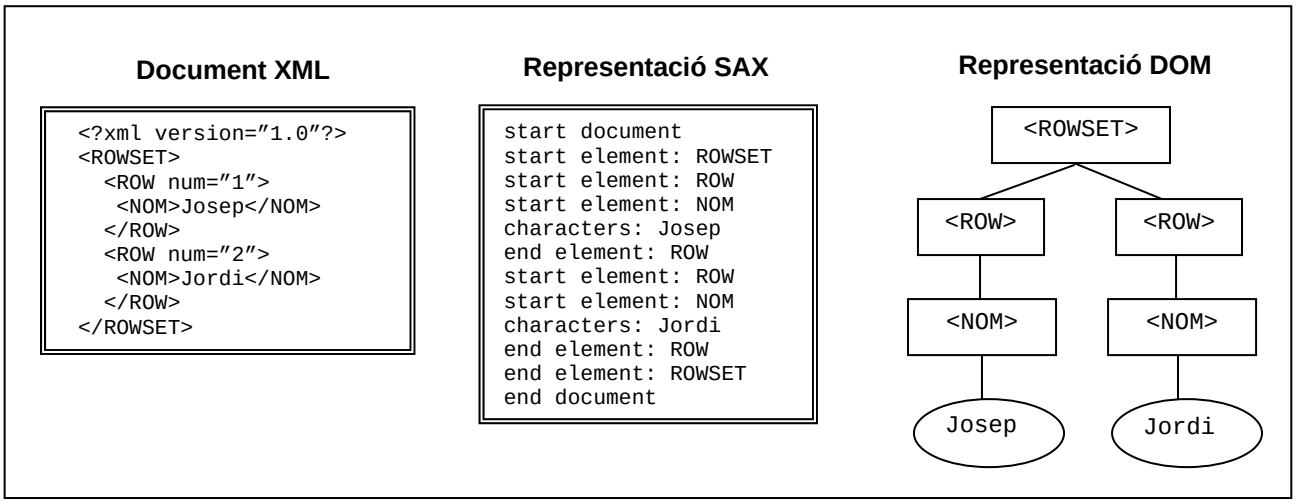

Figura 2.3: Representació d'un document XML en SAX i DOM

2.3 Processadors XSLT

En el mercat existeixen diverses implementacions de processadors XSLT per a la transformació de documents XML en HTML, text, altres documents XML, etc. Hem realitzat un estudi dels principals processadors XSLT per escollir el que més s'adapti a les nostres necessitats tenint en compte els següents punts:

- Independència de plataforma (preferiblement en Java)
- Lliure distribució / codi obert

Alguns exemples de processadors XSLT són :

- **Saxon** : Motor molt potent de XSLT de domini públic (codi obert) que segueix les recomanacions de W3C per a XSLT i XPATH. A més, proporciona APIs per al desenvolupament d'aplicacions Java.
  S'ha analitzat la versió Saxon 7.9.1 (http://saxon.sourceforge.net/).
- **Xalan** : Motor de transformació XSLT desenvolupat per IBM i donat a l'entitat *Apache Software Foundation* per a la seva lliure distribució. Xalan és una de les millors implementacions existents en el mercat i suporta el 100% de la recomanació W3C.
  S'ha analitzat la versió Xalan-Java 2.6.0 (http://xml.apache.org/xalan-j/).
- **JAXP** (Java API for XML Processing): API de JAVA que permet analitzar i transformar documents XML de forma independent de la implementació dels processador XML. Inclòs en les distribucions lliures del JDK de SUN.
  S'ha analitzat la versió JAXP 1.2.4 (http://java.sun.com/xml/jaxp/index.jsp/).

Tot seguit veurem amb més detall les principals característiques de cadascun d'aquests processadors XSLT.

2.3.1 SAXON

El paquet que constitueix el processador Saxon està format per una col·lecció d'eines pel processament dels documents XML. Concretament, els principals components que inclou la versió estudiada (XSLT Saxon 7.9.1) són els següents:

- Un processador XSLT, el qual implementa la versió 2.0 XSLT del W3C. A més, inclou un gran nombre d'extensions que li proporcionen una major potència.
- Un processador XPath (versió 2.0), el qual és accessible mitjançant un API per al desenvolupament d'aplicacions Java.
- Un processador XQuery (versió 1.0), el qual pot ser utilitzat des de línia de comandes o bé, invocat des de una aplicació Java mitjançant l'ús de l'API corresponent.

Així doncs, podem utilitzar Saxon pel processament de plantilles XSLT, per escriure i processar consultes XQuery o bé, per combinar ambdues funcionalitats. Tot això ho podem fer de forma senzilla des d'aplicacions Java fent ús de les API Java que ens proporciona. D'altra banda, l'ús de Java garanteix que Saxon pugui treballar amb independència de la plataforma de treball utilitzada.

2.3.2 XALAN

Xalan és un processador XSLT per a transformar documents XML a HTML, text o altres documents XML. La versió Xalan-Java 2.6.0, versió estudiada, està formada pels següents elements:

- Un processador XSLT que segueix l'especificació XSLT 1.0 del W3C
- Un processador XPath, versió 2.0.

Ambdós processadors poden ser utilitzats tant des de la línia de comanda, com a través d'aplicacions JAVA, ja que en la distribució es faciliten les llibreries necessàries.

Els estàndards que implementa la versió estudiada (2.6.0) són els següents:

- SAX versió 2.0
- DOM nivell 2
- JAXP 1.2

2.3.3 JAXP

L'API JAVA per a processament XML (JAXP), desenvolupada per SUN, permet a les aplicacions analitzar i transformar documents XML utilitzant un API que és independent de qualsevol implementació de processador XML. Mitjançant un esquema connectable, permet als desenvolupadors canviar les implementacions del processador XML sense impactar en les seves aplicacions. És a dir, si tenim una aplicació escrita cap a un processador que soporta JAXP, l'aplicació pot ser migrada cap a un altre processador que també soporti JAXP sense realitzar cap modificació.

L'especificació JAXP 1.2, versió estudiada, soporta els següents estàndards:

- SAX versió 2.0
- DOM nivell 2

També incorpora la implementació d'un processador XSLT, bastant simple, compatible amb l'especificació XSLT 1.0 del W3C. Això representa, que a part de ser una API per independitzar les aplicacions de les implementacions dels processadors XSLT, permeti utilitzar-la alhora com a propi processador XSLT.

L'API JAXP està inclosa en les distribucions lliures del JDK de SUN en diferents versions, concretament en el J2SDK 1.4.2 (versió utilitzada per a l'estudi) inclou l'especificació JAXP 1.2.

2.4 Processadors XSL-FO

En el mercat existeixen diversos processadors XSLT-FO per a la transformació de documents XML-FO en documents PDF, PCL, PS, etc. S'ha realitzat un estudi dels principals processadors XSLT-FO per escollir el que més s'adapti a les nostres necessitats tenint en compte els següents punts:

- Independència de plataforma (preferiblement en Java)
- Lliure distribució / codi obert
- Generació de documents PDF

Alguns exemples de processadors XSLT-FO són :

- **FOP** : És un processador de XSL-FO desenvolupat per Apache XML Project. Es tracta d'una aplicació JAVA que llegeix documents XSL-FO i els transforma en documents PDF, PCL, PS, SVG, AWT, MIF i TXT. És el primer que va aparèixer al mercat i és totalment gratuït. Encara que no és molt potent, és el més utilitzat. Entre les seves avantatges cal destacar que permet treballar amb documents SVG.
  S'ha analitzat la versió FOP 0.20.5 (http://xml.apache.org/fop/).

- **XEP :** Desenvolupat per RenderX. És el processador XSL-FO més avançat, suporta multitud d'elements de formateig, encara que sols genera documents PDF i PS.
  Es tracta d'una eina comercial, encara que hi ha versions d'avaluació amb funcionalitats limitades. Es pot utilitzar tant des de la línia de comandes com des d'aplicacions JAVA.
  S'ha analitzat la versió XEP 3.7.8 (http://www.renderx.com/).

- **XSLFormatter**: És un processador XSL-FO desenvolupat per Antenna House. Es tracta també d'una eina comercial, però aquesta incorpora també una interfície gràfica.
  S'ha analitzat la versió XSL Formatter 3.1 (http://www.antennahouse.com/).

Tot seguit veurem amb més detall les principals característiques de cadascun d'aquests processadors XSLT-FO.

2.4.1 FOP

El FOP (Formatting Object to PDF) és el primer processador XSL-FO que va aparèixer. Va començar a ser desenvolupat en solitari per James Tauber però posteriorment es va incorporar al Apache XML Project, canvi que està accelerant el seu desenvolupament. Encara que no és una eina molt avançada i molt potent, és la més utilitzada, entre altres motius perquè és gratuïta.

Les característiques principals de l'última versió, la versió 0.20.5, són les següents:

- Implementa parcialment la recomanació del W3C XSL-FO 1.0.
- Implementa les transformacions a documents PDF, PCL, PS, SVG, AWT, MIF i TXT.
- Implementa els principals elements de formateig: Taules, imatges, llistes, etc.
- Suporta imatges BMP, EPS, GIF, JPEG i TIFF de forma nativa.
- Suporta imatges PNG a traves de JIMI o JAI i SVG a través de Batik.

El FOP pot ésser utilitzat tant des de la línia de comandes com des d'un aplicatiu JAVA, ja que proporciona les llibreries necessàries.

2.4.2 XEP

El XEP és un processador XSL-FO comercial desenvolupat per RenderX. És, el més avançat i potent. Té implementada l'especificació del W3C XSL-FO 1.0 pràcticament al complet. Això vol dir que suporta quasi tots els elements de formateig definits a l'especificació.

Existeixen dos versions de l'eina, una per a plataforma JAVA, i una altra per la plataforma .NET. La versió analitzada, XEP 3.7.8, genera els següents formats de documents de sortida:

- Adobe Portable Document Format (PDF), versió 1.3
- Adobe PostScript, nivell 2 o 3

I els formats d'elements gràfics que suporta són: PNG, JPEG, GIF, TIFF, SVG i EPS.

Encara que es tracta d'una eina comercial, disposa d'una versió d'avaluació, això si, amb menor nivell de funcionalitats. Tant la versió comercial com d'avaluació poden ésser utilitzades des de la línia de comandes o des d'un aplicatiu JAVA.

2.4.3 XSLFormatter

Es tracta d'un processador XSL-FO, també comercial, desenvolupat per Antenna House. A diferència del FOP o XEP, únicament genera documents PDF. Altres característiques importants de XSL Formatter són:

- Implementa la recomanació del W3C XSL-FO 1.0.
- Suporta imatges BMP, JPEG, PNG, TIFF, GIF, WMF, EMF, EPS, SVG, MathML i CGM.

XSL Formatter pot transformar directament un document XSL-FO a PDF, o també pot interpretar documents XML i traduir-los, basant-se amb una plantilla XSLT.

El XSL Formatter està disponible per a plataformes JAVA i .NET. També per a Windows disposa d'una interfície gràfica (GUI) que ajuda al tractament dels documents.

3 Extracció d'informació amb XML

Abans de començar amb la transformació de les dades en format XML al format PDF, hem d'aconseguir obtenir la informació emmagatzemada dins d'una base de dades qualsevol i extreure-la amb format XML necessari per continuar el procés.

Hem analitzat diferents possibilitats per generar documents XML a partir de consultes realitzades contra una base de dades:

- mitjançant APIs de transformació utilitzant el estàndard SAX o DOM
- mitjançant la utilització de la utilitat XDK d'ORACLE

A continuació veiem com treballen cadascuna d'aquestes eines.

3.1 APIs XML per a base de dades

Per a processar documents XML, la majoria d'eines XML treballen amb els APIs SAX o DOM, permetent treballar amb bases de dades com si fossin document XML.

Com hem vist anteriorment, SAX és una API per a XML basada en events. Amb ella, l'analitzador SAX reporta els events com l'inici o final dels elements de l'aplicació i mentre passen a través del document. Com l'analitzador va reportant els events mentre visita les diferents parts del document, no ha de construir una estructura interna. Per altra banda, la API DOM, realitza una construcció en forma d'arbre. Els elements tenen relacions pare-fill amb altres elements. Amb aquesta API, l'analitzador construeix una estructura interna per on una aplicació pot navegar, ja sigui seqüencialment o amb accés aleatori.

A causa de la estructura altament regular de l'emmagatzematge de dades en una base de dades, la podem transformar fàcilment dins d'un document XML. Per exemple podem transformar una taula d'una base de dades en un document XML amb els següent DTD:

```
<!ELEMENT table rows*>
<!ELEMENT rows (column1, column2, ...)>
<!ELEMENT column1 #PCDATA>
<!ELEMENT column2 #PCDATA>
....
```

Exemple 3.1: DTD per transformar una taula en document XML.

Això implica que, amb una API XML per a base de dades, podem fer que la base de dades sembli un document XML, per a posteriors tractaments amb altres eines, com si d'un document XML es tractés.

En la figura 3.1 veiem quin és aquest procés de transformació de la informació emmagatzemada en una base de dades a un documetn XML.

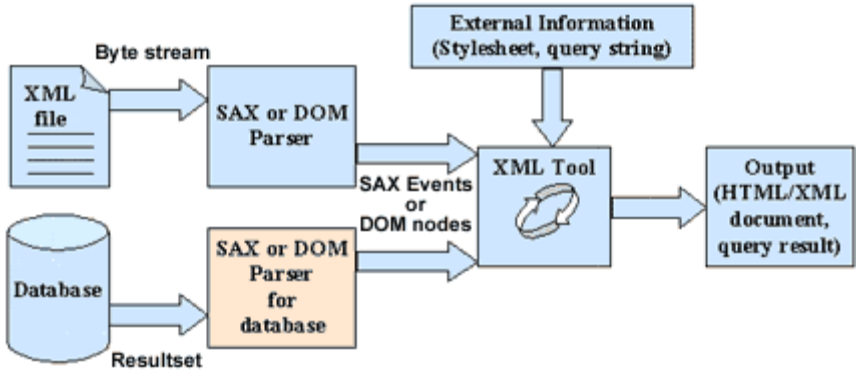

Figura 3.1: Esquema de funcionament de l'extracció d'informació d'una base de dades utilitzant les APIs XML

3.1.1 ORACLE XDK

ORACLE va introduir al 1999 en la seva versió Oracle8i, un primer suport a XML. Realitzava un tractament simple, ja que estava pensat únicament per a realitzar processos d'intercanvi d'informació en format XML. Posteriorment, en la versió Oracle9i, ja es proporciona un suport complet, i des de la pròpia base de dades, per al tractament de documents XML, el XDK. El XDK, XML Developers Kit, proporciona un conjunt d'eines per al processament d'XML tant dins com fóra de la base de dades. Aquest conjunt d'eines són:

- *Parsejadors XML:*

  ORACLE proporciona parsejadors XML per a Java, C, C++ i PL/SQL. Aquests components parsejen els documents XML per tal que puguin ser processats per les aplicacions. Els parsejadors es basen en les interfícies proporcionades per DOM i SAX.

- *Processador XSLT:*

  Eina que permet la transformació dels documents XML a partir de fulles d'estil XSL. Utilitzant el processador XSLT, es poden transformar els documents XML a XML, HTML o qualsevol altre format basat en text.

- *Processador XML Schema*:

  El processador XML Schema parseja i valida els fitxers XML, oferint funcionalitats i característiques que no poden oferir els DTDs.

- *Generador classes XML*:

  Eina capaç de generar automàticament classes Java a partir dels esquemes XML o dels DTDs. És especialment útil quan una aplicació vol enviar dades XML als formularis web o d'altres aplicacions.

- *XSQL Servlet*:

  El Servelt XSQL processa consultes SQL i genera com a sortida els resultats en format XML. És utilitzat pel Parsejador XML de Java i per l'XML SQL Utility per realitzar moltes de les seves operacions.

- *XML SQL Utility*:

  Eina que consisteix en un conjunt de classes Java que permeten:
  - Passar una consulta a la base de dades i generar un document XML a partir dels resultats obtinguts per la consulta (OracleXMLQuery).
  - Escriure dades XML a les taules de la base de dades (OracleXMLSave).

  Per exemple, executant la següent consulta utilitzant la classe OracleXMLQuery que pertany a l'eina XML SQL Utility:

```
SELECT DNI, NOM,COGNOM1,COGNOM2,DATAALTA FROM USUARI WHERE NOM LIKE 'Josep';
```

Exemple 3.2: Sentencia SQL.

es genera el següent document XML:

```
<?xml version = "1.0"?>
<ROWSET>
   <ROW num="1">
      <DNI>23388124E</DNI>
      <NOM>Josep</NOM>
      <COGNOM1>Gonzálvez</COGNOM1>
      <COGNOM2>Cornelles</COGNOM2>
      <DATAALTA>5/3/1979 0:0:0</DATAALTA>
   </ROW>
   <ROW num="2">
      <DNI>32011915D</DNI>
      <NOM>Josep</NOM>
      <COGNOM1>Fontanillas</COGNOM1>
      <COGNOM2>Aragonés</COGNOM2>
      <DATAALTA>8/30/1976 0:0:0</DATAALTA>
   </ROW>
</ROWSET>
```

Exemple 3.3: Document XML resultant de la sentècnia SQL de l'exemple 3.2 amb XDK d'ORACLE.

Podem veure l'esquema de funcionament en el següent gràfic:

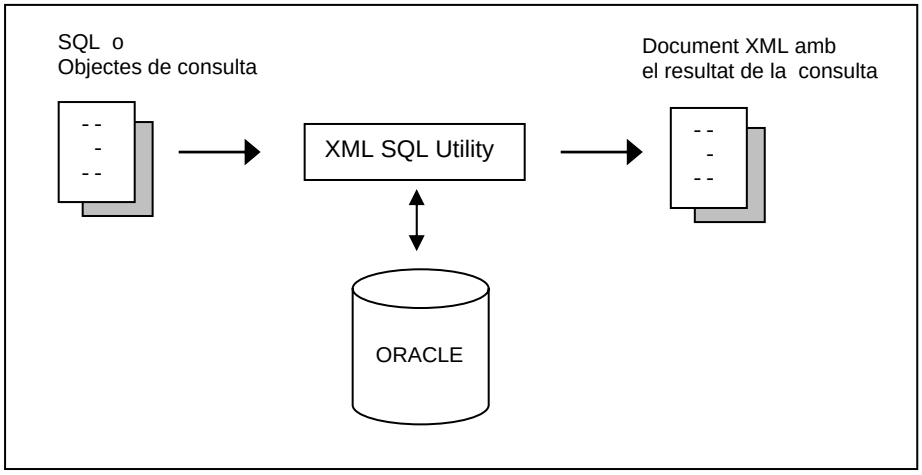

Figura 3.2: Esquema de funcionament de l'extracció d'informació d'una base de dades utilitzant XDK d'ORACLE.

4 Procés de transformació

Un cop hem estudiat les diferents tecnologies i eines a utilitzar en el desenvolupament del projecte podem definir el procés sencer que tindrem que realitzar per a obtenir el document PDF a partir d'informació extreta d'una base de dades.

En essència, aquest procediment consta de dues fases, que alhora són també les parts principals del projecte:

1. Extreure la informació de la base de dades

2. Aplicar el procés de transformació

4.1 Extreure la informació de la base de dades

L'objectiu és extreure de la base de dades la informació necessària per a generar l'informe en format de document XML. En aquesta fase s'obtindrà el conjunt de dades que han d'aparèixer a l'informe, així com l'estructura de la plantilla que defineix el format que tindrà el document a generar.

Per tant, per una banda obtindrem les dades a mostrar a l'informe, per exemple isbn i títol d'un llibre, i per l'altra, com estarà definit el format del document (plantilla), com per exemple, si hi ha capítols i subcapítols, si s'inclou una imatge, taula d'informació, algun text fix, etc. Amb aquests dos conjunts d'informació generarem un document XML que contindrà tant l'estructura com les dades i al qual se li aplicarà el procés de transformació.

Aquesta fase és de gran importància, ja que ens garanteix un procés genèric i independent de l'aplicació o de les dades a l'hora d'aplicar la transformació per generar el document PDF. Això vol dir, que si canviem d'aplicació o es modifica el model de dades, tan sols caldrà modificar el contingut de la plantilla, que està emmagatzemada en taules dins de la pròpia base de dades, sense que la resta del procés es vegi afectat.

4.2 Aplicar el procés de transformació

Aquest procés s'encarregarà, a partir del document XML que es passa, i d'una plantilla de transformació on s'indica com transformar-ho, obtenir un altre document, en aquest cas PDF, amb la informació formatejada.

Aquest procés es pot dividir a la vegada en dues parts:

1. Aplicar al document XML la plantilla XSLT per tal de generar un nou document en format XSL-FO.
2. Transformar el document XSL-FO a PDF o altres formats.

S'utilitzaran eines existents al mercat i estudiades en els punts 2.3 i 2.4 per realitzar cadascuna d'aquestes dues parts, en concret un processador XSLT per al primer punt i un processador XSL-FO per al segon punt.

La següent figura mostra gràficament aquest procés:

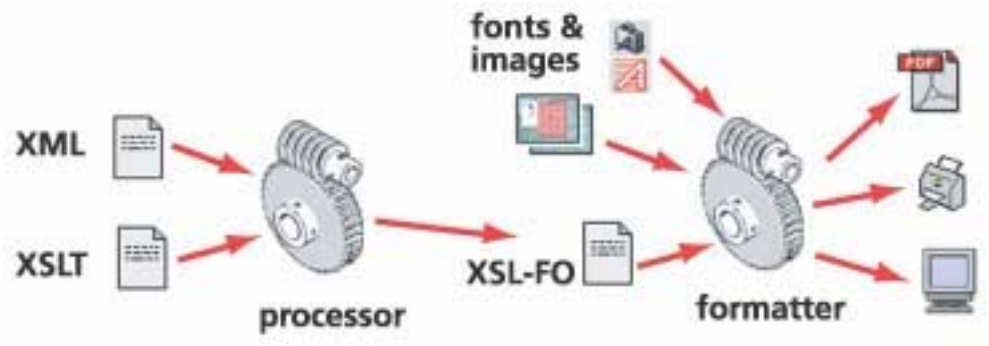

Figura 4.1: Procés de transformació de documents XML a PDF.

Seguint amb l'exemple 2.6 en que ja havíem obtingut el document XSL-FO i aplicant un processador XSL-FO obtenim el resultat en format PDF que mostra la següent figura:

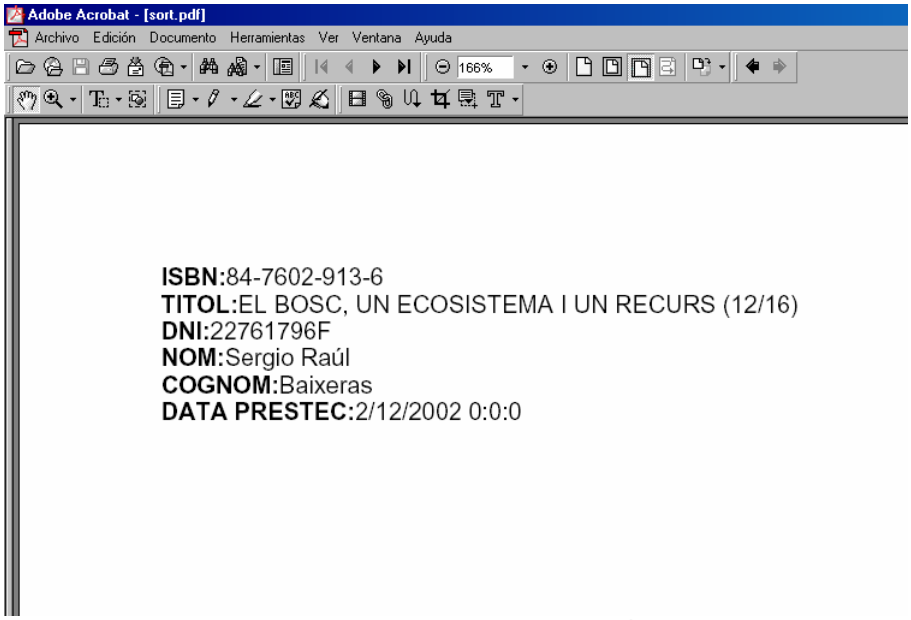

Figura 4.2: Document PDF resultant de la transformació de l'exemple 2.6.

4.3 La nostra eina

Anem a veure com hem implementat la nostra eina de generació de documents en PDF. Es tracta d'una eina desenvolupada en JAVA que permet generar documents en format PDF a partir de l'extracció d'informació d'una base de dades. Aquesta informació inclou, tant les dades a visualitzar com la plantilla que defineix el format que han de tenir les dades.

Primer explicarem quin model de dades hem definit per emmagatzemar l'estructura de les plantilles que definiran el format del document a generar. Posteriorment, definirem quins processadors hem escollit per realitzar la traducció, així com el procés que realitza cadascun d'ells.

4.3.1 Model de dades

Cal definir un model de dades que contindrà els elements necessaris per poder generar informes a partir d'unes plantilles pre-determinades. Aquests informes, definits a partir de les plantilles, poden contenir capítols, text, imatges i taules. Un capítol pot contenir al seu temps, altres capítols, text, imatges i taules.

En base a aquests requisits, cal definir una taula de plantilles de manera que inclogui la informació general del que ha de contenir. També haurà d'existir una taula per guardar els capítols, és la subdivisió de les plantilles, una taula per guardar les possibles imatges a introduir dins un informe, una taula per guardar els texts, i unes taules per guardar les taules d'informació a introduir en els informes.

En el cas dels texts, hi ha la possibilitat de poder introduir meta-dades dins les taules de texts (que facin referència a columnes prefixades de determinades taules). Per exemple, pot haver un text genèric i dins una etiqueta <TITOL_LLIBRE>, de manera que quan surti l'informe tradueixi les etiquetes pel seu valor. Per implementar aquestes meta-dades, haurem d'afegir funcions a la base de dades per recuperar cada camp d'una entitat de l'aplicació origen, per exemple el títol o autor d'un determinat llibre, a partir de la clau primària.

Per tant, un informe està composat per capítols, que al seu temps està composat per altres capítols, texts, imatges o taules d'informació. Cada un dels elements que composa l'informe ho podem considerar com a nodes d'un arbre.

Treballant sobre aquests conceptes i dissenyant la part corresponent de la base de dades, s'arriba a extraure la informació que cal en cada taula. També s'arriba a veure la necessitat de crear una taula d'elements, que contingui informació comuna a tots els elements possibles (capítols, texts, imatges i taules d'informació) i que ens facilitarà el fet de crear una estructura arborescent, on s'indicarà l'ordre de cadascun d'aquests elements.

Els *scripts* que defineixen el model de dades són els següents:

```
create table plantilla (
codi           number(6) not null,
descripcio     varchar2(60) not null,
tipus          varchar2(1) not null,
data_creacio date,
constraint plantilla_pk primary key (codi)
);

create table capitol (
codi           number(6) not null,
nom            varchar2(25) not null,
constraint capitol_pk primary key (codi)
);

create table texte(
codi           number(6) not null,
texte          varchar2(2000) not null,
constraint texte_pk primary key (codi)
);


create table imatge(
codi           number(6) not null,
nom            varchar(200) not null,
imatge         blob,
constraint imatge_pk primary key (codi)
);
```

```
create table elements(
plantilla   number(6) not null,
elements    number(6) not null,
tipus       varchar2(1) not null,
ordre          number(4) not null,
capitol     number(6),
subcapitol     number(6),
constraint elements_pk primary key
(plantilla,capitol,ordre),
constraint elements_capitol foreign key (capitol)
       references capitol(codi),
constraint elements_subcapitol foreign key
(subcapitol) references capitol(codi)
);

create table taula(
codi   number(6) not null,
nom    varchar2(50) not null,
columnes number(2) not null,
constraint taula_pk primary key (codi)
);

create table fila(
codi   number(6) not null,
taula  number(6) not null,
constraint fila_pk primary key (codi,taula),
constraint fila_taula foreign key (taula)
       references taula(codi)
);
```

```
create table columna(                              references fila(taula,codi)
codi   number(6) not null,                     );
taula  number(6) not null,
fila   number(6) not null,                     create table metadades(
texte  varchar2(2000) not null,                metadada   varchar2(50) not null,
constraint columna_pk primary key              funcio    varchar2(2000) not null,
(codi,taula,fila),                             constraint metadades_pk primary key (metadada)
constraint columna_fila foreign key            );
 (taula,fila)
```

Exemple 4.1: Model de dades on s'emmagatzema l'estructura de les plantilles.

4.3.2 Extracció d'informació amb XML

Donat que l'estructura de les plantilles emmagatzemades dins de la base de dades són en forma d'arbre (un capítol pot tenir altres capítols, que a la vegada tenen textos, imatges, etc.), s'ha escollit generar el document XML que englobi tota la informació de la plantilla mitjançant un document DOM. Com hem vist anteriorment, els documents DOM ja s'estructuren en forma d'arbre, a part de que tenen molta més capacitat per la manipulació de documents XML que el SAX.

El procés realitzat ha estat el següent:

1. Es crea un nou document DOM.
2. S'obtenen els diferents elements (capítols, textos, imatges o taules) del primer nivell, és a dir, aquells que no pertanyen a cap capítol.
3. S'adjunta l'element a l'arrel del document DOM, com un node fill.
4. Si l'element és un capítol, a més, es crea un subdocument que contindrà els elements que contingui el capítol i s'adjunta al node que hem afegit al document DOM.

La part de l'aplicació encarregada d'implementar aquest procediment és la següent:

```
   // --

       Statement stmt = conn.createStatement();

       Document document= new DocumentImpl();
       Document docfill= new DocumentImpl();

       Element root = document.createElement("DOCUMENT");
       ResultSet rs = stmt.executeQuery(consulta);
       while (rs.next())
       {
          int n = rs.getInt("ELEMENTS");
          String s = rs.getString("TIPUS");
          docfill=tractarElement(s.charAt(0),n,0);
          root.appendChild(document.importNode(docfill.getDocumentElement(),true));
       }
       document.appendChild( root);

   // --
```

Exemple 4.2: Algorisme de construcció del document DOM.

El procediment *tractarElement* s'encarrega d'obtenir el node fill que contindrà l'element corresponent, que pot ser un únic node o un subarbre si l'element es tractava d'un capítol. És a dir, en funció del tipus d'element que tracta construeix la seva part del document, que posteriorment és afegida al document general. El codi d'aquest procediment és el següent:

```
private static Document tractarElement(char tipus, int codi, int nivell)
  {
    Document doc= new DocumentImpl();
    OracleXMLQuery qry;

    String sql;
    switch (tipus) {

      //Tipus text
      case 'T': {
         sql = "select tractarTexte(texte,'"+clau+"') texte from texte where "+
               "codi="+codi;
         qry = new OracleXMLQuery(conn, sql);
         qry.setRowsetTag("P_TEXTE");
         qry.setRowTag("R_TEXTE");
         doc=qry.getXMLDOM();
         break;
      }

      //Tipus Imatge
      case 'I': {
         try {
            PreparedStatement ps=conn.prepareStatement("select nom,imatge from "+
                                                 "imatge where codi="+codi);
           ResultSet rs = ps.executeQuery();
           int c;
           if(rs.next()){
             File fitxer= new File(rs.getString("NOM"));
             FileOutputStream out = new FileOutputStream(fitxer);
             InputStream in = rs.getBinaryStream("IMATGE");
             while ((c = in.read()) != -1)
               out.write(c);
               out.close();
           }
           sql = "select nom from imatge where codi="+codi;
           qry = new OracleXMLQuery(conn, sql);
           qry.setRowsetTag("P_IMATGE");
           qry.setRowTag("R_IMATGE");
           doc=qry.getXMLDOM();
         }
         catch (Exception e) {
           System.err.println("error: - "+e.getMessage());
         }
         break;
      }

      //Tipus Capitol
      case 'C': {
         Document doc2 = new DocumentImpl();
         Document doc3= new DocumentImpl();
         sql = "select nom from capitol where codi="+codi;
         qry = new OracleXMLQuery(conn, sql);
         qry.setRowsetTag("P_CAPITOL");
         qry.setRowTag("R_CAPITOL");
         doc2=qry.getXMLDOM();
         Element capitol = doc.createElement("CAPITOL"+nivell);
         capitol.appendChild(doc.importNode(doc2.getDocumentElement(),true));
         String consulta="select elements, tipus from elements where"+
                         " plantilla="+plantilla+" and capitol="+codi+
                         " order by ordre";
         try{
          Statement stmt = conn.createStatement();
          ResultSet rs = stmt.executeQuery(consulta);
           while (rs.next()) {
             int n = rs.getInt("ELEMENTS");
```

```
                String s = rs.getString("TIPUS");
                doc3=tractarElement(s.charAt(0),n,nivell+1);
                capitol.appendChild(doc.importNode(doc3.getDocumentElement(),true));
              }
              doc.appendChild(capitol);
            }
            catch (Exception e) {
               System.err.println("error: - "+e.getMessage());
            }
            break;
        }

        //Tipus Taula informació
        case 'L': {
          Document doc2= new DocumentImpl();
          Document doc3= new DocumentImpl();
          Document doc4= new DocumentImpl();
          try{
            String consulta="select nom,columnes from taula where codi="+codi;
            Statement stmt = conn.createStatement();
            ResultSet rs = stmt.executeQuery(consulta);
            Element taula = doc.createElement("TAULA");
            String str=null;;
            int i,cols=0;
            if(rs.next()){
               taula.setAttribute("NOM",rs.getString("NOM"));
               cols=rs.getInt("COLUMNES");
               str="\n";
               for (i=0;i<cols;i++)
               {
                    str=str+"<fo:table-column column-width=\""+160/cols+"mm\"/>\n";
               }
               taula.setAttribute("COLS",str);
               consulta="select codi from fila where taula="+codi+
                        " order by codi";
               rs = stmt.executeQuery(consulta);
               i=0;
               while (rs.next()) {
                    i++;
                    int n = rs.getInt("CODI");
                   sql = "select tractarTexte(texte,'"+clau+"') texte from columna "
                        "where taula="+codi+" and fila="+n+" order by codi";
                    qry = new OracleXMLQuery(conn, sql);
                    qry.setRowsetTag("P_FILA");
                    qry.setRowTag("R_COLUMNA");
                    doc2=qry.getXMLDOM();
                    taula.appendChild(doc.importNode(doc2.getDocumentElement(),true));
                  }
               }
               doc.appendChild(taula);
            }
            catch (Exception e) {
              System.err.println("error: - "+e.getMessage());
            }
            break;
          }
        }
        return doc;
      }
```

Exemple 4.3: Codi del procediment tractarElement.

Com podem observar, per cada tipus d'element es crea un node amb les seves etiquetes corresponents, per tal què, posteriorment, la plantilla XSLT li doni format. Així, els capítols tenen l'etiqueta <P_CAPITOL>, els texts <P_TEXTE>, les imatges <P_IMATGE> i les taules amb les files i columnes tenen <TAULA>, <P_FILA> i <R_COLUMNA> respectivament. Un cas especial

són les etiquetes <CAPITOL0> i <CAPITOL1> que correspon als subarbres d'un capítol i d'un subcapítol.

També, a l'hora de recuperar un text, aquest es processa amb la funció tractarTexte, implementada amb PL/SQL i SQL dinàmic, que substitueix totes les meta-dades que conté el text pel seu valor. El valor l'obté fent una crida a la funció associada a cada meta-dada dins la taula metadades. A continuació podem veure la seva implementació:

```
CREATE OR REPLACE FUNCTION tractarTexte( texte IN VARCHAR2, codi IN VARCHAR2)
return VARCHAR2 IS
cursor c is SELECT metadada,funcio FROM metadades;
sort VARCHAR2(2000);
resultat VARCHAR2(2000);
vcursor  INTEGER;
BEGIN
  sort:=texte;
  FOR r IN c LOOP
    vcursor := DBMS_SQL.open_cursor;
    DBMS_SQL.parse( vcursor, 'select '||r.funcio||'('''||codi||''') from dual',
                  DBMS_SQL.native );
    DBMS_SQL.define_column( vcursor, 1, resultat, 2000 );
    IF DBMS_SQL.execute_and_fetch( vcursor ) > 0 THEN
        DBMS_SQL.column_value( vcursor, 1, resultat );
    ELSE
      RAISE NO_DATA_FOUND;
    END IF;
    DBMS_SQL.close_cursor( vcursor );
    SELECT replace(sort,r.metadada,resultat) INTO sort FROM dual;
   END LOOP;
 return sort;
END tractarTexte;
```

Exemple 4.4: Codi de la funció PL/SQL tractarTexte.

Encara que, com hem vist, a l'hora de generar el document XML global que contindrà tant les dades com l'estructura, utilitzem l'API DOM, per a l'extracció de la informació des de la base de dades utilitzem la utilitat XDK d'ORACLE. En concret l'eina OracleXMLQuery, ja que ens retorna directament les dades en format XML i la nostra funció sols cal que vagi asignant els nodes d'acord a l'estructura definida.

4.3.3 Transformació XML a XSL-FO

Un cop hem extret la informació de la base de dades amb XML, i seguint el procés que hem vist en el punt 4.2, hem de crear una plantilla XSLT per a transformar el document XML en XML-FO, necessari per generar el document PDF que volem obtenir al final. Després, aplicarem al document XML i a la plantilla XSLT un processador XSLT que ens generarà el document XSL-FO, per tant, també hem de seleccionar quin procesador XSLT farem servir.

4.3.3.1 Creació de la plantilla XSLT

Donat que hem construït una eina genèrica, i per tant, el procés d'extracció d'informació sempre ens generarà, independentment de l'aplicació o el model de dades, un document XML amb la mateixa estructura, només ens cal definir una única plantilla XSLT de transformació.

L'objectiu d'aquesta plantilla és definir els canvis a produir en la informació extreta de la base de dades en XML per a donar-li el format XSL-FO. Per tant, la plantilla tracta els diferents elements que hem definit al procés anterior: Capítols, subcapítols, texts, imatges i taules d'informació.

A continuació es mostra la plantilla que s'ha construït:

```
<?xml version="1.0" encoding="ISO-8859-15" ?>
<xsl:stylesheet version="1.0" xmlns:fo="http://www.w3.org/1999/XSL/Format"
                    xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" indent="yes" encoding="ISO-8859-15"/>

<xsl:template match="DOCUMENT">
  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
      <fo:simple-page-master page-height="297mm" page-width="210mm"
          margin="5mm 25mm 5mm 25mm" master-name="PageMaster">
        <fo:region-body margin="20mm 0mm 20mm 0mm"/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="PageMaster">
      <fo:flow flow-name="xsl-region-body">
        <xsl:apply-templates />
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>

<xsl:template match="CAPITOL0">
  <fo:block font-size="24pt"
            font-weight="bold"
            space-after="14pt"
            break-before="page"
            keep-with-next.within-page="always"
            border-after-style="solid"
            border-after-width="2pt"
            id="{generate-id()}">
    <xsl:value-of select="P_CAPITOL/R_CAPITOL/NOM" />
  </fo:block>
  <fo:block>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="CAPITOL1">
  <fo:block font-size="18pt"
            font-weight="bold"
            space-after="10pt"
            space-before="14pt"
            border-after-style="solid"
            id="{generate-id()}">
    <xsl:value-of select="P_CAPITOL/R_CAPITOL/NOM" />
  </fo:block>
  <fo:block>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="NOM">
</xsl:template>

<xsl:template match="P_TEXTE/R_TEXTE">
  <fo:block text-align="justify">
    <xsl:value-of select="TEXTE" />
  </fo:block>
</xsl:template>

<xsl:template match="P_IMATGE/R_IMATGE">
  <xsl:variable name="fitxer"> <xsl:value-of select="NOM" /> </xsl:variable>
  <fo:block>
    <fo:external-graphic src="{$fitxer}">
```

```
      </fo:external-graphic>
    </fo:block>
  </xsl:template>

  <xsl:template match="TAULA">
    <fo:block space-after="10pt">
      <xsl:value-of select="@NOM" />
    </fo:block>
    <fo:table table-layout="fixed">
      <xsl:value-of select="@COLS" />
      <fo:table-body>
        <xsl:apply-templates select="P_FILA" />
      </fo:table-body>
    </fo:table>
    <fo:block space-after="10pt"/>
  </xsl:template>

  <xsl:template match="P_FILA">
    <fo:table-row>
      <xsl:apply-templates select="R_COLUMNA" />
    </fo:table-row>
  </xsl:template>

  <xsl:template match="R_COLUMNA">
    <fo:table-cell border="solid #000000 1px">
      <fo:block>
        <xsl:value-of select="TEXTE" />
      </fo:block>
    </fo:table-cell>
  </xsl:template>

  </xsl:stylesheet>
```

Exemple 4.5: Plantilla XSLT.

Podem observar que la plantilla és molt simple, i es limita a fer el tractament dels diferents elements (capítols, subcapítols, texts, imatges i taules). Podríem millorar el format del document resultant afegint elements com peus de pàgina, capçaleres, taules de contingut, índexs, etc. però això ho anotarem com a futures línies de treball al final d'aquesta memòria.

4.3.3.2 Processador XSLT

En el punt 2.3 hem analitzat diferents processadors XSLT existents en el mercat. Finalment, després d'avaluar les necessitats del projecte i les diferents funcionalitats dels tres processadors XSLT analitzats, s'ha escollit per a la implementació del projecte el processador JAXP, inclòs en la distribució J2SDK 1.4.2 utilitzada com a plataforma de desenvolupament.

Tots tres processadors XSLT cobreixen els nostres objectius inicials: són de lliure distribució, independent de plataforma i en JAVA, però el JAXP a més, és independent de la implementació del processador XSLT. És a dir, podríem substituir el processador XSLT que porta el JAXP, per exemple pel XALAN que n'és compatible, i la nostra eina continuaria funcionant sense realitzar cap modificació.

Com que utilitzem un processador ja implementat, el procés de transformació és molt simple, tant sols cal invocar al procesador, passant-li el document DOM (XML) obtingut en l'extracció d'informació, juntament amb la plantilla XSLT construïda, i ens retorna el document XSL-FO resultant de la transformació.

A continuació podem veure la part de codi que invoca al processador JAXP i obté el document XSL-FO:

```
        //Obtenim el document DOM
        DOMSource source = new DOMSource(document);

        // Fem la transformació a XSL-FO utilitzant la pantilla.xsl
        // Obtenim la instancia per la traduccio.
        TransformerFactory tFactory = TransformerFactory.newInstance();

        // Associem el xls a la traduccio.
        Source xslSource = new StreamSource(new File("plantilla_fo.xsl"));

        // Generem la traduccio.
        Transformer transformer = tFactory.newTransformer(xslSource);

        // Realitzem la traduccio i enviem el resultat a la sortida estàndard.
        transformer.transform(source,new StreamResult(System.out));
```

Exemple 4.6: Codi de la transformació XSLT.

4.3.4 Transformació XSL-FO a PDF

Ja tenim un document XSL-FO. A partir d'aquí, tant sols ens queda l'últim pas, que és aplicar un processador XSL-FO per tal de generar un document amb el format escollit.

Al mercat hi ha múltiples processadors, i cadascun d'ells permet generar diversos formats, encara que els més utilitzats són el format PDF i PostScript. La nostra elecció ha estat el processador FOP i el format de sortida PDF.

A continuació veurem perquè hem escollit el document PDF com a format de sortida i com es realitza aquest últim procés de transformació.

4.3.4.1 Format PDF

El format PDF (Portable Document Format) és l'estàndard de facto per a la distribució i intercanvi segur i fiable de documents electrònics. Es tracta d'un format natiu, creat per Adobe, que es basa en el llenguatge PostScript per a descriure el text, els gràfics i les imatges, independentment de qualsevol plataforma. El format PDF és una especificació de format d'arxiu oberta que es troba a disposició de qualsevol per a la seva utilització.

Els documents PDF mantenen les seves característiques originals, és a dir, format, fonts, imatges, gràfics, format de pàgina, etc. independentment de l'aplicació i plataforma utilitzada per a crear-lo, el que significa que qualsevol ordinador amb qualsevol sistema operatiu pot visualitzar-los correctament. Els arxius PDF son compactes i complets, que es poden compartir, veure o imprimir amb una aplicació gratuïta i disponible per a qualsevol plataforma.

4.3.4.2 Processador XSL-FO

En el punt 2.4 hem analitzat diferents processadors XSL-FO existents en el mercat. Encara que un dels objectius era que fos de lliure distribució, és difícil trobar-ne algun, ja que la majoria són comercials. L'excepció és el FOP, que encara què, com és lògic, no és el més potent, cobreix perfectament les nostres necessitats. Per això l'hem escollit per a la nostra implementació.

Un cop decidit el processador, el procés de transformació, a l'igual que en l'anterior pas, és molt simple. Només cal invocar al processador passant-li el document XSL-FO i obtenir el resultat de la transformació a PDF.

En el cas del FOP, no se li passa el document origen, sinó que cal passar-li un analitzador SAX sobre el document origen, per tal de que el pugui processar. A continuació veiem el procediment de transformació de XML a XSL-FO descrit en l'anterior pas, modificat amb la creació de l'analitzador SAX i encadenant la sortida de la primera transformació amb l'entrada de la segona transformació, per tal d'obtenir finalment en document PDF desitjat.

```
        //Obtenim el document DOM
        DOMSource source = new DOMSource(document);

        Driver driver = new Driver();

        //Creem el logger
        Logger logger = new ConsoleLogger(ConsoleLogger.LEVEL_INFO);
        driver.setLogger(logger);
        MessageHandler.setScreenLogger(logger);

        //Creem el Renderer (format PDF)
        driver.setRenderer(Driver.RENDER_PDF);

        //Creem el fitxer on guardarem el document PDF
        OutputStream out = new java.io.FileOutputStream("Sortida.pdf");
        try
        {
          driver.setOutputStream(out);

          // Fem la transformació a XSL-FO utilitzant la pantilla.xsl
          // Obtenim la instancia per la traduccio.
          TransformerFactory tFactory = TransformerFactory.newInstance();

          // Associem el xls a la traduccio.
          Source xslSource = new StreamSource(new File("plantilla_fo.xsl"));

          // Generem la traduccio.
          Transformer transformer = tFactory.newTransformer(xslSource);
          Result res = new SAXResult(driver.getContentHandler());

          // Realitzem la traduccio i enviem el resultat al fitxer PDF.
          transformer.transform(source,res);
        }
        finally
        {
            out.close();
        }
```

Exemple 4.7: Codi de la transformació XSL-FO.

5 Un cas pràctic: Gestió d'una biblioteca

Com a exemple pràctic del projecte suposem una aplicació de gestió d'una biblioteca, on hi ha la necessitat, entre altres, de que els usuaris puguin imprimir-se un resum dels diferents llibres: pot ser d'un llibre concret, o el resultat d'una cerca per títol, autor, tema, etc.

Es tracta d'una aplicació existent en el mercat, BiblioGes, que cobreix tota la gestió d'una biblioteca, és genèrica i es pot implantar a qualsevol biblioteca. Des de la seva creació i primera implantació, sempre ha gaudit d'una gran fama d'aplicació robusta i molt ajustada a les necessitats de gestió de qualsevol biblioteca. Actualment, s'han realitzat moltes implantacions, però, un cop consolidada la fase de gestió, els clients comencen a demanar noves funcionalitats, bàsicament en el mòdul de sortides impreses, on l'aplicació actual te certes mancances.

Les principals queixes, venen donades pel fet de que quan una biblioteca necessita un informe nou, cal que el sol·liciti a l'empresa, cosa que fa augmentar molt el cost i el temps de resposta. Per altra banda, l'empresa ha de dedicar molts recursos per a desenvolupar i mantenir els diferents informes dels clients, donat que cadascun d'ells desitja tenir un format diferent. Això provoca un cost més elevat que acaba repercutint al client, el qual no sempre està disposat a pagar i es generen enfrontaments.

Per evitar aquests problemes creiem que una solució seria l'aplicació del nostre projecte. Amb la implantació de la nostra eina de generació de documents podem:

- Per una banda, oferir als clients una eina per a que ells mateixos es puguin definir i personalitzar, segons les seves necessitats, qualsevol informe que extregui informació de BiblioGes.

- Per l'altra, simplificar el manteniment que ha de realitzar l'empresa, centrant-lo en el nucli de gestió, i per tant, abaratint el cost i millorant les relacions amb els clients.

Tot i tractar-se d'una eina genèrica per a l'empresa, de cara al client li permet definir-se els informes de forma personalitzada i amb molt poc cost i temps. Creiem que és la millor solució per a les dues bandes.

Així, doncs, anem a veure com incorporem aquesta eina a la nostra aplicació Biblioges. Primer, però veurem, a grans trets, com és l'actual aplicació Biblioges, i posteriorment descriurem com hem anat implementat la nostra eina i el resultat obtingut.

5.1 BiblioGes

BiblioGes és una aplicació desenvolupada amb una arquitectura de tres capes seguint la tecnologia J2EE. La interfície és web, una capa molt fina generada dinàmicament des de la capa de lògica. La capa intermitja o de lògica, està implementada en JAVA i resideix en el servidor d'aplicacions ORACLE IAS 9i. I la capa de dades està implementada en el gestor de base de dades Oracle9i.

L'àmbit funcional de BiblioGes és la gestió d'una biblioteca: llibres, usuaris, registre de préstecs, etc. Com es pot observar, es centra més en el registre dels llibres de que disposa la biblioteca i el registre de préstecs i el seu seguiment. A continuació detallem la seva estructura, donat que posteriorment ens serà d'utilitat per explicar el nostre exemple.

Els *scripts* que defineixen el seu model de dades són els següents:

```
create table autor (
codi           number(6) not null,
nom            varchar2(60) not null,
constraint autor_pk primary key (codi),
constraint autor_uk unique (nom)
);

create table ciutat (
codi           number(6) not null,
nom            varchar2(25) not null,
constraint ciutat_pk primary key (codi),
constraint ciutat_uk unique (nom)
);

create table descriptor (
codi           number(6) not null,
nom            varchar2(30) not null,
constraint descriptor_pk primary key (codi),
constraint descriptor_uk unique (nom)
);

create table editorial (
codi           number(6) not null,
nom            varchar2(35) not null,
constraint editorial_pk primary key (codi),
constraint editorial_uk unique (nom)
);

create table obra (
isbn           varchar2(15) not null,
titol          varchar2(75) not null,
editorial      number(6),
ciutatedicio   number(6),
pagines        number(6) not null,
alcada         number(6,2) not null,
constraint obra_pk primary key (isbn),
constraint obra_editorial_fk foreign key
(editorial)
       references editorial(codi),
constraint obra_ciutat_fk foreign key
(ciutatedicio)
       references ciutat(codi)
);

create table escriure (
obra           varchar2(15) not null,
autor          number(6) not null,
constraint escriure_pk primary key (obra,autor),
constraint escriure_obra_fk foreign key (obra)
       references obra(isbn),
constraint escriure_autor_fk foreign key (autor)
       references autor(codi)
);

create table especificar (
obra           varchar2(15) not null,
descriptor     number(6) not null,
constraint especificar_pk primary key
 (obra,descriptor),
constraint especificar_obra_fk foreign key (obra)
       references obra(isbn),
constraint especificar_descriptor_fk foreign key
(descriptor)
       references descriptor(codi)
);

create table exemplar (
registre       number(6) not null,
obra           varchar2(15) not null,
datacompra     date not null,
constraint exemplar_pk primary key (registre),
constraint exemplar_obra_fk foreign key (obra)
       references obra(isbn)
);

create table usuari (
dni            varchar2(9) not null,
nom            varchar2(25) not null,
cognom1        varchar2(25) not null,
cognom2        varchar2(25),
dataalta       date not null,
constraint usuari_pk primary key (dni)
);

create table prestec (
numero         number(6) not null,
exemplar       number(6) not null,
usuari         varchar2(9) not null,
dataprestec    date not null,
dataretorn     date,
constraint prestec_pk primary key (numero),
constraint prestec_exemplar_fk foreign key
(exemplar)     references exemplar(registre),
constraint prestec_usuari_fk foreign key (usuari)
       references usuari(dni)
);

create index obra_editorial_fk on obra(editorial);
create index obra_ciutat_fk on obra(ciutatedicio);

create index escriure_obra_fk on escriure(obra);
create index escriure_autor_fk on escriure(autor);

create index especificar_obra_fk on
especificar(obra);
create index especificar_descriptor_fk on
especificar(descriptor);

create index exemplar_obra_fk on exemplar(obra);

create index prestec_exemplar_fk on
prestec(exemplar);

create index prestec_usuari_fk on prestec(usuari);
```

Exemple 5.1: Model de dades de BiblioGes.

5.2 Un informe nou per BiblioGes

Davant la insintència dels clients a l'hora de demanar nous informes suposem que l'empresa de BiblioGes s'ha decidit a implantar la nostra eina de generació de documents PDF. A continuació veurem, pas a pas, com s'ha realitzat un nou informe a BiblioGes utilitzant la nostra eina.

5.2.1 Extracció d'informació amb XML

Anem a veure com el que hem explicat fins ara ho portem a la pràctica. Hem afegit a la base de dades de BiblioGes la nostra part del model, definida al punt 4.3.1, i ens interessa crear un informe que donat un identificador d'un llibre (ISBN) ens mostri el seu títol, autor, una foto, les seves dades tècniques i un breu resum. Els passos seguits són el següents:

1. Afegim una nova plantilla, a la taula plantilla, que anomenarem Informació de llibre.

2. Creem els diferents elements que necessitarem, en les seves respectives taules:
   - Un capítol que contindrà tota la informació del llibre. (C0)
   - Una imatge que serà la portada del llibre. (I1)
   - Un text que contindrà el títol del llibre. (T1)
   - Un text que contindrà l'autor del llibre. (T2)
   - Un subcapítol que contindrà les dades tècniques. (C1)
   - Una taula que tindrà 4 files i 2 columnes on especificarem en la primera columna de cada fila els texts ISBN, Editorial, Llengua i pàgines del llibre respectivaments, i en la segona columna els seus respectius valors. (L1)
   - Un subcapítol que contindrà el resum del llibre. (C2)
   - Un text que serà el resum del llibre. (T3)

3. Ordenem els elements a la taula d'elements

Element	Ordre	Dins capítol
C0	1	
I1	2	C0
T1	3	C0
T2	4	C0
C1	5	C0
L1	6	C1
C2	7	C0
T3	8	C2

Finalment, hem comentat que dins els texts hi havia meta-dades de l'estil <TITOL_LLIBRE> per tal de substituir-les pel seu valor a l'hora de generar el document amb la funció tractarTexte. Per tant, hem definit les possibles meta-dades que apareixen dins els diferents texts (T1, T2, T3, i dins les diferents columnes de la taula). Aquest són:

- <TITOL_LLIBRE>
- <ISBN_LLIBRE>
- <PAGINES_LLIBRE>
- <AUTOR_LLIBRE>
- <EDITORIAL_LLIBRE>

Per cada una d'aquestes meta-dades hem associat, a la taula metadades, una funció PL/SQL que ens retornarà el seu valor, a partir de la clau del llibre. Aquesta funció serà la que cridarà la funció

tractarTexte per obtenir el seu valor. Un exemple d'aquestes funcions ho podem veure a l'exemple 5.2 que obté el títol del llibre.

```
Function obtTitolLlibre(p_codi IN  VARCHAR2) RETURN VARCHAR2 IS
sort varchar2(2000);
BEGIN
  select titol into sort from obra where ISBN=p_codi;
  return sort;
END;
```

Exemple 5.2: Codi de la funció PL/SQL obtTitolLlibe.

Un cop tenim definida tota l'estructura de la plantilla, aplicarem el procés d'extracció d'informació amb XML descrit en el punt 4.3.2. Aquest procés ens generarà un document XML, tant amb les dades com amb la seva estructura (capítols, taules, imatges, texts, etc.) . A continuació podem veure el resultat d'aplicar-ho a la plantilla Informació de llibre definida anteriorment:

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<DOCUMENT>
<CAPITOL0>
    <P_CAPITOL>
      <R_CAPITOL num="1">
        <NOM>El Club de Dante</NOM>
      </R_CAPITOL>
    </P_CAPITOL>
    <P_IMATGE>
      <R_IMATGE num="1">
        <NOM>dante.bmp</NOM>
      </R_IMATGE>
    </P_IMATGE>
    <P_TEXTE>
      <R_TEXTE num="1">
        <TEXTE>El Club de Dante</TEXTE>
      </R_TEXTE>
    </P_TEXTE>
    <P_TEXTE>
      <R_TEXTE num="1">
        <TEXTE>de PERAL, MATTHEW</TEXTE>
      </R_TEXTE>
    </P_TEXTE>
    <CAPITOL1>
      <P_CAPITOL>
        <R_CAPITOL num="1">
          <NOM>Dades tècniques</NOM>
        </R_CAPITOL>
      </P_CAPITOL>
      <TAULA NOM="Les dades tècniques d'aquest llibre son:">
        <P_FILA>
          <R_COLUMNA num="1">
            <TEXTE>ISBN:</TEXTE>
          </R_COLUMNA>
          <R_COLUMNA num="2">
            <TEXTE>84-322-9632-5</TEXTE>
          </R_COLUMNA>
        </P_FILA>
        <P_FILA>
          <R_COLUMNA num="1">
            <TEXTE>Editorial:</TEXTE>
          </R_COLUMNA>
          <R_COLUMNA num="2">
             <TEXTE>SEIX BARRAL, S.A.</TEXTE>
          </R_COLUMNA>
        </P_FILA>
```

```
          <P_FILA>
            <R_COLUMNA num="1">
              <TEXTE>Llengua:</TEXTE>
            </R_COLUMNA>
            <R_COLUMNA num="2">
              <TEXTE>Castellana</TEXTE>
            </R_COLUMNA>
          </P_FILA>
          <P_FILA>
            <R_COLUMNA num="1">
              <TEXTE>Pàgines:</TEXTE>
            </R_COLUMNA>
            <R_COLUMNA num="2">
              <TEXTE>472</TEXTE>
            </R_COLUMNA>
          </P_FILA>
        </TAULA>
      </CAPITOL1>
      <CAPITOL1>
        <P_CAPITOL>
          <R_CAPITOL num="1">
            <NOM>Resum del llibre</NOM>
          </R_CAPITOL>
        </P_CAPITOL>
        <P_TEXTE>
          <R_TEXTE num="1">
            <TEXTE>Una novel·la històrica de suspens comparada insistentment amb
                   El Código da Vinci y El Nombre de la Rosa de Umberto Eco.</TEXTE>
          </R_TEXTE>
        </P_TEXTE>
        <P_TEXTE>
          <R_TEXTE num="1">
            <TEXTE>Boston 1865. Importants personalitats estan sent brutalment
                   assassinades per un criminal inspirat en els torments de l'Infern
                   de Dante. Sols els membres del Club Dante, poetes i professors
                   de Harward dirigits per Henry Wadsworth Longfeloww, poden
                   anticipar-se a l'assassí i identificar-lo. Mentre preparen
                   la traducció americana de la Divina Comedia enfrontant-se
                   a l'oposició de la puritana vella guàrdia de Harward, els
                   intel·lectuals hauran de convertir-se en detectius i passar
                   a l'acció</TEXTE>
          </R_TEXTE>
        </P_TEXTE>
      </CAPITOL1>
    </CAPITOL0>
  </DOCUMENT>
```

Exemple 5.3: Document XML resultant del procés d'extracciódе la base de dades.

Podem observar que en aquest document XML ja no apareixen les meta-dades, ja que han estat traduïdes pels seus valors respectius a l'hora de fer l'extracció.

5.2.2 Transformació XML a XSL-FO

En el document XML que hem obtingut en l'anterior punt hem aplicant el procés de transformació amb el JAXP descrit anteriorment, juntament amb la plantilla que hem construït (en el punt 4.3.3.1). El resultat és el següent:

```
<?xml version="1.0" encoding="ISO-8859-15"?>
<fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
 <fo:layout-master-set>
```

```
   <fo:simple-page-master page-height="297mm" page-width="210mm"
                          margin="5mm 25mm 5mm 25mm" master-name="PageMaster">
      <fo:region-body margin="20mm 0mm 20mm 0mm"/>
   </fo:simple-page-master>
</fo:layout-master-set>
<fo:page-sequence master-reference="PageMaster">
   <fo:flow flow-name="xsl-region-body">
      <fo:block font-size="24pt" font-weight="bold" space-after="14pt"
                break-before="page" keep-with-next.within-page="always"
                border-after-style="solid" border-after-width="2pt" id="w0aa">
                El Club de Dante</fo:block>
      <fo:block>
         <fo:block>
            <fo:external-graphic src="dante.bmp"/>
         </fo:block>
         <fo:block text-align="justify">El Club de Dante</fo:block>
         <fo:block text-align="justify">de PERAL, MATTHEW</fo:block>
         <fo:block font-size="18pt" font-weight="bold" space-after="10pt"
                   space-before="14pt" border-after-style="solid" id="w0aab4">
                   Dades tècniques</fo:block>
         <fo:block>
            <fo:block space-after="10pt">Les dades tècniques d'aquest llibre
                                         son:</fo:block>
            <fo:table table-layout="fixed">
               <fo:table-column column-width="80mm"/>
               <fo:table-column column-width="80mm"/>
               <fo:table-body>
                  <fo:table-row>
                     <fo:table-cell border="solid #000000 1px">
                        <fo:block>ISBN:</fo:block>
                     </fo:table-cell>
                     <fo:table-cell border="solid #000000 1px">
                        <fo:block>84-322-9632-5</fo:block>
                     </fo:table-cell>
                  </fo:table-row>
                  <fo:table-row>
                     <fo:table-cell border="solid #000000 1px">
                        <fo:block>Editorial:</fo:block>
                     </fo:table-cell>
                     <fo:table-cell border="solid #000000 1px">
                        <fo:block>SEIX BARRAL, S.A.</fo:block>
                     </fo:table-cell>
                  </fo:table-row>
                  <fo:table-row>
                     <fo:table-cell border="solid #000000 1px">
                        <fo:block>Llengua:</fo:block>
                     </fo:table-cell>
                     <fo:table-cell border="solid #000000 1px">
                        <fo:block>Castellana</fo:block>
                     </fo:table-cell>
                  </fo:table-row>
                  <fo:table-row>
                     <fo:table-cell border="solid #000000 1px">
                        <fo:block>Pàgines:</fo:block>
                     </fo:table-cell>
                     <fo:table-cell border="solid #000000 1px">
                        <fo:block>472</fo:block>
                     </fo:table-cell>
                  </fo:table-row>
               </fo:table-body>
            </fo:table>
            <fo:block space-after="10pt"/>
         </fo:block>
         <fo:block font-size="18pt" font-weight="bold" space-after="10pt"
                   space-before="14pt" border-after-style="solid" id="w0aab5">
                   Resum del llibre</fo:block>
```

```
          <fo:block>
             <fo:block text-align="justify">Una novel·la històrica de suspens comparada
                      insistentment amb El Código da Vinci y El Nombre de la Rosa de
                      Umberto Eco.</fo:block>
             <fo:block text-align="justify">Boston 1865. Importants personalitats estan
                      sent brutalment assassinades per un criminal inspirat en els
                      torments de l'Infern de Dante. Sols els membres del Club Dante,
                      poetes i professors de Harward dirigits per Henry Wadsworth
                      Longfelowww, poden anticipar-se a l'assassí i identificar-lo.
                      Mentre preparen la traducció americana de la Divina Comedia
                      enfrontant-se a l'oposició de la puritana vella guàrdia de
                      Harward, els intel·lectuals hauran de convertir-se en detectius i
                      passar a l'acció</fo:block>
          </fo:block>
       </fo:block>
    </fo:flow>
 </fo:page-sequence>
</fo:root>
```

Exemple 5.4: Document XSL-FO resultant de la transformació de l'exemple 5.3.

Com podem observar, es tracta d'un document, en format XSL-FO, que a part de contenir les dades i l'estructura, que ja contenia a l'anterior document XML, conté també el seu format: tipus de lletra, justificació dels paràgrafs, mida de la fulla, etc. Estem ja preparats per fer l'últim pas.

5.2.3 Transformació XSL-FO a PDF

En el document XML que hem obtingut en el punt 5.2.1 hem aplicant el nou procés de transformació descrit en el punt 4.3.4, primer es transforma a XSL-FO (amb JAXP) i després es transforma a PDF (amb el FOP). El resultat és el que mostra la figura següent:

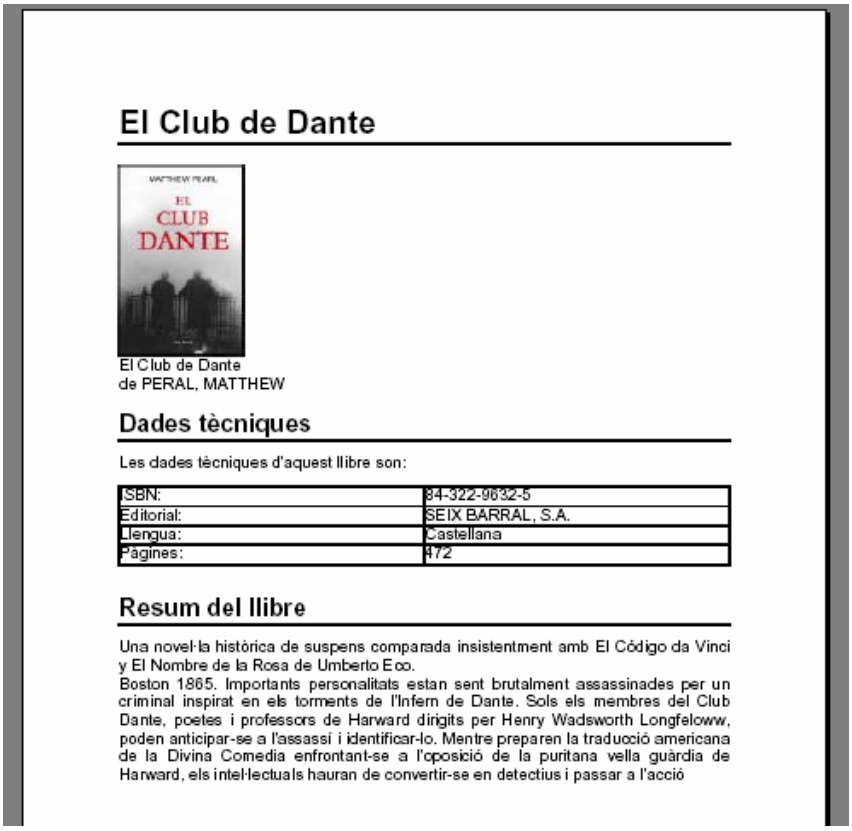

Figura 5.1: Document resultant de la transformació de l'exemple 5.4.

Com podem observar, el document mostra gràficament el que inicialment havíem dissenyat mitjançant unes taules d'elements. La resta, fins arribar a aquest resultat, se n'encarrega la nostra eina d'una forma genèrica.

Ja tenim l'eina construïda, ara és feina dels usuaris dissenyar els seus informes.

5.3 Més informes

Arribat aquest punt, hi ha preguntes obligades: que he de fer per implementar nous informes?, i si en el futur canvia el model de dades de BiblioGes, caldrà modificar l'eina de generació d'informes?

Les respostes són molt simples. Com hem vist, l'eina és molt genèrica i no depén de com és el model de dades sobre el que treballa. Si un dia es modifica el model de dades de BiblioGes, tant sols caldrà modificar les funcions que hem implementat, dins el propi model de BiblioGes, i que són les que obtenen les dades. La resta no es veu afectada.

Si es vol implementar més informes, tant sols cal definir una nova plantilla, com hem vist al punt 5.2.1, construïnt elements i la seva estructura mitjançant la seva inserció a les taules corresponents i definint, si és el cas, noves funcions per obtenir la informació dels meta-dades. La resta, com en el cas anterior no es veu afectada.

6 Conclusions

Davant del repte tecnològic existent en els sistemes d'extracció d'informació de les aplicacions de nova generació, en que per una banda, s'ha de garantir la seva independència dels esquemes de les bases de dades dels quals s'alimenten, i per l'altra, ser capaços de mostrar la informació en múltiples formats, hem realitzat un estudi dels estàndards de la família XML. En concret, hem estudiat els estàndrads referents a la transformació i presentació de la informació (CSS, XSL, XSLT i XSL-FO) i els referents a la creació i manipulació de documents XML (SAX i DOM). Aquest estudi ens ha donat una visió global de l'estat actual d'aquesta tecnologia, i de quines característiques en podem treure per afrontar aquest repte.

L'XML (eXtensible Markup Language) és el llenguatge que ens permet l'intercanvi de dades i informació de forma estàndard i totalment independent de les plataformes i aplicacions utilitzades. Donada aquesta flexibilitat i independència de les fonts de dades que proporciona, aquest és, avui en dia, el llenguatge més òptim per abordar el nostre projecte.

Però, a més, a part del llenguatge, ens cal altres mecanismes que ens ajudin a la construcció d'una eina genèrica, independent de les bases de dades i altament parametritzable per part de l'usuari. L'estudi realitzat dels principals processadors XSLT (Saxon, Xalan i JAXP) i processadors XSL-FO (FOP, XEP i XSLFormatter) que hi ha al mercat, així com de les diferents tecnologies per generar documents XML a partir de consultes realitzades en una base de dades, ens hi ha ajudat. L'ús d'aquestes eines existents han simplificat la nostra tasca.

Ens hem centrat en el disseny i construcció d'un procés d'extracció d'informació genèric i independent de la base de dades, on l'estructura de la informació s'emmagatzema en la pròpia base de dades mitjançant taules que implementen plantilles. Així, la definició d'una nova plantilla es realitza mitjançant insercions a aquestes taules, sense que la resta de l'eina s'hagi de modificar.

En resum, l'ús de la tecnologia XML i d'unes eines existents, els processadors XSLT i XSL-FO, juntament amb el disseny i construcció d'un procediment d'extracció genèric d'informació d'una base de dades, ens ha portat a aconseguir els nostres objectius.

Disposem d'una eina per incloure a qualsevol aplicació, que li facilita la funcionalitat de generar informes amb independència de plataforma i elecció del format de sortida. A més, el fet de canviar d'aplicació o de model de dades no implica canvis en l'eina, així com també permet que cada aplicació o fins i tot cada usuari, es pugui personalitzar l'informe a les seves necessitats.

Finalment hem mostrat, mitjançant l'exemple de creació d'un informe que mostra les dades d'un llibre, la simplicitat i àmplia utilitat que la nostra eina pot oferir a una aplicació.

Degut a l'abast del projecte i la limitació en el temps, no ens ha estat possible, tot i que hagués resultat molt interessant, implementar nous elements en la nostra eina. Així doncs, totes aquestes millores o ampliacions que ens hagués agradat implementar, ho proposem a continuació com a futures línies de treball.

6.1 Futures línies de treball

Els punts que proposem continuar desenvolupant en el futur son les següents:

Tractament d'elements de text en format RTF.

L'eina contempla l'ús de texts fixos o amb meta-dades dins les plantilles emmagatzemades, però en ambdós casos el text es visualitza amb format normal. És a dir, tot el text té el mateix format sense possibilitat de definir-hi cap tipus de lletra, negreta, cursiva, etc.
Seria adient, que l'eina contemplés la possibilitat de que els elements de text poguessin estar definits en format RTF. Això donaria molta més potència de formateig pels documents. També caldria trobar un processador XSL-FO, que fos capaç d'interpretar aquest format.

Inclusió de nous elements en els documents per millorar el seu aspecte.

L'eina té la possibilitat d'incloure dins d'una plantilla capítols, subcapítols, texts, imatges i taules d'informació. En principi són els elements més necessaris, però altres elements que també considerem interessants poden ser llistes d'elements, peus de pàgines, capçaleres, taules de contingut, etc. Caldria afegir el seu tractament dins de l'eina.

Creació d'una aplicació per construir les plantilles.

Hem construït una eina que genera documents PDF a partir de la informació emmagatzemada dins d'una base de dades, i en concret en base d'unes taules que defineixen l'estructura de la plantilla. Però aquestes taules únicament es llegeixen. Caldria una aplicació que permeti a l'usuari poder definir les plantilles d'una forma àgil i còmoda sense necessitat d'anar inserint els elements en les diferents taules.

7 Glossari

API: Interfície de programació per al desenvolupament d'aplicacions.

classe: En JAVA, una classe és un model o prototipus que defineix les variables i el mètodes comuns a tots els objectes de la mateixa classe.

CSS: *Cascading Style Sheets,* llenguatge per modificar la visualització de les planes web. També s'utilitza per especificar la presentació dels documents XML.

document XML: Document format únicament per text pla i que conté informació estructurada i delimitada mitjançant marques.

DOM: *Document Object Model*, interfície de programació orientada a objectes que representa els documents XML mitjançant un conjunt d'objectes en forma d'arbre.

DTD: *Document Type Definitions*, document que defineix la gramàtica dels documents XML.

esquema XML: *XML Schema*, document en sintaxi XML que defineix l'estructura, la gramàtica, el contingut i la semàntica dels document XML de forma més estricta i robusta que els DTD.

event: Fets que s'anucien quan es produeixen, per exemple, el fet de trobar l'etiqueta d'inici d'un element.

full d'estil: document que conté la forma en que s'han de mostrar els elements d'un document XML.

HTML: *Hypertext Markup Language*, llenguatge de marques per a la creació de pàgines web.

interfície: Plataforma de programació que proporciona un conjunt de classes i mètodes per facilitar les tasques de desenvolupament.

JAVA: Llenguatge de programació independent de plataforma creat per SUN Microsystems.

JDK: *Java Development Kit, o SDK Software Development Kit*, conjunt d'eines per desenvolupar en JAVA.

J2EE: *Java 2 Platform, Enterprise Edition*, estàndard que defineix la forma de desenvolupar aplicacions en múltiples capes, generalment en tres.

meta-dada: Etiqueta lògica que representa un valor.

.NET: Plataforma de desenvolupament d'aplicacions web de Microsoft.

ORACLE XDK: *XML Developers Kit*, conjunt d'eines proporcionades per ORACLE per al processament d'XML tant dins com fóra de la base de dades.

PDF: *Portable Document Format*, format de document estàndard.

plantilla: document que conté un esquelet de document que es genera a partir d'aquesta.

PL/SQL: llenguatge de programació utilitzat dins les bases de dades d'ORACLE.

Processador XSLT: Eina que a partir d'un document XML i una plantilla XSLT obté un nou document fruit de la transformació del document XML seguint les pautes de la plantilla XSLT.

Processador XSL-FO: Eina que a partir d'un document XML-FO obté un nou document amb format estàndard, generalment en PDF.

SAX: *Simple API for XML*, interfície de programació simple basada en events i fàcil d'utilitzar per analitzar els documents XML.

script: Conjunt d'instruccions en un llenguatge determinat que realitzen una acció.

SQL: Llenguatge de consulta de les base de dades relacionals.

SQL dinàmic: Capacitat d'ORACLE per generar i executar sentencies SQL en temps d'execució.

W3C: *World Wide Web Consortium,* organisme creat amb l'objectiu de desenvolupar protocols (especificacions, guies d'estil, software, etc.).

XML: *Extensible Markup Language*, llenguatge extensible de marques que defineix el format estàndard per a l'estructuració de dades i informació.

XPath: Llenguatge que permet adreçar parts d'un document XML, seguint l'estructura lògica (arbre de nodes) que descriu el contingut del document.

XSL: *Extensible Stylesheet Language*, llenguatge que aplica format als documents XML indicant com s'han de visualitzar i donant certa capacitat de transformació del contingut dels documents.

XSLT: *Extensible Stylesheet Language Tranformation*, llenguatge que permet transformar un document XML en un altre document XML.

XSLT-FO: *XML Stylesheet Language Formatting Objects,* llenguatge de formateig que descriu de forma precisa com s'ha de visualitzar el contingut dels documents XML

XQuery: Llenguatge de consulta que basant-se en l'estructura dels documents XML, proporciona els mecanismes necessaris per poder interrogar, recuperar i interpretar la informació.

8 Bibliografia

Llibres

Abraham Gutiérrez, Raúl Martínez . "**XML a través de ejemplos**". Editorial Ra-Ma. 2001.

David Flanagan. "**JAVA IN A NUTSHELL: A Desktop Quick Reference**". O'REILLY. 2002.

Elliotte Rusty Harold, W. Scott Means. "**XML IN A NUTSHELL: A Desktop Quick Reference**". O'REILLY. 2001.

Kevin Loney, George Kock. "**ORACLE 8i. The Complet Reference**". Osborne / McGraw-Hill. 2000.

Links

Antenna House. "**How to Develop Stylesheets for XML to XSL-FO Transformation**".
http://www.antennahouse.com/XSLsample/howtoRC/Howtodevelop-en.pdf

Antenna House. "**XSL Formatter V3.1**".
http://www.antennahouse.com/

Antenna House."**XSL-FO Tutorial and Samples**".
http://www.antennahouse.com/XSLsample/XSLsample.htm

Apache XML Project. "**FOP (Formatting Objects Processor) 0.20.5**".
http://xml.apache.org/fop/

Apache XML Project. "**Xalan-Java version 2.6.0**".
http://xml.apache.org/xalan-j/

David Megginson. "**SAX (Simple API for XML)**".
http://www.saxproject.org/

Joaquin Bravo. "**Convertir XML en PDF utilizando XSL-FO y FOP**".
http://www.programacion.net/articulo/joa_pdf/

Joaquin Bravo. "**Tutorial de XML en castellano**". 1999
http://html.programacion.net/xml/index.htm

ORACLE. "**ORACLE XML Developer's Kit** (XDK)".
http://otn.oracle.com/tech/xml/xdkhome.html

ORACLE. "**XML Technology Center**".
http://otn.oracle.com/tech/xml/index.html

SouceForge."**SAXON, The XSLT and XQuery Processor**".
http://saxon.sourceforge.net/

SUN. "**Comparación de las Tecnologías Java para XML**".
http://programacion.com/java/tutorial/javaxml/

SUN. "**Java API for XML Processing (JAXP)**". 2003.
http://java.sun.com/xml/jaxp/index.jsp/

RenderX. "**XEP 3.7.8**".
http://www.renderx.com/

RenderX. "**XSL-FO Tutorial**".
http://www.renderx.com/tutorial.html

W3C. "**Cascading Style Sheets (CSS)**".
http://www.w3.org/Style/CSS/

W3C. "**Document Object Model (DOM)**".
http://www.w3.org/DOM/

W3C. "**Extensible Markup Language (XML)**".
http://www.w3.org/XML/

W3C. "**Extensible Stylesheet Language (XSL) Version 1.0**". W3C Recommendation 15 October 2001.
http://www.w3.org/TR/xsl/

W3C. "**XSL Transformations (XSLT) Version 1.0**". W3C Recommendation 16 November 1999.
http://www.w3.org/TR/xslt/

ANNEX 1: Codi font de l'eina

```
import org.w3c.dom.*;
import org.apache.xerces.dom.DocumentImpl;
import java.io.*;
import java.sql.*;
import java.lang.*;
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.sax.*;
import oracle.jdbc.*;
import oracle.xml.parser.v2.*;
import oracle.xml.sql.query.*;

import org.apache.fop.apps.Driver;
import org.apache.fop.apps.FOPException;
import org.apache.fop.messaging.MessageHandler;

import org.apache.avalon.framework.logger.ConsoleLogger;
import org.apache.avalon.framework.logger.Logger;

import ru.novosoft.dc.rtf2fo.*;
import ru.novosoft.dc.core.*;

public class sql2pdf
{
  static Connection conn;
  static int plantilla=2;
  static String clau="84-322-9632-5";

  public static void main(String args[]) throws Exception
  {
   //Creeem la connexió a la BBDD
    DriverManager.registerDriver(new oracle.jdbc.driver.OracleDriver());
    conn = DriverManager.getConnection("jdbc:oracle:thin:@172.20.20.1:1521:mu_ocat",
                                       "bdpfc","bdpfc");

    String consulta="select elements,tipus from elements where plantilla="+plantilla+
                    " and capitol is null order by ordre";

    Statement stmt = conn.createStatement();

    Document document= new DocumentImpl();
    Document docfill= new DocumentImpl();
    Element root = document.createElement("DOCUMENT");
    ResultSet rs = stmt.executeQuery(consulta);
    while (rs.next())
    {
     int n = rs.getInt("ELEMENTS");
     String s = rs.getString("TIPUS");
     docfill=tractarElement(s.charAt(0),n,0);
      root.appendChild(document.importNode(docfill.getDocumentElement(),true));
    }

    document.appendChild( root);
    DOMSource source = new DOMSource(document);

    //Tanquem la connexió a la BBDD
    conn.close();
```

```
  Driver driver = new Driver();

  //Setup logger
  Logger logger = new ConsoleLogger(ConsoleLogger.LEVEL_INFO);
  driver.setLogger(logger);
  MessageHandler.setScreenLogger(logger);

  //Setup Renderer (output format)
  driver.setRenderer(Driver.RENDER_PDF);

  //Setup output
  OutputStream out = new java.io.FileOutputStream("Sortida.pdf");
  try
  {
    driver.setOutputStream(out);

    // Fem la transformació a XSL-FO utilitzant la pantilla.xsl
    // Obtenim la instancia per la traduccio.
    TransformerFactory tFactory = TransformerFactory.newInstance();

    // Associem el xls a la traduccio.
    Source xslSource = new StreamSource(new File("plantilla_fo.xsl"));

    // Generem la traduccio.
    Transformer transformer = tFactory.newTransformer(xslSource);
    transformer.setOutputProperty("encoding","ISO-8859-15");
    transformer.setOutputProperty("method","xml");
    Result res = new SAXResult(driver.getContentHandler());

    // Realitzem la traduccio i enviem el resultat a la sortida estàndard.
    transformer.transform(source,res);
  }
  finally
  {
      out.close();
  }
}

private static Document tractarElement(char tipus, int codi, int nivell)
{
  Document doc= new DocumentImpl();
 OracleXMLQuery qry;

  String sql;
 switch (tipus) {

   //Tipus Text
    case 'T': {
       sql = "select tractarTexte(texte,'"+clau+"') texte from texte where codi="
             +codi;
        qry = new OracleXMLQuery(conn, sql);
        qry.setRowsetTag("P_TEXTE");
        qry.setRowTag("R_TEXTE");
        doc=qry.getXMLDOM();
     break;
    }

    //Tipus Imatge
    case 'I': {
        try {
         PreparedStatement ps=conn.prepareStatement(
                           "select nom,imatge from imatge where codi="+codi);
          ResultSet rs = ps.executeQuery();
          int c;
        if(rs.next()){
           File fitxer= new File(rs.getString("NOM"));
```

```
            FileOutputStream out = new FileOutputStream(fitxer);
            InputStream in = rs.getBinaryStream("IMATGE");
            while ((c = in.read()) != -1)
              out.write(c);
              out.close();
            }
        sql = "select nom from imatge where codi="+codi;
          qry = new OracleXMLQuery(conn, sql);
          qry.setRowsetTag("P_IMATGE");
          qry.setRowTag("R_IMATGE");
          doc=qry.getXMLDOM();
        }
        catch (Exception e) {
        System.err.println("error: - "+e.getMessage());
       }
      break;
     }

     //Tipus Capitol
     case 'C': {
         Document doc2 = new DocumentImpl();
         Document doc3= new DocumentImpl();

         sql = "select nom from capitol where codi="+codi;
         qry = new OracleXMLQuery(conn, sql);
         qry.setRowsetTag("P_CAPITOL");
         qry.setRowTag("R_CAPITOL");
         doc2=qry.getXMLDOM();
         Element capitol = doc.createElement("CAPITOL"+nivell);
         capitol.appendChild(doc.importNode(doc2.getDocumentElement(),true));
         String consulta="select elements, tipus from elements where plantilla="
                         +plantilla+" and capitol="+codi+
                         " order by ordre";
         try{
         Statement stmt = conn.createStatement();
         ResultSet rs = stmt.executeQuery(consulta);
           while (rs.next()) {
             int n = rs.getInt("ELEMENTS");
             String s = rs.getString("TIPUS");
             doc3=tractarElement(s.charAt(0),n,nivell+1);
             capitol.appendChild(doc.importNode(doc3.getDocumentElement(),true));
           }
           doc.appendChild(capitol);
         }
         catch (Exception e) {
           System.err.println("error: - "+e.getMessage());
       }
         break;
    }

    //Tipus Taula informació
     case 'L': {
      Document doc2 = new DocumentImpl();
        Document doc3= new DocumentImpl();
        Document doc4= new DocumentImpl();
       try{
         String consulta="select nom,columnes from taula where codi="+codi;
           Statement stmt = conn.createStatement();
        ResultSet rs = stmt.executeQuery(consulta);

           Element taula = doc.createElement("TAULA");
         String str=null;;
           int i,cols=0;
           if(rs.next()){
             taula.setAttribute("NOM",rs.getString("NOM"));
             cols=rs.getInt("COLUMNES");
```

```
                str="\n";
                for (i=0;i<cols;i++)
                {
                    str=str+"<fo:table-column column-width=\""+160/cols+"mm\"/>\n";
                  }
                taula.setAttribute("COLS",str);

                consulta="select codi from fila where taula="+codi+
                        " order by codi";

                rs = stmt.executeQuery(consulta);
                i=0;
                while (rs.next()) {
                    i++;
                    int n = rs.getInt("CODI");
                    sql = "select texte from columna where taula="+codi+
                          " and fila="+n+" order by codi";
                    qry = new OracleXMLQuery(conn, sql);
                    qry.setRowsetTag("P_FILA");
                    qry.setRowTag("R_COLUMNA");
                    doc2=qry.getXMLDOM();
                   taula.appendChild(doc.importNode(doc2.getDocumentElement(),true));
                }
            }
            doc.appendChild(taula);
             }
             catch (Exception e) {
               System.err.println("error: - "+e.getMessage());
           }
             break;
        }
      }
      return doc;
    }

}
```

ANNEX 2: Plantilla XSLT

```
<?xml version="1.0" encoding="ISO-8859-15" ?>
<xsl:stylesheet version="1.0" xmlns:fo="http://www.w3.org/1999/XSL/Format"
                      xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="xml" version="1.0" indent="yes" encoding="ISO-8859-15"/>

<xsl:template match="DOCUMENT">
  <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format">
    <fo:layout-master-set>
      <fo:simple-page-master page-height="297mm" page-width="210mm"
          margin="5mm 25mm 5mm 25mm" master-name="PageMaster">
        <fo:region-body margin="20mm 0mm 20mm 0mm"/>
      </fo:simple-page-master>
    </fo:layout-master-set>
    <fo:page-sequence master-reference="PageMaster">
      <fo:flow flow-name="xsl-region-body">
        <xsl:apply-templates />
      </fo:flow>
    </fo:page-sequence>
  </fo:root>
</xsl:template>

<xsl:template match="CAPITOL0">
  <fo:block font-size="24pt"
            font-weight="bold"
            space-after="14pt"
            break-before="page"
            keep-with-next.within-page="always"
            border-after-style="solid"
            border-after-width="2pt"
            id="{generate-id()}">
    <xsl:value-of select="P_CAPITOL/R_CAPITOL/NOM" />
  </fo:block>
  <fo:block>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="CAPITOL1">
  <fo:block font-size="18pt"
            font-weight="bold"
            space-after="10pt"
            space-before="14pt"
            border-after-style="solid"
            id="{generate-id()}">
    <xsl:value-of select="P_CAPITOL/R_CAPITOL/NOM" />
  </fo:block>
  <fo:block>
    <xsl:apply-templates />
  </fo:block>
</xsl:template>

<xsl:template match="NOM"> </xsl:template>

<xsl:template match="P_TEXTE/R_TEXTE">
  <fo:block text-align="justify">
    <xsl:value-of select="TEXTE" />
  </fo:block>
</xsl:template>
```

```
<xsl:template match="P_IMATGE/R_IMATGE">
  <xsl:variable name="fitxer"> <xsl:value-of select="NOM" /> </xsl:variable>
  <fo:block>
    <fo:external-graphic src="{$fitxer}">
    </fo:external-graphic>
  </fo:block>
</xsl:template>

<xsl:template match="TAULA">
  <fo:block space-after="10pt">
    <xsl:value-of select="@NOM" />
  </fo:block>
  <fo:table table-layout="fixed">
    <xsl:value-of select="@COLS" />
    <fo:table-body>
      <xsl:apply-templates select="P_FILA" />
    </fo:table-body>
  </fo:table>
  <fo:block space-after="10pt"/>
</xsl:template>

<xsl:template match="P_FILA">
  <fo:table-row>
    <xsl:apply-templates select="R_COLUMNA" />
  </fo:table-row>
</xsl:template>

<xsl:template match="R_COLUMNA">
  <fo:table-cell border="solid #000000 1px">
    <fo:block>
      <xsl:value-of select="TEXTE" />
    </fo:block>
  </fo:table-cell>
</xsl:template>

</xsl:stylesheet>
```