généraliste comme Java ou C++. C'est vrai, mais cette affirmation doit être largement nuancée dans la pratique. En effet, un des problèmes de XSL en général et de XSLT en particulier, est que ce sont des langages assez déstabilisants, dans la mesure où ils demandent aux programmeurs des compétences dans des domaines qui sont généralement assez peu fréquentés. Un expert XSLT mettra très certainement beaucoup moins de temps à réaliser le traitement demandé que l'expert Java utilisant les API SAX (Simple API for XML) ou DOM (Document Object Model) pour faire le même travail ; le seul problème, à vrai dire, est d'être expert XSLT.

A titre de comparaison, lorsque le langage Java a été rendu public et disponible, un expert C++ ou Smalltalk (ou langage à objets, d'une façon plus générale) à qui l'on présentait ce langage, en faisait le tour en une semaine, le temps de s'habituer aux caractéristiques spécifiques ; mais son mode de pensée restait fondamentalement intact.

A l'inverse, prenez un expert SQL, ou Java, ou C, ou Visual Basic, et présentez-lui XSLT : il n'aura pratiquement rien à quoi se raccrocher ; tout pour lui sera nouveau ou presque, sauf s'il a une bonne culture dans le domaine des langages fonctionnels ou déclaratifs (Prolog, Lisp, Caml, etc.).

Pourtant, il ne faut pas croire que XSLT soit un langage spécialement difficile ou complexe ; il est même beaucoup moins complexe que C++. Mais il est déroutant, parce qu'il nous fait pénétrer dans un monde auquel nous ne sommes en général pas habitués.

En résumé, si vous avez à faire un traitement non trivial sur un document XML, et que vous êtes novice en XSLT, prévoyez une phase importante d'investissement personnel initial dans le calcul de votre délai, ou réalisez votre traitement en programmation traditionnelle, avec des API comme SAX ou DOM. Mais ce n'est pas une solution rentable à moyen terme, car il est certain qu'une fois la compétence acquise, on est beaucoup plus efficace en XSLT qu'on peut l'être en Java ou C++ pour réaliser une transformation donnée.

Le langage XSL

Statut actuel

Le W3C n'est pas un organisme de normalisation officiel ; il ne peut donc prétendre éditer des normes ou des standards. C'est pourquoi un document final du W3C est appelé *Recommendation*, terme qui peut sembler un peu bizarre au premier abord. Mais une *Recommendation* du W3C n'a rien à voir avec un recueil de conseils, et c'est un document qui équivaut de fait à un standard, rédigé dans le style classique des spécifications. Dans toute la suite, nous parlerons néanmoins de standards XSL ou XSLT, même si ce ne sont pas des standards au sens officiel du terme.

Le langage XSL (eXtensible Stylesheet Language) est un langage dont la spécification est divisée en deux parties :

- XSLT (XSL Transformation) est un langage de type XML qui sert à décrire les transformations d'un arbre XML en un autre arbre XML. Le standard XSLT 1.0 est une

W3C Recommendation du 16 novembre 1999 (voir *www.w3.org/TR/xslt*). A noter que les attributs de certains éléments XML de ce langage contiennent des chaînes de caractères dont la syntaxe et la sémantique obéissent à un autre langage, nommé XPath (XML Path Language), qui n'a rien à voir avec XML, et dont le but est de décrire des ensembles de nœuds de l'arbre XML du document source à traiter. Le langage XPath 1.0 a fait l'objet d'une *W3C Recommendation* du 16 novembre 1999 (voir *http://www.w3c.org/TR/xpath*).

- XSLFO (XSL Formating Objects) est un langage de type XML utilisé pour la description de pages imprimables en haute qualité typographique. Le standard XSLFO 1.0 est une *W3C Recommendation* du 15 octobre 2001 (voir *http://www.w3c.org/TR/XSL*).

Ces deux parties s'intègrent dans une chaîne de production résumée à la figure 1-1.

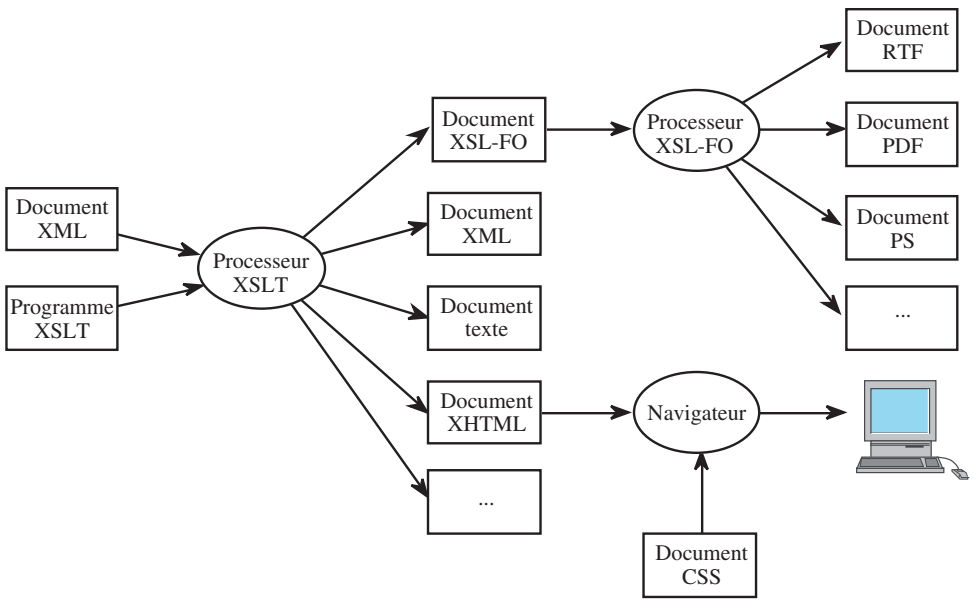


Figure 1-1

Les deux composantes de XSL.

XSLT peut être utilisé seul, et d'ailleurs, pendant les deux ans qui ont séparés la publication des standards respectifs de XSLT et XSLFO, on ne pouvait guère faire autrement que d'utiliser XSLT seul.

Inversement, il ne semble pas très pertinent et encore moins facile d'utiliser XSLFO seul : la description de pages imprimables réclame des compétences qui relèvent plus de celles d'un typographe. A défaut de les posséder, il semble plus prudent de laisser la génération de documents XSLFO à un processeur XSLT équipé de règles de traduction

adéquates et toutes faites, ou tout au moins d'une bibliothèque de règles de plus haut niveau que celles qu'on pourrait écrire en partant de zéro sur la compréhension des objets manipulés par XSLFO. Une autre possibilité pourrait être d'employer un logiciel de transformation, qui génère du FO à partir de documents RTF ou Latex, par exemple, mais on entre ici dans un domaine qui n'a plus rien à voir avec l'objet de ce livre.

Comme on peut le voir sur la figure 1-1, il est possible, sous certaines conditions, d'obtenir du texte, du HTML ou du XML en sortie de XSLT. En effet, bien que XSLT soit toujours présenté comme un langage de transformation d'arbres XML (l'arbre XML du document source est transformé en un autre arbre XML résultat), un processeur XSLT est équipé d'un sérialisateur qui permet d'obtenir une représentation de l'arbre résultat sous la forme d'une suite de caractères. Le processus de sérialisation est paramétrable pour obtenir facilement du texte brut, du HTML ou du XML. Comme on peut obtenir du texte brut non balisé au format XML, on peut donc envisager de programmer des transformations qui aboutissent à des documents RTF ou Latex, voire directement au format Postscript ou PDF. Mais comme les processeurs XSLFO commencent à offrir un fonctionnement stable et utilisable, ce genre de prouesse sera probablement de moins en moins envisagée à l'avenir.

Une des transformations les plus courantes reste bien sûr la transformation XML vers XHTML ou HTML, mais il ne faut pas imaginer que le langage XSLT a été fait pour ça, même si l'allusion aux feuilles de style (stylesheet) dans le nom même d'XSL encourage à cette assimilation beaucoup trop réductrice : la production d'HTML n'est qu'une possibilité parmi d'autres.

Evolutions

XSL est un langage qui évolue sous la pression des utilisateurs, des implémenteurs de processeurs XSLT ou XSL-FO, et des grands éditeurs de logiciels. Le rôle du W3C est entre autres de canaliser ces évolutions, afin de prendre en compte les demandes les plus intéressantes, et de les élever au rang de standard (ou *W3C Recommendation*). Le 20 décembre 2001, le W3C a publié des *Working Drafts* annonçant des évolutions majeures de XSLT et de XPath : *W3C Working Draft XSLT 2.0* et *W3C Working Draft XPath 2.0*. Ces documents déboucheront à terme sur un standard XSLT 2.0 et un standard XPath 2.0. Pour l'instant beaucoup de choses restent en discussion, et d'ailleurs l'un des buts de ces publications est précisément de déclencher les débats et les réactions, dont les retombées peuvent éventuellement être extrêmement importantes dans les choix finalement retenus.

Avant le *W3C Working Draft XSLT 2.0*, il y a eu un *W3C Working Draft XSLT 1.1*, qui est resté et restera à jamais à l'état de Working Draft, car le groupe de travail chargé de la rédaction de la *Final Recommendation* s'est rendu compte que les évolutions proposées étaient vraiment majeures, et s'accordaient donc mal avec un simple passage de 1.0 à 1.1. Les travaux de la lignée 1.xx ont donc été abandonnés, et réintégrés à ceux de la lignée 2.xx.

Divers modes d'utilisation de XSLT

XSLT est un langage interprété ; à ce titre il réclame un interpréteur (souvent appelé processeur XSLT) qui peut être lancé de diverses façons, suivant l'objectif à atteindre.

Mode Commande

Le mode Commande est le mode qui consiste à lancer (à la main ou en tant que commande faisant partie d'un fichier de commandes) le processeur XSLT en lui passant les arguments qu'il attend (le fichier contenant la feuille de style XSLT à exécuter, le fichier XML à traiter, et le nom du fichier résultat, plus divers paramètres ou options).

Ce mode convient pour les traitements non interactifs, et c'est d'ailleurs celui qui offre le plus de souplesse en ce sens qu'il ne requiert aucune liaison particulière entre le fichier XML et la feuille de style de traitement : la même feuille de style peut traiter plusieurs fichiers XML différents, et un même fichier XML peut être traité par plusieurs feuilles de style XSLT différentes.

C'est aussi un mode qui n'impose aucun format de sortie particulier : on peut générer aussi bien du HTML que du XML, du Latex, etc.

Processeurs courants

En mode Commande, les processeurs les plus courants sont Xt, Xalan, et Saxon.

Xt est le premier processeur XSLT à être apparu ; il a été écrit par un grand maître de SGML et de DSSSL (l'équivalent de XSLT pour SGML), James Clark, qui est aussi l'éditeur de la norme XSLT 1.0 du W3C. Xt n'est pas tout à fait complet, et ne le sera jamais, car il n'y a plus aucun développement sur ce produit. Cependant, *Xt* reste aujourd'hui le plus rapide de tous les processeurs XSLT. C'est un produit « open source » gratuit (écrit en Java), que l'on peut obtenir sur *www.jclark.com/xml/xt.html*.

Saxon est le processeur XSLT écrit par Michael Kay, également auteur du premier manuel de référence sur le langage XSLT. C'est un produit libre et gratuit, dont les sources sont disponibles (en Java) sur *http://saxon.sourceforge.net/*.

Pour l'avoir utilisé, mon avis est qu'il est très rapide, bien documenté et très stable. De plus, il est complet, et même plus que complet, puisqu'il implémente les nouvelles fonctionnalités annoncées dans le W3C Working Draft XSLT 1.1, et même, à titre expérimental, celles annoncées dans le W3C Working Draft XSLT 2.0.

Une version empaquetée sous forme d'un exécutable Windows existe aussi (Instant Saxon), mais n'est pas aussi rapide à l'exécution que la version ordinaire. Elle est à conseiller uniquement à ceux qui veulent tester Saxon sur une plate-forme Windows sans avoir à installer une machine virtuelle Java.

Xalan est un processeur « open source » produit par Apache. C'est un produit libre et gratuit, dont les sources sont disponibles, que l'on peut l'obtenir sur *http://xml.apache.org/xalan-j/*.

Comme Saxon, c'est un produit très largement utilisé par la communauté d'utilisateurs, donc très fiable. Il est complet par rapport au standard XSLT 1.0.

Signalons pour finir que MSXSL, le processeur XSLT de Microsoft (fourni avec MSXML3 ou 4), est fait pour être lancé en mode Commande, bien que les transformations XSLT au travers du navigateur Internet Explorer soient plus populaires. Sur les plates-formes Windows, il est plus rapide que Saxon.

Il y a évidemment beaucoup d'autres processeurs XSLT : on pourra, si l'on veut, faire son marché sur certains sites qui les recensent, comme par exemple : *www.w3c.org/Style/XSL*, *www.xmlsoftware.com/xslt*, et *www.xml.com/*.

Mode Navigateur

Le mode navigateur est un mode qui ne fonctionne qu'avec un navigateur équipé d'un processeur XSLT (par exemple Internet Explorer 5 avec MSXML3 (ou 4), IE6 ou Netscape 6). A priori, ce mode de fonctionnement est typiquement fait pour que la transformation XSL aboutisse à du HTML : le serveur HTTP, en réponse à une requête, envoie un document XML et la feuille de style qui génère le code HTML adéquat. Le gain attendu est double (voire triple) :

- Les documents XML qui transitent sur le réseau sont généralement moins bavards que leur équivalent HTML : on gagne de la bande passante.
- Il arrive assez souvent que ce soit la même feuille de style qui puisse traiter plusieurs documents XML différents (en provenance du même serveur) ; dans ce cas les techniques de mémoire cache sur le navigateur économisent à nouveau de la bande passante.
- A plus long terme, si le contenu du Web s'enrichit progressivement de documents de plus en plus souvent XML et de moins en moins souvent HTML, les moteurs de recherche auront moins de mal à sélectionner de l'information plus pertinente, parce que XML est beaucoup plus tourné vers l'expression de la sémantique du contenu que ne l'est HTML.

N'oublions pas non plus que la transformation XSLT sur le poste client décroît de façon notable la charge du serveur en termes de CPU et de mémoire consommée : moins de bande passante réseau utilisée et moins de ressources consommées sur le serveur sont les gains théoriques que l'on peut attendre de ce mode de fonctionnement. Mais bien sûr, il faudra du temps pour que tout cela devienne courant : pour l'instant l'évolution vers le traitement local de pages XML ne fait que commencer.

Processeurs courants

En mode Navigateur, il n'y a guère que IE5 ou IE6 de Microsoft qui soient vraiment aboutis pour les transformations XSLT (bien que Netscape 6 soit depuis peu un concurrent dans ce domaine). A noter que IE5 nécessite une installation auxiliaire, celle de MSXML3 (ou 4), qui offre une bonne implémentation de XSLT 1.0. On peut obtenir MSXML3 ou MSXML4 sur *http://msdn.microsoft.com/xml*.

Netscape 6 est une alternative intéressante pour XSLT. Il peut y avoir encore quelques problèmes d'application de feuilles CSS au résultat obtenu, mais les évolutions sont rapides en ce moment : il faut consulter le site *www.mozilla.org/projects/xslt* pour avoir l'information la plus à jour.

Mode Serveur

Dans le mode serveur, le processeur XSLT est chargé en tant que thread (processus léger), et il peut être invoqué par une API Java adéquate (généralement l'API TrAX) pour générer des pages HTML (ou PDF) à la volée. Typiquement, le serveur HTTP reçoit une requête, et une servlet ou une page ASP va chercher une page XML (qui contient une référence à la feuille de style XSLT à charger). Cette page XML est transmise au thread XSLT qui va la transformer en HTML, en utilisant au passage des données annexes (par exemple le résultat d'une requête SQL). Une autre possibilité est que le résultat du traitement de la requête soit un ensemble d'objets (Java, par exemple) qui résument la réponse à envoyer au client. Cet ensemble d'objets Java peut alors servir de base à la construction d'un arbre DOM représentant un document XML virtuel qui sera lui même transmis au thread XSLT pour être transformé en HTML et renvoyé vers le client.

Ce genre de solution fonctionne très bien, mais a tendance à consommer des ressources sur le serveur. Il est certain que dans l'idéal, il vaudrait mieux demander au client de faire lui-même la transformation XSLT, mais ce n'est pas possible actuellement, à cause de la grande disparité des navigateurs vis-à-vis du support de XSLT 1.0

Même avec IE5, il faut installer le module MSXML3 pour que cela fonctionne, sinon le dialecte XSL intégré par défaut à IE5 est tellement éloigné de XSLT 1.0, qu'on peut dire que ce n'est pas le même langage.

Actuellement la seule solution viable est donc la transformation sur le serveur, parce que c'est la seule qui permet de s'affranchir de la diversité des navigateurs ; il n'y a que si l'on travaille dans un environnement intranet, où les postes clients sont configurés de façon centralisée, que l'on peut envisager de diffuser sur le réseau du XML à transformer à l'arrivée.

Processeurs courants

L'API TrAX (Transformation API for XML) est une API qui permet de lancer des transformations XSLT au travers d'appels de méthodes Java standard. JAXP (Java API for XML Processing), de Sun Microsystems, est une API complète pour tout ce qui est traitement XML, et comporte une partie « transformation » conforme aux spécifications de TrAX. Les processeurs dont il est question ci-dessous implémentent l'API TrAX.

En mode Serveur, on retrouve les deux processeurs Saxon et Xalan, qui peuvent tous les deux être appelés depuis une application Java, via des appels TrAX, et notamment depuis une servlet. Cocoon, produit par Apache, est un framework intégrant, grosso modo, un serveur HTTP, un moteur de servlets, et un processeur XSLT (Xalan).

Typologie des utilisations d'XSL

Il y a au moins trois grands domaines où le langage XSL peut intervenir :

- dans les applications Internet ;
- dans les applications documentaires ;
- dans les applications d'échanges d'informations entre systèmes hétérogènes et répartis.

La figure 1-2 montre une possible architecture d'application Internet, avec les divers endroits où XSL peut (éventuellement) intervenir. Actuellement, la tendance est plutôt d'utiliser XSL pour générer des pages HTML dynamiques, mais ce n'est qu'une tendance, et de nombreuses applications ont été construites sur des architectures utilisant XSLT de façon plus importante : il serait tout à fait possible d'imaginer une architecture dans laquelle les objets métier de l'application Internet sont connectés à des gisements de données répartis (par exemple via JDBC), cette connexion se faisant par l'intermédiaire d'objets DOM (Document Object Model) qui sont adaptés et transformés par un ou plusieurs processus XSLT.

Note

La connexion via JDBC à une base de données est une possibilité, mais ce n'est pas la seule, car il y a des extensions, notamment avec Xalan et Saxon, qui permettent à une feuille de style de se connecter directement à une base de données relationnelle.

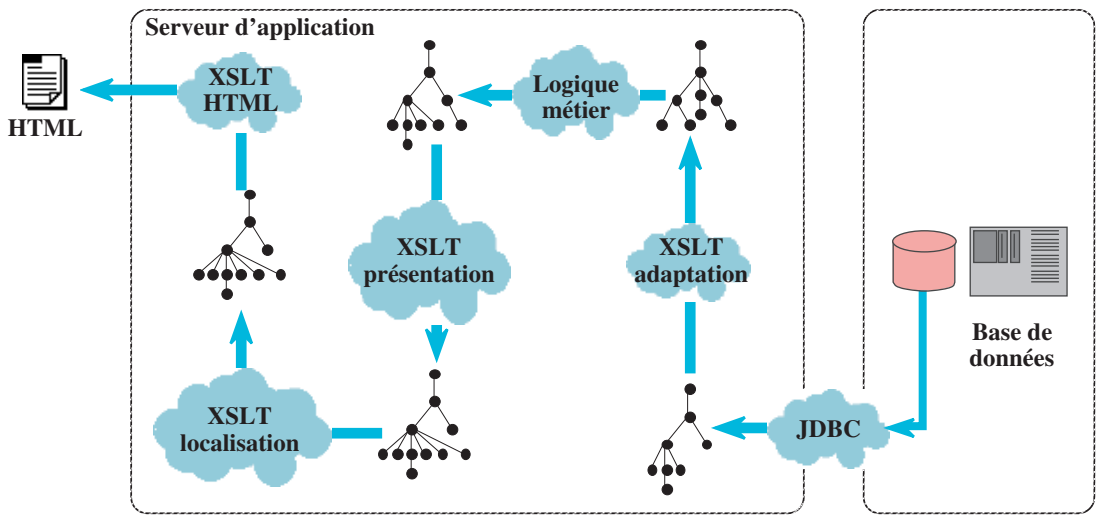


Figure 1-2

Divers emplois d'XSLT dans une application Internet.

Les applications documentaires recouvrent un vaste champ d'applications qui ne font pas forcément partie du domaine strictement informatique ; citons par exemple le domaine de la littérature et les sciences humaines, avec des projets comme la TEI (Text Encoding Initiative, *www.tei-c.org*) ou celui de la composition électronique (PDF, Postscript) de textes anciens ou modernes dans des langues peu communes (grec ancien, arabe ancien, sanscrit, copte, écriture hiéroglyphique, etc.) : voir par exemple *www.fluxus-virus.com/fr/what-critical.html* .

Dans un domaine plus familier aux informaticiens, XSL intervient dans les chaînes de production plus ou moins automatisée de documentations techniques ou commerciales et marketing ; un exemple simple étant la production d'un document de synthèse à partir de données au format XML extraites d'une base de données, qui sont ensuite mises en forme par XSLT et XSL-FO, et donnent au final un document PDF ou Postscript (voir la figure 1-1 , à la section *Le langage XSL*, page 2).

Il faut citer aussi DocBook (*http://docbook.sourceforge.net*), dont on reparlera d'ailleurs dans la suite de ce livre, qui est un système complet et gratuit pour rédiger de la documentation (rapports, articles, livres, documents structurés, etc.) au format XML. DocBook délivre des sorties en HTML (prévisualisation) ou en FO, que l'on transforme ensuite en PDF, par exemple (cette dernière transformation se faisant grâce à un processeur FO comme FOP, XEP ou RenderX).

Enfin, le domaine de l'échange de données entre systèmes hétérogènes et répartis concerne plusieurs courants d'activité, notamment celui de l'EDI (Electronic Data Interchange, ou échange de données informatisées), de l'EAI (Enterprise Application Integration), et des Web Services (*www.w3.org/TR/wsdl*, *www.w3.org/TR/soap12-part1*, *www.xmlbus.com*).

Dans tous les cas, de l'information est échangée au format XML, et des transformations XSLT permettent d'adapter les données propres à un système pour les conformer à une syntaxe (DTD, XML schemas, etc.) à laquelle adhère une certaine communauté d'utilisateurs.

Un avant-goût d'XSLT

Donner un aperçu du langage XSLT n'est pas du tout évident. Pour un langage comme C ou Java, c'est assez facile de donner quelques exemples donnant une première impression, parce que ce sont des langages qui ressemblent à d'autres : on peut donc procéder par comparaison en montrant par exemple un programme Visual Basic et son équivalent en Java, ou un programme Pascal et son équivalent en C. Pour XSLT, il n'y a aucun langage un tant soit peu équivalent et à peu près bien connu qui permette d'établir une comparaison. Seuls des langages fonctionnels ou déclaratifs comme Prolog, Caml ou Lisp donnent une bonne base de départ qui permet de s'y retrouver, mais encore faut-il en avoir l'expérience...

Si on ne l'a pas, il n'est guère possible de se fier à son intuition pour deviner l'effet d'un programme, notamment au tout début de la prise de contact avec ce langage. Il y a néanmoins un angle d'attaque, qui ne donne pas un panorama complet du langage, loin s'en

faut, mais qui permet au moins de voir certains aspects, et de comprendre dans quelles directions il faut partir pour comprendre ce qu'il y a à comprendre dans ce langage.

Nous allons donc commencer par voir ce qu'on appelle une feuille de style simplifiée (Simplified Stylesheet) ou Litteral Result Element As Stylesheet, comme l'appelle fort doctement la spécification XSLT 1.0.

L'avantage est qu'une feuille de style simplifiée utilise une version tellement bridée du langage XSLT que l'on peut assez facilement deviner ce que fait une telle feuille de style sans que des explications très détaillées soient forcément nécessaires.

Mais l'inconvénient est que l'on ne peut pas faire grand chose de plus que ce qu'on va montrer dans cet aperçu.

Remarque

Une feuille de style simplifiée n'est jamais nécessaire : il n'y a jamais rien ici que l'on ne puisse faire avec une vraie feuille de style.

Une feuille de style simplifiée peut rendre service aux auteurs de pages HTML possédant peu de compétences en programmation, dans la mesure où cela leur permet d'écrire assez facilement des pages dynamiques. La seule contrainte à respecter absolument, qui contredit peut être la pratique courante, est que le document HTML à écrire doit respecter la syntaxe XML, c'est-à-dire qu'il faut employer du XHTML : toute balise ouverte doit être refermée, et les valeurs d'attributs doivent être enfermées dans des apostrophes ou des guillemets.

Avant de passer à l'exemple proprement dit, insistons encore une fois sur le fait que cette notion de feuille de style simplifiée n'est présentée ici que pour vous donner une première idée du langage XSLT. Il n'y a aucune autre justification à son usage, et plus jamais nous n'en reparlerons dans la suite de ce livre. Autant dire qu'une feuille de style simplifiée n'a pas grand intérêt dans la pratique...

Exemple

Pour montrer comment générer une page dynamique avec une feuille de style simplifiée, nous allons prendre un exemple, et supposer que nous voulons obtenir une page telle que celle qui est montrée à la figure 1-3.

Cette page est produite à partir d'un fond qui comporte essentiellement de l'HTML XMLisé :

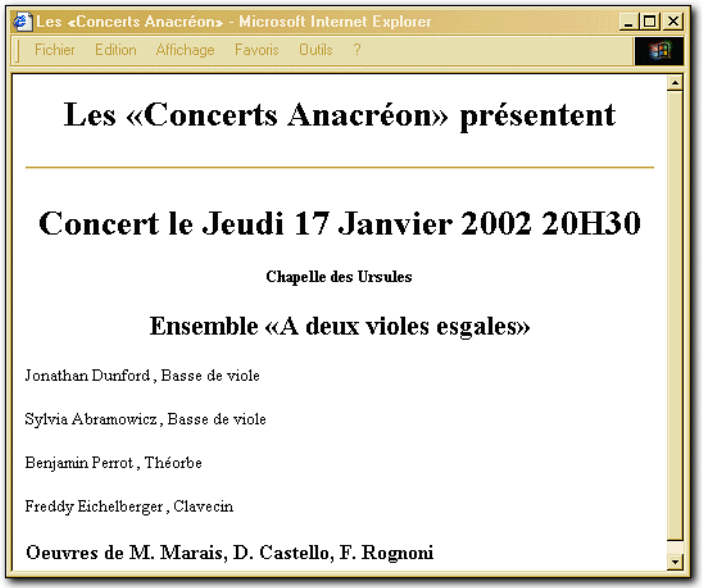
AnnonceConcert.xsl

```
<?xml version="1.0"?>
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform" xsl:version="1.0">

    <head>
        <title>Les "Concerts Anacréon"</title>
```

Figure 1-3

Une page HTML à générer dynamiquement.




```
</head>

<body>

    <H1 align="center">Les "Concerts Anacréon" présentent</H1>

    <hr/>

    <br/>

    <H1 align="center">
        Concert le <xsl:value-of select="/Annonce/Date"/>
    </H1>

    <H4 align="center">
        <xsl:value-of select="/Annonce/Lieu"/>
    </H4>

    <H2 align="center">
        Ensemble "<xsl:value-of select="/Annonce/Ensemble"/>"
    </H2>

    <xsl:for-each select="/Annonce/Interprète">
        <p>
```

```
                <xsl:value-of select="./Nom"/>,
                <xsl:value-of select="./Instrument"/>
            </p>
        </xsl:for-each>

        <H3>
            Oeuvres de <xsl:value-of select="/Annonce/Compositeurs"/>
        </H3>

    </body>
</html>
```

Ce fichier HTML est en réalité un fichier XML, comme le montre la première ligne, dont l'élément racine est l'élément `<html>`, qui introduit donc un document HTML. Les deux attributs accompagnant cet élément sont obligatoires : si vous voulez écrire votre propre exemple, vous pouvez recopier les deux premières lignes telles quelles, sans vous poser de question. Le premier attribut est en fait une définition de domaine nominal, c'est-à-dire la définition d'un jeu de vocabulaire identifié par une URL ad hoc, et abrégée sous la forme d'un préfixe qui est ici `xsl`. Cette URL n'est pas du tout utilisée en tant qu'URL, c'est juste une chaîne de caractères convenue qui représente symboliquement le jeu de vocabulaire XSLT ; cette chaîne de caractères en forme d'URL a été déterminée par le W3C, et le processeur XSLT qui va lire ce fichier ne prend en compte en tant qu'instruction XSLT que les éléments XML dont le préfixe référence cette URL. Il est donc absolument impossible de changer quoi que ce soit à cette déclaration de domaine nominal. Il en est de même pour l'attribut `xsl:version`, du moins tant que la spécification XSLT 2.0 ne sera pas parvenue au stade de *Recommendation* : pour l'instant, donc, la seule valeur possible est 1.0.

Le reste du fichier est essentiellement du XHTML, avec ça et là des instructions XSLT : l'instruction `<xsl:for-each>`, et l'instruction `<xsl:value-of>` dans le cas présent.

Ces instructions ont pour but d'alimenter le texte HTML en données provenant d'un fichier XML auxiliaire. Ces données ne sont pas directement présentes dans le texte XHTML, afin de pouvoir générer autant de pages différentes que nécessaire, ayant toutes le même aspect : il suffit pour cela d'avoir plusieurs fichiers XML différents.

Dans notre exemple, le fichier XML auxiliaire est celui-ci :

Annonce.xml

```
<?xml version="1.0" ?>

<Annonce>

    <Date>Jeudi 17 janvier 2002 20H30
    </Date>
    <Lieu>Chapelle des Ursules</Lieu>

    <Ensemble>A deux violes esgales</Ensemble>
```

```
    <Interprète>
        <Nom> Jonathan Dunford </Nom>
        <Instrument>Basse de viole</Instrument>
    </Interprète>

    <Interprète>
        <Nom> Sylvia Abramowicz </Nom>
        <Instrument>Basse de viole</Instrument>
    </Interprète>

    <Interprète>
        <Nom> Benjamin Perrot </Nom>
        <Instrument>Théorbe</Instrument>
    </Interprète>

    <Interprète>
        <Nom> Freddy Eichelberger </Nom>
        <Instrument>Clavecin</Instrument>
    </Interprète>

    <Compositeurs>
        M. Marais, D. Castello, F. Rognoni
    </Compositeurs>

</Annonce>
```

La génération de la page HTML dynamique consiste à lancer un processeur XSLT en lui fournissant deux fichiers : d'une part le fichier de données XML `Annonce.xml`, et d'autre part le fichier programme `AnnonceConcert.xsl`.

Nous supposons que nous sommes en mode Commande (voir *Mode Commande*, page 5), et que le processeur choisi est Saxon. La ligne de commande pour obtenir la page HTML `annonce.html` serait alors celle-ci :

Ligne de commande (d'un seul tenant)

```
java -classpath saxon.jar com.icl.saxon.StyleSheet
                                  -o annonce.html Annonce.xml AnnonceConcert.xsl
```

Note

Ceux qui ne voudraient pas installer une machine virtuelle Java pourraient utiliser ici Instant Saxon.

Et le fichier obtenu serait celui-ci (voir aussi la figure 1-3) :

Annonce.html

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

```
      <title>Les &laquo;Concerts Anacr&eacute;on&raquo;</title>
   </head>
   <body>
      <H1 align="center">Les &laquo;Concerts Anacr&eacute;on&raquo;
       pr&eacute;sentent</H1>
      <hr><br><H1 align="center">
                  Concert le Jeudi 17 janvier 2002 20H30

      </H1>
      <H4 align="center">Chapelle des Ursules</H4>
      <H2 align="center">
                  Ensemble &laquo;A deux violes esgales&raquo;

      </H2>
      <p> Jonathan Dunford ,
                        Basse de viole
      </p>
      <p> Sylvia Abramowicz ,
                        Basse de viole
      </p>
      <p> Benjamin Perrot ,
                        Th&eacute;orbe
      </p>
      <p> Freddy Eichelberger ,
                        Clavecin
      </p>
      <H3>
                  Oeuvres de
              M. Marais, D. Castello, F. Rognoni

      </H3>
   </body>
</html>
```

Avant de continuer, il faut ici faire une remarque importante : le document source, malgré les apparences, est le fichier XML `Annonce.xml` ; le processeur XSLT transforme ce document XML en un document HTML `annonce.html`, par l'intermédiaire d'un programme XSLT contenu dans le fichier `AnnonceConcert.xsl` : le fichier auxiliaire de données est en fait, du point de vue XSLT, le document principal, et ce qui semblait n'être que le modèle de fichier HTML à produire est en réalité le programme XSLT à exécuter.

Extraction individuelle (pull processing)

Regardons maintenant de plus près le fonctionnement du programme XSLT : il est essentiellement basé sur le principe de l'extraction individuelle (connu en anglais sous le nom de *pull processing*). Typiquement, le début du programme est entièrement basé sur ce principe :

```
<head>
    <title>Les "Concerts Anacréon"</title>
</head>
<body>

    <H1 align="center">Les "Concerts Anacréon" présentent</H1>

    <hr/>

    <br/>

    <H1 align="center">
        Concert le <xsl:value-of select="/Annonce/Date"/>
    </H1>

    <H4 align="center">
        <xsl:value-of select="/Annonce/Lieu"/>
    </H4>

    <H2 align="center">
        Ensemble "<xsl:value-of select="/Annonce/Ensemble"/>"
    </H2>
...
```

Tout ce texte est recopié tel quel par le processeur XSLT dans le document HTML résultat, à l'exception des instructions `<xsl:value-of>` qui sont des instructions d'extraction individuelle : elles sont donc remplacées par leur valeur, valeur qui est prélevée (ou *extraite*, d'où cette notion d'extraction individuelle) dans le fichier de données XML.

Le langage XPath

Le problème est naturellement de définir à quel endroit du document XML se trouve la donnée à extraire. C'est là qu'intervient le langage XPath, qui permet de référencer des ensembles d'éléments, d'attributs et de textes figurant dans un document XML.

Par exemple, l'expression XPath `/Annonce/Date` sélectionne tous les éléments `<Date>` qui se trouvent directement rattachés à un élément `<Annonce>` racine du document. Si vous regardez le document XML, vous verrez qu'il n'y en a qu'un qui réponde à la description. L'instruction :

```
<xsl:value-of select="/Annonce/Date"/>
```

sera donc remplacée par sa valeur, c'est-à-dire par le texte trouvé dans l'élément `<Date>` sélectionné, à savoir :

```
Jeudi 17 janvier 2002 20H30
```

Remarque

L'analogie entre une expression XPath telle que `/Annonce/Date` et un chemin Unix d'accès à un fichier a souvent été soulignée. Dans notre exemple, l'expression XPath lue comme un chemin Unix voudrait dire « le fichier (ou répertoire) `Date` se trouvant dans le répertoire `Annonce` situé à la racine du système de fichiers ». Cette analogie est à mon avis plus dangereuse qu'utile. En effet il y a une énorme différence entre une arborescence de fichiers et une arborescence d'éléments XML : il ne peut jamais y avoir deux fichiers ayant le même nom dans un même répertoire, alors qu'il peut fort bien y avoir plusieurs éléments de mêmes noms rattachés au même élément. Dans notre fichier XML d'exemple, voyez comment interpréter l'expression `/Annonce/Interprète` : il s'agit de l'ensemble de tous les éléments `<Interprète>` directement rattachés à la racine `<Annonce>`. Cette possible multiplicité d'éléments intervient à tous les niveaux : dans une expression comme `/Annonce/Interprète/Instrument`, rien n'interdit qu'il y ait plusieurs `<Interprète>` par `<Annonce>` (déjà vu), et plusieurs `<Instrument>` par `<Interprète>` (il n'y a pas d'exemple dans notre fichier XML, mais cela pourrait arriver : voir plus bas).

La conséquence, et c'est là ce qu'on voulait montrer, est qu'il est dès lors impossible de lire une expression XPath comme on lirait un chemin Unix : c'est en cela que l'analogie est mauvaise et nocive. Un chemin Unix se lit normalement, de gauche à droite. Mais le même chemin, considéré comme une expression XPath, doit se lire à l'envers, car c'est le seul moyen de préserver la multiplicité des éléments sélectionnés : `/Annonce/Interprète/Instrument` doit se lire « les instruments qui sont rattachés à des interprètes qui sont rattachés à la racine Annonce ».

En appliquant cette instruction `xsl:value-of` aux divers chemins XPath concernés, on voit donc facilement que le fragment de programme ci-dessus va produire le résultat suivant :

```
<head>
    <title>Les "Concerts Anacréon"</title>
</head>

<body>

    <H1 align="center">Les "Concerts Anacréon" présentent</H1>

    <hr/>

    <br/>

    <H1 align="center">
        Concert le Jeudi 17 janvier 2002 20H30
    </H1>

    <H4 align="center">
        Chapelle des Ursules
    </H4>

    <H2 align="center">
        Ensemble "A deux violes esgales"
    </H2>
...
```

La suite du programme comporte une répétition <xsl:for-each> :

```
<xsl:for-each select="/Annonce/Interprète">
    <p>
        <xsl:value-of select="./Nom"/>,
        <xsl:value-of select="./Instrument"/>
    </p>
</xsl:for-each>
```

Ici, l'instruction <xsl:for-each> sélectionne un ensemble d'éléments XML contenant tous les <Interprète> directement rattachés à la racine <Annonce> ; et dans le corps de cette répétition, on dispose d'un élément courant, qui est bien sûr un <Interprète>, que l'on peut référencer dans une expression XPath par la notation ".". Ainsi, l'expression Xpath "./Nom" veut dire : « tous les <Nom> rattachés directement à l'élément <Interprète> courant ». Comme un tel <Nom> est unique, l'instruction :

```
<xsl:value-of select="./Nom"/>
```

est donc remplacée par la valeur du nom de l'interprète courant, et il en est de même pour l'instruction :

```
<xsl:value-of select="./Instrument"/>
```

Au total, cette répétition produit la séquence suivante dans le document résultat :

```
<p> Jonathan Dunford ,
                Basse de viole
</p>
<p> Sylvia Abramowicz ,
                Basse de viole
</p>
<p> Benjamin Perrot ,
                Th&eacute;orbe
</p>
<p> Freddy Eichelberger ,
                Clavecin
</p>
```

Finalement, on obtient donc le fichier HTML déjà montré plus haut.

Autre exemple

Les feuilles de style simplifiées se résument en gros à ce qu'on vient de voir ; il n'est guère possible de faire plus, à part utiliser quelques instructions XSLT complémentaires, par exemple un <xsl:if>. Pour illustrer ceci, supposons maintenant que le fichier XML à traiter soit constitué ainsi :

Annonce.xml

```
<?xml version="1.0" ?>

<Annonce>
```

```
    <Date>Jeudi 17 janvier 2002 20H30
    </Date>
    <Lieu>Chapelle des Ursules</Lieu>

    <Ensemble>A deux violes esgales</Ensemble>

    <Interprète>
        <Nom> Jonathan Dunford </Nom>
        <Instrument>Basse de viole</Instrument>
    </Interprète>

    <Interprète>
        <Nom> Sylvia Abramowicz </Nom>
        <Instrument>Basse de viole</Instrument>
    </Interprète>

    <Interprète>
        <Nom> Benjamin Perrot </Nom>
        <Instrument>Théorbe</Instrument>
        <Instrument>Luth</Instrument>
        <Instrument>Chitarrone</Instrument>
        <Instrument>Vihuela</Instrument>
        <Instrument>Angelique</Instrument>
    </Interprète>

    <Interprète>
        <Nom> Freddy Eichelberger </Nom>
        <Instrument>Clavecin</Instrument>
    </Interprète>

    <Compositeurs>
        M. Marais, D. Castello, F. Rognoni
    </Compositeurs>

</Annonce>
```

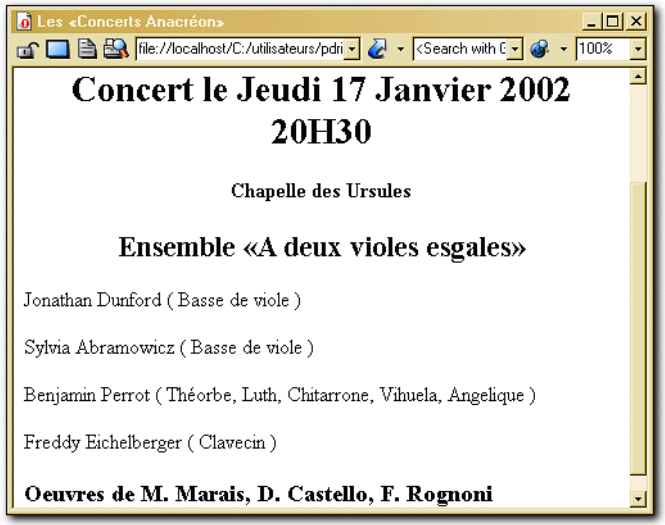
Le problème à traiter, ici, est qu'il peut y avoir une liste d'instruments pour chaque interprète. On veut que le résultat soit celui montré à la figure 1-4.

Fondamentalement, il n'y a rien de changé au programme. Simplement, il va falloir placer une deuxième instruction de répétition, pour donner la liste des instruments par interprète, ce qui complique un peu les choses, car une liste implique la présence de séparateurs d'éléments (ici, c'est une virgule).

Cette virgule doit apparaître après chaque nom d'instrument, sauf le dernier : d'où la nécessité d'utiliser une instruction `<xsl:if>` qui va nous dire si l'`<Instrument>` courant est le dernier ou non.

Figure 1-4

Une page HTML un peu plus compliquée à générer dynamiquement.



AnnonceConcert.xsl

```
<?xml version="1.0" ?>
<html xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
      xsl:version="1.0">

    <head>
        <title>Les "Concerts Anacréon"</title>
    </head>

    <body>

        <H1 align="center">Les "Concerts Anacréon" présentent</H1>

        <hr/>

        <br/>

        <H1 align="center">
            Concert le <xsl:value-of select="/Annonce/Date"/>
        </H1>

        <H4 align="center">
            <xsl:value-of select="/Annonce/Lieu"/>
        </H4>

        <H2 align="center">
```

```
            Ensemble "<xsl:value-of select="/Annonce/Ensemble"/>"
        </H2>

        <xsl:for-each select="/Annonce/Interprète">
            <p>
                <xsl:value-of select="./Nom"/>
                <xsl:text> ( </xsl:text>
                <xsl:for-each select="./Instrument">
                    <xsl:value-of select="."/>
                    <xsl:if test="position() != last()">
                        <xsl:text>, </xsl:text>
                    </xsl:if>
                </xsl:for-each>
                <xsl:text> ) </xsl:text>
            </p>
        </xsl:for-each>

        <H3>
            Oeuvres de <xsl:value-of select="/Annonce/Compositeurs"/>
        </H3>

    </body>
</html>
```

Ici la première instruction de répétition `<xsl:for-each select="/Annonce/Interprète">` sélectionne un ensemble d'éléments `<Interprète>`, et chacun d'eux devient tour à tour l'`<Interprète>` courant, noté `"."` dans les deux premiers attributs `select` qui viennent ensuite :

- `select="./Nom"` : sélectionne tous les `<Nom>` qui sont directement rattachés à l'`<Interprète>` courant (il n'y en a qu'un) ;
- `select="./Instrument"` : sélectionne tous les `<Instrument>` qui sont directement rattachés à l'`<Interprète>` courant (il peut y en avoir plusieurs).

La deuxième instruction de répétition `<xsl:for-each>` produit donc une liste d'instruments, en copiant dans le document résultat le texte associé à l'`<Instrument>` courant, et en le faisant suivre d'une virgule, sauf si l'`<Instrument>` courant est le dernier de la liste.

Par ailleurs on remarque l'utilisation de l'instruction `<xsl:text>`. Cette instruction est très utile, mais son rôle reste très modeste, en tout cas ici. Elle sert à délimiter exactement un texte littéral à produire dans le document résultat, sans que des espaces, tabulations, et autres sauts de ligne ne viennent s'ajouter de façon intempestive au résultat.

La différence entre :

```
<xsl:value-of select="./Nom"/>
   <xsl:text> ( </xsl:text>
```

et :

```
<xsl:value-of select="./Nom"/>
   (
```

est que dans le premier cas, on voit qu'il y a exactement un espace avant la parenthèse, alors que dans le deuxième, on voit qu'il y en a plusieurs (combien ? difficile à dire) et qu'il y a aussi un saut de ligne. On pourrait se passer de l'instruction `<xsl:text>` en écrivant :

```
<xsl:value-of select="./Nom"/> ( <xsl:for-each select="./Instrument">
```

mais ce n'est pas très agréable, car cela impose la présentation du programme en empêchant de placer des sauts de lignes où on veut afin d'aérer la disposition.

Voici pour finir le fichier HTML obtenu (on pourra comparer avec la figure 1-4), en deux versions : la première obtenue avec le programme XSLT tel qu'il apparaît avant (fichier `AnnonceConcert.xsl`) :

annonce.html

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

      <title>Les &laquo;Concerts Anacr&eacute;on&raquo;</title>
   </head>
   <body>
      <H1 align="center">Les &laquo;Concerts Anacr&eacute;on&raquo;
        pr&eacute;sentent</H1>
      <hr><br><H1 align="center">
                    Concert le Jeudi 17 janvier 2002 20H30

      </H1>
      <H4 align="center">Chapelle des Ursules</H4>
      <H2 align="center">
                    Ensemble &laquo;A deux violes esgales&raquo;

      </H2>
      <p> Jonathan Dunford  ( Basse de viole ) </p>
      <p> Sylvia Abramowicz  ( Basse de viole ) </p>
      <p> Benjamin Perrot  ( Th&eacute;orbe, Luth, Chitarrone, Vihuela, Angelique )
          </p>
      <p> Freddy Eichelberger  ( Clavecin ) </p>
      <H3>
                    Oeuvres de
               M. Marais, D. Castello, F. Rognoni

      </H3>
   </body>
</html>
```

Et la deuxième version :

annonce.html

```
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=utf-8">

      <title>Les &laquo;Concerts Anacr&eacute;on&raquo;</title>
   </head>
   <body>
      <H1 align="center">Les &laquo;Concerts Anacr&eacute;on&raquo;
        pr&eacute;sentent</H1>
      <hr><br><H1 align="center">
                  Concert le Jeudi 17 janvier 2002 20H30

      </H1>
      <H4 align="center">Chapelle des Ursules</H4>
      <H2 align="center">
                  Ensemble &laquo;A deux violes esgales&raquo;

      </H2>
      <p> Jonathan Dunford
                        (
                        Basse de viole
                        )

      </p>
      <p> Sylvia Abramowicz
                        (
                        Basse de viole
                        )

      </p>
      <p> Benjamin Perrot
                        (
                        Th&eacute;orbe
                              ,
                            Luth
                              ,
                            Chitarrone
                              ,
                            Vihuela
                              ,
                            Angelique
                        )

      </p>
      <p> Freddy Eichelberger
                        (
                        Clavecin
                        )
```

```
      </p>
      <H3>
                  Oeuvres de
            M. Marais, D. Castello, F. Rognoni

      </H3>
   </body>
</html>
```

Cette deuxième version est obtenue en supprimant les instructions `<xsl:text>` du programme précédent, comme ceci :

```
<xsl:for-each select="/Annonce/Interprète">
    <p>
        <xsl:value-of select="./Nom"/>
        (
        <xsl:for-each select="./Instrument">
            <xsl:value-of select="."/>
            <xsl:if test="position() != last()">
                ,
            </xsl:if>
        </xsl:for-each>
        )
    </p>
</xsl:for-each>
```

Conclusion

Nous venons de voir ce qu'on peut faire avec la notion de feuille de style simplifiée. Cela se résume à de l'extraction individuelle (pull processing), agrémentée de la possibilité d'utiliser quelques instructions d'XSLT, permettant de faire des répétitions, des tests, des choix multiples, et quelques autres traitements complémentaires.

Mais il manque essentiellement le pendant de l'extraction individuelle, qui est la *distribution sélective* (ou *push processing*), qui donne toute sa puissance à XSLT (mais qui en fait aussi la difficulté). Il manque également des instructions qui sont interdites (ou plutôt impossibles) avec les feuilles de style simplifiées, notamment toutes les instructions dites de premier niveau : cela inclut la déclaration de variables globales, de clés associatives (`<xsl:key>`), d'instructions d'importation d'autres feuilles de style, et la définition de modèles nommés (qui sont des structures jouant à peu près le même rôle que les fonctions ou les sous programmes dans les autres langages). Avouez que cela finit par faire vraiment beaucoup, et que dans ces conditions, les feuilles de style simplifiées n'ont pas beaucoup d'intérêt, à part d'être très simples et faciles à comprendre intuitivement. Mais pour quelqu'un qui connaît bien XSLT, on peut aller jusqu'à dire qu'elles n'ont strictement aucun intérêt par rapport aux vraies feuilles de style XSLT.

Nous avons vu au passage que le langage XSLT possède son propre jeu d'instructions, au format XML, mais identifié par le domaine nominal *http://www.w3.org/1999/XSL/Transform* ; et qu'on y utilise un autre langage, le langage XPath, qui n'a rien à voir avec XML.

Ce langage, qui permet de sélectionner divers ensembles de fragments d'un document XML, est en fait nettement plus compliqué que ce qu'on a pu montrer dans les exemples précédents, et il faudra un gros chapitre pour en venir à bout.

Parcours de lecture

Nous venons de voir l'intérêt du langage XSLT, à quoi il sert et où il se situe. Bien que partie intégrante de XSL, XSLT est un langage qui peut être considéré comme indépendant. De plus il est extrêmement différent, dans sa philosophie et dans les compétences qu'il met en jeu, du langage XSL-FO, l'autre versant de XSL.

C'est pourquoi dans la suite de ce livre, on se consacrera exclusivement à XSLT lui-même, et à son compère XPath.

Nous verrons donc d'abord XPath, puis les principes du langage XSLT. Le style adopté est un style à l'opposé d'un manuel de référence, en ce sens qu'on cherche plus à favoriser la compréhension du sujet que l'exhaustivité du propos. Mais ceci doit être tempéré par le fait que plus on avance dans la lecture, et plus on en sait : on est donc plus à même d'accepter des détails ou des subtilités vers la fin de la présentation que vers le début.

Si vous êtes pressé et voulez lire l'essentiel pour comprendre XSLT, sans entrer dans les détails, et pour comprendre le mode de pensée à adopter pour être en phase avec ce langage, il n'y a qu'un chapitre à lire : le chapitre *Au cœur du langage XSLT*, page 75.

Après avoir fait le tour de la plus grande partie du langage en quelques chapitres, on s'intéressera à la notion de « pattern » de conception XSLT, en mettant en évidence des grands thèmes qui reviennent fréquemment, aussi bien dans le domaine microscopique de la programmation par des techniques particulières, que dans celui plus macroscopique de la transformation d'arbres.

Enfin, dans les annexes, on trouvera des éléments complémentaires, notamment la description sommaire de quelques instructions dont la compréhension ne pose pas de difficulté particulière, ou dont la compréhension est plus du ressort de tests effectifs avec un processeur XSLT que de celui d'une lecture argumentée. On y trouvera aussi des éléments syntaxiques, et une description des fonctions prédéfinies XPath et XSLT.

A noter que dans ce livre, les exemples fournis ont tous été testés avec les processeurs Xalan et Saxon, et que le mode de fonctionnement sous-jacent des exemples proposés est le mode Commande ; cela n'a aucune influence sur la sémantique du langage XSLT, mais cela permet de fixer les idées.