tion, et générera un document résultat pouvant être une page XHTML, ou tout type de format selon le média visé (XML, WML, texte, etc.). Voir figure 6-5-a.

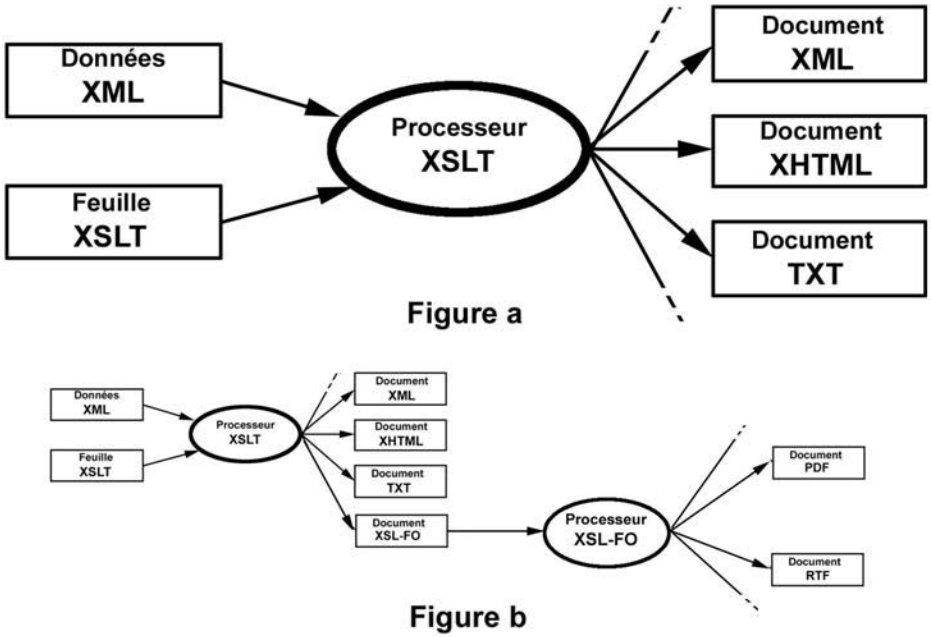

Figure 6-5
Principe de la transformation XSLT.

XSL permet aussi de transformer un document XML en page imprimable au format PDF ou RTF, par exemple. Dans ce cas, le processeur XSLT doit produire un format XSL-FO qui sera ensuite traité par un second processeur XSL-FO dédié à cet usage (voir figure 6-5-b).

Organisation d'un document XSLT

La structure d'un document XSLT

Un document XSLT étant un document XML, il doit donc commencer par un prologue XML (revoir l'introduction au XML si besoin est) :

```
<?xml version="1.0" encoding="iso-8859-1"?>
```

Ce prologue peut quelquefois être accompagné d'une définition dans la DTD interne, faisant référence à l'équivalent Unicode de certaines valeurs fréquemment utilisées (c'est le cas notamment des pages XSLT générées par Dreamweaver) :

```
<!DOCTYPE xsl:stylesheet  [
<!ENTITY nbsp   " ">
...
<!ENTITY euro   "&#8364;">
]>
```

Vient ensuite l'élément racine du document XSLT :

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

L'élément racine doit toujours comprendre la version XSLT utilisée, afin qu'elle soit identifiée par le processeur (**`version="1.0"`**). L'attribut **`xsl`** définit l'espace de nom et l'associe à une URL (**`xmlns:xsl="http ..."`**) afin de qualifier d'une manière explicite tous les éléments XSLT.

À l'intérieur de l'élément racine, il est possible de spécifier le format de sortie du document produit par le processeur XSLT. Pour cela, il faut utiliser l'instruction **`xsl:output`** en précisant la méthode correspondant au format de sortie (html, xml, text...) et le type d'encodage (UTF-8, ISO-8889-1...).

Exemple :

```
<xsl:output method="html" encoding="iso-8859-1"/>
```

Les modèles de transformation (template)

Lorsque le processeur XSLT traite un document XML, il commence par créer une structure arborescente (appelée « arbre source ») à l'image du document XML. Le principe de la transformation consiste ensuite à appliquer à cet arbre source des modèles de transformation (appelés *template*) contenus dans la feuille XSLT, afin de produire un arbre résultat. L'arbre résultat ainsi créé permettra d'élaborer le document de sortie (on appelle cette phase la « sérialisation ») pouvant être par exemple une structure XHTML (voir figure 6.6).

Les motifs de sélection (pattern)

Un modèle de transformation (template) effectue un traitement sur un motif de sélection (appelé *pattern*). Concrètement un motif de sélection correspond à un nœud, ou à un ensemble de nœuds, sur lequel le modèle va être appliqué et pour lequel son contenu sera remplacé dans l'arbre résultat. Les motifs de sélection (pattern) sont exprimés selon le langage X-Path (voir figure 6.6).

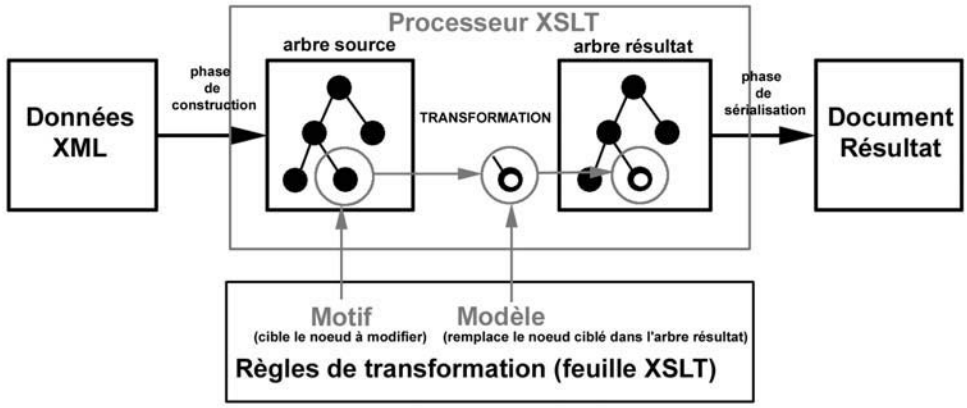

Figure 6-6
Principe d'une règle de transformation : le motif cible le nœud (ou groupe de nœuds) à remplacer par le modèle.

Les règles de transformation (xsl:template match="pattern")

Un programme XSLT se compose principalement d'une série de règles de transformation qui définissent si l'élément courant doit être traité (grâce au motif) et, si oui, par quoi il doit être remplacé (grâce au modèle). Chaque règle est donc constituée d'un motif (pattern) et d'un modèle de transformation (template). Voir figure 6-6.

Une règle de transformation se présente sous la forme qui suit :

```
<xsl:template match="MOTIF">
MODELE
</xsl:template>
```

Avec pour `MOTIF` (pattern) une expression X-Path désignant la sélection devant être traitée, et pour `MODELE` le code qui doit remplacer la sélection désignée par le motif.

Le `MODELE` peut contenir un simple texte, des éléments XML ou encore d'autres instructions XSLT, comme les deux instructions fondamentales que nous allons présenter ci-après.

Contrainte concernant les expressions X-Path du motif

Le motif (pattern) d'une règle est la valeur de l'attribut **`match`**. Cette valeur sera matérialisée par une expression X-Path (revoir si besoin est la partie sur le langage de navigation X-Path). Cependant, l'expression X-Path d'un motif doit utiliser exclusivement les axes **`child::`** ou **`attribute::`** à l'exclusion de tous les autres axes comme **`parent::`**, **`self::`**, etc. (et leur abréviation : revoir tableau 6-1). Par exemple, vous pourrez vous servir, comme motif, des chemins suivants : **`child::immeuble/proprietaire`** (ou l'abréviation : « **`immeuble/proprietaire`** »), **`child:: proprietaire/attribute::nom`** (ou l'abréviation : « **`proprietaire/@nom`** »). Mais vous ne pourrez pas utiliser **`self::node()`** (ou l'abreviation « **`.`** »), **`parent::node()`** (ou l'abreviation « **`..`** »). Par contre, pour un prédicat, vous pourrez employer tous les types d'axes sans restriction.

Règles utilisant l'instruction xsl:value-of

Une règle de transformation utilisant l'instruction **xsl:value-of** se présente sous la forme ci-dessous :

```
<xsl:template match="MOTIF">
...
<xsl:value-of select ="Chemin X-Path" />
...
</xsl:template>
```

Dans ce cas, au moment du traitement du processeur XSLT, l'instruction **<xsl:value-of select ="Chemin X-Path" />** est remplacée par la valeur textuelle de ce qui est désigné par le **Chemin X-Path** de l'attribut **select**.

Exemple :

source.xml

```
<?xml version="1.0" encoding="iso-8859-1"?>
<immeuble  ville="Malakoff" >
           <proprietaire nom="Defrance">
               <etage>2eme</etage>
               <piece>3</piece>
               <email>jmdefrance@aol.com</email>
           </proprietaire>
           <proprietaire nom="Bertaut" >
               <etage>1er</etage>
               <piece>3</piece>
               <email>abertaut@aol.com</email>
           </proprietaire>
</immeuble>
```

test1.xsl

```
<?xml version="1.0" encoding="iso-8859-1"?><!-- DWXMLSource="test1.xml" -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="iso-8859-1"/>
<xsl:template match="/">
<html>
<head>
<title>Test1</title>
</head>
<body>
<p>Nom :<xsl:value-of select="immeuble/proprietaire/@nom"/></p>
<p>Etage :  <xsl:value-of select="immeuble/proprietaire/etage"/></p>
```

```
<p>E-mail : <xsl:value-of select="immeuble/proprietaire/email"/></p>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Si on exécutait cet exemple, on obtiendrait alors le résultat suivant :

Nom : Defrance
Etage : 2eme
E-mail : jmdefrance@aol.com

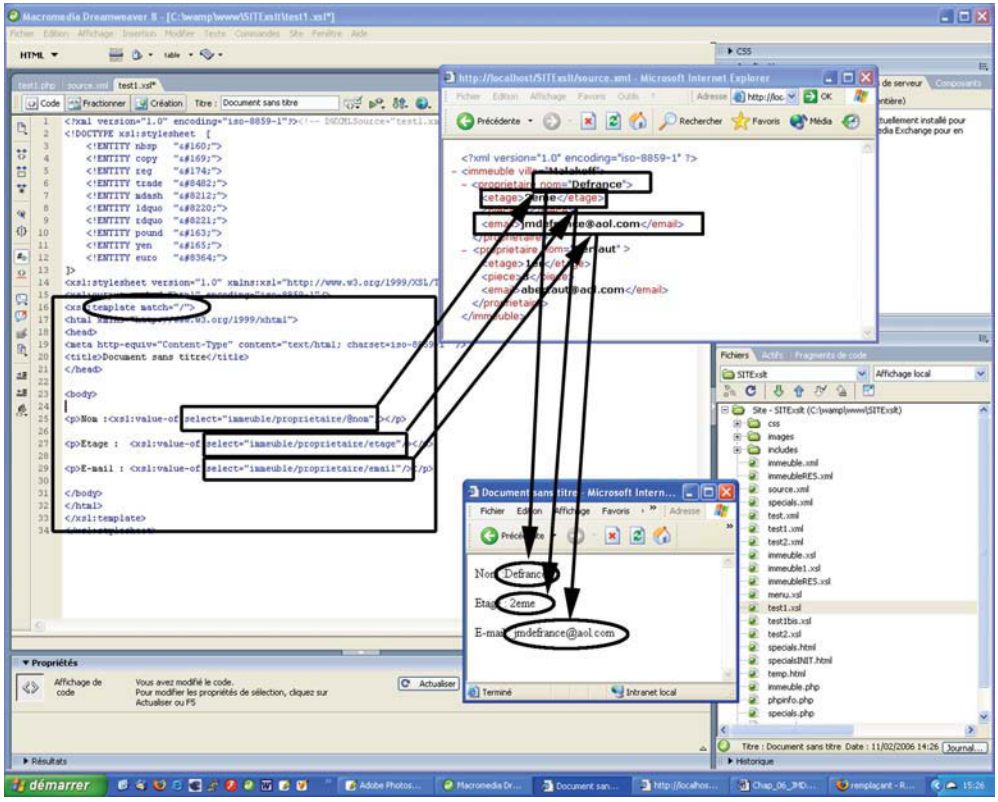

Figure 6-7
Exemple de règle utilisant une instruction xsl:value-of.

On remarque que seules les informations du premier propriétaire ont été renvoyées dans le résultat. Cela est lié au fait que XSLT sélectionne par défaut le premier nœud `<proprietaire>`

(dont l'attribut nom est **Defrance** dans notre exemple). Il est toutefois possible de préciser dans l'attribut **select** un nœud spécifique, en précisant son indice entre crochets, après son nom (exemple **proprietaire[n]**). Ainsi, le même résultat que l'exemple ci-dessus pourrait être aussi obtenu si nous avions utilisé le chemin suivant : **immeuble/proprietaire[1]/@nom** (idem pour la sélection des éléments **etage** et **email**). Par conséquent, de cette manière il devient facile de choisir le second nœud **<proprietaire>** (ou un nœud quelconque) pour obtenir, par exemple, les informations du propriétaire « Bertaut » en remplaçant les trois instructions **value-of** par le code ci-dessous :

```
<p>Nom :<xsl:value-of select="immeuble/proprietaire[2]/@nom"/></p>
<p>Etage :  <xsl:value-of select="immeuble/proprietaire[2]/etage"/></p>
<p>E-mail : <xsl:value-of select="immeuble/proprietaire[2]/email"/></p>
```

On obtiendrait dans ce cas le résultat suivant :

Nom : Bertaut
Etage : 1er
E-mail : abertaut@aol.com

Règles utilisant l'instruction xsl:apply-template

Une règle de transformation utilisant l'instruction **xsl:apply-templates** se présente sous la forme suivante :

```
<xsl:template match="MOTIF">
...<xsl:apply-templates />
...
</xsl:template>
```

Dans ce cas, au moment du traitement du processeur XSLT, l'instruction **<xsl: apply-templates />** génère un fragment du document source correspondant au contenu du nœud courant (voir repères 1 et 2 de la figure 6-8). Ce fragment sera ensuite interprété par les autres règles, et cela d'une manière récursive. Ce concept spécifique aux langages déclaratifs (auxquels XSLT appartient) n'étant pas évident, je vous propose de l'illustrer par un exemple pratique.

Exemple (le document XML est le même que dans l'exercice précédent, **source.xml**) :

test2.xsl

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="iso-8859-1"/>
<xsl:template match="/">
<html>
<head>
```

```
<title>Test2</title>
</head>
<body>
<xsl:apply-templates />
</body>
</html>
</xsl:template>
<xsl:template match="immeuble">
   <p>Liste des propriétaires : </p>
      <xsl:apply-templates />
   <p>Copyright 2006 </p>
</xsl:template>
<xsl:template match="proprietaire">
   <p>Nom :<xsl:value-of select="@nom"/></p>
   <p>Etage :<xsl:value-of select="etage"/></p>
   <p>E-mail :<xsl:value-of select="email"/></p>
</xsl:template>
</xsl:stylesheet>
```

Si on exécutait cet exemple, on obtiendrait alors le résultat suivant :

Liste des propriétaires :
Nom :Defrance
Etage : 2eme
E-mail : jmdefrance@aol.com
Nom : Bertaut
Etage : 1er
E-mail : abertaut@aol.com
Copyright 2006

Lorsque le processeur exécute la première instruction **<xsl: apply-templates />** (le nœud courant est alors égal à la racine / , voir figure 6-8 repère 1), un fragment du document source correspondant au contenu du nœud courant est alors généré. Lorsque la seconde règle est exécutée, elle traite alors ce premier fragment de document. Dans notre exemple, des éléments texte étant présents dans le modèle de cette seconde règle, ils sont copiés dans le document résultat à ce moment. Puis la seconde instruction **<xsl: apply-templates />** est analysée par le processeur, et un second fragment correspondant cette fois au contenu du nouveau nœud courant (soit, dans notre exemple, **<immeuble>** ; voir figure 6-8 repère 2) est produit. Ce dernier est ensuite examiné dans la troisième règle qui exploite des instructions **<xsl: value-of />** (voir figure 6-8 repère 3). D'ailleurs, les attributs **select** de ces instructions **<xsl: value-of />** sont configurés d'une manière relative à ce second fragment dont le nœud courant est maintenant l'élément **<proprietaire>** (**select="@nom"**, **select="etage"** et **select="email"**).

Nœud courant

Dans la partie consacrée aux expressions X-Path, nous avions utilisé la notion de nœud contexte pour définir l'évaluation d'un élément relatif à un nœud. Par contre, pour les transformations XSLT, on emploie le terme de nœud courant pour décrire l'exécution d'un modèle, relative à un nœud.

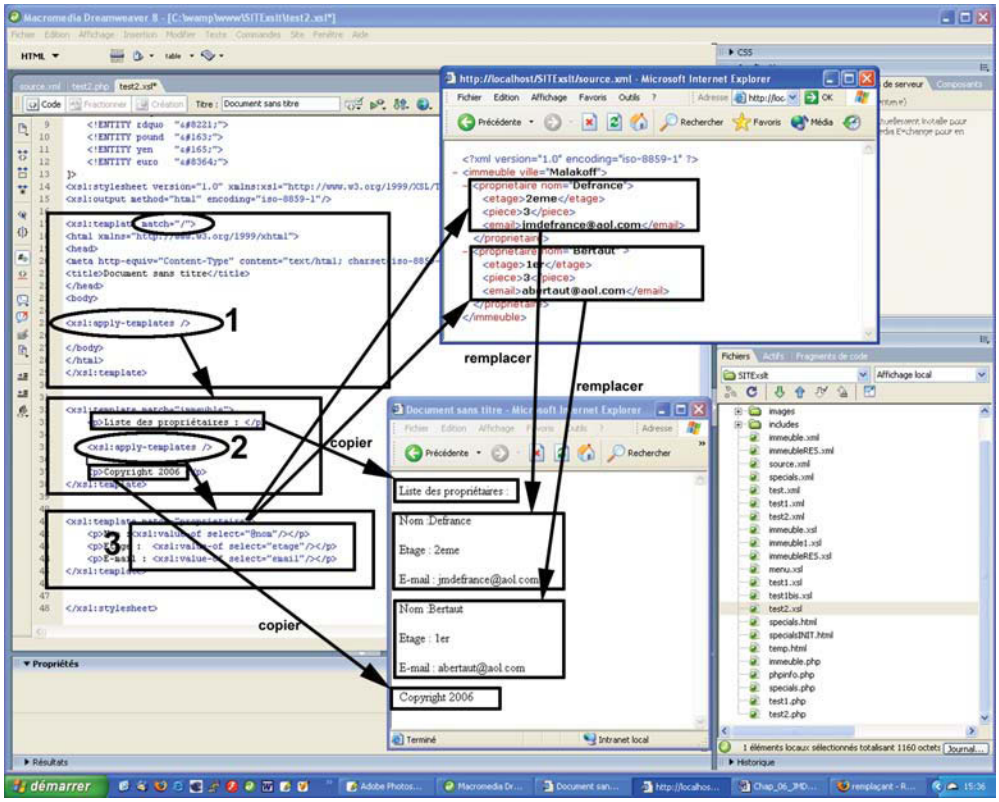

Figure 6-8

Exemple de règle utilisant une instruction xsl:apply-templates.

Instructions XSLT

Dans la précédente partie, nous avons présenté les deux instructions de base du XSLT (`xsl:value-of` et `xsl:apply-templates`). Vous trouverez ci-après une sélection de commandes complémentaires qui faciliteront la mise en œuvre d'applications XSLT dans vos futurs projets.

Création d'attributs (xsl:attribute)

Si vous désirez générer un attribut dans le document résultat (pour créer un lien hypertexte par exemple), il faut alors utiliser l'instruction `xsl:attribute`.

Elle se présente sous la forme suivante :

```
<xsl:attribute name="nomAttribut" >
...
</xsl:attribute>
```

Passons dès maintenant à un exemple concret pour l'illustrer. Nous vous proposons de créer un lien « mailto » – un lien « mailto » permet de déclencher l'ouverture du gestionnaire de messagerie du client, en préconfigurant l'e-mail du destinataire – dont le nom et l'attribut **href** devront être créés dynamiquement à partir du nom du destinataire et de son e-mail, tous les deux stockés dans un fichier XML. Le fichier XML source sera le même fichier **source.xml** déjà utilisé dans les exemples précédents.

test3.xsl

```
<?xml version="1.0" encoding="iso-8859-1"?><!-- DWXMLSource="test1.xml" -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="iso-8859-1"/>
<xsl:template match="/">
<html >
<head>
<title>Test 3</title>
</head>
<body>
<a>
<xsl:attribute name="href">mailto:<xsl:value-of select="immeuble/proprietaire/
email"/>
</xsl:attribute>
<xsl:value-of select="immeuble/proprietaire/@nom"/>
</a>
</body>
</html>
</xsl:template>
</xsl:stylesheet>
```

Lors de l'exécution de l'instruction **xsl:attribute**, la balise **<a>** hérite d'un nouvel attribut dont le nom est fourni par l'attribut **name** de l'instruction. La valeur de l'attribut **href** ainsi créé est ensuite récupérée par la commande **xsl:value-of** dont l'attribut **select** est configuré avec l'e-mail du propriétaire (**immeuble/proprietaire/email**). Le nom du lien cliquable est, quant à lui, retrouvé par une seconde instruction **xsl:value-of** dont l'attribut **select** est paramétré avec le nom du propriétaire (**immeuble/proprietaire/@nom**). L'ensemble de ces instructions produira le code suivant dans le document de sortie :

```
<a href="mailto:jmdefrance@aol.com" >Defrance</ a>
```

À noter qu'il existe aussi une instruction `xsl:element` destinée à générer des balises « élément » dans le document de sortie, mais qu'elle est rarement utilisée en raison de la facilité à créer une balise élément en l'insérant simplement dans le modèle (exemple : `<title>Test 3</title>`).

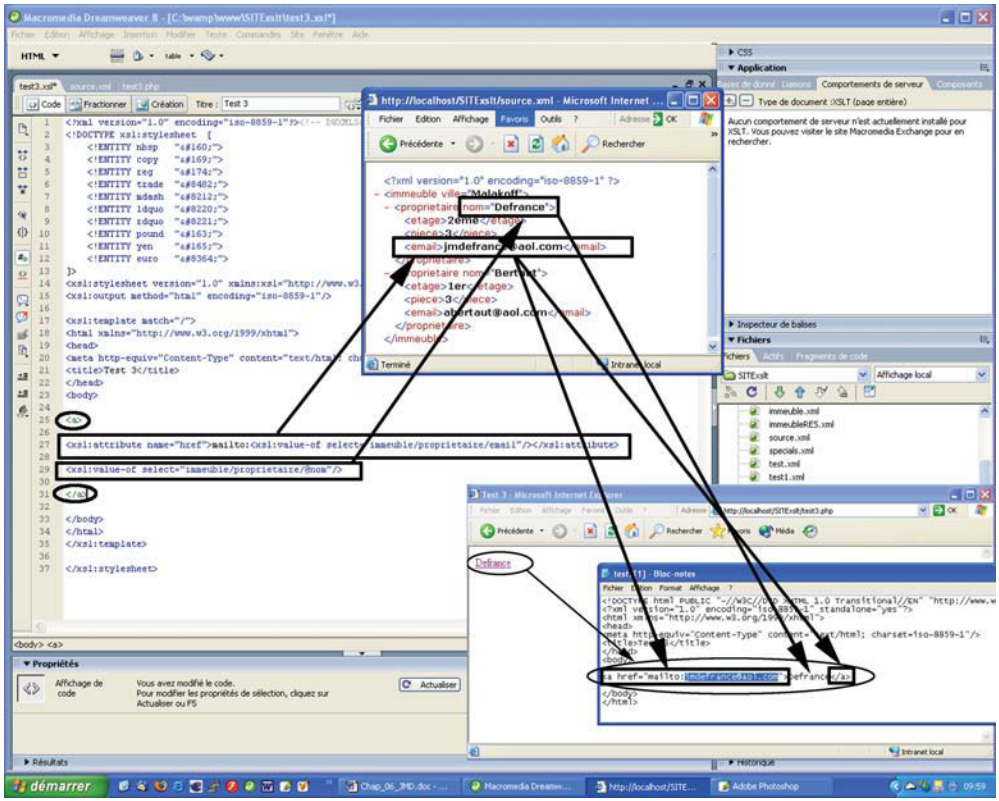

Figure 6-9

Test d'une feuille XSLT utilisant une instruction xsl:attribute pour créer un lien mailto.

Instructions de test

Il est souvent intéressant, au niveau du document de sortie, de placer une expression de condition qui détermine la création d'une partie de code. Il faut alors utiliser `xsl:if`.

Cette instruction `xsl:if` se présente sous la forme suivante :

```
<xsl:if name="nomAttribut" >
   <!—fragment de code conditionné -->
</xsl:attribute>
```

Elle permet de créer une condition simple « si... alors ». Elle possède un attribut **test** qui spécifie l'expression de condition qui doit être évaluée pour générer, ou pas, un fragment de codesur lequel portera la condition. Dans l'exemple ci-dessous, nous vous proposons de créer une liste des noms des propriétaires, séparés par une virgule. L'instruction **xsl:if** permettra de conditionner l'insertion de cette virgule qui ne devra pas être ajoutée si l'élément traité est le dernier de la liste. Le fichier XML source sera cette fois le fichier **source4.xml** (voir contenu ci-dessous). Le document résultat affichera le texte suivant à l'écran :

Liste des propriétaires :

Defrance , Bertaut , Fionda

source4.xml

Attention, il est important, dans cet exemple, de supprimer tous les espaces ou autres caractères de mise en forme entre les différentes balises du fichier :

```
<?xml version="1.0" encoding="iso-8859-1"?>
<immeuble >
<proprietaire>Defrance</proprietaire>
<proprietaire>Bertaut</proprietaire>
<proprietaire>Fionda</proprietaire>
</immeuble>
```

test4.xsl

```
<?xml version="1.0" encoding="iso-8859-1"?><!-- DWXMLSource="test1.xml" -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="iso-8859-1"/>
<xsl:template match="/">
<html >
<head>
<title>Test 4</title>
</head>
<body>
<p>Liste des propriétaires : </p>
<xsl:apply-templates />
</body>
</html>
</xsl:template>
<xsl:template match="immeuble/proprietaire">
<xsl:value-of select="."/>
   <xsl:if test="not(position()=last())">
   ,
```

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

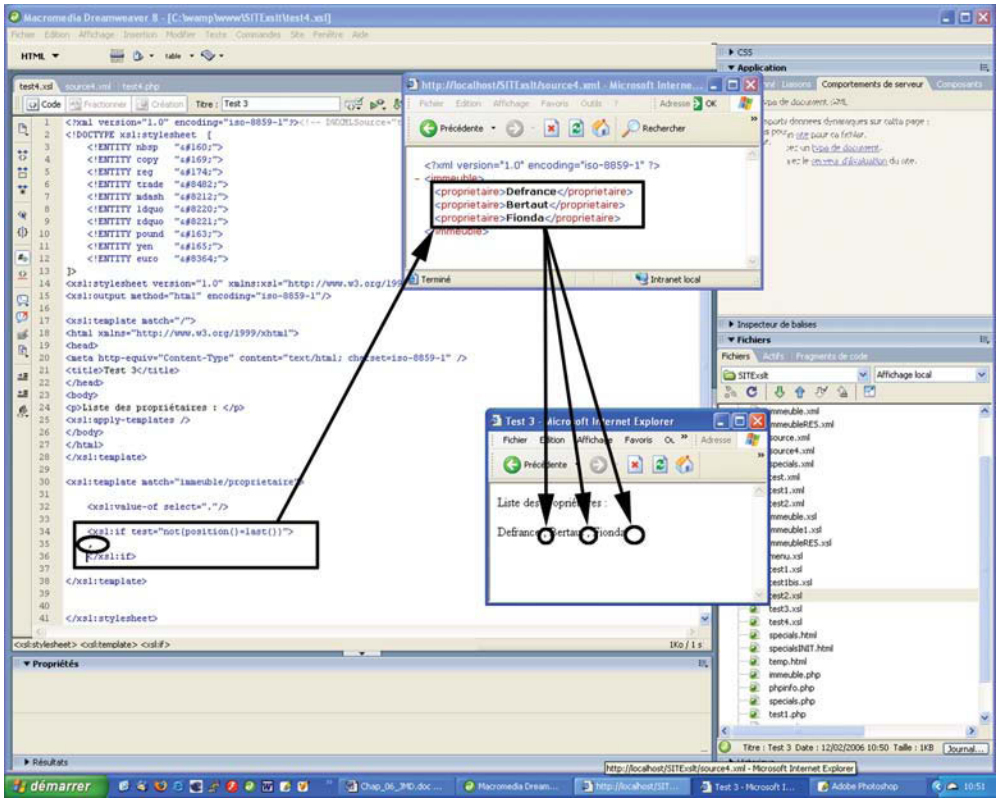

Figure 6-10
Test d'une feuille XSLT utilisant une instruction xsl:if pour conditionner l'affichage d'une virgule.

Instruction de boucle

Nous avons déjà vu, dans la partie dédiée aux règles de transformation, qu'il était possible de générer la répétition d'un fragment de code personnalisé selon les nœuds à traiter, en utilisant l'instruction `xsl:apply-template` (revoir figure 6-8). Cependant, il existe une autre instruction `xsl:for-each` (plus proche de la programmation procédurale à laquelle nous sommes habitués) qui s'apparente à la création d'une boucle, et qui permet d'obtenir ainsi des résultats semblables. Attention, contrairement à son nom qui pourrait être trompeur, l'instruction `xsl:for-each` est différente des boucles réalisées en PHP avec l'instruction `for()`. En effet, l'évolution d'un compteur n'est pas possible avec `xsl:for-each` du fait que XSLT est un

langage déclaratif et non procédural, comme PHP. Nous allons vous présenter cette instruction dans la partie ci-dessous, mais nous vous rappelons que le XSLT étant un langage déclaratif, il est plutôt conseillé d'utiliser l'instruction **xsl:apply-template** (en créant plusieurs modèles), afin de faciliter la maintenance de votre code et d'en augmenter la modularité (ce qui vous permettra ainsi de reprendre facilement les mêmes modèles dans d'autres programmes).

Cette instruction **xsl:for-each** se présente sous la forme ci-dessous :

```
<xsl:for-each select="nomAttribut" >
 <!—fragment de code répété -->
</xsl:for-each>
```

L'instruction **xsl:for-each** contient un fragment de code qui sera répété et personnalisé pour chaque nœud sélectionné par l'expression X-Path contenue dans l'attribut obligatoire **select**. Pour vous démontrer les similitudes entre ces deux instructions, nous vous proposons d'illustrer l'utilisation de **xsl:for-each** en recréant le même programme que celui de la figure 6-8, réalisé avec une instruction **xsl:apply-template**. Le fichier XML source sera le même fichier **source.xml** déjà utilisé dans les exemples précédents.

test5.xsl

```
<?xml version="1.0" encoding="iso-8859-1"?><!-- DWXMLSource="test1.xml" -->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="iso-8859-1"/>
<xsl:template match="/">
<html>
<head>
<title>Test 5</title>
</head>
<body>
<xsl:apply-templates />
</body>
</html>
</xsl:template>
<xsl:template match="immeuble">
   <p>Liste des propriétaires : </p>
   <xsl:for-each select="proprietaire" >
         <p>Nom :<xsl:value-of select="@nom"/></p>
         <p>Etage :  <xsl:value-of select="etage"/></p>
         <p>E-mail : <xsl:value-of select="email"/></p>
   </xsl:for-each>
```

```
    <p>Copyright 2006 </p>
</xsl:template>
</xsl:stylesheet>
```

Le résultat obtenu avec cette feuille XSLT est le suivant :

Liste des propriétaires :
Nom : Defrance
Etage : 2eme
E-mail : jmdefrance@aol.com
Nom : Bertaut
Etage : 1er
E-mail : abertaut@aol.com
Copyright 2006

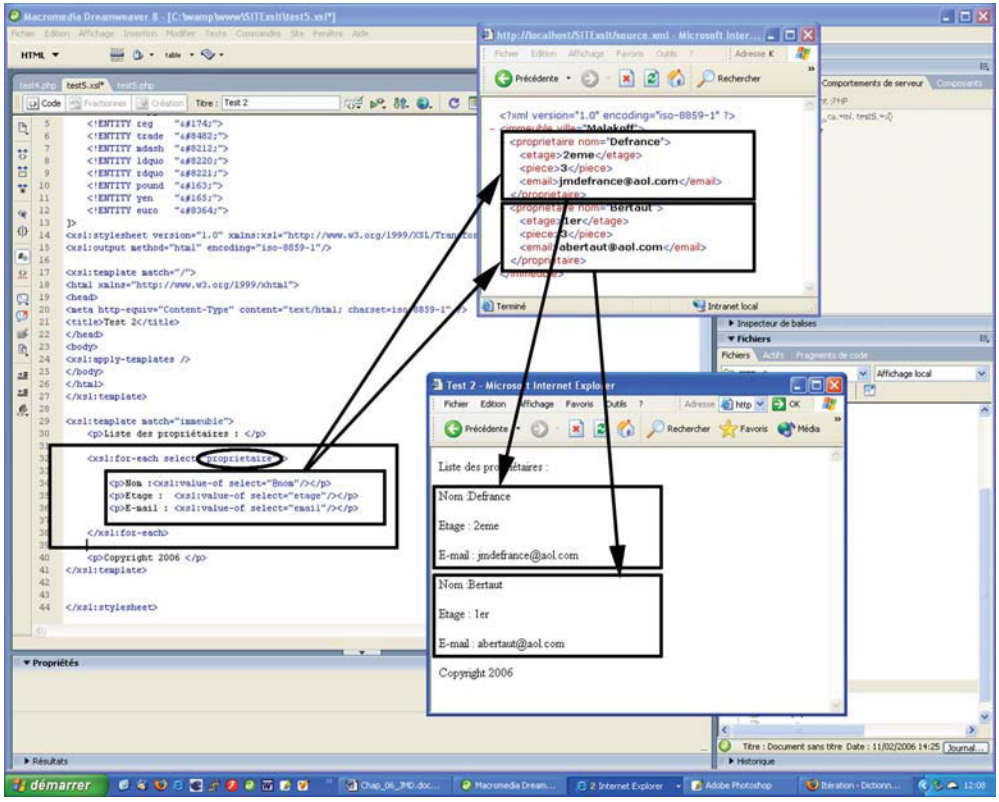

Figure 6-11
Test d'une feuille XSLT utilisant une instruction xsl:for-each.