Authoring Transformations by Direct Manipulation for Adaptable Multimedia Presentations

Lionel Villard
Opéra Project
INRIA Rhône-Alpes Research unit
Zirst - 655 avenue de l'Europe - Montbonnot
38334 Saint Ismier Cedex
France.
Tel: +33 (0)4 76 61 53 82
Fax: +33 (0)4 76 61 52 07
lionel.villard@inrialpes.fr

Abstract

In this paper, we present a method for authoring generic and adaptable multimedia presentations. This method relies on document transformations. For the currently available tools, designing the XML content and the transformation sheets is a tedious and error prone experience. We propose a framework based on an incremental transformation process. Incremental transformation processors represent a better alternative to help in the design of both the content and the transformation sheets. We believe that such authoring tools are a first step toward fully interactive transformation-based authoring environments. In this paper, we focus on the authoring of transformation sheets by direct manipulation. In particular, we study the authoring of transformations for the XSLT language defined at the World Wide Web Consortium.

Keywords: Incremental transformations, Authoring tools, Multimedia, Document model, XSLT, XML.

1 Introduction

The advent of the XML standard at the World Wide Web Consortium has triggered the definition of an incredible amount of vocabulary groups in different areas of content representations. Although these vocabulary groups are defined separately and designed for a variety of purposes such as content structure descriptions, layout languages, vector graphics or mathematical formulas rendering, they can also be inter-mixed in a single and same document. These vocabulary groups share the same encoding language at the syntactic level thanks to XML and are combined using another XML companion standard called XML Namespaces [11]. However, in most cases the content is encoded using third party DTD's while the resulting documents are encoded using rendering vocabularies such as XSL [3]. The result is an increasing diversity of document classes and vocabularies and a lack of authoring tools to cope with this diversity.

Two categories of existing authoring tools for editing XML content have been identified. Authoring tools that belong to the first category deal with rendering vocabularies. These tools are designed specifically for one or many rendering vocabularies. For example, Adobe illustrator allows the authoring of SVG documents, Grins [12] is designed for authoring SMIL documents, Amaya [18] allows the authoring of MathML[1], SVG[2] and XHTML[3]. In the second category, authoring tools are generic and deal with any kind of XML documents. Therefore, the only way to edit XML documents is through a lower-level text representation or at most through an enhanced representation such as a graphical tree. More recently, some authoring tools [14] give the author the ability to edit and attach style to XML elements. This association simplifies the authoring of a document by making the XML content more accessible to the user through the graphical interface. However, style sheets remain of a very limited help when considering more complex presentations, in particular those obtained by transformation. Moreover, authoring style sheets is relatively easy unlike transformation languages which are more complex to edit in a convenient way.

Another feature that must be addressed when presenting XML documents is the adaptation of document content to the current presentation context. This operation must take into account user capabilities, user preferences, physical location, network and system resources, etc. A lot of current content adaptation architectures are based on transformation process [6][16]. However, as we said previously, transformation sheets are hard to design. The challenge is then to provide authoring tools to help the author to design adaptable documents by authoring transformation sheets.

In [17] we have presented a framework that allows the editing of XML documents through one or more of its rendered presentations. In order to produce and to edit complex and adapted presentations, in particular multimedia presentations, this framework is based on an incremental transformation process. In

[1] Mathematical Markup Language (MathML) Version 2, available at http://www.w3.org/TR/MathML2/
[2] Scalable Vector Graphics (SVG) 1.0 Specification, available at http://www.w3.org/TR/SVG/
[3] XHTML™ 1.0: The Extensible HyperText Markup Language, available at http://www.w3.org/TR/xhtml1/

[17] we used this framework for editing source documents of transformations. The next step is to extend this framework for editing transformation sheets.

The goal of this paper is to present a method that allows authoring of transformations by direct manipulation. In particular, this paper focuses on the XSLT transformation language. XSLT [21] is a general-purpose transformation language that is used in many applications: data conversion, database queries, presentation generation, etc. The latter is probably the most popular application directly visible to the end user. In this paper, we focus on this kind of applications. We consider the reader familiar with the XSLT language. The vocabulary used in this paper that is related to XSLT is defined in the section 3.4.

The remainder of the paper is organized as follows: in the second section, we describe existing tools that allow the authoring of transformations and the authoring of queries. In the third section, we present the context of work presented in this paper. In particular, we describe the framework based on an incremental transformation processor. In the fourth section, we give an overview for authoring transformation sheets by direct manipulation. This overview is described through an editing use case study. In the fifth section, we describe the transformation rules that are generated for a subset of editing operations. In the last, section we give some conclusions and draw some perspectives.

2 Related work

We have classified the related work in two categories. The first category represents authoring tools designed specifically for XSLT. The second category gathers work about the authoring of presentation models (such as style sheets) and the connection with a database. The latter covers the XPath [20] aspect of the XSLT language.

2.1 Authoring tools for XSLT

Two kinds of existing authoring tools for XSLT have been identified. The goal of authoring tools that belong to the first category is to create general-purpose XSLT transformations [1][2]. Such tools are equivalent to interactive development environments (IDE). They provide a textual view of XSLT code and a debugger. When the target document is an HTML document, these tools provide a formatted view of the transformation result. An approach based on visual programming is presented in [13]. The presented tool allows the visualization and the editing of general-purpose transformations. In particular it proposes to edit XSLT transformations. Although this tool increases the author efficiency, it requires a good knowledge of the transformation language.

Authoring tools that belong to the second category allow the authoring of transformations for generating exclusively HTML or XHTML presentations. Therefore these tools take advantage of this specific context to provide high level authoring features. **<xsl>Composer** [19] allows the generation of XSLT transformations by direct manipulation. The main window is composed of four parts. The first part shows hierarchically the opened XML document. The second part shows the list of templates. The third part shows applied templates and the last part shows the final presentation and the XSLT code. Dragging and dropping XML elements in the final presentation achieves the creation of new templates. The authoring of the template content, which is HTML, is done in a separate WYSIWYG view. Although this tool allows the generation of XSLT transformation sheets in a convenient way, it does it in a very limited way. A limited part of the XSLT expressiveness is considered. In particular, this tool does not allow nesting of templates: templates are applied in sequence only. Therefore, it is not possible to create presentations for recursive document models. Moreover, the transformation process is executed from scratch after each modification of the transformation sheet. The result is an increasing processing cost proportional to the size of the document. In [9] is presented a method for generating XSLT transformation sheets by demonstration. The source document and the target document are both HTML. The method relies on the recording of user interactions when he/she edits an HTML document. At the end of the recording, transformation rules (or instructions) are generated from the history of user interactions. This method has the following drawbacks. First, the source document and the target document belong to the same document class. And secondly, as this method relies on demonstrational interfaces, it inherits its drawbacks. In particular, demonstrational interfaces do not provide a static representation of the program. Therefore, the reuse, the modification and the revision of the program are not possible [10].

2.2 Authoring of document models and database connection

The authoring of transformation sheets is similar to the editing of presentation models and the connection with a database. Microsoft Office allows the creation of presentation models based on style sheets. The content of the model can be filled manually by the author or generated automatically from a relational database. The selection of data is made through several forms. As we said previously, editing style sheets is easier than editing transformation sheets.

Macromedia Dreamweaver and UltraDev allow the creation of presentation models for HTML documents. UltraDev allows the creation of JSP[4] and ASP[5] code by direct manipulation. The edition of HTML pages is achieved through three views: the design view, the live data view and the code view. The design view shows language instructions, such as database queries. The live data view shows the final presentation with real data. The code view shows the ASP/JSP code embedded inside the HTML code. The authoring of SQL queries is performed externally either through wizards or using a textual view. When a query is dragged and dropped inside the design view, the corresponding JSP or ASP code is generated. Although ASP or JSP languages have some similarities with XSLT language, they are different. In particular, XSLT is a rule-based language and it applies on XML documents, while ASP/JSP code is applied on database. Macromedia Ultradev does not need to deal with templates and contextual queries. Moreover, the generation does not deal with multiple dimensions (spatial, temporal, etc.). It simply generates HTML documents.

[4] Java Server Pages, Specification available at http://java.sun.com/products/jsp/index.html

[5] Active Server Pages, http://msdn.microsoft.com/workshop/server/asp/ASPover.asp

2.3 Synthesis

None of the previous tools allows the friendly authoring of transformation sheets. Either they are difficult to use or they handle a subset of the transformation languages. In particular, they do not take into account the capability to transform recursive documents (for example a list inside a list). At the XSLT level, it is translated by the incapacity to handle relative expressions. Another lack of the friendly tools is their incapacity to include in a same tool specific modifications and generic modifications. In particular, the live data view of Macromedia UltraDev considers the ASP/JSP code as monolithic: no specific modification is allowed. Moreover, all of theses tools allow only the generation of HTML documents. None of the tools provide a way to edit fully XPath expressions. Finally the time of transformation executions (and also the execution of the ASP/JSP code) prevents the creation of reactive authoring tools. In this paper, our contribution is to propose a complete authoring tool that handles any kind of XML documents, in particular recursive documents, and that produces multimedia documents. This authoring tool relies on our incremental transformation processor that we have extended in order to handle transformation sheets modifications.

3 Context of work

3.1 The Kaomi multimedia authoring tool

Kaomi [8] is an authoring tool that allows the creation and the modification of multimedia documents. The edition of multimedia documents is achieved through multiple views of the document. The main view, called *execution view*, allows playing the document. Various other views convey comprehending information on the document: its structure, its temporal scenario, its content, etc. These views can support editing actions and can be synchronized on object selection. For example the author can directly change the layout of the presentation in the execution view. In the remainder of this paper, such views are called *target views*. They show the result of the transformation process: the target document. Moreover, Kaomi allows the presentation of XML documents that belong to classes. Typical examples of document classes are tourist guides, slideshow presentations, technical documentation (for installation and maintenance), courseware and photo album. The presentation of XML documents is achieved by transforming the document to presentation document models such as Madeus [7] (c.f. the next section). The authoring of XML documents is performed in *source views* and also through target views. The goal of this paper is to reuse the same authoring paradigm for editing XSLT transformation sheets.

3.2 The Madeus model

The Madeus model is a model for describing multimedia documents. It allows composition of media objects (text, audio, 3D animation, etc.) in temporal, spatial and hypermedia dimensions. The description of composition is distributed over four modules: general definitions module, media definition module, temporal scenario module and spatial layout modules [16]. Its syntax is formally described as a XML DTD and therefore it takes full advantage of all XML existing tools. The DTD itself can be found at [15].

In order to simplify the description of rules generation, we use a subset of the Madeus model composed only of the media and the spatial modules. The media definition module allows the declaration of objects that belong to the presentation. The spatial layout module allows the organization of object in space. These two modules define two independent hierarchical structures. Links between these structures are made by ID references. Principles of rules generation presented in this paper can easily be applied to the rest of Madeus modules. More generally these principles can be applied to other presentation languages such as SMIL, XSL-FO, etc.

3.3 Photo album model

The source document model used throughout this paper is the photo album model. This model represents a set of photos gathered in albums (cf. Figure 1). Each album is composed of meta-data gathered inside the header element and a list of photos. Meta-data are the title, the date of creation and the author of the album. A photo is characterized by the src attribute that references a file name containing photo data. The aperture and the speed of diaphragm used during the photo shot can eventually be specified using aperture and speed attributes. The location where the photo has been taken is given by the location element. Textual and/or audio comments about photo are added inside the comment element. The content of textual comments can contain paragraphs (para element) and lists of items (list, listitem elements). Nested lists are also allowed.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<album>
    <header>
        <title>Vacancy in Corsica</title>
        <date>June 2000</date>
        <author>Lionel Villard</author>
    </header>
    <photo src="Corse/p1.gif" speed="11" aperture="3">
        <author>Lionel Villard</author>
        <location>In the ferry boat</location>
        <comment>
            <para>First contact with the corse isle</para>
        </comment>
    </photo>
    <photo src="Corse/p2.gif" speed="auto" aperture="auto">
        <comment src="Corse/p2_comment.mp3"/>
    </photo>
</album>
```

Figure 1. An instance of photo album model.

3.4 Incremental transformations

The authoring of transformation sheets can be interactively achieved only if results of transformation modifications are visualized instantaneously. Batch transformation is resource intensive and therefore is not suitable for interactive authoring. The best way for updating transformation result is to use an incremental version of transformation process [17]. The core of this processor is the internal representation of the transformation execution: the execution flow tree (see Figure 2). The execution flow tree structure is composed of so-called *execution nodes*. Execution nodes of type **flow** (apply-templates, for-each, if, etc.) contain the value (as hierarchical links) of their associated expression. For instance, apply-templates execution nodes have template nodes as direct children. Template nodes, and only these nodes, have links to the source nodes. Therefore, from apply-templates nodes we can retrieve source nodes that compose the context node list. **Producer** nodes (value-of instruction, literal

elements such as Region element) contain data related to the target tree. This data will be use to restore the target context. For example, the literal result elements (Region and Flow elements in the figure) have a link to the element it generates. For a character producer such as value-of instruction, only the number of generated characters needs to be stored.

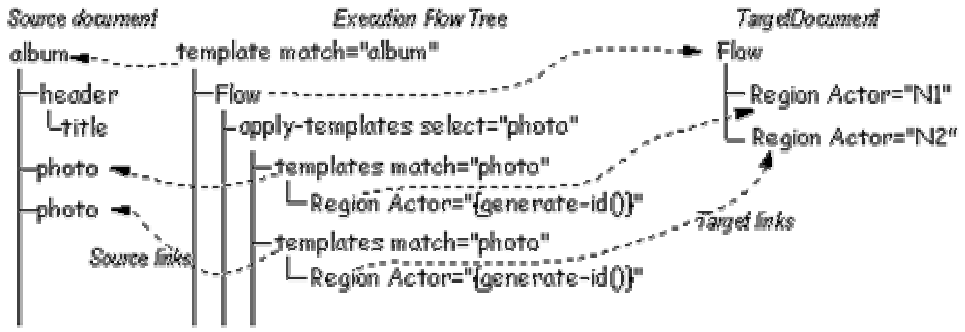

Figure 2 Fragment of the execution flow tree for photo album transformation.

This data structure is widely used, in particular, for updating the target document after transformation modifications. In this paper, we consider two kinds of modifications:

1. **Instructions (other than template) addition:** when an instruction is added in a template, the target document can easily be updated thanks to the execution flow tree. This is done in three steps:
   1. Execution nodes corresponding to the parent of the newly inserted instruction are found.
   2. For each of the execution nodes, the processor context and the target context are restored (c.f. [17])
   3. The newly inserted instruction is then executed. When this instruction is a producer instruction, the target document is updated.
2. **Template instruction addition:** when a template node is added, it's necessary to re-consider all previous template instantiations. For each node selected by apply-templates instructions, a template is sought. In [17] is described an optimization that minimizes the number of such re-considerations.

As we will see later, the execution flow tree will be used for helping the generation of transformation instructions.

4 Use case study for authoring a photo album presentation

This section gives an illustrative use case study for editing transformation sheets by direct manipulation. The main goal of this use case study is to present editing operations from the author point of view.

4.1 General approach

The authoring of presentations is achieved for an XML instance of a document model. The author edits multimedia presentations directly in target views. The authoring of XPath expressions is performed on the hierarchical view of the XML document. The insertion of XPath expressions in transformation sheets is achieved by dropping expressions in one of the target views. In this paper, we do not consider a more generic approach consisting in authoring transformations from document models. Indeed, most of the XML documents do not require having a formal definition of document type (DTD). However, authors of these documents want to create complex presentations and reuse them for similar documents. Moreover the using of document instances allows the visualization of the presentation obtained by transformations. We expect that this visualization simplifies the editing of transformation sheets. We propose a multimedia authoring tool for such requirements.

4.2 Creation of photo albums

The Figure 3 illustrates the multimedia presentation that we want to create. The presentation is composed of two parts. On the left a list of photo thumbnails is presented vertically. The right part presents a photo in their original size. At the bottom of the right part is displayed textual data about the current photo: author, aperture, speed, date and filename. Navigation buttons are provided in order to see the preceding or the following photo.

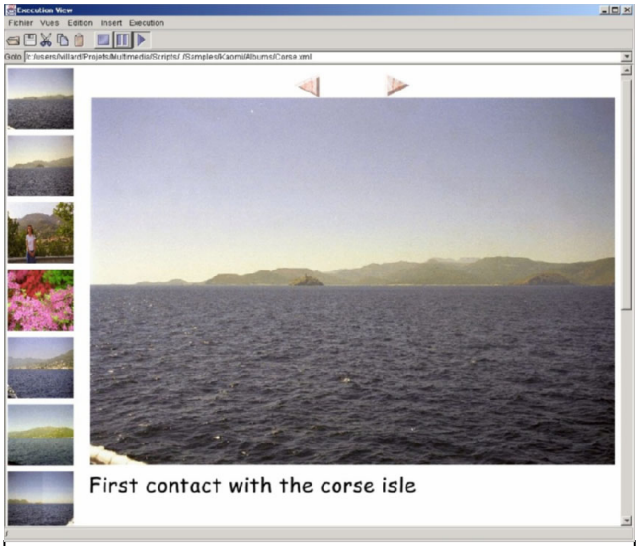

Figure 3 A presentation for the photo album model.

To build this presentation, the author starts by creating a new empty presentation. In the execution view, he inserts two spatial groups inside the root region, one for the left part and one for the right part. Then he sets the right spacing by inserting a spatial relation between these groups.

In the left part he inserts an empty image and opens attributes view of this image. This view shows in particular the FileName attribute of the newly created image. In order to set this filename

attribute, the author opens the source view. In this view, he selects the src attribute for all photos (that correspond to the photo/@src expression) and drags it in the filename field. As the evaluation of the expression returns several src attributes, the authoring system generates a for-each instruction in order to produce an image for each photo.

By default, when a region contains an image, the region size corresponds to the intrinsic dimension of the image. When the author wants to create photo thumbnails, he must resize each generated region. The resizing of the first (or other regions) is applied only on this region. He can repeat this operation for each generated region, but it is tedious. Moreover this operation can be performed only for the photos in the document instance. Therefore, the execution view must visualize the presentation in a more generic way (c.f. Figure 6). The author perceives just one region augmented by a header representing the repetition of photos. When the author resizes the region in this view mode, each region generated by the photos is resized.

In order to create the presentation of the right part, the author proceeds in the same way. He uses alternatively the specific and the generic representation of the presentation. To create the presentation of item lists, the author add a spatial group in the presentation. Then he drops the list expression inside the new spatial group. He turns on the generic mode, adds a new spatial group and drops the listitem expression in the spatial group. The authoring system generates a spatial group for each listitem as described previously. In order to generate nested lists, the author selects a header that represents the repetition of the list expression. Then he drags and drops it in the spatial group generated by the listitem elements. The authoring system generates a template for listitem elements and applies the template for the first level and the others levels of list elements.

The result of this use case study is a complete multimedia presentation. This presentation can be reused for each document that belongs to the photo model. Moreover, the generated transformation sheets can be easily modified in order to produce presentations adapted to other devices. In this section we have presented the editing of transformations from the author point of view. In the next section, we describe the generation of rules from the developer point of view.

5 Rules generation

In this section, we start by giving a set of rule patterns to apply when the author makes a particular modification in a target view. Then we propose a graphical user interface to edit XPath expressions. The third section describes the modifications to apply in transformation sheets when the author drops an expression in a target view. As the transformation specification is visualized through multiple views, the last section shows how to keep the selection synchronization between these views.

5.1 Rules generation from target views

As illustrated in the use case study, the author can edit transformations directly in a target view. The authoring is achieved using classical multimedia editing operations. For example, the author can add a video object, modify the content of a text item, change the style of an object, remove a temporal relation, etc. The result of these authoring operations is a set of transformation rules. In this section, we study the different cases of rules generation when the author performs such authoring actions. Most of the time, these actions are performed in a formatted view of the document. In this paper, we do not cover the conversion from the formatted view to the initial specification of the multimedia document. A number of previous works have covered this subject [4].

With the currently available tool, the author can edit the presentation in a specific way. Target views show the result of transformation applied to a particular source document instance. Although some authoring operations need to be done in such views, they are not sufficient to perform more complex editing operations. For example, the author cannot add a media object inside all groups generated from each photo. Therefore, the authoring tool must provide a more generic view that allows the visualization of transformation instructions. This visualization is achieved by extending existing views and/or by creating new views, as we will see later. The extension is done by providing two visualizations mode: the specific mode and the generic mode.

In the following sections, we firstly describe what rules are generated when the authoring occurs in target views turned in specific mode. Then we describe how generic target views are built and we give rule patterns to apply when the editing is achieved in this kind of views.

5.1.1 Specific views

Specific views show the target document resulting from transformation. Each modification on this view is converted to transformation instructions that allow meeting the author intents. For example, when the author adds an image in the multimedia presentation, say in the fourth spatial group, an image element is added in the transformation sheet and a region element is added as child of the selected spatial group (see Figure 4). As views are specific, the author expects to make a specific action. In the previous example, the Figure 4b illustrates a bad generation: a region is added for *all* photos and not for the fourth photo. Indeed, as a spatial group is generated for each photo, the selected spatial group must be firstly isolated. This separation is realized by using the pattern illustrated Figure 4c. This pattern is called the *isolation pattern*.

```
<xsl:template match="album" mode="ac">
</xsl:template>

<xsl:template match="album" mode="sp">
    <xsl:for-each select="Photo">
        <S-Group/>
    </xsl:for-each>
    …
```

(a)Before authoring

```
<xsl:template match="album" mode="ac">
    <Image ID="image"/>
</xsl:template>

<xsl:template match="album" mode="sp">
    <xsl:for-each select="Photo">
        <S-Group>
            <Region Actor="image"/>
        </S-Group>
    </xsl:for-each>
```

…

(b) Bad generation

```
<xsl:template match="album" mode="ac">
    <Image ID="image"/>
</xsl:template>

<xsl:template match="album" mode="sp">
    <xsl:for-each select="Photo">
    <xsl:choose>
        <xsl:when test="position()=4">
        <S-Group>
                <Region Actor="image"/>
        </S-Group>
        </xsl:when>
        <xsl:otherwise>
            <S-Group/>
        </xsl:otherwise>
    </xsl:choose>
    </xsl:for-each>
…
```

(c) Good generation

Figure 4 Result of the generation after the addition of an image object in a specific view.

The generation of the isolation pattern needs to compute the condition that allows the separation of the rule. This separation is needed only when the transformation rule can be generated many times. In XSLT, multiple generations occurs only when an instruction is inside a for-each or a template element. For each of these instructions that are ancestor of the selected rule, the position of context node is computed. The only way to perform this computation is to retrieve the corresponding execution flow nodes (c.f section 3.4). Indeed, only this structure contains the list of selected nodes.

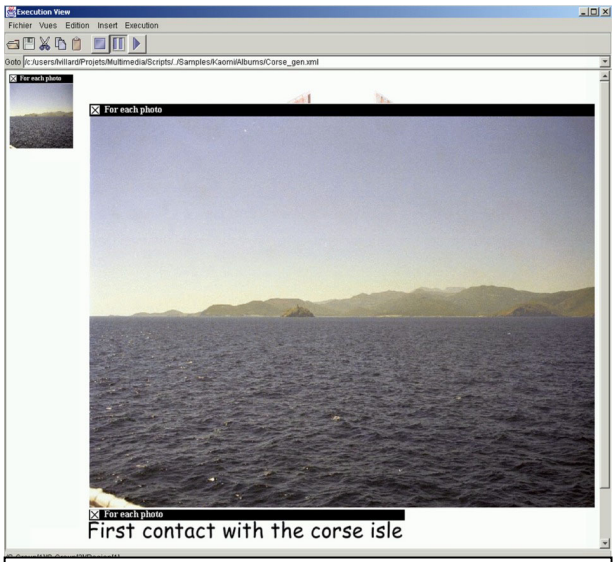

Figure 6. Generic mode representation of the execution view.

5.1.2 Generic views

The generic views show the transformation specification in a semi formatted manner (see Figure 6). This kind of views is built as follows: the instructions for-each and apply-templates are applied only for the first selected node. The instructions that generate characters, such as value-of and text, generate characters that represent the instruction. For example, the expression value-of generates the expression attached to it. When the mouse cursor is over a target representation, a header is shown. This header contains the representative name of the instruction and the expression attached to the instruction (if it exists). The author can change the authoring mode from generic to specific by clicking on the swap button. While target views are synchronized, the header is shown in all views. Moreover, inside a same view, the header can appear many times. For example, when the body of a for-each instruction contains two spatial groups, then two headers are attached to these two groups.

The generation of rules in generic views is easier than in its specific counter-part. However, as the target document has many dimensions, the difficulty is to generate the links between these dimensions. In Madeus, multi-dimensions links are represented by unique identifier (UID) references. Static identifiers are not enough to guarantee the uniqueness. The generation of UID rely on the generate-id() function that generates an UID for a specific source node.

```
<xsl:for-each select="photo">
    <Region/>
</xsl:for-each>
```

(a) Before

```
<xsl:for-each select="photo">
    <Image ID="A-photo{generate-id()}"/>
</xsl:for-each>

<xsl:for-each select="photo">
    <Region Actor="A-photo{generate-id()}"/>
</xsl:for-each>
```

(b) After

Figure 5 Result of rules generation after the insertion of an image in generic-mode execution view.

5.1.3 Mixing representations

Most of editing operations requires some parts of the multimedia presentation to be specific and other parts to be generic. For example, when two for-each instructions are nested, the author can add an object inside all elements generated by the first for-each instruction and just for the third element generated by the second for-each instruction. Kaomi allows the swapping between the specific representation and the generic representation for fragments of the target document.

5.2 Authoring of location path expressions and patterns

Authoring expressions are achieved in hierarchical source view. The figure 5 shows the authoring of

/album/photo/descendant::author expression. The user edits the expression by selecting one or many elements or attributes in the hierarchical source view. He can add constraints on each step of the expression using the filter dialog. The status bar shows the expression currently being edited. The expression can also be edited directly in the status bar.

In order to shorten the edition of expressions, two authoring modes are provided: the single-selection mode and the multiple-selection mode. The single-selection mode allows the selection of a particular node. For instance, when the user selects the second photo element in the source view, the /album[1]/photo[2] expression is generated. In contrary, the multiple-selection mode generates expression that potentially selects a set of nodes. In this mode, when the user selects the second photo element, the photo

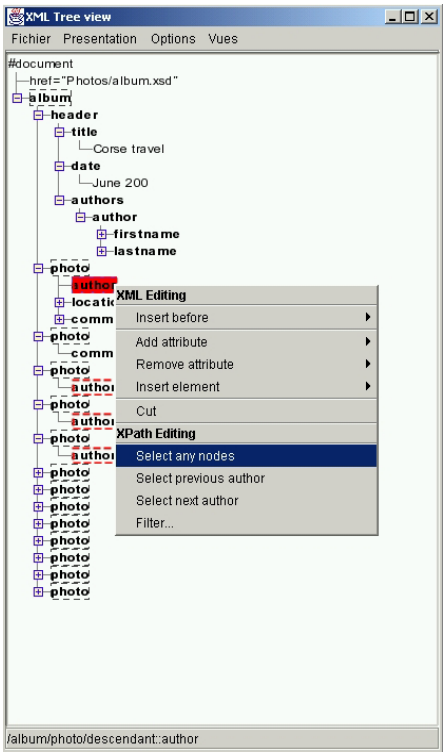

Figure 7 Authoring of expressions in hierarchical source view

expression is generated. Independently from theses modes, the author can restrain or extend the node selection by applying editing operations described below.

The generation of expression depends on the key modifiers used during the selection.

No modifier: the expression is re-initialized. The generated expression depends on the current mode. In single-selection mode, the expression that selects the node is generated. It's an absolute expression. In multiple-selection mode, the expression consists of just one step. This step is either the selected node name (for element node type) or the selected node type (for other node type).

Control modifier: a step is added (or removed) to the current expression. Depending on the current selection position, the choice of the axis between the previous step and the new step follows these rules:

- The level of current selection is the same as the previous selection level. If the current selection is after the previous selection, the authoring system generates either a following-sibling axis or a following axis. In contrary, if the current selection is before the previous selection, either a previous-sibling axis or a previous axis is generated.
- The difference between the level of current selection and the level of the previous selection is equal to one. If the current selection belong to the same path of the previous selection, two cases occurs:
  - When the current selection is before the previous selection, parent axis is generated.
  - Otherwise, child axis is generated
- There is a difference of two or more levels between the level of the current selection and the level of the previous selection. This case is similar to the previous one. The descendant axis is generated in place of the child axis and the ancestor axis is generated in place of the parent axis.

Shift modifier: a union operator is created.

More than visualizing the expression that is being edited, this view shows the result of the expression evaluation. The nodes that belong to the result of the expression are shown with a red box around the nodes representation. Intermediate nodes resulting from expression steps are also shown using black boxes. When the expression is not absolute, the user can select the context node for testing the expression.

With this view, the author can edit simply any kind of location path expressions. Moreover he can check through an example the result of the expression.

5.3 Generation of rules from expressions

After the creation of an expression in a source view, the author can drop this expression somewhere in a target view. In this section, we describe what transformation rules are generated after this kind of editing operation. A basic rules generator is firstly described. Then we give some techniques that increase the power of the basic rule generator: the creation of templates and the computation of similar expressions.

5.3.1 Basic rules generator

When the author drops a source expression in a target view, he identifies *a target node* and a *position* inside the target node. When the dropping destination is near the borders of the target node representation, the generator marks the target node as the destination of the dropping. Otherwise, the generator marks the content of the target node as the destination of the dropping. For example, when in the use case study the author drops the expression @src inside the filename field and "far" of the borders, the value of the attribute FileName that belong to the image element is marked as the dropping destination. The owner template matches any album elements.

From the target node, the context source node of the template can be easily retrieved. This source node is used to evaluate the source expression. The rule generator produces different rules depending on the result of the source expression evaluation:

1. The source expression generates no node. An error is

displayed and nothing is generated.

2. The source expression generates only one node. The following cases occur depending on the type of target node and the position inside of the target node:
   1. The target node is the value of an attribute. The generator adds an attribute value template (AVT) containing the evaluation of the source expression. For example:

<Image FileName=””/>

(a) Before

<Image FileName=”**{photo[1]/@src}**”/>

(b) After

   2. The target node is the content of an element. The generator adds a new object inside the target element or modifies an attribute of the target element. The type of the new object depends on the nature of the target element and the source node. For example, when the value of the source node references an image file and when the target element is a spatial group, the generator adds a new image media inside the spatial group.
   3. The target node is an attribute or an element. The generator adds a test based on the source expression for generating the target attribute or element.
3. The source expression generates more than one node. The following cases depending on the type of target node and the position inside of the target node:
   1. The target node is the value of an attribute. A for-each instruction is generated. The select attribute contains the fragment of the source expression that generates many nodes. The content of the for-each instruction is an attribute value template (as describe previously). For example, the following generation after the author drops the photo/@src expression in filename field:

```
<xsl:template match="album" mode="ac">
    <Image ID="A-1"/>
</xsl:template>

<xsl:template match="album" mode="sp">
    <S-Group>
        <Region Actor="A-1"/>
    </S-Group>
</xsl:template>
```

(a) Before

```
<xsl:template match="album" mode="ac">
    <xsl:for-each select="photo">
        <Image ID="A-1{generate-id()}"
            FileName="{@src}"/>
    </xsl:for-each>
</xsl:template>

<xsl:template match="album" mode="sp">
    <S-Group>
     <xsl:for-each select="photo">
      <Region Actor="A-1{generate-id()}"/>
     </xsl:for-each>
    </S-Group>
</xsl:template>
```

(b) After

   2. The target node is an attribute. An error is displayed. The XML language enables to set just one attribute with the same name on an element.
   3. The target node is the content of an element. When target element is a composite, the generator produces many new objects. Rules to choose the type of objects are the same that those presented previously (case 2.2). When the target element is a leaf, the previous rules cannot be applied for the same reason invoked in the previous case. An error is then displayed.
4. The target node is an element. The target element is encapsulated inside a for-each instruction.

The previous generation patterns can be applied for the two target authoring modes (generic/specific). When the part of view is in specific mode, the isolation pattern (c.f. section 5.1.1) is applied before the previous patterns.

Most of the time, as the target document describes multimedia presentations, rules are created in several templates with different modes. These rules are strongly linked. The modification of rules in a particular template implies the modification of linked rules in other templates. For example when the author changes the photo expression in the previous example (the case 3.1) to another one, the two select attributes in the two templates are changed.

By default, the basic generator generates for-each instructions rather than template/apply-templates instructions for performance reasons. The creation and the application of template instructions are presented in the next section.

5.3.2 Generation and application of template instructions

A natural way to generate template instructions is to convert a for-each instruction to an apply-templates/template instruction couple. The select attribute of the apply-templates instruction is the same as the select attribute of the for-each instruction. In order to generate the template instruction, the for-each expression must be converted to a pattern. The result pattern matches at least all the nodes selected by the expression. The template mode is the same as the template that contains the for-each instruction. This conversion must be performed for all for-each instructions linked together (c.f. previous section).

This conversion is performed by dragging and dropping the header of a for-each instruction to another location in space and time. While the presentation is dynamic, this location can be not visible. That is why a template view is provided. This view allows the visualization of the existing templates. The author instantiates a template by dragging and dropping a template in a target view. Moreover he can create new templates by dragging and dropping a pattern in the template view.

5.3.3 Generation of relative and similar expressions

In the target views, the author has not a direct knowledge of the template context. Therefore, it is not easy to create the exact expression to use in a particular context. To add more flexibility during the dropping of an expression in a target view, the authoring tool can calculate some expressions that are close to the

initial one. This happens when nothing is produced in the destination context. For example, nothing is generated by the @src expression when it is dropped in a spatial group generated by a template that matches album elements. The authoring tool can propose to drop the photo/@src or the photo/comment/@src expressions rather than the @src expression. These expressions are deduced from the photo album model. This model allows the specification of the src attribute only on photo and comment elements. The computation of similar expressions is relatively complex. It relies on the analysis of transformation sheets and document models. In our paper, we don't describe such computations.

5.4 Selection synchronization

The synchronization on object selection is a fundamental concept when the authoring is performed on multi-views. Indeed, the drawback of multi-views is the partial and the fragmented perception of object properties. The synchronization allows the visual grouping of these properties. The Kaomi authoring tool already allows the synchronization between targets views. We have extended this synchronization between source views and target views.

When the author selects an object in a view, one or more document nodes are selected. The synchronization is performed for each of these nodes. The realization of the synchronization depends on the document selected:

> A target node is selected: the execution flow tree contains references to target nodes. Therefore, the corresponding execution nodes can be easily retrieved. Each expression of these execution nodes is evaluated. The result is the list of source nodes to synchronize.
> A source node is selected: the system must find all expressions that return the selected source node. The result of expression evaluation is not conserved for memory consumption reasons. Therefore all the expressions of transformation sheets must be evaluated. Note that such computations can require a lot of time.

6 Conclusion and perspectives

In this paper, we have presented a complete method for authoring XSLT transformation sheets by direct manipulation. This method has been implemented into the Kaomi multimedia authoring tool. The source view has been extended in order to edit XPath expressions. The editing of transformations has been implemented only for the execution view. This view provides a specific representation of the transformation. It provides also a generic representation for a subset of the XSLT language (for-each and template instructions). The author can drag and drop expressions in the specific and the generic representations. The template view has been also implemented.

Compared to existing authoring tools, our proposal go further in several points. First we propose a convenient user interface for editing any XPath expression. This GUI is not based on forms as <xsl>Composer. Second, we allow the editing of transformations using both a specific and a generic representation of the generated document. None of the existing tools allows specific modifications of the transformation sheets.

We propose a set of rule patterns that allows specific modifications. The generic representation is similar to the design view of UltraDev. We have reconsidered this view in order to take into account the expressiveness and the hierarchical structure of XSLT. Furthermore, while all existing tools deal with HTML target, we have considered a broader use of transformations through the production of multimedia documents. The temporal dimension of multimedia documents has an important impact on the authoring tool, in particular on the generic representations.

In short term, we plan to extend the implementation to the other views of Kaomi. In particular, the timeline view that allow the visualization and the editing of the temporal scenario. Moreover, we plan to consider the modification and the removal of transformation rules. We want also to extend the coverage of the XSLT language to the modularization instructions and the variables instructions. Another point is to provide a set of services that helps the author. In particular, the computation of similar expressions can add some flexibility during the editing. Moreover, it is well known that the knowledge of document models increases the efficiency of authoring XML instances. How that can be applied to the authoring of XSLT transformations, this remains an open question.

References

[1] ActiveState, *"Visual XSLT"*, http://aspn.activestate.com/ASPN/Downloads/VisualXSLT, 2001.

[2] Excelon, "Stylus Studio", http://www.stylusstudio.com/, 2001.

[3] *"Extensible Stylesheet Language (XSL) Version 1.0"*, S. Adler and Co, W3C Working Draft, available at http://www.w3.org/TR/xsl/, 21 November 2000.

[4] Richard Furuta, Jeffrey Scofield and Alan Shaw, *"Document Formatting Systems: Survey, Concepts and Issues"*, ACM Computing Surveys, Vol. 14, N°. 3, pp. 417-472, September 1982.

[6] Masahiro Hori, Goh Kondoh, Kouichi Ono, Shin-ichi Hirose, and Sandeep Singhal *"Annotation-Based Web Content Transcoding,"* In Proceedings of Ninth International World Wide Web Conference, Amsterdam, Netherlands, 15-19 May 2000.

[7] Muriel Jourdan, Nabil Layaïda, Cécile Roisin, Loay Sabry-Ismail, Laurent Tardif, "Madeus, an Authoring Environment for Interactive Multimedia Documents", *ACM Multimedia'98,* pp. 267-272, ACM, Bristol (UK), September 1998.

[8] Muriel Jourdan, Cécile Roisin, and Laurent Tardif, "A Scalable Toolkit for Designing Multimedia Authoring Environments", Special number, Multimedia Authoring and Presentation: Strategies, Tools, and Experiences of Multimedia Tools and Applications Journal, Kluwer Academic Publishers, 1999.

[9] Teruo Koyanagi, Kouichi Oni, and Masashiro Hori, *"Demonstrational Interface for XSLT Stylesheet Generation"*, Graphic Communications Association, Extreme Markup Languages Conference, 17 August 2000.

[10] Brad A. Myers, *"Demonstrational Interfaces: A Step Beyond Direct Manipulation"*, IEEE Computer, 25(8), pp. 61-73, 1992.

[11] *"Namespaces in XML"*, T. Bray, D. Hollander, A. Layman, W3C Recommendation, available at http://www.w3.org/TR/REC-xml-names, 14 January 1999.

[12] Oratrix, *"GRiNS"*, http://www.oratrix.com/, 2001.

[13] Emmanuel Pietriga, Vincent Quint and Jean-Yves Vion-Dury, "VXT: A Visual Approach to XML transformations", submitted to Symposium on Document Engineering, 2001.

[14] SoftQuad, *"XMetal 2.0"*, http://www.xmetal.com/, 2000.
[15] Lionel Villard, *"Madeus model DTD"*, available at http://www.inrialpes.fr/opera/madeusmodel.dtd, 2000.
[16] Lionel Villard, Cécile Roisin and Nabil Layaïda, *"An XML-based multimedia document processing model for content adaptation"*, Digital Documents and Electronic Publishing (DDEP00), September 2000.
[17] Lionel Villard and Nabil Layaïda, *"iXSLT: An Incremental XSLT Transformation Processor for XML Document Manipulation"*, submitted to World Wide Web Journal, Kluwer publisher, 2001.
[18] W3C, "Amaya", http://www.w3.org/Amaya/, 2001.
[19] WhiteHill, *"<xsl>Composer"*, http://www.whitehill.com/, 2001.
[20] "XML Path Language (XPath)", J. Clark and S. DeRose, W3C Recommendation, available at http://www.w3.org/TR/xpath.html, 16 November 1999.
[21] *"XSL Transformations (XSLT)"*, J. Clark, W3C Recommendation, available at http://www.w3.org/TR/xslt, 16 November 1999.