Printing Structured Text without Stylesheets*

Helena Ahonen[1], Barbara Heikkinen[2],
Oskari Heinonen[1], and Mika Klemettinen[1]

[1] University of Helsinki, Department of Computer Science,
P.O. Box 26 (Teollisuuskatu 23), FIN–00014 University of Helsinki, Finland
Phone: +358 9 1911, Fax: +358 9 191 44441
[2] Nokia Research Center, P.O. Box 407 (Itämerenkatu 11–13), FIN–00045 Nokia Group, Finland

Abstract. As more and more XML documents start to appear, e.g. on the WWW, the users face a new problem: opposite to HTML tags, XML tags do not tell the semantics of a structure element. This means that if a document does not come with layout, e.g. XSL or CSS, specifications, it is not easy to say how the document should be formatted for presentation in print or on screen. In this paper we describe a tool and a process with which a document without any stylesheets or styling information can be automatically transformed to be viewed with different target media (paper, WWW browser, WAP phone, etc.). Our approach is based on DTD generalization and element mapping, and transformation and styling with XSLT. We give hands-on examples of the automatic transformation process pipeline from the original stylesheetless document to the transformed result document with media-specific styling, and show that the approach works in practice.

Prof. Helena Ahonen-Myka, Ph.D. Mika Klemettinen, Ph.D. Barbara Heikkinen, and M.Sc. Oskari Heinonen have their backgrounds in structured documents and data mining. Ahonen-Myka got her Ph.D. in 1996 with a thesis titled "Generating grammars for structured documents using grammatical inference methods". Klemettinen obtained his Ph.D. in 1999; his thesis was about knowledge discovery in telecom area. Heikkinen got her Ph.D. in 2000 with a thesis titled "Generalization of Document Structures and Document Assembly". Heinonen is finishing his Ph.D. studies. All the authors except Heikkinen work currently at the Department of Computer Science at the University of Helsinki; Heikkinen is with the Nokia Research Center in Helsinki.

Keywords: XML, XSL, XSLT, CSS, transformation, stylesheets

1 Introduction

As more and more XML documents start to appear, e.g. on the WWW, the users face a new problem: opposite to HTML tags, XML tags do not tell the semantics of a structure element. This means that if a document does not come with layout, e.g. XSL or CSS, specifications, it is not easy to say how the document should be formatted for presentation in print or on screen.

We have developed a tool which automatically creates a stylesheet for a structured document given [1, 2]. In the first phase, the tool studies XML element structures and finds a general meaning, like section, paragraph, list, or title, for each element. Then the element tags are replaced with the respective generic class names (or augmented with this information in an attribute). The set of generic classes is fixed. Hence, it has been possible to define layout specifications for each generic class. In the second phase, the tool chooses the best style definition for each element of the XML document, taking into account the generic class and context of the element, as well as the requirements of the target media. For instance, an element classified as a paragraph is presented in a different way when it appears within a picture caption on screen than when it appears within a footnote in paper.

In addition to the need of fast, online formatting of unknown documents, as described above, the tool can also facilitate the design process of layout specifications for document types. In this

* Authors' e-mail addresses: *{Helena.Ahonen-Myka, Oskari.Heinonen, Mika.Klemettinen}@cs.Helsinki.FI*, *Barbara.Heikkinen@nokia.com*. This article has been presented in XML Scandinavia 2000, Gothenburg, Sweden, May 2-4, 2000.

case, the original element names are not replaced with the generic classes, but the layout specification of a respective generic class is used as a basis for the layout specification for the original element. The designer of the layout can then easily see the result and make the necessary modifications to the draft specifications, without the need to write the specifications from scratch.

This paper is organized as follows. In Section 2 we first introduce the methods we use in the automatic transformation process, namely generalization and element mapping (Section 2.1), and transformation and styling with XSLT (Section 2.2). Then in Section 3 we discuss the automatic transformation process pipe from the original stylesheetless document to the transformed result document with media-specific styling. Finally, Section 4 is a short conclusion.

2 Methods

The core of our automatic transformation process, element mapping, is described in Section 2.1. The transformation and styling phases with XSLT are then discussed in Section 2.2.

2.1 Generalization and Element Mapping

Our approach is based on the fact that although no semantics can be inferred from the names of XML elements, some knowledge can be extracted from the elements: which other elements an element can contain, and how long the text contents of the element and its subtree are on average. This information is extracted from the document instances, which may be the only source available. Various *components*, like strings, paragraphs and sections, as well as various *structures*, like element hierarchies and element containers wrapping logical components together, can be recognized. Using the extracted knowledge each element type is mapped to some predefined element type called *generic class*. This mapping process is called *element-type classification*.

Each generic element has an element type definition, which describes the structure of the element type in terms of other generic classes. The definitions of the generic classes constitute a generic DTD (see Figure 1).

The basic elements of the generic hierarchy are *sections* (Section), *paragraphs* (Para) and *strings* (String). Typically, strings are short sequences of characters, paragraphs consist of a number of strings, and sections in turn consist of paragraphs and other sections. Some container elements without any text content of their own are also needed to wrap a logical unit together. For example, a typical list structure in a document consists of items, which consist of one or several paragraphs. In this case, the generic container structure would be a *paragraph group* (ParaGrp) consisting of *paragraph containers* (ParaCont), which in turn consist of one or several paragraphs. The corresponding container structures for strings are a *string group* (StrGrp) and a *string container* (StrCont). Furthermore, there are two different types of containers wrapping sections, namely *section groups* (SectGrp) and *string sections* (StrSect).

The basic idea of element-type classification is the following. In the beginning of the classification process information is collected from the document instances of a document class. For each element type in a DTD the following information is extracted: names of the children in the element instances, the average text length of all the element instances, and finally, the average text length of the subtrees of all the element instances. This information is then used to decide which generic class would be closest to the original element type.

The generic DTD presented in Figure 1 is only one solution to describe structures. It is suitable particularly for typical electronic documents such as books, articles, reports, news messages, encyclopedias, and manuals. For other kinds of documents, different generic structures may be better.

Examples of decision rules (maxstr = 60, minsect = 2000):

- **String**

  If ((textlength < maxstr) **and** (subtreelength < minsect)) **then** Element is a String.

  A typical use of String is to emphasize some word in a paragraph. A String may also contain other Strings, and even paragraphs: a string may contain a footnote, which in turn contains a paragraph. The condition for the length of the subtree prevents a String from containing a Section.

– **Para**

If ((maxstr < textlength < minsect) **and** (subtreelength < minsect)) **then** Element is a Para.

The classes String and Para have identical content models in the generic DTD. The basic difference is that Para is a longer sequence of characters than String.

– **ParaCont**

If ((one child is Para) **and** (textlength = 0) **and** (subtreelength < minsect)) **then** Element is a ParaCont.

A container wraps other elements, and it is not allowed to directly contain any text (PCDATA). The class ParaCont must contain paragraphs.

It is common to make a distinction between *block-level* and *inline* elements: block-level elements generally begin a new line, whereas inline elements do not. On the other hand, block-level elements may contain both inline and block-level elements, whereas inline elements contain text and other inline elements. Block-level structures are generally larger than inline structures, because block-level elements are typically interior nodes of the structure tree, whereas inline-elements are often leaves or located near leaves.

Typically, the generic classes String, Fig, Formula, and Ref behave like inline elements, whereas the rest of the generic classes, such as Section, Para, ParaCont, and StrCont, are more clearly block-level elements. However, some elements may act as inline or block-level elements depending on the context. As an example, consider strings: if a string is located inside a paragraph, it is clearly an inline element, whereas if a StrCont has several strings as children, each string should begin a new line.

```
<!ENTITY % sec "Section|SectGrp|StrSect">
<!ENTITY % par "Para|ParaCont|ParaGrp">
<!ENTITY % sml "meta|Empty|Ref|Formula|FormCont|math|Fig|FigCont|
String|Title|StrCont|StrGrp|Unknown">
<!ENTITY % sml2 "meta|Empty|Ref|Fig|FigCont|String|Title|StrCont|
StrGrp|Unknown">
<!ENTITY % sml3 "meta|Empty|Ref|String|Title|StrCont|StrGrp|Unknown">
<!ELEMENT VirtDoc ((#PCDATA)|%sml;|%par;|%sec;|table)*>
<!ELEMENT Unknown ANY>
<!ELEMENT Section ((#PCDATA)|%sml;|%par;|%sec;|table)*>
<!ELEMENT SectGrp (%sml;|%sec;|table)*>
<!ELEMENT StrSect (%sml;|StrSect|table)*>
<!ELEMENT ParaGrp (%sml;|ParaCont|ParaGrp|table)*>
<!ELEMENT ParaCont (%sml;|%par;|table)*>
<!ELEMENT Para ((#PCDATA)|%sml;|%par;|table)*>
<!ELEMENT StrGrp (%sml3;|table)*>
<!ELEMENT StrCont (meta|Empty|Ref|String|Title|Unknown)*>
<!ELEMENT String ((#PCDATA)|%sml;|%par;|table)*>
<!ELEMENT Title ((#PCDATA)|%sml;|%par;|table)*>
<!ELEMENT FigCont (%sml2;|table)*>
<!ELEMENT Fig EMPTY>
<!ELEMENT FormCont (%sml;|table)*>
<!ELEMENT Formula EMPTY>
<!ELEMENT Ref EMPTY>
<!ELEMENT Empty EMPTY>
<!ELEMENT meta EMPTY>
<ENTITY % tableX PUBLIC "-//W3C//DTD XHTML Table Model//EN">
<ENTITY % mathML PUBLIC "-//W3C//DTD Mathematical Markup Language//EN">
%tableX; %mathML;
```

Fig. 1. The generic DTD.

2.2 Transformation and Styling

With the generic, fixed DTD and the decision rules introduced in Section 2.1, each element type of any incoming document can be mapped to some predefined element type. Since all possible elements that may occur in the documents after the element mapping are known, stylesheets (or transformations) can be created for all target media. For each generic DTD, it is enough to create the stylesheet only once for each target media, and then to use it for all incoming documents.

In Figure 2, an example stylesheet (or transformation) excerpt for creating HTML is given.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="html" doctype-public="-//W3C//DTD HTML 4.0//EN"/>
<xsl:template match="*[@METAMAP='VirtDoc']">
<xsl:element name="HTML">
<xsl:attribute name="CLASS">
<xsl:value-of select="@METAMAP"/>
</xsl:attribute>
<xsl:element name="HEAD">
<xsl:element name="TITLE">
<xsl:value-of select=".//*[@METAMAP='Title']"/>
</xsl:element>
</xsl:element>
<xsl:element name="BODY">
<xsl:attribute name="STYLE">text-align: left; margin: 5pt; background-color:
#FFFFFF</xsl:attribute>
<xsl:apply-templates/>
</xsl:element>
</xsl:element>
</xsl:template>
<xsl:template match="*[@METAMAP='Section' or @METAMAP='SectGrp'
or @METAMAP='StrSect']">
<xsl:element name="DIV">
<xsl:attribute name="CLASS">
<xsl:value-of select="@METAMAP"/>
</xsl:attribute>
<xsl:attribute name="STYLE">text-align: left; margin: 15pt</xsl:attribute>
<xsl:apply-templates/>
</xsl:element>
</xsl:template>
...
<xsl:template match="*[@METAMAP='Title']">
<xsl:element name="H3">
<xsl:attribute name="CLASS">
<xsl:value-of select="@METAMAP"/>
</xsl:attribute>
<xsl:attribute name="STYLE">text-align: left</xsl:attribute>
<xsl:apply-templates/>
</xsl:element>
</xsl:template>
...
```

Fig. 2. An excerpt from an XSLT stylesheet for generating HTML.

3 Automatic Transformation Process

The automatic transformation process pipe from the original stylesheetless document to the transformed result document with terminal-specific styling goes as follows (see Figure 3):

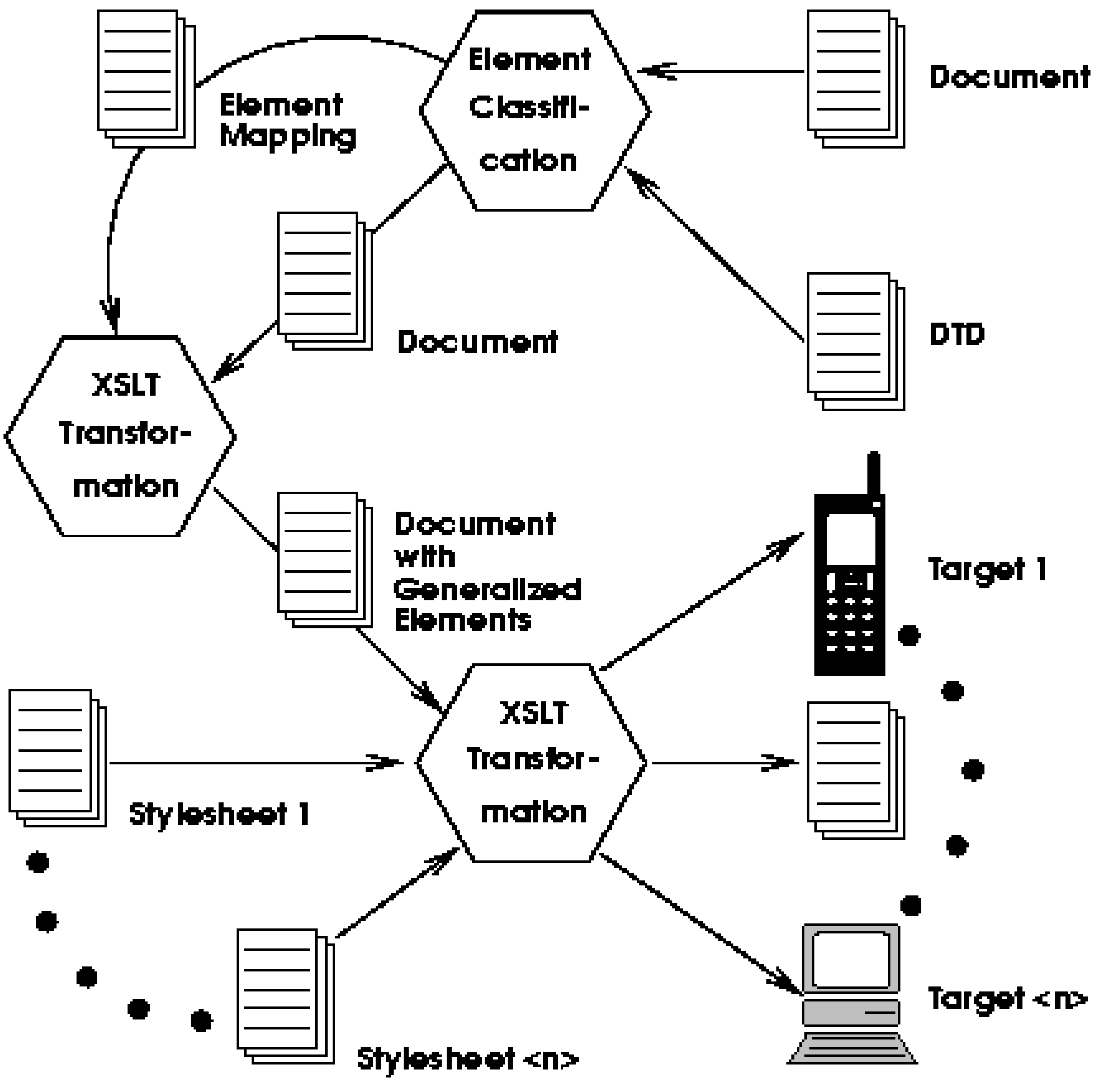

Fig. 3. Automatic conversion process.

1. The original document and its DTD (if available) are entered.
2. The document and its elements (document structure, elements lengths) are analyzed.
3. The element mapping is done using the decision rules for a certain generic DTD.
4. A METAMAP attribute telling the generic element class is added to each element of the original document with XSLT.
5. The document is transformed and styled for different target media or format (XML, HTML, LaTeX, etc.) with XSLT using prewritten stylesheets.

We have made transformation experiments with, e.g., different Shakespeare's plays. In Figure 4, the original DTD for Shakespeare's plays is given. The mapping from the original DTD to the generic element classes of the generic DTD is presented in Figure 5; the generic DTD itself is given in Figure 1 in Section 1. Next, Shakespeare's play "Pericles, Prince of Tyre" in XML format with the generic element classes in the METAMAP attributes is given in Figure 6.

An excerpt from a transformed Shakespeare's play with an HTML/CSS styling is shown in Figure 7 (a screenshot from Netscape Navigator). The transformation has been done using the XSLT stylesheet described in Figure 2 in Section 2.2.

```
<!DOCTYPE PLAY PUBLIC "-//Free Text Project//DTD Play//EN">
<!ELEMENT PLAY (TITLE, FM, PERSONAE, SCNDESCR, PLAYSUBT, INDUCT?,
PROLOGUE?, ACT+, EPILOGUE?)>
<!ELEMENT TITLE (#PCDATA)>
<!ELEMENT FM (P+)>
<!ELEMENT P (#PCDATA)>
<!ELEMENT PERSONAE (TITLE, (PERSONA | PGROUP)+)>
<!ELEMENT PGROUP (PERSONA+, GRPDESCR)>
<!ELEMENT PERSONA (#PCDATA)>
<!ELEMENT GRPDESCR (#PCDATA)>
<!ELEMENT SCNDESCR (#PCDATA)>
<!ELEMENT PLAYSUBT (#PCDATA)>
<!ELEMENT INDUCT (TITLE, SUBTITLE*, (SCENE+|(SPEECH|STAGEDIR|SUBHEAD)+))>
<!ELEMENT ACT (TITLE, SUBTITLE*, PROLOGUE?, SCENE+, EPILOGUE?)>
<!ELEMENT SCENE (TITLE, SUBTITLE*, (SPEECH | STAGEDIR | SUBHEAD)+)>
<!ELEMENT PROLOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
<!ELEMENT EPILOGUE (TITLE, SUBTITLE*, (STAGEDIR | SPEECH)+)>
<!ELEMENT SPEECH (SPEAKER+, (LINE | STAGEDIR | SUBHEAD)+)>
<!ELEMENT SPEAKER (#PCDATA)>
<!ELEMENT LINE (#PCDATA | STAGEDIR)*>
<!ELEMENT STAGEDIR (#PCDATA)>
<!ELEMENT SUBTITLE (#PCDATA)>
<!ELEMENT SUBHEAD (#PCDATA)>
```

Fig. 4. Original DTD for Shakespeare's plays (by J. Bosak).

```
<!DOCTYPE metamap SYSTEM "metamap-xml.dtd">
<metamap doctype="PLAY">
<map elem="PLAY" class="VirtDoc"></map>
<map elem="ACT" class="StrSect"></map>
<map elem="SCENE" class="StrSect"></map>
<map elem="INDUCT" class="StrSect"></map>
<map elem="PERSONAE" class="StrGrp"></map>
<map elem="EPILOGUE" class="StrGrp"></map>
<map elem="PROLOGUE" class="StrGrp"></map>
<map elem="FM" class="StrCont"></map>
<map elem="PGROUP" class="StrCont"></map>
<map elem="SPEECH" class="StrCont"></map>
<map elem="P" class="String"></map>
<map elem="PERSONA" class="String"></map>
<map elem="GRPDESCR" class="String"></map>
<map elem="SCNDESCR" class="String"></map>
<map elem="PLAYSUBT" class="String"></map>
<map elem="STAGEDIR" class="String"></map>
<map elem="SPEAKER" class="String"></map>
<map elem="LINE" class="String"></map>
<map elem="SUBHEAD" class="String"></map>
<map elem="SUBTITLE" class="String"></map>
<map elem="TITLE" class="Title"></map>
</metamap>
```

Fig. 5. Mapping from the original DTD for Shakespeare's plays to the generic element classes of the generic DTD.

```
...
<SPEECH METAMAP="StrCont">
<SPEAKER METAMAP="String">SIMONIDES</SPEAKER>
<LINE METAMAP="String">Opinion’s but a fool, that makes us scan</LINE>
<LINE METAMAP="String">The outward habit by the inward man.</LINE>
<LINE METAMAP="String">But stay, the knights are coming: we will
withdraw</LINE>
<LINE METAMAP="String">Into the gallery.</LINE>
</SPEECH>
<STAGEDIR METAMAP="String">Exeunt</STAGEDIR>
<STAGEDIR METAMAP="String">Great shouts within and all cry ’The mean
knight!’</STAGEDIR>
</SCENE>
<SCENE METAMAP="StrSect"><TITLE METAMAP="Title">SCENE III. The same.
A hall of state: a
banquet prepared.</TITLE>
<STAGEDIR METAMAP="String">Enter SIMONIDES, THAISA, Lords, Attendants,
and Knights, from tilting</STAGEDIR>
<SPEECH METAMAP="StrCont">
<SPEAKER METAMAP="String">SIMONIDES</SPEAKER>
<LINE METAMAP="String">Knights,</LINE>
<LINE METAMAP="String">To say you’re welcome were superfluous.</LINE>
<LINE METAMAP="String">To place upon the volume of your deeds,</LINE>
<LINE METAMAP="String">As in a title-page, your worth in arms,</LINE>
<LINE METAMAP="String">Were more than you expect, or more than’s fit,</LINE>
<LINE METAMAP="String">Since every worth in show commends itself.</LINE>
<LINE METAMAP="String">Prepare for mirth, for mirth becomes a feast:</LINE>
<LINE METAMAP="String">You are princes and my guests.</LINE>
</SPEECH>
<SPEECH METAMAP="StrCont">
<SPEAKER METAMAP="String">THAISA</SPEAKER>
<LINE METAMAP="String">But you, my knight and guest;</LINE>
<LINE METAMAP="String">To whom this wreath of victory I give,</LINE>
<LINE METAMAP="String">And crown you king of this day’s happiness.</LINE>
</SPEECH>
<SPEECH METAMAP="StrCont">
<SPEAKER METAMAP="String">PERICLES</SPEAKER>
<LINE METAMAP="String">’Tis more by fortune, lady, than by merit.</LINE>
</SPEECH>
...
```

Fig. 6. Shakespeare’s play “Pericles, Prince of Tyre” in XML format with the generic element classes in the METAMAP attributes.

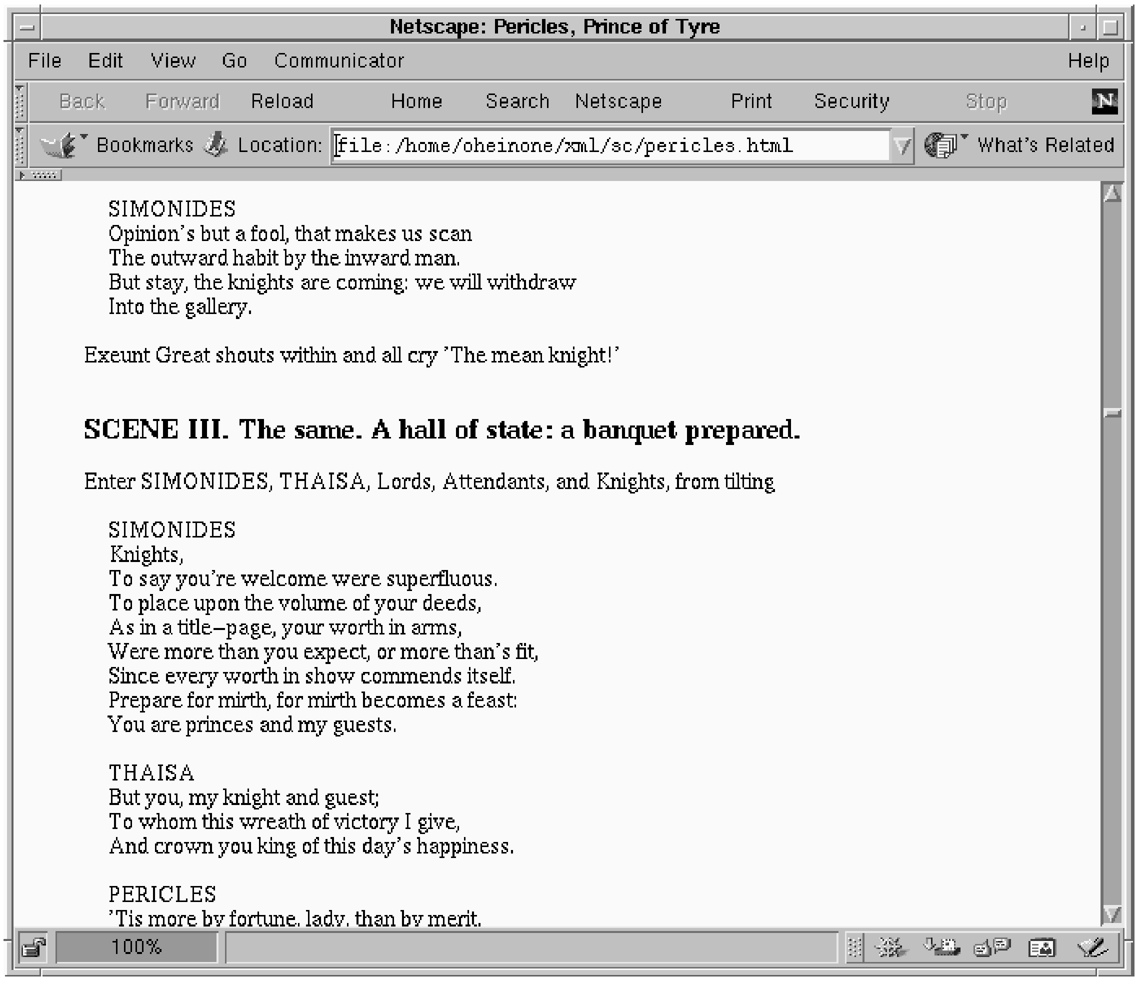

Fig. 7. An excerpt from a transformed Shakespeare's play with HTML/CSS styling.

4 Conclusions

In this paper we have described a tool and a process with which a document without any stylesheets or styling information can be automatically transformed to be viewed with different target media (paper, WWW browser, WAP phone, etc.). The incoming document and the attached DTD are analyzed, and the elements in the document are mapped to element classes of a fixed generic DTD, e.g., based on their length and location in the original DTD tree structure. For this generic DTD, one can create stylesheets for different target media in advance.

With a generic DTD and a fully automatic process the resulting styling might not be an optimal one, but the approach allows us to generate material for different target media without any user interference. Our experiments with real document material and different target media have shown us that the approach gives useful results and thus works in practice.

References

1. Helena Ahonen, Barbara Heikkinen, Oskari Heinonen, Jani Jaakkola, and Mika Klemettinen. Analysis of document structures for element type classification. *Proceedings of the 4th International Workshop on Principles of Digital Document Processing, PODDP '98*, Saint Malo, France, March 29-30, 1998. Number 1481 in Lecture Notes in Computer Science, Springer-Verlag.
2. Barbara Heikkinen. Generalization of Document Structures and Document Assembly. Ph.D. thesis, University of Helsinki, Department of Computer Science, April 2000.