Improving Document Reusability with Adaptive XML Inclusions

Author Tomas Pitner

Assistant Professor Masaryk University in Brno, Faculty of Informatics

Botanicka 68aBrno↵
↵
Czech Republic↵
tomp@fi.muni.cz↵

Tomas Pitner, Ph.D. is an assistant professor at Masaryk University in Brno, Czech Republic. His research interests include processing of semistructured data and appropriate software architectures. He applies the results mainly in the fields of e-learning and blended learning.

Abstract The concept of XML Inclusions (XInclude, *Marsh and Orchard 2004*) is well known and used in many areas of XML document authoring and processing in order to facilitate authoring and enable document reuse. However, as the included documents or fragments are included unchanged, the potential for the reuse is limited. This paper introduces the Adaptive XML Inclusions (AXI) which adopt, i.e. transform the included document. AXI is not limited to any specific markup of documents and has potentially wide range of applications. AXI has been implemented in Java as open-source software and is available at Sourceforge (*Pitner 2005*).

Keywordset DocBook XSLT SAX Content conversion Document creation

1 Introduction

The concept of XML Inclusions and the tools implementing them are well known and used in many areas of XML document authoring and processing. XML Inclusions enable specification of the document to be included, its encoding as well as fallback behavior if the included document is not available. As the documents containing XInclude elements do not require the DTD to be provided nor a validating parser to be used, the technique is applicable in various environments. The availability of advanced, user-friendly authoring tools grows quickly and more documents can originally be written in XML, so XInclude is a helpful tool. However, as the included documents or fragments are included unchanged, the potential for reuse is limited.

Let us imagine a conference with presentations and printed proceedings that are also available in a web-friendly form. The editors choose and prescribe the format for the full papers in the proceedings - assume it is *DocBook* (*Walsh 2004*). Moreover, the editors want to have also the slides shown at the presentation included in the proceedings. Many author choose for the presentation *DocBook Slides* which is a specific customization of DocBook (*Walsh 2004*). Thus, the sources of the proceedings constitute a mixture of papers in DocBook and copies of slides in DocBook Slides. With legacy processing tools such as XSLT processor and XML Includer, each slides file must have been separately converted to DocBook and subsequently included among other papers into the proceedings. The reuse of slides for both presentation and inclusion into the proceedings is possible but complicated and uncomfortable.

In this paper, we introduce a solution that would significantly facilitate such process - Adaptive XML Inclusions (AXI) which adopt, i.e. transform, the included document before it is included. It is not limited to any specific markup of documents and has potentially wide range of applications.

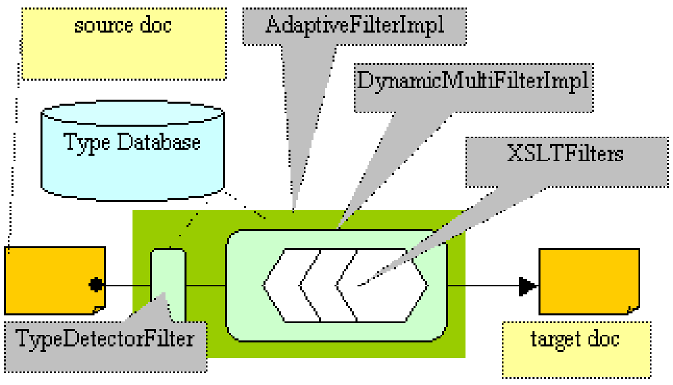
Initially, look at the main principle of Adaptive XML Inclusions which is based on Adaptive Filtering.

2 Principle of Adaptive XML Inclusions

2.1 Adaptive Filtering

The idea behind Adaptive (SAX) Filtering is simple. The type (markup) of the input document is detected. The client requires the output in a different type (markup) than the input one. The Adaptive Filter finds the appropriate transformation between the input and output markup and subsequently transforms the input document. For instance, input document type is DocBook, the client wants the output as HTML. So, technically, an XSLT stylesheet transforming DocBook to HTML is found, then a SAX filter (for explanation of SAX processing model, see *Megginson 2004*) doing the XSLT transformation according to the stylesheet is created, and finally the input document is parsed and it passes through the transforming filter to the output.

The figure below shows the architecture of the Adaptive Filter which is implemented in the AXI package available at Sourceforge (see *Pitner 2005*): the Type Detector Filter is the first component. It detects the type of the incoming document based on the characteristics found in the Type Database. The info about the type is used together with the required output type to feed the Dynamic MultiFilter with the appropriate transformation which is also found in the Type Database. The transformation is represented by the XSLTFilters.

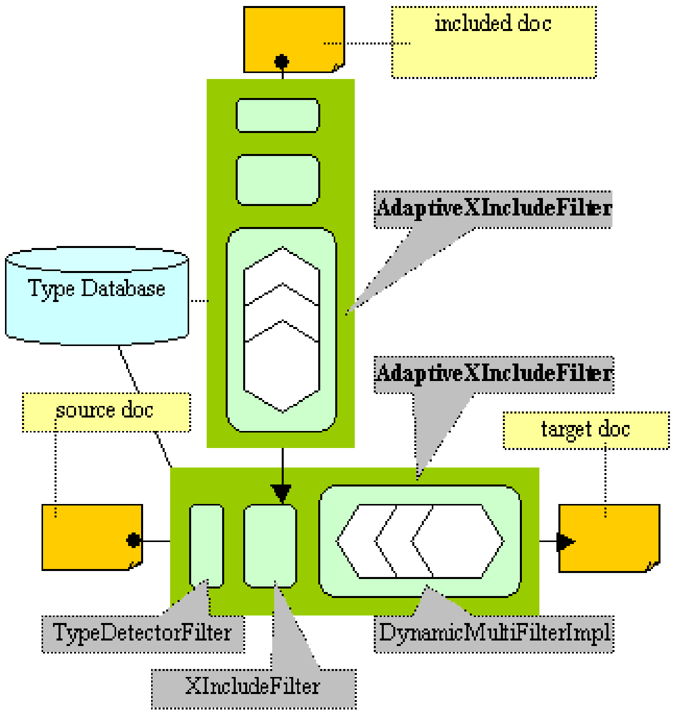


2.2 Adaptive XML Inclusions

Adaptive XML Inclusions are a natural extension to the Adaptive Filtering. The core principle of the document processing remains the same: the type of the source document is detected, the client wishes to get the output in a different markup. However, unlike with the Adaptive Filtering, the source document may contain also Adaptive XML Inclusions references to other documents. For instance, when writting this paper, the following inclusion could be used to easily insert the info about the author - even if the info about the author is originally stored not in DocBook `author` element but as a *vCard* (see *vCard 2001*) record.

```
<article xmlns:axi="http://tomp.sf.net/ns/axi/1.0">
  <title>Improving Document Reusability with Adaptive XML Inclusions</title>
  <author>
    <axi:include   href="http://www.fi.muni.cz/~tomp/vcard.xml"/>
  </author>
  ...
```

The difference between normal and Adaptive XML Inclusions is the following. With the classical XML Inclusions, the included document is included "as is", without significant changes - with some minor exceptions such as character encoding conversion. The Adaptive XML Inclusions, in contrast to this, enable transparent, automatical adoption of the type of the included document according to the context where it is included. This adoption is realized using an Adaptive XML Includer again. Thus, the implementation of Adaptive XML Includer is based on an extended Adaptive Filter that uses other Adaptive XML Includers when it fetches the included documents.



3 Types and Transformations

Now, we will concentrate on specific components of the Adaptive XML Includer which is an open-source Java implementation of the Adaptive XML Inclusions idea.

3.1 Type Database

The Type Database is a component holding the following information:

- how to detect the type of the document and
- what transformation will be applied to convert the source and target types.

The detection of the type relies on a simple database holding one record for each type of XML document. This database can be persistently stored in an XML file - as in the reference implementation. Other representations, such as relational database, are also possible.

3.2 Type Identification

The Type Database contains identification and description of XML types. The *type* is identified by the pair of *typing schema* (such as "by-doctype-public") and an *identifier* (the document’s DOCTYPE PUBLIC ID itself). This pair (unique within the database) identifies the type. The XML DOCTYPE PUBLIC identifier (as defined in the current XML specification) is commonly used to publicly identify the type of XML documents. In the Type Database, it is permitted to use any other type identification schema but adhering to the standardized identification schemata – such as DOCTYPE PUBLIC ID – is generally recommended.

```
<type by-doctype-public   ="-//OASIS//DTD DocBook XML V4.1.2//EN">
   ...
</type>
```

3.3 Type Inheritance

Sometimes, it is useful (and even necessary) to distinguish between a basic type and its *subtype*. By a subtype S of a (parent) type P we mean such a type that any document s of type S is also of type P. For instance, there exist several *customizations* of the DocBook markup, such as the markup this paper is written in: `-//IDEAlliance//DTD Conference Paper DocBook XML Subset V1.1//EN`. Many of the customizations are just a subsets of the original markup and thus they are subtypes in the above defined sense.

```
<type by-doctype-public="-//OASIS//DTD DocBook XML V4.1.2//EN">
   ...
</type>
<type by-doctype-public="-//IDEAlliance//DTD Conference Paper DocBook XML Subset V1.1//
   <subtype-of by-doctype-public    ="-//OASIS//DTD DocBook XML V4.1.2//EN"/>
</type>
```

Labeling a type to be a subtype has consequences for determining the transformation between the types. If the transformation of the parent type exists, the same transformation would correctly transform documents of the subtype. Details are described later.

3.4 Typing the Context

Even more important issue concerns typing contexts in document. Let us imagine an XML representation of a vCard entry:

```
<adressbook:entry xmlns:adressbook="http://tomp.sf.net/ns/adressbook/1.0"
                  xmlns="http://www.w3.org/2001/vcard-rdf/3.0#" >
    <FN>Tomas Pitner</FN>
    <N>
      <Family>Pitner</Family>
      <Given>Tomas</Given>
    </N>
    <EMAIL vcard:TYPE="http://www.w3.org/2001/vcard-rdf/3.0#internet">
       tomp@fi.muni.cz</EMAIL>
    <ORG>
      <Orgname>Masaryk University in Brno</Orgname>
      <Orgunit>Faculty of Informatics</Orgunit>
    </ORG>
</adressbook:entry>
```

This card can be reused, for example, in writing an DocBook *article* in the `/article/articleinfo/author` context - the info about the author will be extracted from the card by a simple XSLT transformation into the form similar to

```
<firstname>Tomas</firstname>
<surname>Pitner</surname>
<affiliation>
    <orgname>Masaryk University in Brno, Faculty of Informatics</orgname>
</affiliation>
```

However, using the card in a different context in the same document should produce different output. We may want the fullname and email be presented inside a `para` element like this:

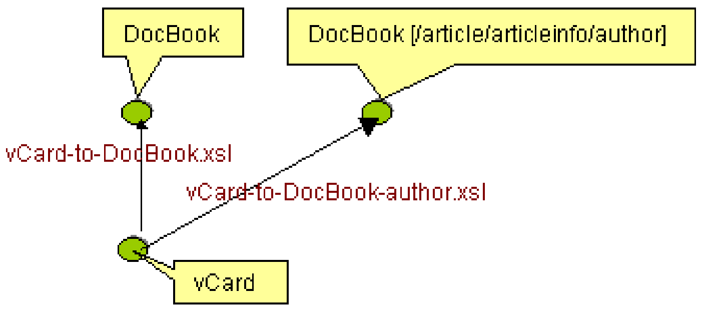
```
<ulink url="mailto:tomp@fi.muni.cz">Tomas Pitner</ulink>
```

To achieve this, the transformation must be completely different.

So, it is important to be able to differentiate among the contexts where documents are included. The type system in Adaptive XML Inclusions offers a mechanism to define a new type based on an existing type but limited to a certain context within documents of the existing type. For example, the following type definition corresponds to the above described situation: since we have two different types, the vCard now can be rendered differently to DocBook `author` context and differently to other contexts in the DocBook text.

```
<type by-doctype-public="-//OASIS//DTD DocBook XML V4.1.2//EN">
   ...
</type>
<type by-doctype-public="-//OASIS//DTD DocBook XML V4.1.2 [author]//EN">
    <part-of   by-doctype-public="-//OASIS//DTD DocBook XML V4.1.2//EN"
             context ="/article/articleinfo/author"/>
</type>
```

The situation where two different transformation styles are applied when transforming the same source to different contexts is depicted on the following figure.


DocBook
DocBook [/article/articleinfo/author]
vCard-to-DocBook.xsl
vCard-to-DocBook-author.xsl
vCard

3.5 Type Detection

The document type may be detected according to characteristics that are both representative and easy to obtain, such as:

- declared PUBLIC (or even SYSTEM) DOCTYPE
- root namespace URI and/or
- root element name and/or root element attributes.

These characteristics can be easily determined very early - at the beginning of the document. This gives a great advantage for the pipeline-based processing – the type is determined just at the beginning and the appropriate transformation can immediately follow and no tree needs to be built in order to "cache" the document before it can be transformed.

In future releases, the set of characteristics will be extended. For instance, a processing instruction preceding the root element can signalize the document type, a combination of more nested elements than just the root can be detected. The detection model should also be able to combine several subconditions with logical operators (AND, OR, NOT).

The following snippet from the Type Database identifies the DocBook 4.1.2 document type and instructs the Type Detector how to detect the type according to the root element names or DOCTYPE PUBLIC.

```
<type by-doctype-public="-//OASIS//DTD DocBook XML V4.1.2//EN">
    <root  name="article" ns="" />
    <root name="book" ns="" />
    <root name="chapter" ns="" />
    <doctype  public ="-//OASIS//DTD DocBook XML V4.1.2//EN" />
</type>
```

If a document has either root element name `article`, `book`, `chapter` or with declared `DOCTYPE PUBLIC -//OASIS//DTD DocBook XML V4.1.2//EN`, it will be detected as type `[doctype-public:-//OASIS//DTD DocBook XML V4.1.2//EN]`.

3.6 Type Transformation

The Adaptive XML Inclusions must transform the included document in order to adapt them to the target context. So, the next task of the Type Database is to store information on the transformations between documents of different types. Currently, two basic types of transformation are supported:

- SAX event filter transformation defined by a class implementing `org.xml.sax.Filter` (or its more powerful extension `net.sf.tomp.xtcl.filter.XTFilter`) interface or
- XSLT transformation defined by an XSLT stylesheet or
- STX transformation defined by an STX stylesheet (for more info about *Streaming Transformations for XML*, see *Becker 2004*).

The figure below presents a record holding prescription stating that documents of type `DocBook Slides v3.1.0` can be transformed to `OASIS DocBook v4.1.2` with the XSLT transformation defined by the `slides2docbook-chapter.xsl` stylesheet.

```
<transformation source-doctype-public="-//Norman Walsh//DTD Slides V3.1.0//EN"
                target-doctype-public="-//OASIS//DTD DocBook XML V4.1.2//EN">
   <style   href="file:xtech/slides2docbook-chapter.xsl">
      <param name="preserve-speakernotes" value="1"/>
   </style>
</transformation>
```

As the listing shows, additional parameter assignments (such as `preserve-speakernotes=1` in the example) for the XSLT transformation can also be specified here in the form of embedded param elements. Which transformation is selected may also depend on an optional variant parameter (attribute) which can be specified in the Adaptive XML Inclusions element `axi:include`.

3.7 Finding Transformation Path

The AdaptiveXIncludeFilter is also able to find and construct a composite transformation. If there is no direct transformation between the given source and target document type but there exists a sequence of transformations t_i (i is from {0, 1, .., n-1}) such as t_i: T_i -> T_{i+1} where T_i denotes the document type i (T_0 is the source type and T_n is the target type), then a composite transformation t from the source to target type T: T_0 -> T_{i+1} is created as a chain of transformations $t = t_n \circ t_{n-1} \circ t_{n-2} \circ \ldots \circ t_1 \circ t_0$.

The sequence of transformations (also denoted as *transformation path*) is found by searching the shortest path between the source and target types. Every transformation, either defined by a stylesheet or a SAX filter, has assigned a positive *costs*-value. For simplicity, every SAX filter has costs of 1, while the transformation specified by an XSLT/STX stylesheet has costs of 10. This, of course, does not fully reflect the real complexity of the transformations but serves as a mechanism to prefer SAX filtering over time- and space-complex XSLT transformations in most real situations.

An identity transformation is introduced between each subtype *S* and its supertype (parent) *P*. This is a consequence of the fact that any document of type *S* is automatically of type *P*, i.e. the identity transformation converts documents from type *S* to *P*. However, in order to prefer the direct transformation between *S* and a target type *T* over the path *S* -> *P* -> *T*, the identity transformation *S* -> *P* has also small costs.

4 Working Examples

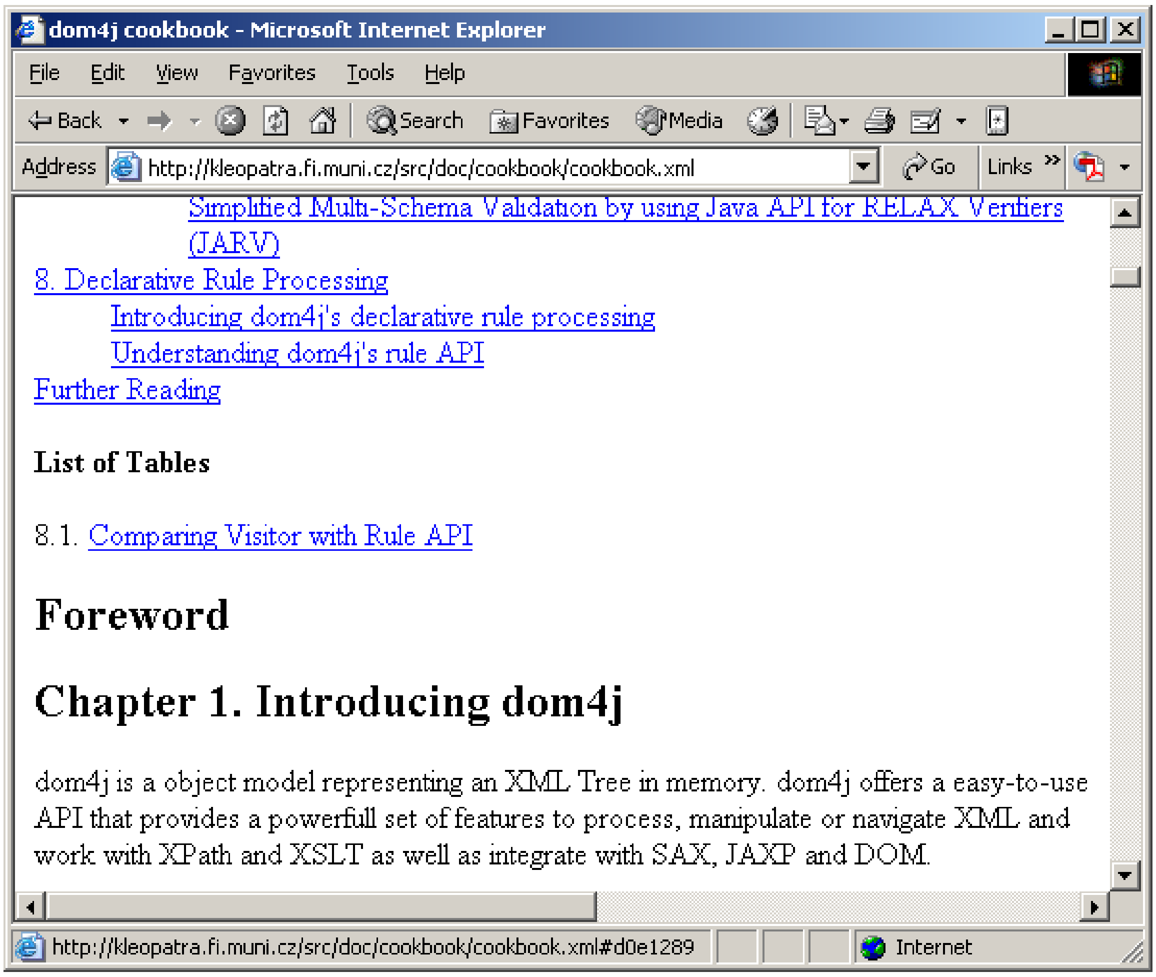
In this section, several working examples are presented. All of them can be found at the project website.

4.1 Browsing XML Files with Adaptive Filtering

Adaptive filtering serves as a basis for a handy application for remote browsing XML files via a regular web browser. This example application can be found with XTCL the Sourceforge distribution under the name *xtclbrowser*. It acts as a very simple HTTP server - when an XML file is remotely accessed via the HTTP GET method, the xtclbrowser tries to transform the content of the file into a format acceptable for the browser, i.e. typically (X)HTML. The user may change this output format to anything else. This allows to quickly and easily browse through XML files on the remote computer. The xtclbrowser can be started similarly to this:

```
C:\xtclbrowser>java net.sf.tomp.xtclbrowser.BrowserServer -r /devel/dom4j-1.4 -v
Using URL mapping file mapping.properties
Serving files under root \devel\dom4j-1.4
Type database read from type-database.xml
BrowserServer listening on port 80
Hit Enter to stop.
```

Now, it is possible to access any XML file under `/devel/dom4j-1. 4` directory on the host where the xtclbrowser is launched. The accessed file `cookbook.xml` written in DocBook is transformed on-the-fly into HTML and sent to the client web browser:


dom4j cookbook - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://kleopatra.fi.muni.cz/src/doc/cookbook/cookbook.xml
Go Links
http://kleopatra.fi.muni.cz/src/doc/cookbook/cookbook.xml#d0e1289
Internet

Simplified Multi-Schema Validation by using Java API for RELAX Verifiers (JARV)
8. Declarative Rule Processing
Introducing dom4j's declarative rule processing
Understanding dom4j's rule API
Further Reading

List of Tables

8.1. Comparing Visitor with Rule API

Foreword

Chapter 1. Introducing dom4j

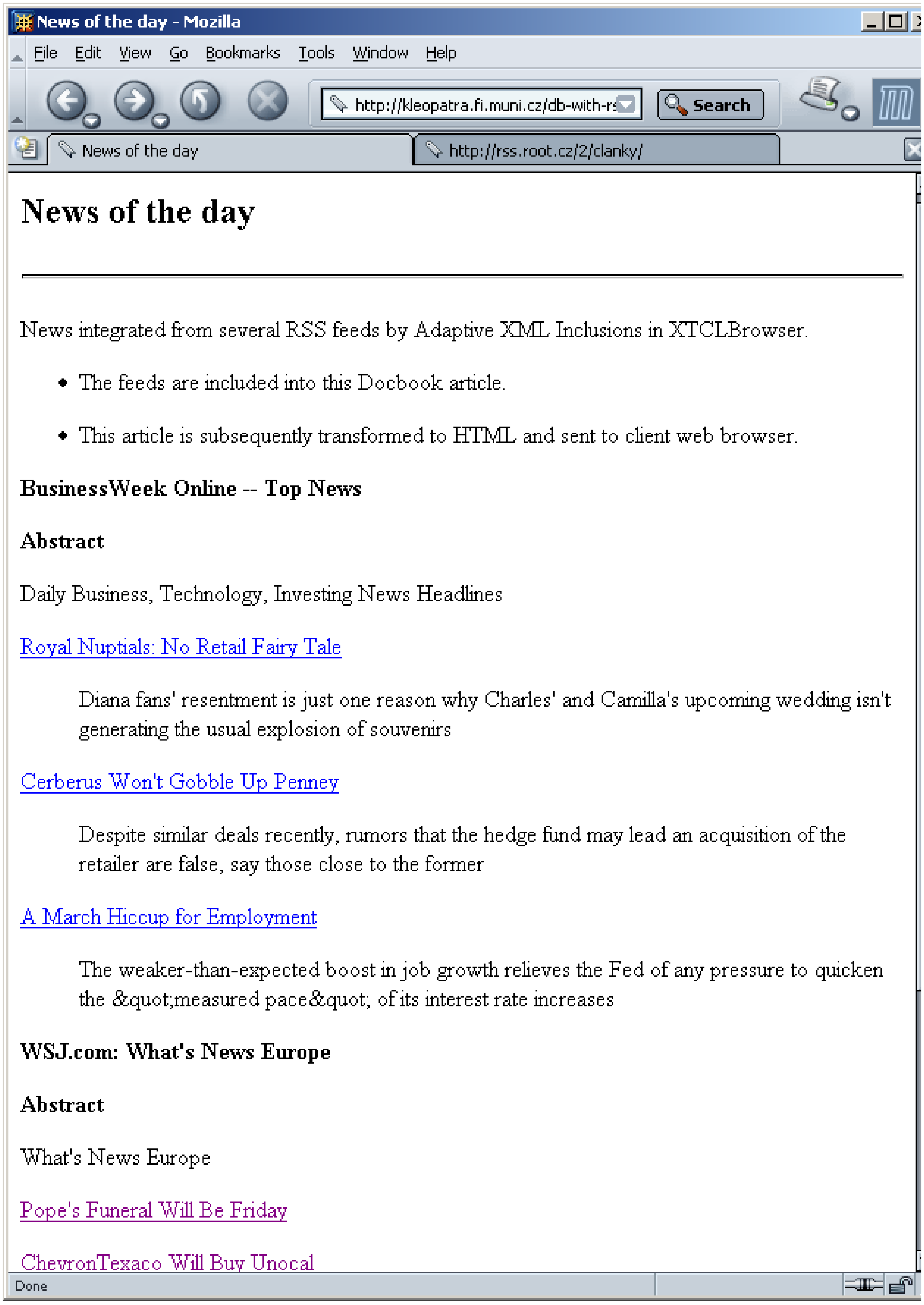
dom4j is a object model representing an XML Tree in memory. dom4j offers a easy-to-use API that provides a powerfull set of features to process, manipulate or navigate XML and work with XPath and XSLT as well as integrate with SAX, JAXP and DOM.

4.2 Assembling various XML Resources

Adaptive XML Includer can be used for assembling various XML resources from the web into one document that is subsequently visualized. The following source refers to several files with RSS format - not DocBook.

```
<article xmlns:xi="http://tomp.sf.net/ns/axi/1.0">
    <title>News of the day</title>
    <para>News integrated from several RSS feeds
          by Adaptive XML Inclusions in XTCLBrowser.</para>
    <itemizedlist>
    <listitem><para>The feeds are included into this DocBook article.</para></listitem>
    <listitem><para>This article is subsequently transformed to HTML
                and sent to client web browser.</para></listitem>
    </itemizedlist>
    <xi:include    href="http://www.businessweek.com/rss/bwdaily.rss">
        <param name="maxitems" value ="3"/>
    </xi:include>
    <xi:include    href="http://online.wsj.com/xml/rss/0,,3_7012,00.xml"/>
</article>
```

When this file is fetched via xtclbrowser, the types of the included documents are autodetected, adopted file contents are merged and visualized as HTML. As you can see, even the number of shown RSS items may be limited by setting `maxitems` parameter on the inclusion.


News of the day - Mozilla
File Edit View Go Bookmarks Tools Window Help
http://kleopatra.fi.muni.cz/db-with-rs
Search
News of the day
http://rss.root.cz/2/clanky/
Done

News of the day

News integrated from several RSS feeds by Adaptive XML Inclusions in XTCLBrowser.

- The feeds are included into this Docbook article.
- This article is subsequently transformed to HTML and sent to client web browser.

BusinessWeek Online -- Top News

Abstract

Daily Business, Technology, Investing News Headlines

Royal Nuptials: No Retail Fairy Tale

Diana fans' resentment is just one reason why Charles' and Camilla's upcoming wedding isn't generating the usual explosion of souvenirs

Cerberus Won't Gobble Up Penney

Despite similar deals recently, rumors that the hedge fund may lead an acquisition of the retailer are false, say those close to the former

A March Hiccup for Employment

The weaker-than-expected boost in job growth relieves the Fed of any pressure to quicken the &quot;measured pace&quot; of its interest rate increases

WSJ.com: What's News Europe

Abstract

What's News Europe

Pope's Funeral Will Be Friday

ChevronTexaco Will Buy Unocal

4.3 Preparing Presentation and Printed Matter in once

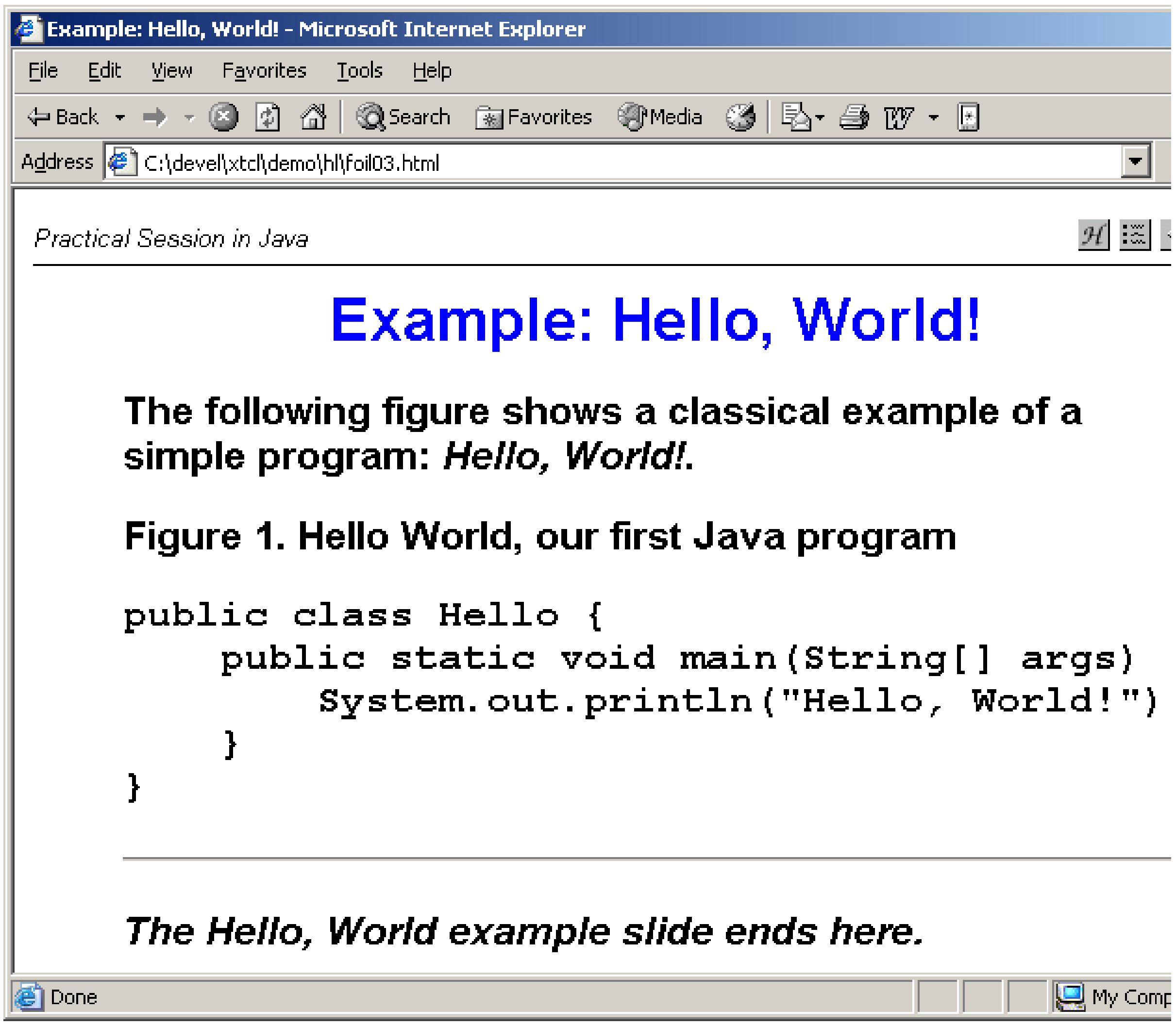
The third example shows a combination of Adaptive XML Inclusions and Adaptive Filtering to different target formats. The motivation stems from a typical situation with so-called blended- (hybrid-) learning where the classical teaching instruments are combined with electronic support.

The effective preparation of a real blended-learning course demands that several different kinds of material be prepared. These materials typically include *slides* for an in-class presentation given by the teacher, printable *full-text* materials and *browsable* web-oriented materials for self-study in front of the computer.

The authoring of the sources for such materials should avoid as much repetitive work as possible. For instance, what can be taken from slides, need not be manually rewritten into full-text material but included in it. Also the extent of the presentations differs significantly – on slides there is not enough space to show everything, on the paper, no "live" behaviour can be presented. So assembling the material must be context, target and purpose dependent.

- The original source is in DocBook slides markup.
- It may contain inclusions of differently marked fragments - for example computer program descriptions.
- At the output, various formats can be achieved - slides (i.e. chunked HTML), browsable HTML (all-in-one-file) and printable PDF.

The figures below illustrate these different outputs: slides and all-in-one HTML file.


Example: Hello, World! - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address C:\devel\xtcl\demo\hl\foil03.html
Done
My Comp

Practical Session in Java

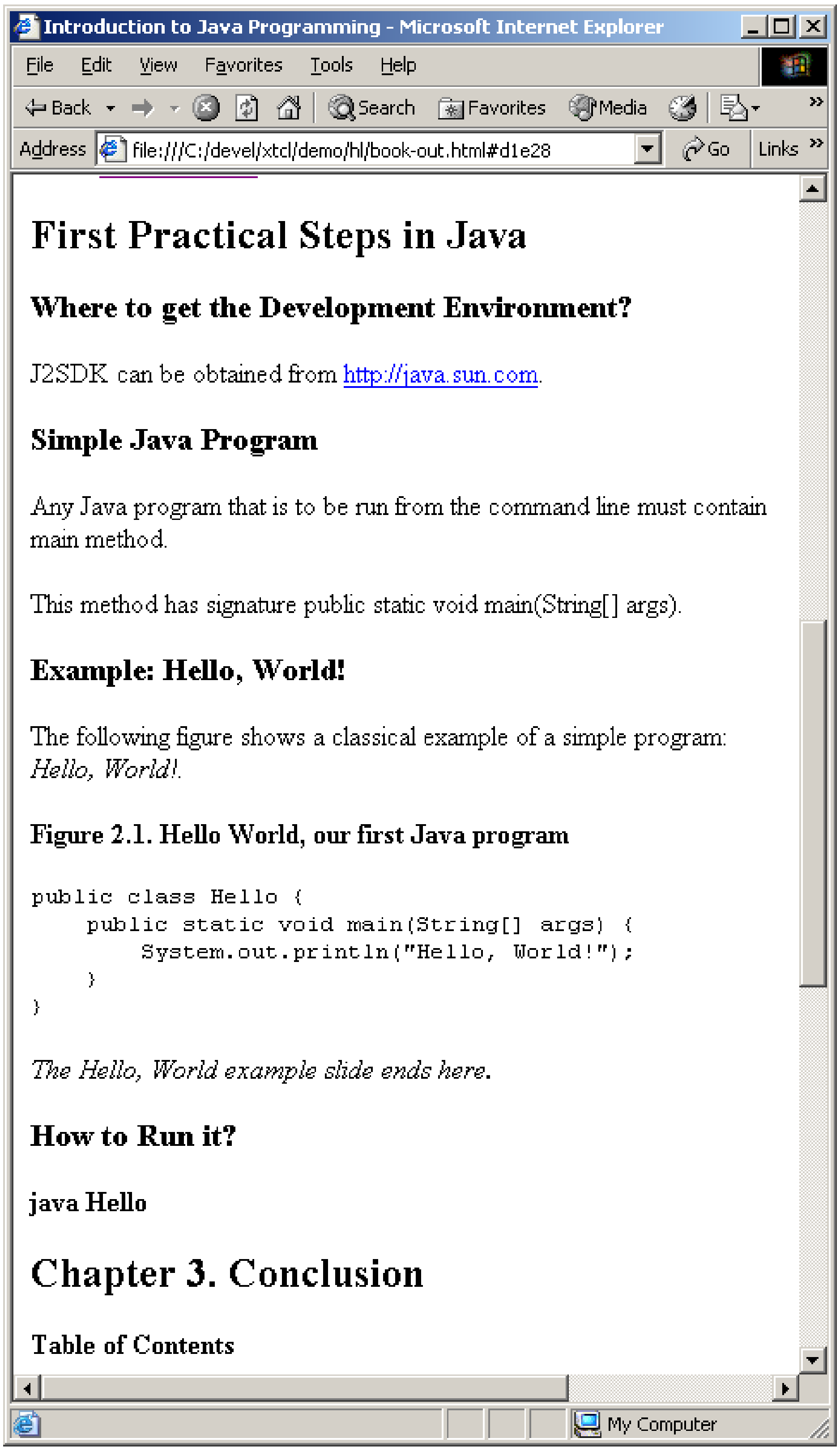
Example: Hello, World!

The following figure shows a classical example of a simple program: *Hello, World!*.

Figure 1. Hello World, our first Java program

```
public class Hello {
    public static void main(String[] args)
        System.out.println("Hello, World!")
    }
}
```

The Hello, World example slide ends here.


Introduction to Java Programming - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address file:///C:/devel/xtcl/demo/hl/book-out.html#d1e28
Go Links
My Computer

First Practical Steps in Java

Where to get the Development Environment?

J2SDK can be obtained from http://java.sun.com.

Simple Java Program

Any Java program that is to be run from the command line must contain main method.

This method has signature public static void main(String[] args).

Example: Hello, World!

The following figure shows a classical example of a simple program: *Hello, World!*.

Figure 2.1. Hello World, our first Java program

```
public class Hello {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```

The Hello, World example slide ends here.

How to Run it?

java Hello

Chapter 3. Conclusion

Table of Contents

Adaptive XML Inclusions make the preparation of the learning content easier, reducing the redundant work and enable multipurpose reuse of the material. For more details, see *Pitner 2004*.

5 Conclusion

Adaptive XML Inclusions have proven as a handy, flexible, and generally applicable tool for preparing and reusing various XML content. In the future, the pool of available transformations will be extended with XQuery and the input type may be detected also according to its MIME-type or file name extension.

ACKNOWLEDGEMENT: The research and development has been supported by the Czech National Programme "Information Society", Grant No. 1ET208050401.

Becker 2004*Serielle Transformationen von XML – Probleme, Methoden, Lösungen; PhD Thesis, Humboldt Universität zu Berlin*, 2004

Marsh and Orchard 2004*Marsh, J., Orchard, D.: XML Inclusions (XInclude) Version 1.0, W3C Recommendation* 2004

Megginson 2004*Megginson, D.: Simple API for XML (SAX), http://www.saxproject.org* 2004

Pitner 2004*Adaptive XML Inclusions for the Effective Support of Hybrid Learning, in Proc. of I-KNOW 2004* 2004

Pitner 2005*Adaptivity XML Tools, http://tomp.sf.net* 2005

vCard 2001*Representing vCard Objects in RDF/XML, http://www.w3.org/TR/vcard-rdf, W3C Note* 2001

Walsh 2004*Walsh, N.: DocBook home page, http://docbook.org* 2005

XML 2004*Yergeau, F., Cowan, J., Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler E.: XML 1.1, W3C Recommendation* 2004