COMPSCI 732: Software Tools and Techniques
Assignment 3: XSLT-based Mapping Generator

Worth 16.8% of your final grade
This assignment is due on 31st May 2005
This assignment must be done individually

Aims

The aim of this project is to give you some experience in the use of a simple mapping language. This language will be used to describe a series of simple mappings between XML-based representations of publications. You will use the mapping specification to generate XSLT capable of performing mappings in either direction for XML documents based on the specified DTD.

The mapping language

In order to specify a mapping we need some formalism to describe it. As we wish to generate XSLT for both directions from one mapping specification we aim for a high-level, declarative approach to the formalism. Below I describe the components of a basic mapping language. In this formalism I use the symbol ↔ to represent an equivalence between information on one side of the symbol and that on the other side of the symbol.

Top level specification

We need to capture which schemas a mapping is being specified for. Therefore at the top level in the formalism we specify the DTDs:

interSchema(*DTD_URI* ↔ *DTD_URI*, …)

Element level specification

After listing the schemas (DTDs) we need to describe which elements are mapped between. We do this by listing the elements, with the first coming from the initial DTD in the interSchema and the second from the second DTD in the interSchema:

interElement(*ElementName* ↔ *ElementName*, …)

Following this we describe the invariants which control whether this mapping should be used for a particular instance of a class. These invariants will be simple equalities. Remember that an invariant which restricts a mapping in one direction is used as an initialiser when the mapping is run in the reverse direction.

Data level mapping specification

Within the element level association we then describe all mappings between individual element values in the schema. This takes the form of:

Equation ↔ *Equation*

The *Equation* on the left-hand side of the equivalence symbol refers to data in the left-hand side *ElementName* from this interElement, and similarly for the right-hand side *Equation*. An equation can be of the following form:

- An element name, or path following element references (as in XPath)
- A default value
- A mathematical equation involving a single arithmetic operator and one element name. The allowable operators are the XPath operators of:

- +
- −
- *
- div

Assumptions for this assignment

In order to reduce some of the other complexities that can occur when processing XML documents and DTDs we make the following assumptions about the types of mapping and structure of DTDs we will be handling:

- The order of elements is equivalent in both DTDs. Here we assume that we can generate the resultant XML file by processing the data level mappings in the order specified. Without this assumption we would have to load the DTD for each schema and determine the correct order to write out element values.
- The elements described as part of the invariants are the first elements in the DTD, occurring in the same order as they are listed in the mapping specification.
- The equations are very simple, so rewriting the equations to perform an inverse mapping is greatly simplified. A mathematical equation will only have one element on each side of the equivalence.
- The invariant specification is very simple with the assumption that all invariants describe a simple equality between a constant value and an element.
- A catalogue element wraps all data being mapped between. You should generate XSLT to handle this top level element (not shown in mappings).

XML form of the mapping language

So that you do not need to consider parsing the mapping formalism the mapping examples for this assignment have been coded in a XML format. The DTD for this XML format, which reflects the mapping language above, is as follows:

```
<!ELEMENT interSchema (dtdURI, dtdURI, interElement+) >
<!ELEMENT dtdURI (#PCDATA) >
<!ELEMENT interElement (elementName, elementName, invariant*, mapping+) >
<!ELEMENT elementName (#PCDATA) >
<!ELEMENT mapping (equation, equation) >
<!ELEMENT equation (elementName | value | arithmetic) >
<!ELEMENT value (#PCDATA) >
<!ELEMENT arithmetic (elementName, value) >
<!ATTLIST arithmetic op (plus | minus | times | div) #REQUIRED>
<!ELEMENT invariant (invExpression, invExpression) >
<!ELEMENT invExpression (elementName | value) >
```

Task One (65%)

In the first part of this project you will need to handle a one-to-one mapping between single elements in an XML file. In this part of the project the only change which is mapped is to modify the name of elements in the XML documents. The only equation which will be present is an equivalence between two elements. Two example DTDs are as follows:

publication1.dtd	**publication2.dtd**
<!ELEMENT catalogue (publication+) > <!ELEMENT publication (title, creator, isbn,	<!ELEMENT catalogue (publication+) > <!ELEMENT publication (title, author, isbn,

subject?, description?, tableOfContents?, cost) > <!ELEMENT title (#PCDATA) > <!ELEMENT creator (#PCDATA) > <!ELEMENT isbn (#PCDATA) > <!ELEMENT subject (#PCDATA) > <!ELEMENT description (#PCDATA) > <!ELEMENT tableOfContents (#PCDATA) > <!ELEMENT cost (#PCDATA) >	classification?, description?, contents?, price) > <!ELEMENT title (#PCDATA) > <!ELEMENT author (#PCDATA) > <!ELEMENT isbn (#PCDATA) > <!ELEMENT classification (#PCDATA) > <!ELEMENT description (#PCDATA) > <!ELEMENT contents (#PCDATA) > <!ELEMENT price (#PCDATA) >

The mapping which is required for this transformation can be found in the file 'one2one.xml' and has the following specification:

```
interSchema(publication1.dtd ↔ publication2.dtd,
        interElement(publication ↔ publication,
                title ↔ title,
                creator ↔ author,
                isbn ↔ isbn,
                subject ↔ classification,
                description ↔ description,
                tableOfContents ↔ contents,
                cost ↔ price
        )
)
```

To complete this task you could consider the problem in the following manner:

1. By hand, write out the XSLT to perform the mapping in one direction
2. Look at how this relates to the mapping specification in the XML version of the mapping
3. Determine a strategy for transforming parts of the XML-based mapping into XSLT
4. Load the XML-based mapping into a DOM
5. Step through the DOM and apply your transformations from 3) to build up a new DOM-based structure
6. Write out the XSLT (DOM) for one mapping direction
7. Test that your XSLT properly transforms the example XML file (publication1.xml)
8. Step through the DOM and apply the inverse transformations from 3) to build a new DOM-based structure
9. Write out the XSLT (DOM) for the other mapping direction
10. Test that your XSLT properly transforms the example XML file (publication2.xml)

Task Two (15%)

Now extend the system you developed above to handle equations as well as simple equivalence. You need only handle equations with a single operator (i.e., +, -, *, div).

A first test is to use the DTDs and XML files from stage 1 but with the mapping in the file 'equation.xml'. In this mapping the cost is defined as price * 0.5855 (US$ conversion).

Task Three (20%)

Finally, extend your system to allow invariants to control which mapping specification is used for a class in a schema. For example, if one schema uses a

publication class to represent all types of publications (e.g., books, magazines, CDs, DVDs, etc) and another schema has individual classes for each type of publication then we need to use an invariant to define when the general publication can be mapped to each of these individual classes.

publications.dtd	**catalogue.dtd**
<!ELEMENT catalogue (publication+) > <!ELEMENT publication (type, title, creator, subject?, description?, tableOfContents?, cost) > <!ELEMENT type (#PCDATA) > <!ELEMENT title (#PCDATA) > <!ELEMENT creator (#PCDATA) > <!ELEMENT subject (#PCDATA) > <!ELEMENT description (#PCDATA) > <!ELEMENT tableOfContents (#PCDATA) > <!ELEMENT cost (#PCDATA) >	<!ELEMENT catalogue (book*, cd*, dvd*) > <!ELEMENT book (title, author, classification?, description?, tableOfContents?, price) > <!ELEMENT title (#PCDATA) > <!ELEMENT author (#PCDATA) > <!ELEMENT classification (#PCDATA) > <!ELEMENT description (#PCDATA) > <!ELEMENT tableOfContents (#PCDATA) > <!ELEMENT price (#PCDATA) > <!ELEMENT cd (title, band, musicType?, description?, songList?, price) > <!ELEMENT title (#PCDATA) > <!ELEMENT band (#PCDATA) > <!ELEMENT musicType (#PCDATA) > <!ELEMENT description (#PCDATA) > <!ELEMENT songList (#PCDATA) > <!ELEMENT price (#PCDATA) > <!ELEMENT dvd (title, producer, movieType?, description?, price) > <!ELEMENT title (#PCDATA) > <!ELEMENT producer (#PCDATA) > <!ELEMENT movieType (#PCDATA) > <!ELEMENT description (#PCDATA) > <!ELEMENT price (#PCDATA) >

The mapping which is required for this transformation can be found in the file ‘invariant.xml’ (there is example data in the files publications.xml and catalogue.xml) and has the following specification:

```
interSchema(publications.dtd ↔ catalogue.dtd,
        interElement(publication ↔ book,
                invariants(type = “BOOK”),
                title ↔ title,
                creator ↔ author,
                subject ↔ classification,
                description ↔ description,
                tableOfContents ↔ tableOfContents,
                cost ↔ price * 0.5855
        )
        interElement(publication ↔ cd,
                invariants(type = “CD”),
                title ↔ title,
                creator ↔ band,
                subject ↔ musicType,
                description ↔ description,
                tableOfContents ↔ songList,
                cost ↔ price * 0.5855
        )
        interElement(publication ↔ dvd,
                invariants(type = “DVD”),
                title ↔ title,
                creator ↔ producer,
```

```
                    subject ↔ movieType,
                    description ↔ description,
                    tableOfContents ↔ "",
                    cost ↔ price * 0.5855
        )
)
```

Remember that invariants (e.g., *type = "BOOK"*) restrict which object will be mapped when going from the publications DTD to the catalogue DTD. In the reverse direction the invariant will initialise the value of *type* in the publications DTD to the value *"BOOK"*.

Deliverables

You should provide the following deliverables for this project (assuming you use Java for your development):

1. A single JAR file containing all of your code for stages 1, 2 and 3. Name this file 'mapping.jar'.
2. A Word document which states how far you got in your implementation and also contains instructions on how to run your mapping system for tasks 1, 2 and 3.

You should plan to spend no more than 25 hours on this assignment. This is an individual project, so make sure you develop your own solutions.

Hand in the electronic copy of your prototype through the web drop-box. You can make as many submissions as you like, only your last submission will be marked. If you make multiple submissions ensure that you send all files for each submission.

Online Sources

For general resources on XML, XSLT, and DOM look at:

- XML Tutorial: The XML Revolution: http://www.brics.dk/~amoeller/XML/
- The Java Web Services Tutorial: http://java.sun.com/webservices/tutorial.html
- XPath Language: http://www.w3.org/TR/xpath