7 Translating Data to XML

- How to translate existing data formats to XML? (and why?)
- XW (XML Wrapper)
  - "XML wrapper description language" (kääreenkuvauskieli)
  - developed in XRAKE project, Univ. of Kuopio, 2001–02

XRAKE Project

- "XML-rajapintojen kehittäminen" (Developing XML-based interfaces)
- Studied definition and implementation of XML-based interfaces, and their application in
  - integration of heterogeneous data sources
  - management of mass printing
  - assembly and manipulation of electronic patient records

XRAKE - Support

- National Technology Agency of Finland (TEKES) and seven local IT companies/organizations
  - DEIO IS
  - Enfo Group
  - JSOP Interactive
  - Kuopio University Hospital
  - Medigroup
  - SysOpen
  - TietoEnator

XW: Motivation

- XML-based protocols developed for e-business, medical messages, …
- Legacy data formats need to be converted to XML – How?

XML Wrapping

- Need "XML-wrappers" (aka extractors)
  - (kääre); interface/conversion program to produce an XML representation of source data

- How to wrap?
  1. With an interface integrated to source
     - E.g. XML-interfaces of database systems
     - OK, if available
  2. With an ad-hoc written translator
     - E.g. JDBC+Java, or separator-encoded text form + Perl
     - Conversion possibly efficient
     - Development and maintenance tedious
  3. Generic source-independent wrapping
     - requires a file/message/report produced by the system
     - normally available
     - development and maintenance of wrappers should become easier
     => Wrapper description language XW

XW (XML Wrapper)

- XML-based, declarative wrapper description language
- To convert
  - from a textual or binary source
  - currently (XW 1.59) only text sources supported
  - to an XML form
XW: Design principles

- A concise and natural XML syntax
  - description of simple and typical conversion tasks should be simple
- Solving the key problem: Initial conversion of a legacy data format to XML
  - more general post-processing with XSLT/SAX/DOM
  - necessary for being able to apply XML techniques

XW: Influences

  - for separating XW commands and result elements
- XML Schema
  - description of alternative and repetitive structures
    (CHOICE, minOccurs, maxoccurs)
  - data types of binary source data
    (string, byte, int, ...)
- XSLT
  - template-based description of result documents
  - variables for storing result fragments

How does XW look like?

```xml
  xw:sourcetype="text" xw:inputencoding="Cp850" ... >
  <invoice note="XW-generated" xw:starter="\"INVOICE\">
  <identifierdata ...>
    Inserted result text ...
  </identifierdata>
  <specification xw:starter="\"PHONE SPECIFICATION\">
    ...
  </specification>
  ...
  </invoice> 
</xw:wrapper>
```

XW architecture (1)

source data

wrapper description

result document

post-processing

```
source data
  wrapper description
  result document
```

XW architecture (2)

```
source data
  wrapper description
```

SAX events

```
startElement(part-a, ...)
startElement(e1, ...)
characters("x1")
```

XW architecture (3)

```
source data
  wrapper description
```

```
application
```

```
result document
```

XW Basic Ideas

- Wrapper description ~ a grammar for source
- Wrapping ~ parsing the source data
  - split data into parts according to the description
  - Result document = XML for the parse tree of the source

XW Syntax

```xml
  <invoice ...>
    <identifierdata ...>
      ...
    </identifierdata>
    <specification ...>
      ...
    </specification>
  </invoice>
</xw:wrapper>
```
Recognition of content parts (1)

- by separators (e.g., \^); For example:
  <invoice xw:starter="^INVOICE">
  <identifier data xw:childterminator="\n" ... />
- by position (within surrounding part):
  <invoice number xw:position="53 64" />
  (Invoice number is in positions 53..64 of the first row of an identifier data part)

Recognition of content parts (2)

- In binary data by content data types;
  For example:
  <xw:wrapper xw:sourcetype="binary"...>
    <A xw:type="byte"/>
    <B xw:type="string" xw:stringLength="20"/>
    <C xw:type="int"/>
  </xw:wrapper>
  - Split input to a byte, a string of 20 characters, and an integer (=> elements A, B and C)

Recognition of content parts (3)

- Repetition:
  <line xw:terminator="\n" xw:minoccurs="2" xw:maxoccurs="2"/>
  - 2 input lines \rightarrow 2 line elements
- Alternative parts:
  <xw:CHOICE xw:maxoccurs="unbounded">
    <A xw:starter="^aa" xw:terminator="\n" />
    <B xw:starter="^bb" xw:terminator="\n" />
  </xw:CHOICE>
  - arbitrary number (at least 1) lines starting with "aa" or "bb" \rightarrow elements A or B

XW: Modifying the structure of data

- Limited modifications supported:
  - discarding parts of data
  - collapsing levels of hierarchy
  - adding levels of hierarchy
  - generating verbatim content and attributes
  - re-arranging existing data

Discarding parts of data

- Input parts not matched by wrapper elements are ignored
  <spec xw:starter="SPEC"
    xw:childterminator="\n" />
  <!-- Split the "SPEC" into rows: -->
  <!-- Ignore the first three rows: -->
  <xw:ignore xw:minoccurs="3" xw:maxoccurs="3" />
  ...
  </spec>

Collapsing hierarchy

- Split source data into parts according to specified separators

Collapsing hierarchy (2)

<table>
<thead>
<tr>
<th>START</th>
<th>17.8.1996</th>
<th>95.50</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>END</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  - Split source data into parts according to specified separators

Collapsing hierarchy (3)

- split parts into sub-parts, according to sub-elements
Collapsing hierarchy (4)

```xml
<date>17.8.1996</date>
<sum>95.50</sum>
```

Collapsing hierarchy (5)

- **Input part**
  - `+ <date />` result
  - `+ <xw:collapse />`

  ```xml
  <date>17.8.1996</date>
  <sum>95.50</sum>
  ```

Adding levels of hierarchy

- **Example: Recognizing IP addresses in binary data**

  ```xml
  <xw:ELEMENT xw:name="IP-address">
    <a xw:type="byte"/>
    <b xw:type="byte"/>
    <c xw:type="byte"/>
    <d xw:type="byte"/>
  </xw:ELEMENT>
  ```

Adding levels of hierarchy (2)

- **Binary data = string of bytes**

  ```
  193 167 232 253
  ```

Adding levels of hierarchy (3)

- **NB: an xw:ELEMENT does not correspond to parts of input data (like ordinary result elements do):**

  ```xml
  <!-- Wrap first two lines as INTRO: -->
  <!--lines are matched by these elements:-->
  <data xw:childterminator="\n" />
  <xw:ELEMENT xw:name="INTRO">
  </xw:ELEMENT>
  ```

Rearranging content

- **Content can be rearranged by storing results temporarily in variables:**

  ```xml
  <data xw:childterminator="\n" />
  <xw:STORE xw:name="lines">
  <!-- lines are matched by these elements:-->
  </xw:STORE>
  ```

Rearranging result structures

- **<xw:STORE xw:name="xx">**

  ```xml
  <a xw:starter="A" xw:terminator="$"/>
  <b xw:starter="B"/>
  <b1 xw:starter="1"/>
  <b2 xw:starter="2"/>
  <b3 xw:starter="3"/>
  </b>
  ```

Rearranging result content

- **<xw:VALUE-OF xw:select="xx">**

  ```xml
  <a>xy..</a>
  <b>z$???</b>
  ```
XW: Implementation

- Prototype implemented with Java
- Apache Xerces 2.0.1 used as a SAX parser
  - to read the wrapper description, which is represented internally as ...
- **a wrapper tree** (käärintäpuu)
  - guides the parsing of source data

Wrapper Tree

- Wrapper tree node
  - corresponds to an element of wrapper description
  - used for matching parts of source data
  - includes sets $S$, $B$, $T$, and $F$ of delimiter strings
    - computed from wrapper description
    - $S$: element's own starter strings
    - $B$: strings that can begin part of element
      - $S \cup$ starters of subelements that can begin the part of the element
    - $T$: terminating delimiters for the part of element
    - $F$: strings that can follow the part of element

Executing a wrapper (simplified)

- Traverse the wrapper tree; In each node:
  - scan input until the start of corresponding part found (= a delimiter belonging to set $B$)
  - report `startElement(...)`
  - Either
    - process child nodes recursively, or
    - report `characters(...)` for a leaf-level element
  - scan input until the end of the part (using sets $T$ and $F$)
  - report `endElement(...)`
  - if node iterative, and a string in $B$ found, reprocess node

Development history and status

- Fall 2001: language designed from concrete examples
- 2002: Design of implementation principles, implementation
  - wrapping of separator-based and positional text data implemented
  - wrapping of binary data (and few other details) unimplemented

XW: Some possible extensions

- Evaluation of expressions
  - for generating computed attributes (**implemented**)
  - for guiding repetition (**min/maxoccurs**) by content values
- Namespace support for results
- Describing recursive (unlimited nesting) source structures
  - recognizing LL(k) languages
    - Usefulness for wrapping data formats?

XW: Summary

- XW: a convenient "XML wrapper description language"
  - Developed at Univ. of Kuopio
  - for translating legacy data to XML
  - for declarative wrapper description
  - Easier than writing ad-hoc conversion programs
  - Working prototype implementation