7 Translating Data to XML

- How to translate existing data formats to XML? (and why?)
- XW (XML Wrapper)
  - an "XML wrapper description language"
  - developed in XRAKE project, Univ. of Kuopio, 2001–02
  - Ek, Hakkakainen, Kilepäinen, Kuikka, Penttinen: Describing XML Wrappers for Information Integration. In Proc. of XM
  - Math, Univ. of Kuopio.

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
  - DICE IS
  - Enfo Group
  - JSCP Interactive
  - Kuopio University Hospital
  - Medigroup
  - SysOpen
  - TietoElinor

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)
  - interface/conversion program to produce an XML representation for source data

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
   - OK; conversion possibly efficient
   - Development and maintenance tedious
How to wrap?  (2)

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 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

- XML Namespaces
  - 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
    inputencoding="Cp850" >
  <identifier data="INVOICE">...
    <identifierdata>...
      inserted result text ...
    </identifierdata>
  </identifier>
  <specification data="PHONE SPECIFICATION" ... >
    <data data="all"/>
    <data data="maxOccurs=unbounded"/>
    ...
  </specification>
</ws:wrapper>
```

XW-architecture (1)

```
source data \rightarrow wrapper description \rightarrow result document
| \hspace{1cm} | \hspace{1cm} | \hspace{1cm} |
| AA x1 | 33 | y2 | I1 x2 |
| \hspace{1cm} | \hspace{1cm} | \hspace{1cm} |
| XW-engine | \rightarrow | \rightarrow | \rightarrow |
| post-processing | \hspace{1cm} | \hspace{1cm} | \hspace{1cm} |
| XSLT | \hspace{1cm} | \hspace{1cm} | \hspace{1cm} |
| SAX | \hspace{1cm} | \hspace{1cm} | \hspace{1cm} |
| DOM | \hspace{1cm} | \hspace{1cm} | \hspace{1cm} |
```
XW-architecture (2)

source data  wrapper description  XW-engine
AA x2
BB y1 y2
C x1
D y1 y2

SAX events
StartElement(event-att...) startElement(event-att...)
characters(String)

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

Recoginition of content parts (1)

- by separators; For example:
  <invoice xm:starter=""INV""><invoice number=""53 64""/>
  <xml:identifier data=""A""...>

- by position (within surrounding part):
  <invoice number=""53 64""/>
  (Invoice number is in positions 53..64 of the first row of an identifier data part)

XW-architecture (3)

source data  wrapper description  result document
AA x2
BB y1 y2
C x1
D y1 y2

XW Syntax

<xml:wrapper xm:sourceType="text">
  <invoice>
    <xml:identifier data=""A""...>
    <xml:specification...>
    </xml:identifier data=""A""...>
    <xml:invoice>
      <xml:wrapper/>
  </xml:invoice>
</xml:wrapper>

Splitting of source content into parts (-> elements)

Recognition of content parts (2)

- In binary data by content data types;
  For example:
  <xml:wrapper xm:sourceType="binary">
    <xml:byte/>
    <xml:string xm:stringLength="20"/>
    <xml:int/>
  </xml: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:
  ```xml
  <line xw:terminator="\n" xw:minoccurs="2" xw:maxoccurs="2"/>
  ```
  - 2 input lines --> 2 line elements

- Alternative parts:
  ```xml
  <xw:CHOICE xw:maxoccurs="unbounded">
    <CA 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" --> elements A or B

Discarding parts of data

```xml
<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

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

```xml
<data xw:starter="START"
  xw:terminator="END"
  xw:childterminator="\n">
  <!-- 'data' is made of rows -->
  <xw:collapse>
    <data xw:position="S 14"/>
  </xw:collapse>
  <sum xw:position="S 16 21"/>
  <xw:collapse>
</data>
```

Collapsing hierarchy (2)

```xml
<START 17.5 1996 95.50
  17.5 1996 95.50
  17.5 1996 95.50
  END>
```

- Split source data into parts according to specified separators

Collapsing hierarchy (3)

```xml
<data>
  <xw:collapse>
    <START 17.5 1996 95.50
      17.5 1996 95.50
    </xw:collapse>
  </data>
```

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

```
<data>
  <sum>7.8.1990</sum>
</data>
```

Collapsing hierarchy (5)

```
Input part  wrapper element
+ <data />  result
<data>7.8.1990</data>
```

Adding levels of hierarchy

Example: Recognizing IP addresses in binary data

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

Adding levels of hierarchy (2)

```
Binary data = string of bytes
<ox:collapse />
```

Adding levels of hierarchy (3)

```
<ox:ELEMENT ox:name="IP-address">
  <ox:collapse />
</ox:ELEMENT>
```

Rearranging content

```
Content can be rearranged by storing results temporarily in variables:
<data ox:childterminator="\n">
  <ox:store ox:name="lines" />
  <!-- lines are matched by these elements: -->
  <ox:collapse />
  <ox:store />
  <ox:copy-of ox:select="lines" />
</data>
```
Development 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 recently)
  - 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"
  - for translating legacy data to XML
  - declarative wrapper description
  - easier than procedural ad-hoc conversion programs
  - working prototype implementation
  - > to be available at
    [www.cs.ukk.fi/research/XWAKE](http://www.cs.ukk.fi/research/XWAKE)