8 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

XRAKE Project

- "XML-rajapintojen kehittäminen" (Developing XML-based interfaces)
- Studies 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)
  - interface/conversion program to produce an XML representation for 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
   - 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 should be available
   - with a proper methodology development and
     maintenance should become easier
=> Wrapper description language XW

XW (XML Wrapper)

- XML-based, declarative wrapper description
  language
- To convert from a
  - textual or binary source
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

How does XW look like?

```xml
<xw:wrapper xw:name="phone-invoice" xw:sourcetype="text"
  <invoice xw:starter="\^INVOICE" xw:maxoccurs="unbounded">
    <identifierdata ...
    ...
  </invoice>
  <specification xw:starter="\^PHONE SPECIFICATION" ...
  ...
  <invoicedata xw:starter="\^---------- ...
  ...
</xw:wrapper>
```

XW-architecture (1)
**XW-architecture (2)**

- **Source data**
  - AA x1
  - BB y1
  - z1

- **Wrapper description**
  - SAX events
    - startElement(part-A, ...)
    - characters("x1")

- **XW-engine**

**XW-architecture (3)**

- **Source data**
  - AA x1
  - BB y2
  - z1

- **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>
</wrapper>
```

**Recognition of content parts (1)**

- **by separators** For example:
  - `<invoice xw:starter="^INVOICE"...` for subparts

- **by position** within surrounding part:
  - `<invoiceNumber xw:position="53 64"/>
  (Invoice number is in positions 53..64 of the first row of an identifierdata or part)

**Recognition of content parts (2)**

- **In binary data by content data types**
  - For example:
    - `<xw:wrapper xw:sourcetype="binary"...` for elements A, B, and C
Recognition of content parts (3)

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

XW: Modifying the structure of data

- Limited modification possible:
  - discarding parts of data
  - collapsing levels of hierarchy
  - adding levels of hierarchy
- Not supported (yet):
  - generating new data
  - re-arranging existing data

Discarding parts of data

```xml
<spec x:w:starter="SPEC" x:w:childterminator="\n">
<!-- Split the "SPEC" into rows: -->
<!-- Ignore the first three rows: -->
<xw:ignore x:w:minoccurs="3" x:w:maxoccurs="3"/>
. . .
</spec>
```

Collapsing hierarchy

```xml
<data x:w:starter="START" x:w:terminator="END" x:w:childterminator="\n">
  <!-- 'data' is made of rows -->
  <xw:collapse>
    <date x:w:position="5 14"/>
    <sum x:w:position="16 21"/>
  </xw:collapse>
  . . .
</data>
```

Collapsing hierarchy (2)

- Split source data into parts according to specified separators

```
<data>
  <START>17.8.1996 95.50</START>
  . . .
</data>
```

Collapsing hierarchy (3)

- Split parts into sub-parts, according to sub-elements

```
<data>
  <xw:collapse>
    17.8.1996 95.50
  </xw:collapse>
  . . .
</data>
```
Collapsing hierarchy (4)

Adding levels of hierarchy

Example: Recognizing IP addresses in binary data

Adding levels of hierarchy (2)

Adding levels of hierarchy (3)

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

XW: Implementation

Wrapper Tree

- 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
  – guides the parsing of source data
Executing a wrapper (simplified)

- Traverse the wrapper tree; in each node:
  - Scan input until the start of corresponding part found (member of 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 E and F)
  - Report endElement(…)
  - If node iterative and a string in B found, reprocess node

Development status

- Fall 2001: language designed from concrete examples
- Spring 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: Possible extensions

- Generation of attributes and data content
- Re-arrangement of content
- Describing recursive (unlimited nesting) source structures
  => Recognizing LL(k) languages
  (Usefulness for wrapping data formats?)

Summary

- XW: a convenient "XML wrapper description language"
  - For translating legacy data to XML
  - Declarative wrapper description
  - Easier than develop and maintain ad-hoc conversion programs
  - Running prototype implementation