

## 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
  - Ek, Hakkarainen, Kilpeläinen, Kuikka, Penttonen: Describing XML Wrappers for Information Integration. In Proc. of *XML Finland 2001*, Tampere, Finland, Nov. 2001, 38–51.

SDPL 2002

Notes 8: XML Wrapping

1

## 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

SDPL 2002

Notes 8: XML Wrapping

2

## 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

SDPL 2002

Notes 8: XML Wrapping

3

## XW: Motivation

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

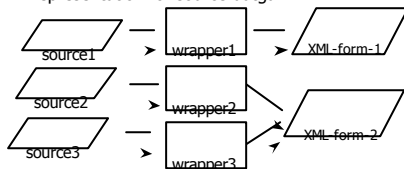
SDPL 2002

Notes 8: XML Wrapping

4

## XML-wrapping

- Need "XML-wrappers" (aka **extractors**)
  - interface/conversion program to produce an XML representation for source data



SDPL 2002

Notes 8: XML Wrapping

5

## 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



SDPL 2002

Notes 8: XML Wrapping

6

## 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

SDPL 2002

Notes 8: XML Wrapping

7

## XW (XML Wrapper)

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

SDPL 2002

Notes 8: XML Wrapping

8

## 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

SDPL 2002

Notes 8: XML Wrapping

9

## 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

SDPL 2002

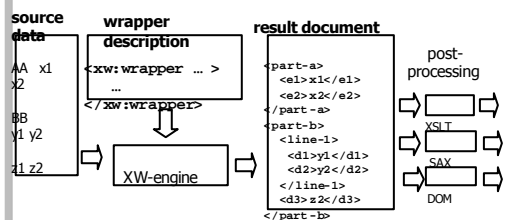
Notes 8: XML Wrapping

10

## How does XW look like?

```
<xw:wrapper xw:name="phone-invoice" xw:sourcetype="text"
  xmlns:xw="http://www.cs.uku.fi/XW/2001" >
  <invoice xw:starter="\^INVOICE" xw:maxoccurs="unbounded">
    <identifierdata ...>
      ...
    </identifierdata>
    <specification xw:starter="\^PHONE SPECIFICATION" ...>
      ...
    </specification>
    <invoicedata xw:starter="\^-----" ...>
      ...
    </invoicedata>
  </invoice>
</xw:wrapper>
```

## XW-architecture (1)

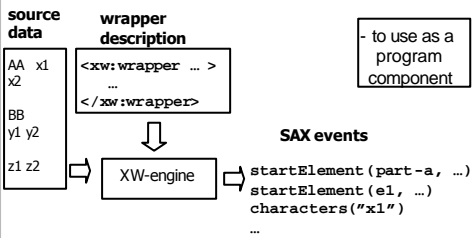


SDPL 2002

Notes 8: XML Wrapping

12

## XW-architecture (2)

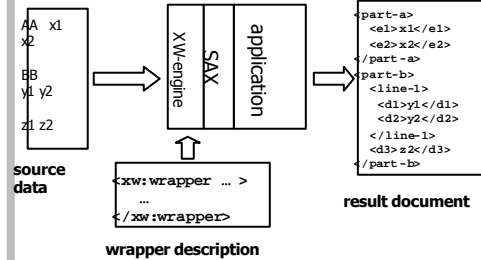


SDPL 2002

Notes 8: XML Wrapping

13

## XW-architecture (3)



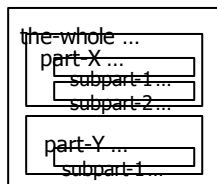
SDPL 2002

Notes 8: XML Wrapping

14

## 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



SDPL 2002

Notes 8: XML Wrapping

15

## XW Syntax

```
<xw:wrapper xw:sourceType="text"
  xmlns:xw="http://www.cs.uku.fi/XW/2001">
  <invoice ... >
    <identifierdata ... >
      ...
    </identifierdata>
    <specification ... >
      ...
    </specification>
  </invoice>
</xw:wrapper>
```

The diagram shows the XML code for an XW wrapper. A callout box points to the <identifierdata> element, stating 'Splitting of source content into parts (-> elements)'. This indicates that the content of the <identifierdata> element is split into parts.

SDPL 2002

Notes 8: XML Wrapping

16

## Recognition of content parts (1)

- by **separators**. For example:
 

```
<invoice xw:starter="^\^INVOICE" ...
  <identifierdata
    xw:childterminator="\n" ...
```

The diagram shows the XML code for an XW wrapper. A callout box points to the <identifierdata> element, stating 'for sub-parts'. This indicates that the content of the <identifierdata> element is split into parts.
- by **position** (within surrounding part):
 

```
<invoicenumber
  xw:position="53 64"/>
```

(Invoice number is in positions 53..64 of the first row of an identifierdata-part)

SDPL 2002

Notes 8: XML Wrapping

17

## 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)

SDPL 2002

Notes 8: XML Wrapping

18

## Recognition of content parts (3)

- Repetition:
 

```
<line xw:terminator="\n"
      xw:minoccurs="2" maxoccurs="2"/>
```

 - 2 rows -> 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" -> elements A or B

SDPL 2002

Notes 8: XML Wrapping

19

## 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

SDPL 2002

Notes 8: XML Wrapping

20

## Discarding parts of data

```
<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>
```

SDPL 2002

Notes 8: XML Wrapping

21

## Collapsing hierarchy

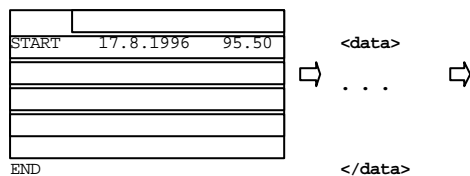
```
<data xw:starter="START"
      xw:terminator="END"
      xw:childterminator="\n">
  <!-- 'data' is made of rows -->
  <xw:collapse>
    <date xw:position="5 14"/>
    <sum xw:position="16 21"/>
  </xw:collapse>
  . . .
</data>
```

SDPL 2002

Notes 8: XML Wrapping

22

## Collapsing hierarchy (2)



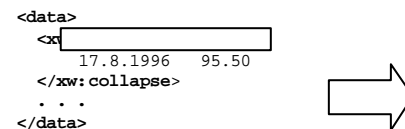
- Split source data into parts according to specified separators

SDPL 2002

Notes 8: XML Wrapping

23

## Collapsing hierarchy (3)



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

SDPL 2002

Notes 8: XML Wrapping

24

## Collapsing hierarchy (4)

```

<data>
  <xw:collapse>
    <date>17.8.1996</date>
    <sum>65.50</sum>
  </xw:collapse>
  . . .
</data>
  
```

→

```

<data>
  <date>17.8.1996</date>
  <sum>65.50</sum>
  . . .
</data>
  
```

SDPL 2002

Notes 8: XML Wrapping

25

## Adding levels of hierarchy

- Example: Recognizing IP addresses in binary data

```

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

SDPL 2002

Notes 8: XML Wrapping

26

## Adding levels of hierarchy (2)

- Binary data = string of bytes

```

193 167 232 253
  
```

⇒

```

<a>193</a>
<b>167</b>
<c>232</c>
<d>253</d>
  
```

⇒

```

<IP-address>
  <a>193</a>
  <b>167</b>
  <c>232</c>
  <d>253</d>
</IP-address>
  
```

SDPL 2002

Notes 8: XML Wrapping

27

## Adding levels of hierarchy (3)

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

```

<!-- Wrap first two lines as INTRO: -->
<data xw:childterminator="\n"/>
  <xw:ELEMENT xw:name="INTRO">
    <!-- parts recognised by childterminators:-->
    <xw:collapse /><xw:collapse />
  </xw:ELEMENT>
  ...
</data>
  
```

SDPL 2002

Notes 8: XML Wrapping

28

## 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**
  - guides the parsing of source data

SDPL 2002

Notes 8: XML Wrapping

29

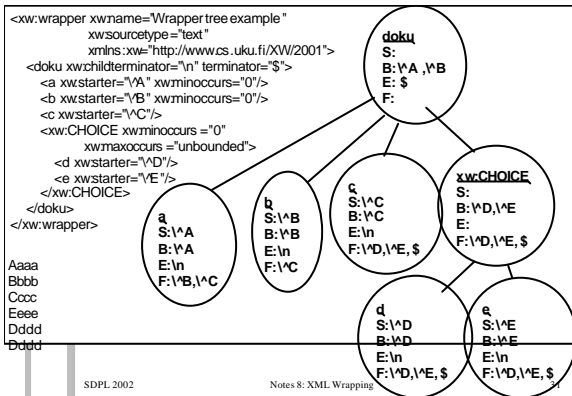
## Wrapper Tree

- Wrapper tree node
  - corresponds to an element of wrapper description
  - used for matching parts of source data
  - includes sets **S**, **B**, **E** and **F** of strings
    - computed from wrapper description
    - S**: element's own starter strings
    - B**: strings that can begin part of element
    - E**: strings that can end part of element
    - F**: strings that can follow the part of element

SDPL 2002

Notes 8: XML Wrapping

30



## 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
- SDPL 2002 Notes 8: XML Wrapping 32

## 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
- SDPL 2002 Notes 8: XML Wrapping 33

## 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?)
- SDPL 2002 Notes 8: XML Wrapping 34

## 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
- SDPL 2002 Notes 8: XML Wrapping 35