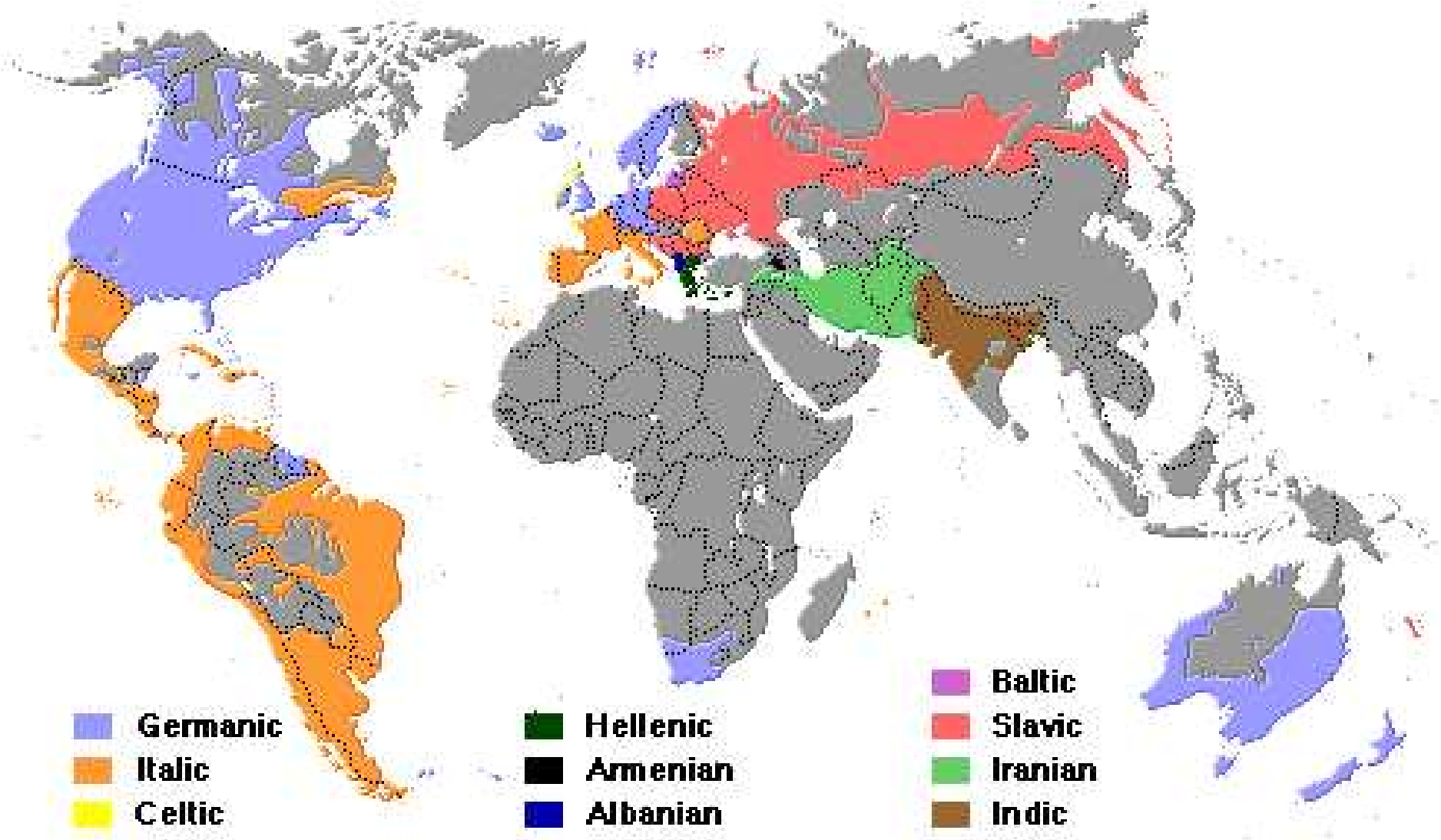
XML – from a Programming Language Perspective

Bengt Nordström

bengt@cs.chalmers.se

ChungAng University, Seoul, Korea

on leave from Chalmers University, Göteborg, Sweden


Germanic
Italic
Celtic
Hellenic
Armenian
Albanian
Baltic
Slavic
Iranian
Indic

Sweden


0 50 100 km
0 50 100 mi
Kiruna
Norwegian Sea
Tärnaby
Luleå
Umeå
Gulf of Bothnia
FINLAND
Sundsvall
Hudiksvall
NORWAY
Gävle
Uppsala
Åland Is.
(FINLAND)
Karlstad
Örebro
STOCKHOLM
ESTONIA
Norrköping
Linköping
Jönköping
Göteborg
Gotland
Kattegat
Kalmar
LATVIA
Halmstad
Karlshamn
Öland
Helsingborg
Karlskrona
Baltic Sea
Sölvesborg
LITH.
Malmö
Bornholm
(DENMARK)
DENMARK
RUSSIA
GERMANY
POLAND

Introductory remark

There is a strong tendency in Computing Science (especially in the USA) to study existing formalisms

- programming languages like C or Java,
- specification languages like UML,
- document description languages like SGML and XML

as if they were part of Nature, in the same was as natural scientists study objects in Nature.

Introductory remark

There is a strong tendency in Computing Science (especially in the USA) to study existing formalisms

- programming languages like C or Java,
- specification languages like UML,
- document description languages like SGML and XML

as if they were part of Nature, in the same was as natural scientists study objects in Nature.

This is a big mistake.

In Computing Science (and Mathematics) we create our own objects to study. Instead of studying poorly constructed artifacts we should create better ones. The only excuses for studying them are

- to avoid mistakes
- to get a better understanding of the problem they solve

This is also the reason I have been studying XML for a few months.

Question

How would XML look if it was based on modern ideas about programming lanugages and type systems?

Question

How would XML look if it was based on modern ideas about programming lanugages and type systems?

Overview:

- History
- Description of the language
- Problems
- Some solutions

Some history

1969 GML, Goldfarb, Mosher, Lorie: Generalized Markup Language. Describe structure of legal documents inside IBM.

1980 SGML. Standard in the USA.

1985 SGML. ISO standard.

1990 HTML, simplified subset of SGML. Tim Berners-Lee, to describe formatting of documents on the web.

1996 HTML becomes more and more complicated, XML starts to be developed.

1998 XML standardized.

Who developed XML?

- Computer industry: Sun Microsystems, Hewlett-Packard, Microsoft, Netscape, Adobe, Fuji, Xerox
- SGML vendors and system integrators: ArborText, Inso, SoftQuad, Grif Texcel
- Academic and research community: Text Encoding Initiative (TEI), NCSA, James Clark
- Recent additions (after XML 1.0): IBM, Oracle.

Minimal syntax of XML

What in mathematics is written

$$f(e_1, \ldots, e_n)$$

where f is a functional constant and $e_1 \ldots, e_n$ are expressions is in XML written:

<f> $e_1 \ldots e_n$ </f>

Minimal syntax of XML

What in mathematics is written

$$f(e_1, \ldots, e_n)$$

where f is a functional constant and $e_1 \ldots, e_n$ are expressions is in XML written:

<f> $e_1 \ldots e_n$ </f>

Some of the e_i may be character strings:

```
<address> <street>Ankarhjelmsvagen 30
D</street <city>Goteborg</city> </address>
```

The characters “< > & /” are not allowed inside the brackets

and “< > &” are not allowed inside the character strings.

Comparisions

XML:

```
<address>
  <street>Ankarhjelmsvägen 30 D</street>
  <city>Göteborg</city>
</address>
```

Mathematics:

```
address(street('Ankarhjelmsvägen 30 D'),
        city('Göteborg'))
```

Lisp:

```
(address (street ('Ankarhjelmsvägen 30 D')
         (city 'Göteborg')))
```

Latex:

```
\address{\street{Ankarhjelmsvagen 30 D}
         \city{Goteborg}}
```

XML syntax: question

An element looks like

`<tag> e1 ... en </tag>`

where some of `ei` can be strings.

Consider now the following element:

`<tag></tag>`

Two alternatives

- this element has no components
- this element has one component, an empty string

Which is it?

XML syntax, cont'd

Tags can have attributes:

```
<elementname attr1='string1' attr2='string2>
  ...
</elementname>
```

The attributes must be strings. Why?

The previous example

```
  <address>
    <street>Ankarhjelmsvägen 30 D</street>
    <city>Göteborg</city>
  </address>
```

could be written (what is the difference?)

```
  <address
     street = 'Ankarhjelmsvagen 30 D'
     city   = 'Goteborg'
  </address>
```

Syntactic well-formedness

Syntactic well-formedness of an XML document is divided into two parts:

- **well-formed document**: In a document, each tag must be properly matched, attribute names must be unique, etc.

Syntactic well-formedness

Syntactic well-formedness of an XML document is divided into two parts:

- **well-formed document**: In a document, each tag must be properly matched, attribute names must be unique, etc.
- **valid document**: Each document must have a type. Types in XML are called DTDs (document type definitions). Each tag is assigned a regular expression which expresses what kind of elements which may occur inside the tag.

Regular expressions in XML

Summary of XML regular expressions:

$e ::= A \parallel$	the tag A occurs
$e_1, e_2 \parallel$	the expression $e1$ followed by $e2$
$e* \parallel$	0 or more occurrences of e
$e? \parallel$	0 or 1 occurrences
$e+ \parallel$	1 or more occurrences
$e_1 \vert e_2 \parallel$	either e_1 or e_2
(e)	grouping

This looks very nice, but it is a simplification.

Typing system

Example:

```
<!ELEMENT slideshow (slide+)>
<!ELEMENT slide (title, item*)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT item (#PCDATA | item)* >
```

The regular expressions have strange restrictions:

> . . . a fi nite state automaton may be constructed from the content model using the standard algorithms, e.g. algorithm 3.5 in section 3.9 of Aho, Sethi, and Ullman [Aho/Ullman]. In many such algorithms, a follow set is constructed for each position in the regular expression (i.e., each leaf node in the syntax tree for the regular expression); if any position has a follow set in which more than one following position is labeled with the same element type name, then the content model is in error and may be reported as an error.

So I have to know automata theory to write a correct type?

Definitions in XML is a mess

There is a number of definitional mechanisms:

- parameter entities: can only occur inside a DTD
- internal entities: abbreviations of text strings, characters, etc
- external entities: reference data external to the given document.

An entity named for instance '**doc**' is referenced as '**&doc;**'.
Only text strings can be abbreviated.
Why not elements?

Why not list of elements?

Summary

XML is a language to define data. A value is a labelled tree with arbitrary number of subtrees. The leaves in the tree are text strings.

Advantages:

- data and the program which manipulates it are distinct, i.e. programs in different programming languages can share data.
- simpler than SGML
- **standardized**: wide spread acceptance

Disadvantages:

- It is easy to check if an XML document is syntactically correct, but how to prove that every XML document which is produced by a program is correct? This requires a programming language which is designed together with its type system.
- Lack of orthogonality, for instance type of attributes, definitions.
- **standardized**. A lot of odd features is there to make committee representatives happy, for instance the conformity to the SGML standard.

What should be done?

We can go in two opposite directions:

- **Remove definitions** (entities) and other things to make a simpler language for data. Attributes can also be removed.
- **Add definitions** to make it into a full programming language. In this way we can guarantee that all documents resulting from a computation will be valid.

The first suggestion is to make XML into a language for data.

The second is to make it into a programming language.

How to add definitions?

In XML we have only primitive constants (elements, constructors), they are not defined, i.e. get their meaning from outside. The idea is simply to introduce definitions (so that the document carry their meaning). A definition could have this form:

```
<?define lhs rhs>
```

Where the shape of the left hand side is (p is a pattern)

```
<elementname> p </elementname>
```

and the right hand side is any expression built up from variables in the language. The pattern p is an expression built up from variables and constructors.

The intuition is that an expression which fits the left hand side of a definition will be replaced by its right hand side (making proper substitutions for variables).

How could a new type system look?

Labelled unions:

```
<List> #A </List>=                 |   \List #A =
 <oneof>                           |     \oneof{
    <nil> <Empty/> </nil>          |        \nil{\Empty}
    <cons> #A <List> #A            |        \cons{#A \List{#A}
              </List>              |             }
    </cons>                        |           }
 </oneof>                          |
```

An element in `<List> B </List>` either has the shape `<nil/>`

or the shape `<cons> a as </cons>` , where `a` is in `B` and `as` is in

`<List> B </List>`

How could a new type system look?

Labelled products:

```
Address =
 <address>
      <street> <String/> </street>
      <city>   <String/> </city>
 </address>
```

An element in `Address` always have the shape

```
<address>
      <street> a </street>
      <city>   b </city>
 </address>
```

where both a and b are elements in `<String/>`.

Implementation

A group of students implemented the definitional mechanism a few years ago.

Future work

Dependent types are important. An example of such a type is a dependent labelled product, where the type of one part may depend on the value of previous parts.
Examples:

- a vector of length 7 is represented as a tuple containing a 7 and a vector of that length.
- a trip from A to B could be represented as a list of pairs of cities, the first city in the first pair being A, the second city in each pair being identical to the first city in the next pair and finally the second city in the last pair being B.
- the shape of an adress is dependent on the country of the adress. An adress in Sweden contains street, nr, city and zipcode. This is not the case in Korea. So the type of the adress depends on the value of the country-component of it.

References

- Extensible Markup Language (XML) 1.0 (Second Edition) http://www.w3.org/TR/REC-xml
- Haruo Hosoya and Benjamin C. Pierce. XDuce: A typed XML processing language. ACM Transactions on Internet Technology, 2002. www.cis.upenn.edu/ bcpierce/papers/

Questions?