Scalable XML Programming Techniques

Burak Emir
EPF Lausanne
Burak.Emir@epfl.ch

outline

* effective and elegant ways to write programs that deal with XML ?
* there is a lot of context
  * XML, XSD, DTD, XQuery, XSLT...
* today: treat XML world like onion
  * start high-level, outer layers
  * gradually expose technicalities

outline

* layer 1) where we are (a lot of Xs)
* layer 2) <scala/xml>
* interlude) applications
* layer 4) what is missing (more on XSDs)
* layer 5) what to do about it

layer 11 where we are

xml

- **the** syntax for data and documents
  - information exchange/transport
  - schemata: define data models
- "everybody is using it"
  - Oracle 10, MusicXML, SystemsBiologyML,...
- but **hard to write programs handling it**

XML is in Unicode

<fml>∫∫a→b</fml>

visit (xml.web.cern.ch)

```
<fruitSalad>
  <apple q="2"/>
  <orange q="1"/>
</fruitSalad>
```

xml

* saves you from writing a parser
* easy to make HTML and PDF out of it!
* application data model (e.g. schema) is a design artifact
* platform, language independent
* **but**: must also specify "semantics" of your data (schema will not suffice)
* **and**: XML (and schemata) can be verbose

Programming Paradigms

* **declarative**, special-purpose languages W3C
  * querying (XQuery)
  * recursive transformation (XSLT)
* **imperative**, general-purpose (e.g. Java)
  * today, every language has XML libraries
  * unsafe (w.r.t. schema validity)
  * "object-oriented" is orthogonal to these

* non W3C language efforts
  * XDuce (Hosoya et al)
  * CDuce (Frisch et al)
  * Xtatic (Gapeyev, Levin et al)
  * XEN, now C-omega (Microsoft)
  * XOBE (Linnemann et al)
  * E4X (BEA Systems)
  * XJ (IBM)
  * JWig, XAct ... (Moller, Schwartzbach)
  * <scala/xml> and iron (EPFL)

state of the art

(all ignored by XML practitioners:-o)

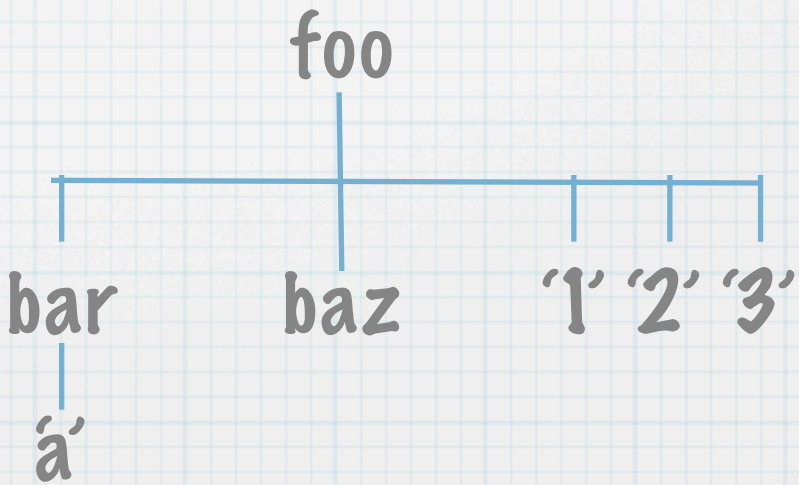
state of the art

* projects aim at
  * static type checking (using schemata!)
    * type checking is the most efficient means of program verification!
  * the "right" operations to construct, decompose, update, navigate XML
  * integrate XML syntax
  * modularity/extensibility

Tree Model

* XML documents can be considered a tree
* underlying all XML related specs
* but with slight differences...

```
<foo>
  <bar>a</bar>
  <baz/>
  123
</foo>
```



XML and Types

* schema languages to **constrain** tree structure
* DTD (Document Type Definition), part of XML spec
* mixed content
* leaves contain only text (#PCDATA)

```
<!ELEMENT foo (bar,baz*)>
<!ELEMENT bar (#PCDATA)>
```

A 'foo' element has a 'bar' and an arbitrary number of 'baz' elements as children.

XML and Types

- later: XML Schema (tries to be OO)
- will not talk about
  - Relax NG (regular tree languages)
  - DSD (boolean formulas)
  - Schematron (checks assertions in XSLT)

XPath

* In a tree, there is unique path to every node
* An XPath expression is a simple query describing a path to "select" sets (sequences) of nodes
* XML should be **typed** (valid w.r.t. some schema) if we make assumptions about result
* otherwise, unsafe

```
<fruitSalad>
  <apple q="2"/>
  <orange q="1"/>
</fruitSalad>
```

```
doc("fruitsalad.xml")/fruitSalad/apple
```

→ `<apple q="2"/>`

```
doc("fruitsalad.xml")//*[@q = 1]
```

→ `<orange q="1"/>`

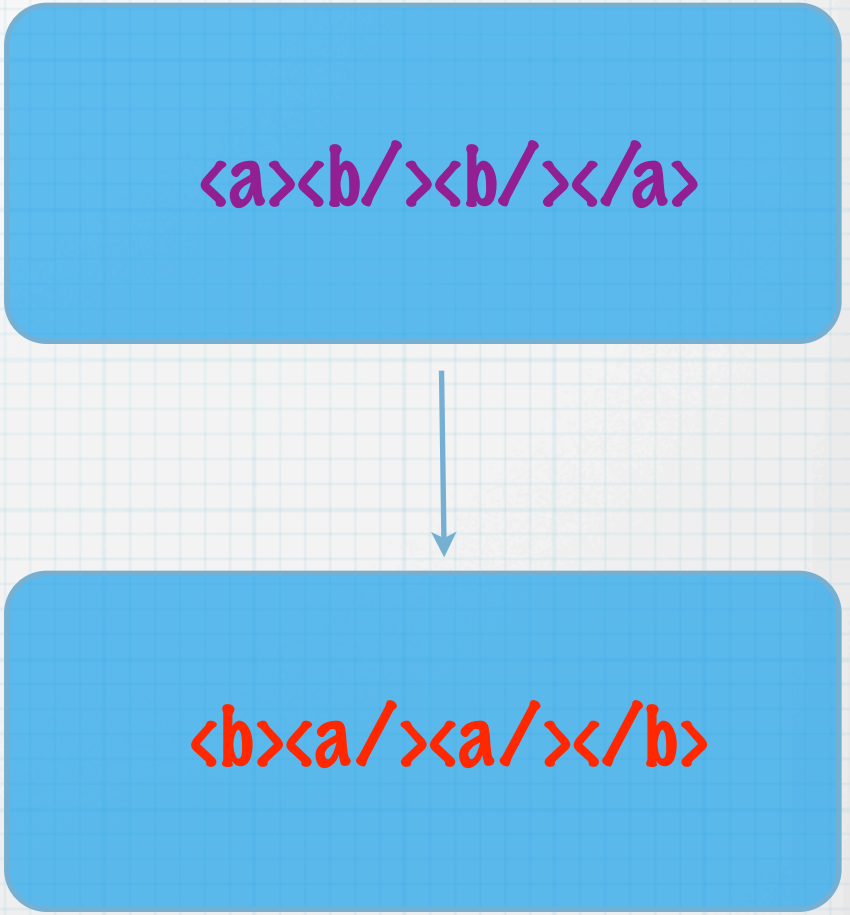
XSLT example

```
<xsl:stylesheet>

  <xsl:template match="a">
    <b><xsl:apply-templates/></b>
  </xsl:template>

  <xsl:template match="b">
    <a><xsl:apply-templates/></a>
  </xsl:template>

</xsl:stylesheet>
```



XQuery example

* "XQuery is to XML what SQL is to relational tables"
* static typing (optional)
* non XML syntax
* XML expressions

```
for $x in doc("books.xml")/bookstore/book
where $x/price>30
return <result>{ $x/title }</result>
```

taken from http://www.brics.dk/~amoeller/XML/index.html (very nice overview)

Java example

* can use Java plus some library to handle XML trees
  * SAX, DOM, JDOM, DOM4J...
* conceptual breach:-(
* remedy data binding frameworks (later)

```
import java.io.*;
import org.xml.sax.*;
import org.xml.sax.helpers.*;
import org.apache.xerces.parsers.SAXParser;

public class Flour extends DefaultHandler {

  float amount = 0;

  public void startElement(String namespaceURI,
                           String localName,
                           String qName,
                           Attributes atts) {
    if (namespaceURI.equals("http://recipes.org")
        && localName.equals("ingredient")) {
      String n = atts.getValue("","name");
      if (n.equals("flour")) {
        String a = atts.getValue("","amount"); // assume 'amou
        amount = amount + Float.valueOf(a).floatValue();
      }
    }
  }

  public static void main(String[] args) {
    Flour f = new Flour();
    SAXParser p = new SAXParser();
    p.setContentHandler(f);
    try { p.parse(args[0]); }
    catch (Exception e) {e.printStackTrace();}
    System.out.println(f.amount);
  }
}
```

layer 2) <scala/xml>

introducing Scala

* object-oriented (every value is an object)
* functional (every function is a value)
* compiled to either Java VM or to .NET
* can use all Java / .NET libraries
* vision: language with advanced type system for scalable component development
* has XML expressions
* has fancy ways to define operators

Scala XML expressions

```
import java.util.Calendar;
def doGetXML() =
  <html>
    <head>
      <title>Hello World</title>
    </head>
    <body>
      <h1>Hello World</h1>
      <p>
        The time is <b>{
Calendar.getInstance().getTime().toString()
     }</b>
      </p>
    </body>
  </html>;
```

One can mix code and XML

Scala recursive matching

```
def transform( n: Node ): Seq[Node] =
  n match {
  case <a>{ children@_* }</a> =>
    <b>{ transform( children.toList ) }</b>
  case <b>{ children@_* }</b> =>
    <a>{ transform( children.toList ) }</a>
}
def transform(s: Seq[Node]): Seq[Node] =
  List.flatten(s map { x =>
   transform(x).toList })
```

* recursive functions and pattern matching can emulate XSLT programming style
* regular expression pattern matching useful for non-XML data

Burak Emir. Scalable XML Programming Techniques. CERN Computing Seminar 2005-06-15

Scala querying

```
for (val z <- doc("books.xml")\"bookstore"\"book";
      z \ "price" > 30)
yield z \ "title"
```

* for-comprehensions and XPath methods can emulate XQuery programming style
* This is made possible by Scala's syntactic sugar for method application

n \ "foo" becomes n.\("foo")

e1 op e2 becomes e1.op(e2)

Scala XML library

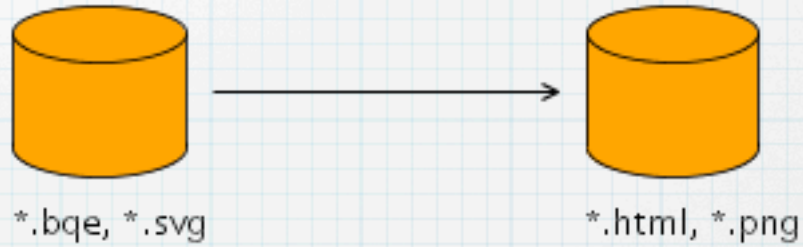
* two syntaxes
  * either XML or call constructors directly
  * XML is always immutable
* routines for parsing, validation
* Scala's type system allows reuse, this reduce amount of code one has to type

mixins

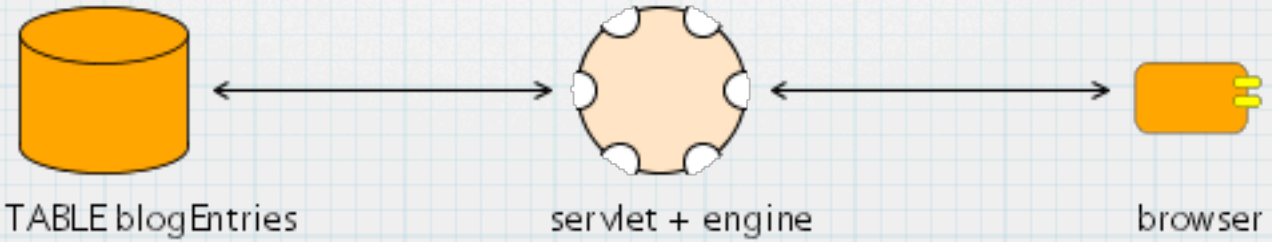
interlude) applications

bqbase - a blogging tool

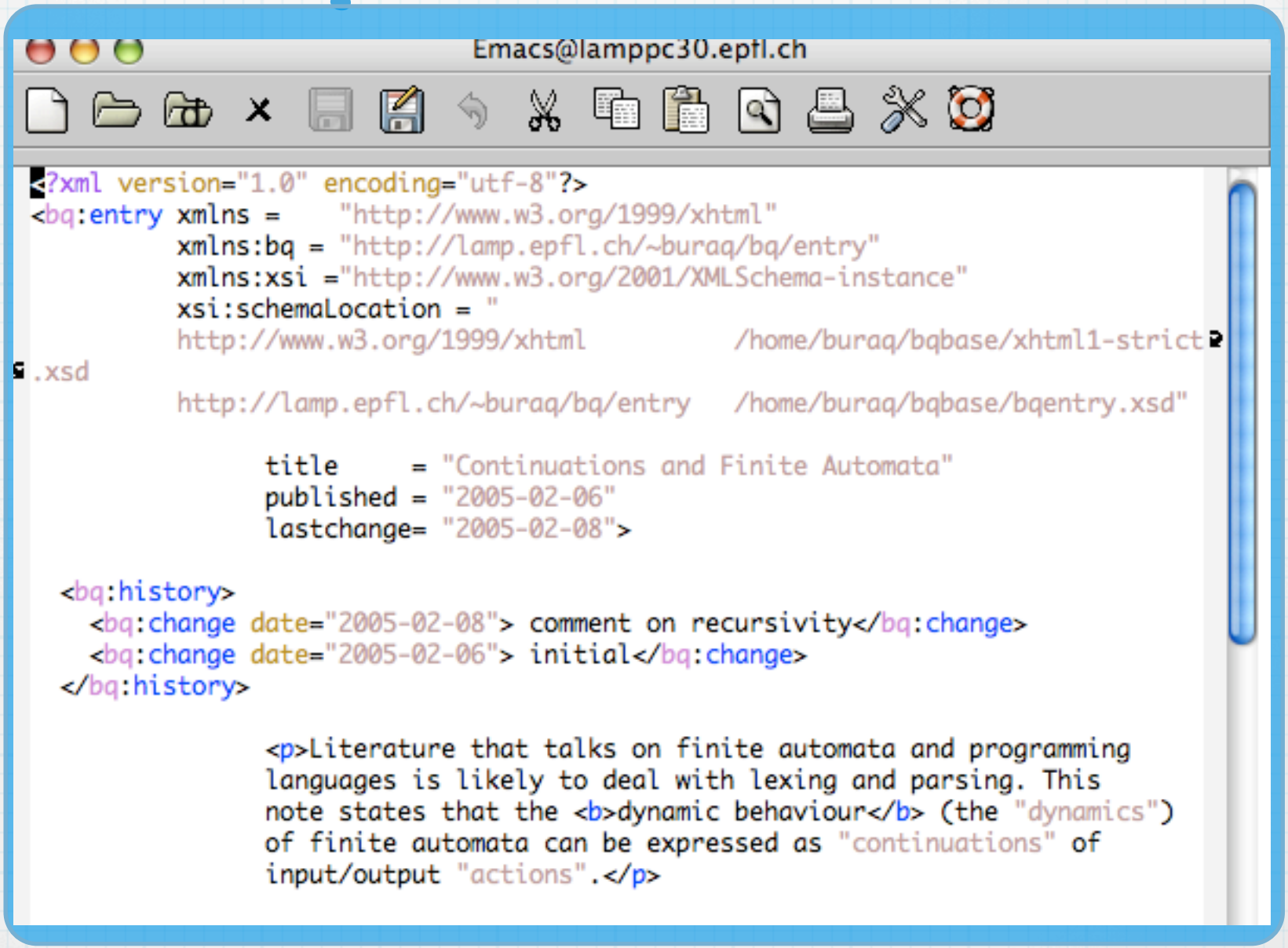
* online publishing from XML sources
* static generation



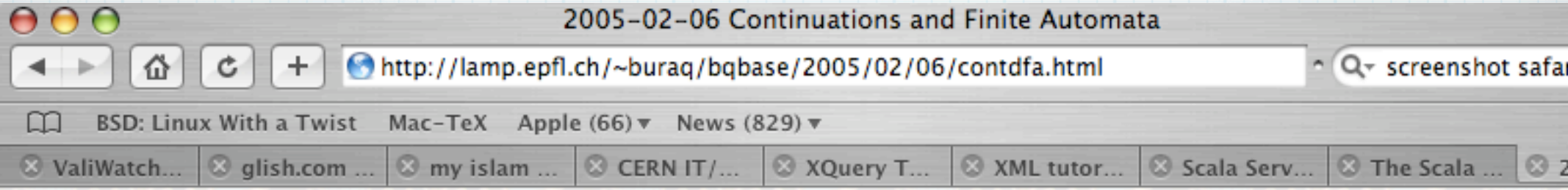
* other possibility: dynamic generation



bqbase


Emacs@lamppc30.epfl.ch
<?xml version="1.0" encoding="utf-8"?>
<bq:entry xmlns = "http://www.w3.org/1999/xhtml"
xmlns:bq = "http://lamp.epfl.ch/~buraq/bq/entry"
xmlns:xsi ="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation = "
http://www.w3.org/1999/xhtml /home/buraq/bqbase/xhtml1-strict
.xsd
http://lamp.epfl.ch/~buraq/bq/entry /home/buraq/bqbase/bqentry.xsd"
title = "Continuations and Finite Automata"
published = "2005-02-06"
lastchange= "2005-02-08">
<bq:history>
<bq:change date="2005-02-08"> comment on recursivity</bq:change>
<bq:change date="2005-02-06"> initial</bq:change>
</bq:history>
Literature that talks on finite automata and programming
languages is likely to deal with lexing and parsing. This
note states that the dynamic behaviour (the "dynamics")
of finite automata can be expressed as "continuations" of
input/output "actions".

bqbase - result



[back to bqbase index]

Continuations and Finite Automata

Literature that talks on finite automata and programming languages is likely to deal with lexing and parsing. This note states that the **dynamic behaviour** (the "dynamics") of finite automata can be expressed as "continuations" of input/output "actions".

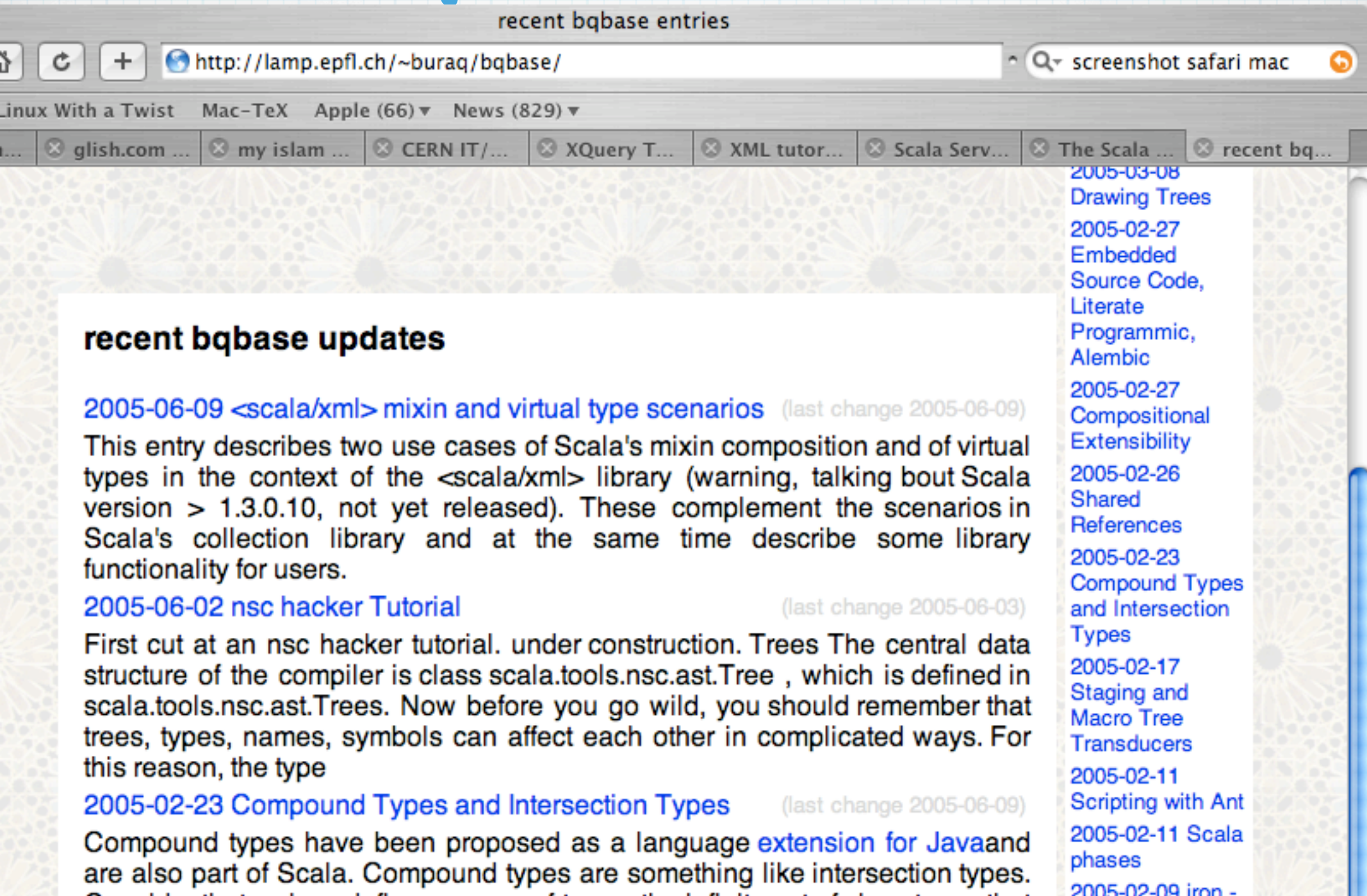
(Dear reader, if you find this entry too cryptic, please check back in a few days; I will flesh out examples and add diagrams.) We will not use a subset of "final" states but an output map beta, more general.

We'll use some vocaulary from Arbib, Manes' treatment of automata (Ch. 6 of the "Arrows, Structures and Functors"), but will not make categorical statements.

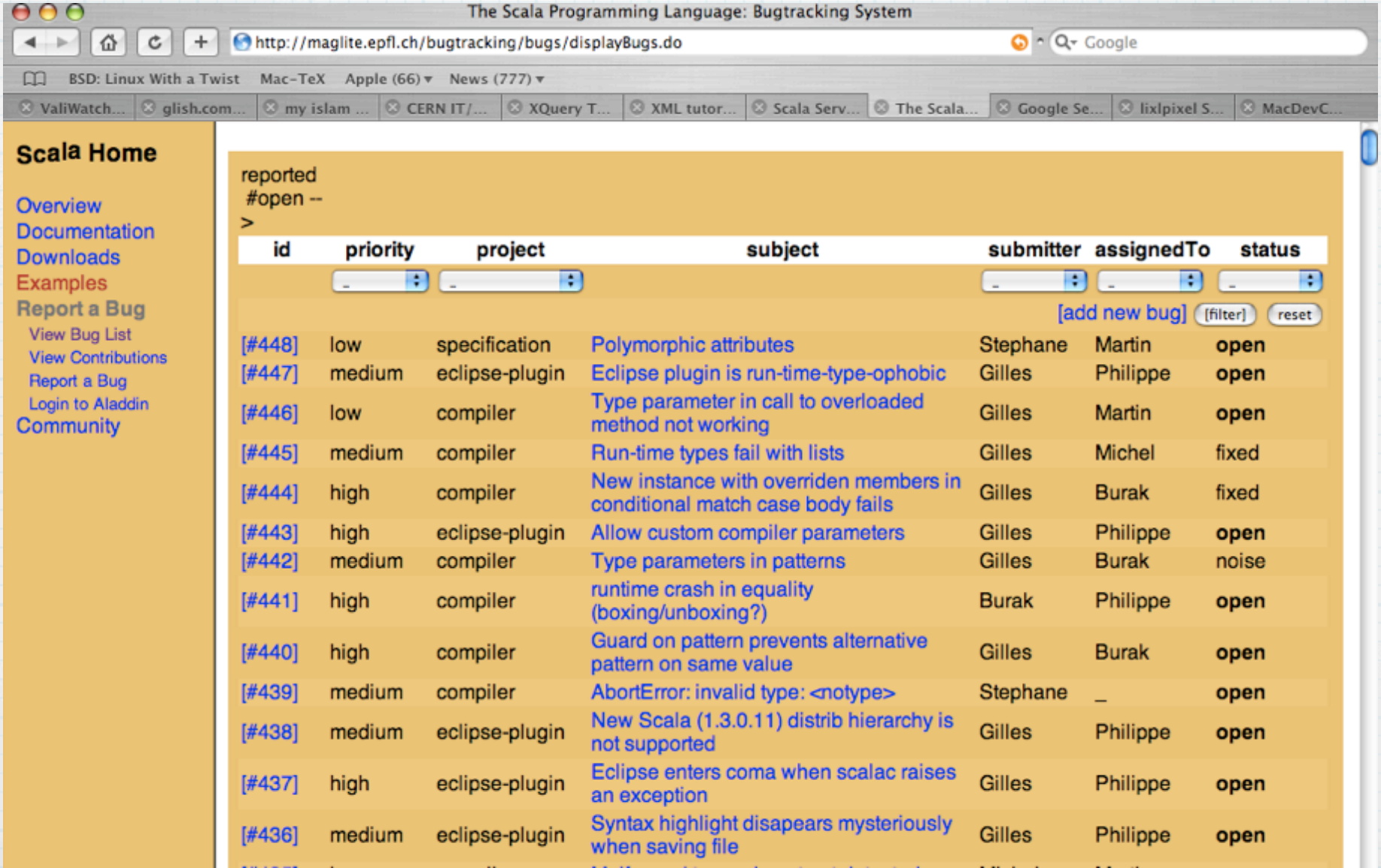
Let (X,Q,delta,q0,Y,beta) be a sequential machine. delta:Q x X → Q is the **dynamics** (transition mapping), beta:Q → Y an output map. We could refer to delta and beta as static (description of) behaviour.

The **run map** delta*: Q × X* → Q is the free extension of delta to monoid X*. The **response** of (Q,delta,beta) **started in state q** is M_q: X* → Y with M_q(w)=beta(delta*(q,w)). The **observability map** sigma: Q → Y^{X^*} assigns M_q to each q.

bqbase - result



aladdin - scala bugtracking system



id	priority	project	subject	submitter	assignedTo	status
[#448]	low	specification	Polymorphic attributes	Stephane	Martin	open
[#447]	medium	eclipse-plugin	Eclipse plugin is run-time-type-ophobic	Gilles	Philippe	open
[#446]	low	compiler	Type parameter in call to overloaded method not working	Gilles	Martin	open
[#445]	medium	compiler	Run-time types fail with lists	Gilles	Michel	fixed
[#444]	high	compiler	New instance with overriden members in conditional match case body fails	Gilles	Burak	fixed
[#443]	high	eclipse-plugin	Allow custom compiler parameters	Gilles	Philippe	open
[#442]	medium	compiler	Type parameters in patterns	Gilles	Burak	noise
[#441]	high	compiler	runtime crash in equality (boxing/unboxing?)	Burak	Philippe	open
[#440]	high	compiler	Guard on pattern prevents alternative pattern on same value	Gilles	Burak	open
[#439]	medium	compiler	AbortError: invalid type: <notype>	Stephane	_	open
[#438]	medium	eclipse-plugin	New Scala (1.3.0.11) distrib hierarchy is not supported	Gilles	Philippe	open
[#437]	high	eclipse-plugin	Eclipse enters coma when scalac raises an exception	Gilles	Philippe	open
[#436]	medium	eclipse-plugin	Syntax highlight disapears mysteriously when saving file	Gilles	Philippe	open

aladdin - scala bugtracking system

* web application
  * web service usable through a browser
* built with Java components (Apache Tomcat, Struts)
* servlets now 100% Scala

some aladdin code

```
class DisplayList extends MyScalaAction{
  def   executeAction(...) = {
    ...
    def con2row(br: BugContrib): Seq[scala.xml.Node] = {
      <td><a href={ quicklink() } >[#{br.id}]</a></td>
      <td>{ br.project }</td>
      <td><a href={ quicklink() }>{ br.subject }</a></td>
      <td>{ br.contributor }</td>
      <td>{ br.category }</td>
      <td>{ if(br.hasReport()) {
          <a href={ displayLink() }>{ br.status }</a>
        } else {
         br.status;
        }
      }</td>
    }
    ...get BugContrib from database, call con2row, send to client ...
}
```

layer 4) what is missing

integration of XML Schema

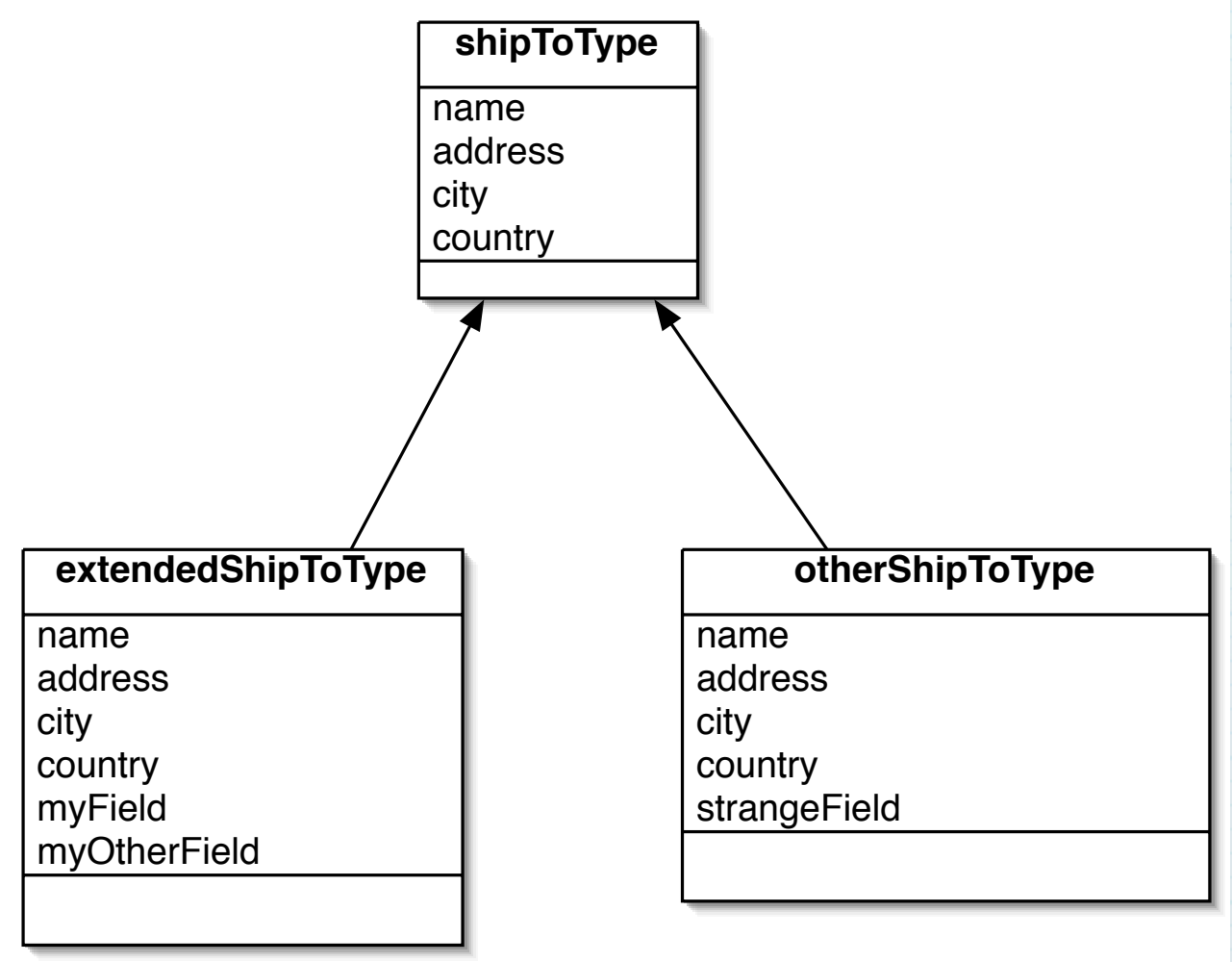
* more recent schema language
* adds datatypes (#PCDATA is history)
* adds object-orientation to the soup
  * type derivation
  * substitution groups
* problem: far too many features...
* XSD = XML Schema Definition

XSD example

```
<xs:element name="shipto">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="name" type="xs:string"/>
   <xs:element name="address" type="xs:string"/>
   <xs:element name="city" type="xs:string"/>
   <xs:element name="country" type="xs:string"/>
  </xs:sequence>
 </xs:complexType>
</xs:element>
```

```
<shipto>
 <name>Ola Nordmann</name>
 <address>Langgt 23</address>
 <city>4000 Stavanger</city>
 <country>Norway</country>
</shipto>
```

Type Hierarchies via Extension



type derivation in XSD

* type extension: add elements to the right
* type restriction: refine model (language inclusion)
* problem: cannot simulate OO style inheritance :-(

all-groups cannot participate in type extension

* problem: tedious to write up in proper XSD syntax

XSD and object/XML mismatch

* why is there no general-purpose language that is object-oriented and is compatible with XML Schema?
* missing extension of all-groups
* more generally: classes unordered (sets), XML content models ordered (sequences)
* one implication: closed-world assumption

date.month

foo / bar [3]

Data Binding

* a nice method to develop software
  * let designers make XSD data model
  * use tool to turn into Scala classes...
  * ...which get instantiated by XML parser
  * let programmer use "XML nature" or object view, depending on what is needed
* pro: independent design document :->
* con: must be aware of object mismatch :-<

layer 5) what to do next

introducing Iron

* dream: A general-purpose language whose type system encompasses (a reasonable subset of) XML Schema
  * compatible with Scala
  * but schema validity statically verified
  * plus support for reactive programming
    * concurrency, web applications
* type systems should be validated with real-life examples

iron design principles

* node types are either
  * classes (unordered, like Java, no regexp)
  * structs (ordered, like XSD)
* classes can be extended, structs can be extended and restricted
* every object is an XML node on some site, objects react to each other's messages :-)
* avoid "THE schema fallacy": allow objects to change type during computation

Conclusions

* DTDs are simple, but yield simple models
* XSDs powerful, but have many traps
* To use XML optimally, still need expert programmers
* But opting for XML keeps options open
* novel languages can steepen learning curve
* Trend: much easier to write web applications than to use native GUI libraries

scala.epfl.ch

Thanks