Native XML Processing in Object Oriented Languages

Calling XMHell from PurgatOOry

The Essence of XML

"So the Essence of XML is this: the problem it solves is not hard, and it does not solve the problem well."

[Siméon, Wadler – POPL'03]

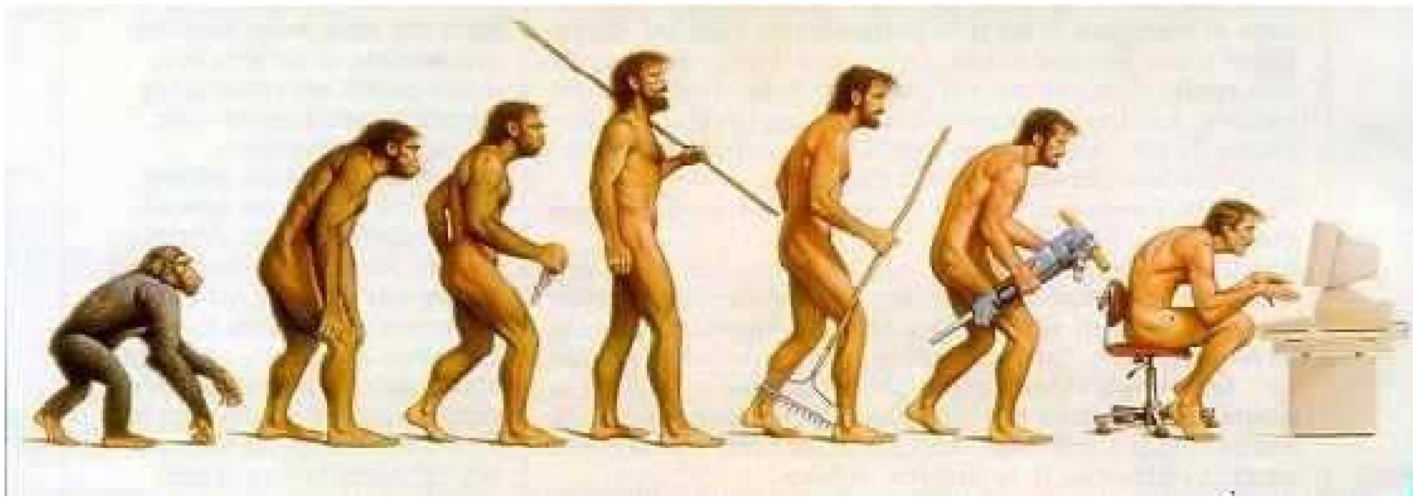
The road to XML is paved with good intentions...

- XML data is pervasive
  $\Longrightarrow$ need powerful tools to manipulate it
- XML has a rich data model
  $\Longrightarrow$ integrate it with the OO data model
- This talk is about the practical integration of the XML and OO data models
- This talk is not about
  - XML standards
    - Schema, Relax NG, ...
  - non-OO XML manipulation languages
    - XQuery, XDuce, CDuce, ...

Native XML manipulation in OO languages

- The evolution of XML integration
  From Strings to Regular Types
- Practical aspects of XML manipulation
  Generation X: XJ, Xact, and Xtatic
- Future challenges
  Xen and the Art of Language Design?

The Evolution of XML manipulation



A simple XML address book

```
<addrbk>
  <entry>
    <name>Pat</name>
    <tel>314-1593</tel>
    <email>Pat@pat.com</email>
  </entry>
  <entry>
    <name>Jo</name>
    <tel>271-8282</tel>
    <email>Jo@jo.com</email>
  </entry>
</addrbk>
```

A simple XML address book

```
<addrbk>
  <entry>
    <name>Pat</>
    <tel>314-1593</>
    <email>Pat@pat.com</>
  </entry>
  <entry>
    <name>Jo</>
    <tel>271-8282</>
    <email>Jo@jo.com</>
  </entry>
</addrbk>
```

The Stone Age
Strings

Strings

```
"<addrbk>
    <entry>
      <name>Pat</>
      <tel>314-1593</>
      <email>Pat@pat.com</>
    </entry>
    <entry>
      <name>Jo</>
      <tel>271-8282</>
      <email>Jo@jo.com</>
    </entry>
 </addrbk>"
```

- Used widely...
  - CGI
  - Java servlets
- ...with difficulties
  - Tedious to write and maintain
  - Output might not be well formed

The Bronze Age
Concrete Data Structures

Concrete Data Structures

- DOM (Document Object Model) like JDOM
  - Provide a generic, standardized AST for XML values
  - Provide an API to manipulate it
- Advantages
  - Many parsers and pretty printers available
  - Generates well formed XML
- Annoyances
  - Little or no check of validity
  - Low-level API
  - Very concrete representation
    - White space may be significant and cannot be ignored

Address book in DOM

```
Element
  name =  "addrbk"
  children ─┐
            ├─ Text
            │    data =  "  \n"
            │
            ├─ Element
            │    name =  "entry"
            │    children ─┐
            │              ├─ Element
            │              │    name =  "name"
            │              │    children ─┐
            │              │              └─ Text
            │              ├─ Text               data =  "Pat"
            │              │    data =  "  "
            │              │
            │              ├─ Element
            │              │    name =  "tel"
            │              │    children ─┐
            │              │              └─ Text
            │              └─ Element            data =  "314-1593"
            │                   name =  "email"
            │                   children ─┐
            ├─ Element                    └─ Text
            │    name =  "entry"                 data =  "Pat@pat.com"
                 . ..
```

The Middle Ages
Data Binding

Data Binding

XML language bindings are "software mechanisms that transform XML data into values that programmers can access and manipulate from within their language of choice."

[Simeoni et. al. – IEEEE Internet Computing, 2003]

- Most XML documents follow a restricted model
- Many description systems: DTD, XML-Schema, Relax...
- Translate ("bind") XML types S to classes $[\![S]\!]$ and XML values d satisfying S to objects $[\![d]\!]_S$ of class $[\![S]\!]$
- Address book type:

$$\begin{aligned} Addrbk &= \texttt{<addrbk>} Entry * \texttt{</>} \\ Entry &= \texttt{<entry>} \\ &\quad \texttt{<name>} pcdata \texttt{</>}, \texttt{<tel>} pcdata \texttt{</>}, \texttt{<email>} pcdata \texttt{</>} \\ &\texttt{</entry>} \end{aligned}$$

Binding Structure

- Reflect XML structure in the OO type system.

```
type Addrbk =
  <addrbk> Entry* </>

type Entry =
  <entry>
    <name>pcdata</>,
    <tel>pcdata</>,
    <email>pcdata</>
  </entry>
```

```
class Addrbk {
  List entries; }

class Entry {

   Name   name;
   Tel    tel;
   Email  email;
}

class Name  { String value; }
class Tel   { String value; }
class Email { String value; }
```

Binding Values

- Reflect XML Values as objects

```
<addrbk>

  <entry>
    <name>Pat</>
    <tel>314-1593</>
    <email>Pat@pat.com</>
  </entry>

  <entry>
    <name>Jo</>
    <tel>271-8282</>
    <email>Jo@jo.com</>
  </entry>

</addrbk>
```

```
Addrbk ab = new Addrbk(
  new List(
    new Entry(
      new Name("Pat"),
      new Tel("314-1593"),
      new Email ("Pat@pat.com")
  ),
  new List(
    new Entry(
      new Name("Jo"),
      new Tel("271-8282"),
      new Email("Jo@jo.com")
  ),
  EmptyList))
)
```

Data Binding

Advantages

- Cleaner representation, easier to navigate
- Automatic generators (Castor, JAXB, Relaxer)
- Some statically checked constraints (OO type system)

Annoyances

- Application (or schema) specific
- Errors reported at the level of the host language
- Some features are tricky to reflect
  - Union (no union of classes)
  - Distributivity laws

```
<acq> (<friend/> | <work/>) </acq> =
        (<acq> <friend/> </acq>) | (<acq> <work/> </acq>)
```

Enlightenment
The rise of Regular Types

Regular Types [Hosoya, Vouillon, Pierce – ICFP'00]

Do not *reflect* XML structure, *add* it as *types*!

- Regular expressions. . .

$$T = () \mid T_1, T_2 \mid T_1|T_2 \mid T*$$

- . . . containing trees. . .

$$T = () \mid T_1, T_2 \mid T_1|T_2 \mid T* \mid \texttt{<l>}T\texttt{</l>}$$

- . . . and recursive definitions (vertical recursion)

$$T = () \mid T_1, T_2 \mid T_1|T_2 \mid T* \mid \texttt{<l>}T\texttt{</l>} \mid X$$

$$E = \{\texttt{type}\ X = T\}$$

$$\texttt{type}\ Folder = \texttt{<folder>}Name, (Folder|File)*\texttt{</>}$$

$$\texttt{type}\ File = \texttt{<file>}Name, Content\texttt{</>}$$

$$\texttt{type}\ Name = \texttt{<name>}pcdata\texttt{</>}$$

$$\texttt{type}\ Content = \texttt{<content>}pcdata\texttt{</>}$$

Technical note: This defines more than regular tree languages

$\Longrightarrow$ restrict the position of variables inside an element

Regular Types as a language

- Types correspond to a language (a set of sequences of trees)
- Intuitive denotation of regular types

$$
\begin{aligned}
[\![()]\!] &= \{()\} \\
[\![T_1, T_2]\!] &= \{t_1, t_2 \mid t_1 \in [\![T_1]\!],\ \ t_2 \in [\![T_2]\!]\} \\
[\![T_1|T_2]\!] &= [\![T_1]\!] \cup [\![T_2]\!] \\
[\![T*]\!] &= \{t_1, \ldots, t_n \mid n \geqslant 0,\ \ \forall k \in [1..n].t_k \in [\![T]\!]\} \\
[\![\texttt{<l>}T\texttt{</l>}]\!] &= \{\texttt{<l>}t\texttt{</l>} \mid t \in [\![T]\!]\} \\
[\![X]\!] &= [\![T]\!] \qquad \text{if } (\texttt{type } X = T) \in E
\end{aligned}
$$

- Typing is set membership $\quad t : T \iff t \in [\![T]\!]$

Types and Values

```
type Addrbk = <addrbk>(Friend | Colleague) * </addrbk>
type Friend = <entry> <acq><friend/></>, <name>pcdata</>, <tel>pcdata</>,
                      (<email>pcdata</>)?, <addr>pcdata</>  </entry>
type Colleague = <entry> <acq><work/></>, <name>pcdata</>, <tel>pcdata</>,
                      <email>pcdata</>, <dept>pcdata</>  </entry>
```

```
<addrbk>
  <entry><acq><friend/></>, <name>Pat</>, <tel>314-1593</>,
         <addr>42, Wallaby Way</> </entry>
  <entry><acq><work/></>, <name>Jo</>, <tel>271-8282</>,
         <email>Jo@jo.com</>, <dept>CIS</> </entry>
</addrbk>
```

Practical Aspects of XML Manipulation

- Creation, exploration, and modification of XML values.
- Subtyping; interaction of regular types with OO types.
- Compilation and run-time representation.

Generation X

XJ Bordawekar, Burke, Harren, Raghavachari, Sharkar, Shmueli

- IBM Research, Thomas J. Watson Research Center

Xobe Kempa, Linnemann

- Universität zu Lübeck

Xact Christensen, Kirkegaard, Møller, Schwartzbach

- BRICS

Xtatic Gapeyev, Levin, Pierce, Schmitt, Sumii

- University of Pennsylvania

An overview. . .

	XJ	Xobe	Xact	Xtatic
Language	Java	Java	Java	C#
Exploration	XPath	XPath	XPath	Pattern Matching
Mutation	Imperative	Declarative	Declarative	Declarative
XML in Objects	Yes	Yes	Yes	Yes
Objects in XML	No	No	No	Objects as Labels
Subtyping	Nominal	Structural	?	Structural
Type Checking	Dynamic	Static	Static	Static
XML at Runtime	DOM	DOM	Lazy List	Lazy List

- Creation, exploration, and modification of XML values.
- Subtyping; interaction of regular types with OO types.
- Compilation and run-time representation.

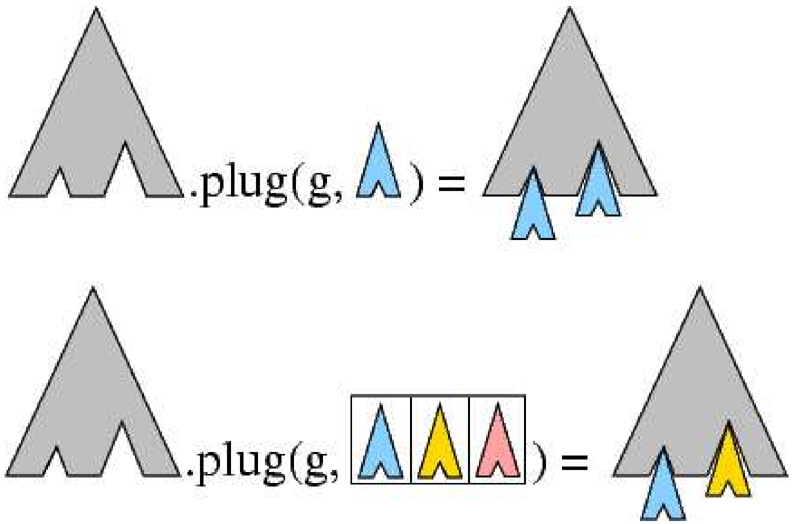
Creating XML

Most languages embed XML concrete syntax with some escaping mechanism (pcdata, variables):

```
[[Friend]] pat = [[<entry> <acq><friend/></>, <name>`Pat`</>,
                           <tel>`314-1593`</>, <addr>`42, Wallaby Way`</>
                   </entry>]]
[[Addrbk]] ad = [[<addrbk>pat</>]]
```

Creating XML: the Xact way

- XML templates: XML with named holes
- XML templates may be plugged into holes



[Schwartzbach – `http://www.brics.dk/~ck/jaoo2003/`]

Exploring trees using XPath

Where does my friend Pat live? 42, Wallaby Way

The XPath way: Giving directions and returning all results

`//entry[acq/friend][name/text() = "Pat"]/addr/text()`

1. Find all `entry` children anywhere
2. Consider those that have a `<acq><friend/></>` child
3. Consider those that also have a `<name>Pat</>` child
4. Look at what is in the `<addr>···</>` child
5. Return the text there

```
<addrbk>
 <entry><acq><friend/></>, <name>Pat</>, <tel>314-1593</>,
         <addr>42, Wallaby Way</> </entry>
 <entry><acq><work/></>, <name>Jo</>, <tel>271-8282</>,
         <email>Jo@jo.com</>, <dept>CIS</> </entry>
</addrbk>
```

Exploring trees using Patterns

Where does my friend Pat live? 42, Wallaby Way

The pattern matching way: giving a map [Hosoya, Pierce – POPL'01]

```
<addrbk>any,
 <entry><acq><friend/></>, <name>Pat</>, any,
         <addr>pcdata x</> </entry>
      any
</addrbk>
```

```
<addrbk>
 <entry><acq><friend/></>, <name>Pat</>, <tel>314-1593</>,
         <addr>42, Wallaby Way</> </entry>
 <entry><acq><work/></>, <name>Jo</>, <tel>271-8282</>,
         <email>Jo@jo.com</>, <dept>CIS</> </entry>
</addrbk>
```

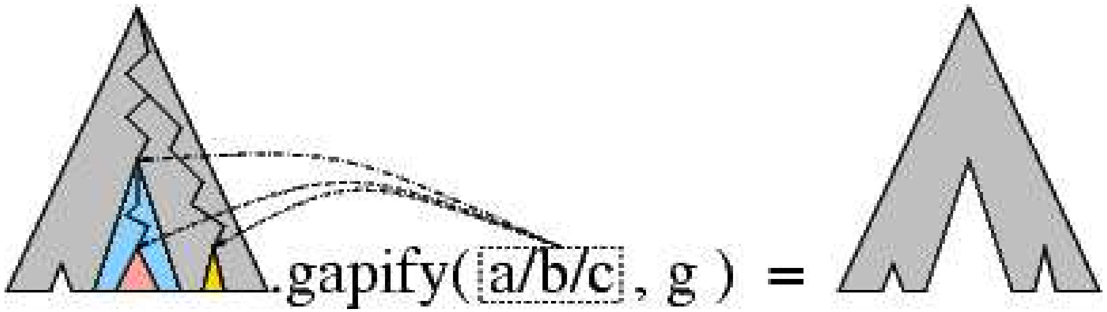
Modifying XML in XJ

- Imperative assignment
  - Closer to OO style
- Substructure extraction using XPath
- Modification pointed by an XPath expression

```
‘/addrbk/entry[name/text() = "Pat"]/addr/text()‘ = "4, Privet Drive"
```

Modifying XML in Xact

- Declarative approach (XML data is immutable)
  - Sharing of substructures, Concurrency, Static Analysis
- Extraction of substructures using XPath
  - To select a subtree
- Named holes may be created in a template
  - To select the context of a subtree



Modifying XML in Xtatic

- Declarative approach
- XML fragment extraction using pattern matching, followed by simple recombination

```
match (person) {
 case [[ <entry>Acq k, Name n, Tel t, any</entry> ]]:
      res = [[ <entry>k, n, t</> ]];
}
```

- Creation, exploration, and modification of XML values.
- Subtyping; interaction of regular types with OO types.
- Compilation and run-time representation.

A type is a type is a type. . . Subtyping

The essence of subtyping:

> *If an operation is guaranteed to be safe on a value of the supertype, then it is safe on a value of the subtype.*

Subtyping for OO types

In the OO world, there already are two forms of subtyping:

Structural (OCaml):

- Subtyping of two classes depends on the presence and type of their fields and methods
- Independent of class hierarchy
- Rich (and complex)

Nominal (Java, C#):

- Subtyping is *declared* (inheritance)
- Class hierarchy checked to satisfy structural subtyping
  - Nominal subtyping implies structural subtyping
- Simplifies type checking

Subtyping for Regular Types

As in the OO world, two forms of subtyping can be considered:

Structural $T+ \sqsubset_{\mathcal{S}} T*$

(A sequence of 1 or more Ts is a sequence of 0 or more Ts)

Nominal $Km \sqsubset_{\mathcal{S}} Distance$

(A distance in `km` is a distance)

$$\texttt{type}\ Distance = \texttt{<distance>}\ Value,\ (\texttt{<km/>}|\texttt{<miles/>})\ \texttt{</>}$$

$$\texttt{type}\ Km = \texttt{<distance>}\ Value,\ \texttt{<km/></>}$$

$$\texttt{type}\ Value = \texttt{<val>int</>}$$

Structural subtyping for Regular Types

- Each Regular Type is a language
- Subtyping is simply language inclusion
  $T \sqsubset_{\mathcal{S}} T' \iff [\![T]\!] \subseteq [\![T']\!]$
  - Intuitive: $t \in [\![T]\!]$ and $T \sqsubset_{\mathcal{S}} T'$ implies $t \in [\![T']\!]$
  - Immediately satisfies many properties
    - Distributivity of union over sequences and trees

$$[\![\texttt{<acq>}\ (\texttt{<friend/>} \mid \texttt{<work/>})\ \texttt{</acq>}]\!] =$$
$$[\![(\texttt{<acq>}\ \texttt{<friend/>}\ \texttt{</acq>}) \mid (\texttt{<acq>}\ \texttt{<work/>}\ \texttt{</acq>})]\!]$$

    - Associativity of sequence concatenation

Nominal Subtyping of Regular Types

Several approaches to nominal subtyping

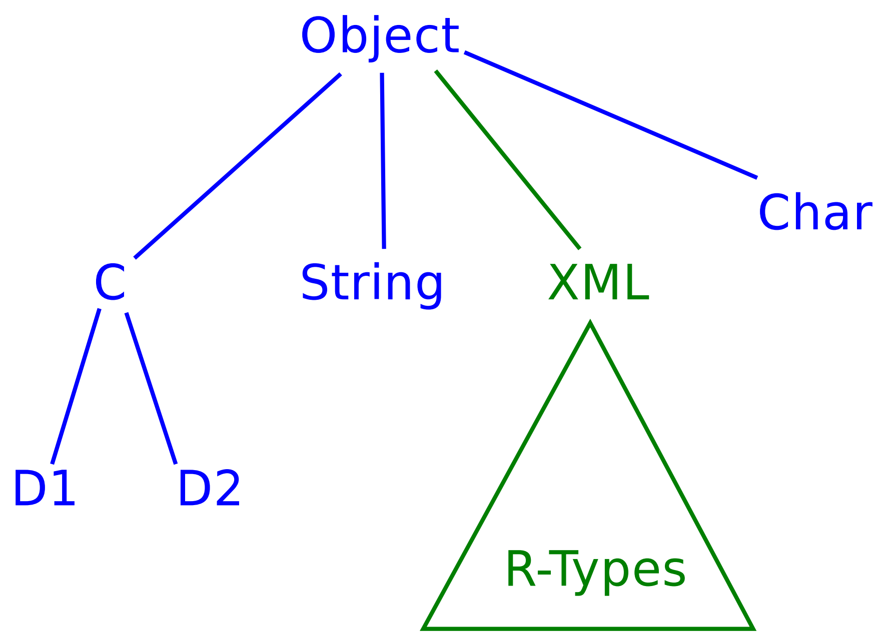
- Purely nominal: every type declared has a name
- Structural horizontally, Nominal vertically
  - Language inclusion of *regular expressions of labels*
    $T = () \mid T_1, T_2 \mid T_1|T_2 \mid T* \mid \mathcal{L}$
  - Declare subtyping of *elements* by their label in $\mathcal{L}$
  - In Schema, labels are pairs (element, type name)
- Allows finer distinctions (Mars Climate Orbiter):

$$miles \neq km \implies \texttt{<}\mathbf{height} :: miles\texttt{>}int\texttt{</>} \neq \texttt{<}\mathbf{height} :: km\texttt{>}int\texttt{</>}$$

- Subtyping is faster
- Must still be structural: $T \sqsubset_{\mathcal{N}} T' \implies T \sqsubset_{\mathcal{S}} T'$
- Need to explicitly state all subtyping relations

Mixing XML and Objects

- Sequences are objects of class *XML*
  - May be used in collections



- Most languages follow this approach

Labels as Objects in Xtatic

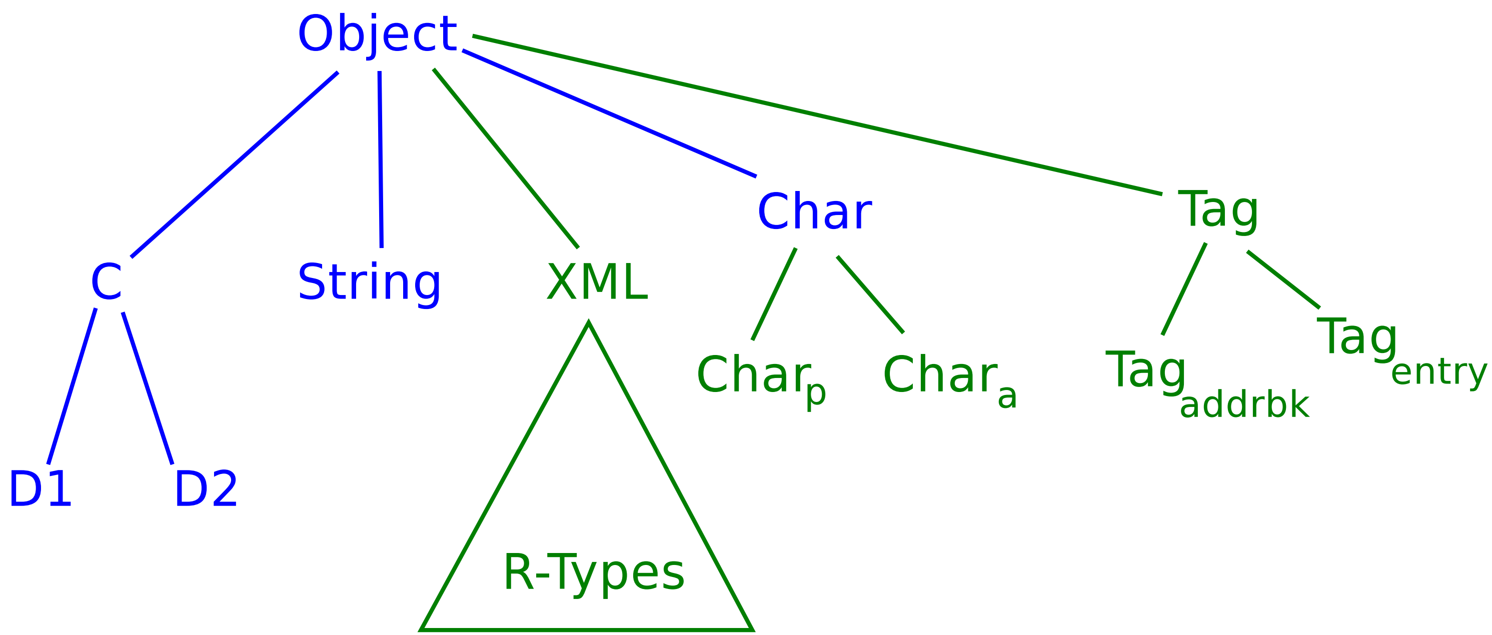
- Labels are objects, Label types are classes

$$T = () \mid T_1, T_2 \mid T_1 | T_2 \mid T* \mid <(\texttt{C})>T</>$$

- XML tags are singleton classes, subclass of Tag:
  $<\texttt{addrbk}> \cdots </> \equiv <(\texttt{Tag}_{\texttt{addrbk}})> \cdots </>$
- Characters are singleton classes, subclass of $Char$:
  $`\texttt{Pat}` \equiv <(\texttt{Char}_{\texttt{P}})/><(\texttt{Char}_{\texttt{a}})/><(\texttt{Char}_{\texttt{t}})/>$
  - Pattern matching used for string regular expressions

```
regtype url_protocols [[ ‘http‘ | ‘ftp‘ | ‘https‘ ]]
regtype url [[ url_protocols , ‘://‘ , (url_char *) ]]
...
case [[ url u, any rest ]] :
  res = [[ res , <a href = u>u</> ]]; p = rest;
```

The Class Struggle



Mixing Structural and Nominal Subtyping

- Structural subtyping for sequences
- Nominal subtyping for labels
  - Use the class hierarchy

$$\mathit{Miles} \not\sqsubset_{\mathcal{C}} \mathit{Km} \implies <\mathtt{height}><(\mathtt{Miles})/></> \not\sqsubset_{\mathcal{S}} <\mathtt{height}><(\mathtt{Km})/></>$$

but

$$\mathit{Miles} \sqsubset_{\mathcal{C}} \mathit{Int} \implies <\mathtt{height}><(\mathtt{Miles})/></> \sqsubset_{\mathcal{S}} <\mathtt{height}><(\mathtt{Int})/></>$$

- Interesting theoretical construction [Gapeyev, Pierce – Ecoop'03]

- Creation, exploration, and modification of XML values.
- Subtyping; interaction of regular types with OO types.
- Compilation and run-time representation.

Source to source translations

All these XML manipulation languages. . .

- Are language extensions
- Provide access to all language features
- Provide access to all libraries

$\Longrightarrow$ either

- Write a full Java / C# compiler
- Write a source to source compiler
  - Translation of regular types and values
  - Type checking
  - Run-time representation

The Holy Grail

Faithful Data Binding (regular types as OO types)

- Translation $[\![\,]\!]$ of types and values to target language
- Exact correspondence for typing and subtyping:
  $v :_{ext} T \iff [\![v]\!] : [\![T]\!]$ and $T \sqsubset_{ext} T' \iff [\![T]\!] \sqsubset [\![T']\!]$
- Uses existing typing/introspection infrastructure
- May still require type checking for the extension
  - Precise error localization and reporting
  - Type inference

but not there yet. . .

- May be impossible with structural subtyping

Heterogeneous vs Homogeneous translation

Heterogeneous *Fitting square pegs into round holes*

- Approximates faithful data-binding
- Add *coercions* to regain lost subtyping relations
- Complex to design
- Efficiency?

Homogeneous *Where did my type go?*

- Simpler compilation: forget about regular types
- But. . . first need to typecheck them
- What to do when types are needed?
  - Method overloading → name mangling
  - Separate compilation → store types
  - Introspection (reflection) → type stamps

Type Checking

XJ [Haren et al – IBM RC23007]

- Usual type checking (regular types in the language)
- XPath expressions typed with XAEL [Fokoué – Unpublished]
- Imperative XML modifications typed dynamically

Xact [Kirkegaard, Møller, Schwartzbach – BRICS RS-03-19]

- Static validation on demand
  - Symbolic evaluation of XML transformations
  - Based on control flow graphs
- Guarantees satisfaction of a given DTD

Xtatic [Gapeyev, Pierce – Ecoop'03]

- Usual type-checking (regular types in the language)
  - Based on Xduce [Hosoya, Vouillon, Pierce – ICFP'00]
- Inference of types of bound variables in patterns

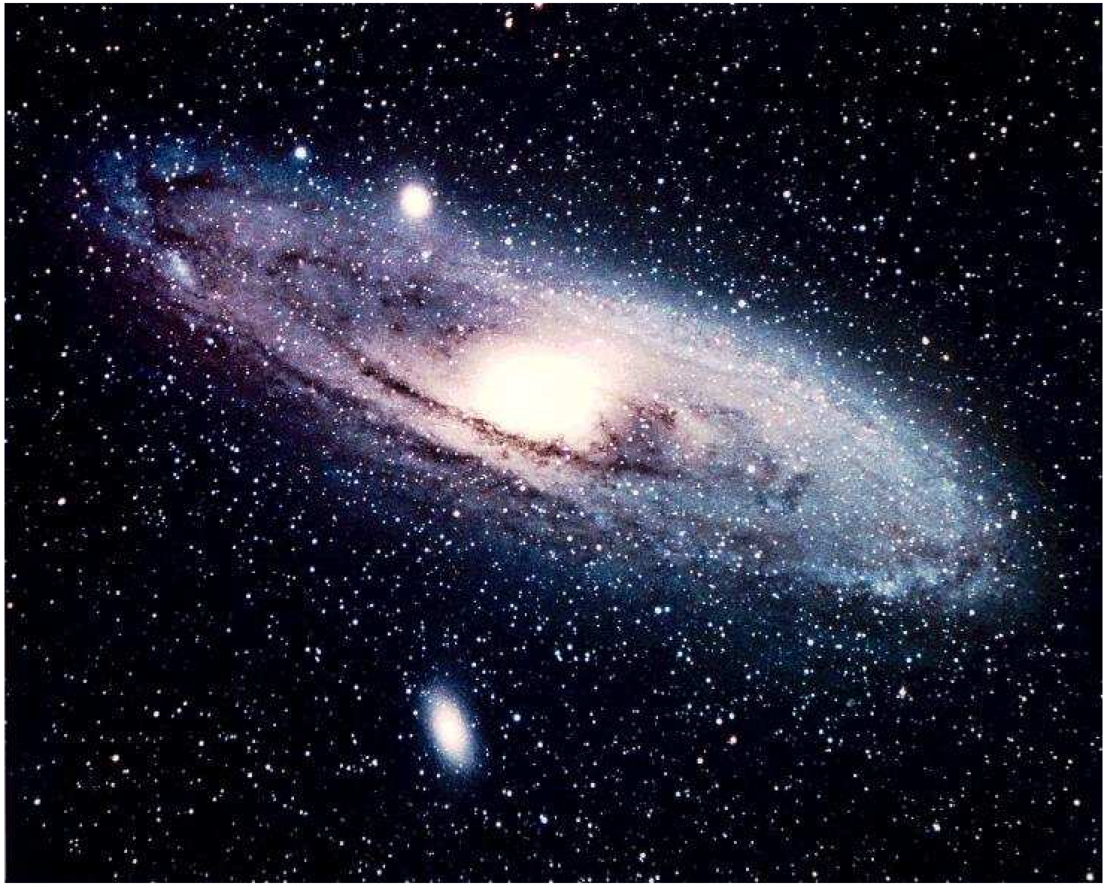
Xtatic: Type Inference in Patterns

```
static [[ Phbk ]] mkPhbk ([[ Addrbk ]] addr) {
 [[ PhPers* ]] res = [[ ]];
 [[ <addrbk> (Friend|Colleague)* pers</> ]] = addr;
 bool cont = true;
 while (cont) {
  match (pers) {
   case [[ <entry>Acq k, Name n, Tel t, any</entry>, any rest ]]:
        res = [[ res, <entry>k, n, t</> ]];
        pers = rest;
   case [[ ]]:
        cont = false;
  } }
  return [[ <addrbk>res</> ]]; }
```

Run-time representations

- XJ and Xobe use a DOM representation
  - Mutable doubly linked tree
  - Useful for XJ (imperative modification of XML)
- Xact and Xtatic use a custom representation
  - Immutable singly linked tree
    - Sharing of substructures
    - Lazy concatenation for efficiency
  - Xact: [Christensen, Kirkegaard, Møller – BRICS RS-03-29]
  - Xtatic: [Levin – ICFP'03], [Gapeyev, Levin, Pierce, Schmitt – MS-CIS-03-43]

To Infinity and Beyond



Boolean object types

- Needed for precise type inference of bound variables

  $$\texttt{case}\ [[<(\texttt{A x})/> \mid <(\texttt{B x})/>]] : \ldots$$

  `x` should have type `A` | `B`

- Integrates nicely with an homogeneous compilation framework: only need to extend the typechecker.

- Current work extends FJ [Igarashi, Pierce, Wadler – OOPSLA'99] with union [Nagira, Igarashi – JSSST'03]

Filters

- Regular extension of pattern-matching clauses [Hosoya – PlanX'04]
- A clause is a pattern and an expression
- Example: transform every entry of an address book

```
static [[ Phbk ]] mkPhbk ([[ Addrbk ]] addr) {
 filter addr {
  <addrbk>
   ( <entry>Acq k, Name n, Tel t, any</entry> {<entry>k, n, t</>} )*
  </addrbk>
 }
}
```

- Similar to Cduce `map` or `transform` [Benzaken, Castagna, Frisch – ICFP'03]
- Integrates language features (loops) into pattern matching

Strategies of Pattern Matching

- Greedy [Frisch, Cardelli – PlanX'04]
  - Most common approach, simple to implement
  - Approximation of longest match
- Lazy
  - Very useful in practice (Find the first URL)
  - Recovered by stateful loops and first match policy

```
while (cont) {
 match (curr) {
   case [[ url u, any rest ]]:  curr = rest; ...
   case [[ one_char c, any rest ]]:  curr = rest; ...
   case [[ ]]:  cont = false
 }
}
```

  - Interesting typing questions (Type of pcdata without any URL?)

Strategies of Pattern Matching

- Multi
  - Return all results
  - May bridge the gap between XPath and pattern matching
- Deep
  - Apply a transformation anywhere in the tree
    - Extension of filters with vertical recursion
  - Avoids boilerplate code
  - Challenging design and typing issues

Deeper Integration with OO

Types

- Mixing nominal and structural systems
- Integration of structural regular subtyping with languages that have structural OO subtyping (OCaml: CamlDuce?)

Sequences as objects

- XJ: sequences are Java lists
  - `sequence.size()`
- Scala: For-Comprehensions
  - List to list transformation
  - `for {val p <- persons; p.age > 20} yield p.name`
  - Defined using `map`, `filter`, and `flatMap`
    $\Longrightarrow$ not restricted to lists

Xen and the Art of Language Design?

[Meijer, Schulte, Bierman – XML'03]

- Aims at a tight integration of OO, XML, and SQL (for C#)
- Includes Streams, Tuples, Union, Join Patterns (asynchronous programming)
- Map, Filter, and Fold on streams
- More details on the type system?
  - Aim at a seamless integration
    - No distinction between old and new types
  - What kind of subtyping integration?
    - Challenging issue

Take-home points

- Regular types are an expressive data model for XML
- Type systems and subtyping integration are crucial for a tight coupling of the two data models
- We need a better understanding of the relationship between nominal and structural subtyping

Do you want to know more?

Xobe http://www.ifis.mu-luebeck.de/projects/XOBE/XOBE.html (in German)

XJ http://www.google.com/search?hl=en&q=xj%20xml

Xact http://www.brics.dk/~amoeller/Xact/

Xtatic http://www.cis.upenn.edu/~bcpierce/xtatic/