Making XT XML capable

Martin Bravenboer

mbravenb@cs.uu.nl

Institute of Information and Computing Sciences

University Utrecht

The Netherlands

Introduction

Contents

- goals, applications, motivation
- quick comparison of XML and ATerms
- bridge implementation
- ATermification
- generation of ATermification
- conclusions, open issues

Goals

- apply XML tools on ATerms
- apply ATerm tools on XML
- XML tools can use the efficient ATerm library

Most obvious ‘ATerm’ tool we want to apply on XML: Stratego.

Why Stratego for transforming XML?

Stratego is

- not just for program transformations,
- not just for ATerms,
- a transformation language for regular tree languages.

Try Google: `+"stratego" +"regular tree language"` .
$\rightarrow$ Not a *single* match!

Related work

- schemas $\rightarrow$ native data structures:
  - HaXml: generation of Haskell data types from DTDs
  - Haskell ATerm library
  - XML mappings for database systems
- languages with XML as data type:
  - XDuce: typed XML processing language, powerful subtyping on regex types
  - XMLλ: functional, DTDs
  - XSLT, XPath, XQuery, XML-QL

Comparing XML and ATerms

- abstract data types
- ‘extensible’
- nowadays used in very similar ways: exchange of data in tree-like data structures between components (web-services, XT)
- provide similar functionality:
  - not targeted at special kind of data
  - regular tree languages

Comparing XML and ATerms (2)

ATerm *instances* are more structured and typed.

- XML: attributes not structured
- ATerms: int, real, blob
- ATerms: list, tuples
- XML: namespaces

XML documents can be structured and typed by creating an *interpretation* against a schema.

Comparing XML and ATerms (3)

Different conventions

- ATerms: annotations not widely used to represent non-structural information
- XML: interleaving, mixed-content
- Stratego: optional data with Option

Concrete applications

- transforming or analysing XSLT
- XSLT with concrete syntax
- XSLT Compiler in Stratego with concrete syntax
- XML schema tools: verifiers, converters
- server-side applications in Stratego: transformation to XHTML
- XML processing with Stratego and concrete syntax

Running example

```
<trade-history date="2002-05-20">

  <stock-trade>
    <symbol>SUNW</symbol>
    <time>08:45:19</time>
    <price>86.24</price>
    <quantity>500</quantity>
  </stock-trade>

  <stock-trade>
    <symbol>MSFT</symbol>
    <time>08:45:20</time>
    <price>22.26</price>
    <quantity>1000</quantity>
  </stock-trade>

</trade-history>
```

Relax NG Schema (compact syntax)

```
TradeHistory =
  element trade-history {Date, Trade*}

Date =
  attribute date {xsd:date}

Trade =
  element stock-trade {
      element symbol   {xsd:token },
      element time     {xsd:time  },
      element price    {xsd:double},
      element quantity {xsd:int   }
  }
```

Relax NG Schema (standard syntax)

```
<define name="TradeHistory">
  <element name="trade-history">
    <attribute name="date">  <data type="date"/>  </attribute>
    <zeroOrMore>             <ref name="Trade"/>  </zeroOrMore>
  </element>
</define>

<define name="Trade">
  <element name="stock-trade">
    <element name="symbol">   <data type="token"/>   </element>
    <element name="time">     <data type="time"/>    </element>
    <element name="price">    <data type="double"/>  </element>
    <element name="quantity"> <data type="int"/>     </element>
  </element>
</define>
```

Stratego signature

```
module trade-history
signature
  constructors
    trade-history: Date * List(Trade) -> TradeHistory
    date         : Int * Int * Int  -> Date

    stock-trade  : Symbol * Time * Price * Quantity -> Trade
    symbol       : String -> Symbol
    time         : Int * Int * Int -> Time
    price        : Real -> Price
    quantity     : Int -> Quantity
```

Regular Tree Grammar

```
N = {TradeHistory,  Trade, Symbol, Quantity, Price, Time, Date}
T = {trade-history, trade, symbol, quantity, price, time, date}
S = {TradeHistory}
P = {TradeHistory -> trade-history (Date, Trade*),
     Date         -> date (Int, Int, Int),
     StockTrade   -> stock-trade (Symbol, Time, Price, Quantity),
     Symbol       -> symbol (Token),
     Time         -> time (Int, Int, Int),
     Price        -> price (Real),
     Quantity     -> quantity (Int)
    }
```

Note that Stratego is in fact using Regular Tree Grammars. Without ‘or’, with parameterization

XML to ATerm Bridge

Bridge implementation

XML Parsing requires more than just a grammar:

- namespaces
- doc-types, entities
- whitespace

Every platform nowadays has a decent XML parser.

Introduction to SAX

$\rightarrow$ universal interface to implement bridge Simple

API for XML: de facto standard to pass logical content of XML documents in an efficient way.

- DOM builders implement with SAX
- applications implemented with SAX
- chains implemented with SAX: XSLT, Verifiers

Introduction to SAX

```
interface XMLReader {
  void parse(InputSource input)
  void setContentHandler(ContentHandler handler)
  ...
}


interface ContentHandler {
  void characters(char[] ch, int start, int length)
  void startDocument()
  void endDocument()

  void startElement(String nsURI, lName, qName, Attributes atts)
  void endElement(  String nsURI, lName, qName)

  ...
}
```

Event-based API for ATerms

```
interface ATermContentHandler {
  void startSession();
  void endSession();

  void value(int    value, ATermList annos);
  void value(String value, ATermList annos);
  void value(double value, ATermList annos);

  void startApplication(String function, ATermList annos);
  void endApplication();

  void startList(ATermList annotation);
  void endList();
}
```

Bridge interface

Bridge interface:

`ContentHandler` $\leftrightarrow$ `ATermContentHandler`

```
interface Bridge {
  ATermContentHandler toSAX(ContentHandler      handler);
  ContentHandler      toSAA(ATermContentHandler handler);
}
```

XML in Stratego signature

Signature is a lot like the classes of the DOM.

```
Document : Element -> Document
Element  : String * Namespace * List(Attribute) * List(Content)
           -> Element
Attribute: String * String * AttributeType * Namespace
           -> Attribute
Text     : String -> Content

NoNamespace:  Namespace
Namespace  :  String -> Namespace
```

Fragments of the implementation

```
public void startDocument() {
  _contentHandler.startSession();
  _contentHandler.startApplication("Document");
}

public void endDocument() {
  _contentHandler.endApplication(); // of Document
  _contentHandler.endSession();
}

public void characters(char[] ch, int start, int length) {
  String val = new String(ch, start, length);
  _contentHandler.startApplication("Text");
  _contentHandler.value(val);
  _contentHandler.endApplication();
}
```

Fragments of the implementation

```
void startElement(String nsURI, lName, qName, Attributes atts) {
  _contentHandler.startApplication("Element");
  _contentHandler.value(lName);

  namespace(nsURI);

  _contentHandler.startList(); // of Attributes
  int nrOfAtts = atts.getLength();
  for(int i = 0; i < nrOfAtts; i++) {
    _contentHandler.startApplication("Attribute");
    ...
    _contentHandler.endApplication(); //of Attribute
  }

  _contentHandler.endList();
  _contentHandler.startList(); // of Content
}
```

Fragments of the implementation

```
void endElement(String namespaceURI, String localName, String qName) {
  _contentHandler.endList(); // of Content
  _contentHandler.endApplication(); // of Element
}
```

SAX $\rightarrow$ SAA: clear and efficient implementation.

XML in ATerms

```
Document(
  Element("trade-history", NoNamespace,
    [Attribute("date","2002-05-20",CDATA,NoNamespace)],
    [Element("stock-trade", NoNamespace,[],
      [Element("symbol",    NoNamespace,[], [Text("SUNW")]),
       Element("time",      NoNamespace,[], [Text("08:45:19")]),
       Element("price",     NoNamespace,[], [Text("86.24")]),
       Element("quantity",  NoNamespace,[], [Text("500")])]),
     Element("stock-trade", ... )
    ]))
```

We don’t want to transform *elements*, we want to transform *data*.

Imploding XML

```
xml-implode = ?Document(<id>); topdown-wannos( try(Implode) )

Implode:
  Element(s, _, [atts*], children) -> s#(children){atts*}

Implode:
  Text(s) -> s

Implode = Implode-Special-Attribute <+ Implode-Attribute

Implode-Attribute:
  Attribute(name, value, _, _) -> (name, value)

Implode-Special-Attribute:
  Attribute(name, value, NMTOKENS, _) -> (name, <split-at-space> value)

topdown-wannos(s) = rec x(topdown( s; id{map(x)} ))
```

After the implosion

```
trade-history(
  stock-trade(
    symbol("SUNW"),
    time("08:45:19"),
    price("86.24"),
    quantity("500")
  ),
  stock-trade(
    symbol("MSFT"),
    time("08:45:20"),
    price("22.26"),
    quantity("1000")
  )
){("date","2002-05-20")}
```

Direct translation of XML to ATerms

ATermification

Problems

- variable number of arguments $\rightarrow$ list
- primitives: String $\rightarrow$ Int, Real
- attributes
  - move to content?
  - keep in annotation?
- namespaces
  - drop?
  - keep in annotation?
  - different atermification?

ATermification

```
ATermify:
  "trade-history"#([trades*])
    ->
  trade-history(<get-prop(!"date"); string-to-date>, [trades*])

ATermify:
  "stock-trade"#([s, t, p, q]) -> stock-trade(s, t, p, q)

ATermify:
  "time"#([s]) -> <string-to-time> s

ATermify:
  "price"#([s]) -> price(<string-to-real> s)

string-to-date =
    <tokenize> (<id>, "-"); map(string-to-int)
  ; ?[y, m, d]; !date(y, m, d)
```

After the ATermification

```
trade-history(
  date(2002,5,20),
  [ stock-trade(
      symbol("SUNW"),
      time(8,45,19),
      price(8.6239e+01),
      quantity(500)
    ),
    stock-trade(
      symbol("MSFT"),
      time(8,45,20),
      price(2.2260e+01),
      quantity(1000)
    )
  ]
)
```

Handcrafted ATermification: problems

- ad-hoc and tiresome
- correctness depends on implementation

ATermification must be generated from a schema.

Generation of ATermification

1. Generate a regular tree grammar from a schema.
2. Generate a Stratego signature
3. Compute *interpretation* of an instance document against the regular tree grammar.
4. Generate Stratego transformation that uses the production rules for ATermification.
5. Generate the inverse to XML.

Generation of ATermification

We choose Relax NG as input schema language.

Why?

- covers the full class of regular tree grammars: DTD, W3C XML Schema $\rightarrow$ Relax NG
- closed under $\cup$, $\cap$ and $-$
- well-defined formal semantics
- subjective: popular, clear, concrete syntax

Relax NG to Regular Tree Grammar

- choice $\rightarrow$ more production rules
- interleave $\rightarrow$ Consider optional specification to declare interleaving semanticly irrelevant in special cases. If not irrelevant:
  - simple cases:
    optional content
  - more complex cases:
    more production rules
  - most complex cases:
    introduce common type

Relax NG to Regular Tree Grammar (2

- list $\rightarrow$ tokenize
- group $\rightarrow$ tuples
- enumeration $\rightarrow$ non-terminal
- zero/oneOrMore $\rightarrow$ list
- any content $\rightarrow$ XML signature or just implode?

$\rightarrow$ Study multi-schema validators

Final remarks: conclusions, open issues

So, what’s new?

- generalization of the data structures Stratego can operate on
- Common formalism to reason about ATerms and XML
- New application areas for Stratego and other XT tools
- XT could provide the XML community with a lot of interesting and powerful tools.
- danger: interesting to many people

Now available

1. bridge, implemented in Java
2. imploding bridge, implemented in Java
3. imploder implemented in Stratego
4. data oriented XML signature
5. event-based ATerm API

To do

1. create tools for Relax NG
2. generate regular tree grammars from Relax NG
3. formalism for removing productions
4. interpretation creator
5. ATermification generators
6. example applications, experience with XML transformations

Open issues

- How to integrate attributes/annotations in regular tree grammars?
- How to handle ‘any’ content?
- Should we prefer an XML parser in SDF + Stratego?
- What if the number of productions is still getting out of control?
- How to handle namespaces: detect conflicting terminals?
- Should we use annotations for attributes?