Type Safe Programming of XML-based Applications

Martin Kempa
sd&m AG
software design & management
Carl-Wery-Str. 42
D-81739 München, Germany
E-mail: martin.kempa@sdm.de

Volker Linnemann
Universität zu Lübeck
Institut für Informationssysteme
Ratzeburger Allee 160, Geb. 64
D-23538 Lübeck, Germany
E-mail: linnemann@ifis.uni-luebeck.de

Abstract: There is an emerging amount of software for generating and manipulating XML documents. This paper addresses the problem of guaranteeing the validity of dynamically generated XML structures statically at compile time. In the XOBE (**X**ML **OBJE**CTS) project XML Schema is used for describing sets of valid XML documents. An XML schema provides a vehicle to define new classes, i.e. each element declaration in a schema defines a new class of objects (XML objects). Each object within a class represents an XML structure which is valid according to the underlying XML schema. XML objects are created by a new language construct called XML object constructor. XML object constructors are expressed in XML syntax. Previously generated XML objects can be inserted according to the declared XML schema.

The main focus of the paper is the type system of XOBE. Among others, this type system provides the basis for checking the validity of assignments of XML objects to variables. The type system will be described and we present formally a type checking algorithm based on this type system.

1 Introduction

XML [W3C98b] plays an important role for internet data. Due to this fact, there is an emerging amount of software for generating and manipulating XML documents. Therefore, programming language concepts and tools for this purpose are needed. The approaches that are currently in use are not sufficient because they cannot guarantee that only valid XML documents are being dealt with. In the XML context a valid XML document is a document which is correct according to an underlying XML Schema [W3C01] or an XML Document Type Definition DTD [ABS00], i.e. the document is an element of the language defined by the XML schema or XML DTD. Since most current languages and tools do not allow to guarantee the validity of dynamically generated XML documents at compile time, extensive runtime checking is necessary in order to achieve valid documents.

The **X**ML **Obje**cts project (XOBE) [LK02] at the University of Lübeck addresses this mismatch by defining XML objects representing XML fragments and by treating them as first-class data values. We extend Java [AG98] for this purpose. XOBE overcomes the different representations of XML fragments as strings and as nested object structures in

the same source code. Instead, a running XOBE program works only with XML objects. A text form of XML objects with explicit tagging is constituted for communication with the outside world only. XML objects are created by a new language construct called XML object constructor. XML object constructors are expressed in XML syntax. Previously generated XML objects can be inserted according to the declared XML schema or DTD. The XML schema is used for typing the XML objects.

The main aspect of this paper is the type system of XOBE. This type system is the basis for checking the validity of assignments of XML objects to variables. The type system will be described and we present formally a type checking algorithm based on this type system.

The paper is organized as follows. In the next section we survey related work summarizing the state of the art in the field of programming of XML-based applications. Our approach for programming these applications by using XML objects in Java is introduced in Section 3. The corresponding type system is described informally and formally in Section 4. In Section 5 we present implementation details of the XOBE preprocessor. Concluding remarks and an outlook for future work conclude the paper. An appendix provides some performance measures.

2 Related Work

Run time validation. The most elementary way to deal with XML fragments is to treat XML documents as ordinary strings without any structure. One prominent representative of this technique is given by Java Servlets [Wil99]. In former CGI scripts [Gai95] the programming language Perl [WS92] was used. The technique is rather tedious when constant XML fragments are being generated. Java Server Pages [PLC99, FK00] provide an improvement over pure string operations by allowing to switch between XML parts and Java. This switching is done by special markings. Java Server Pages share with string operations the disadvantage that not even well-formedness is checked at compile time. An improvement is to provide classes for nodes of an XML document tree thus allowing to access and manipulate arbitrary XML fragments by object-oriented programming. Representatives of this approach, sometimes called low-level bindings, are the Document Object Model (DOM) [W3C98a] and Java DOM (JDOM) [JDO]. Both are widely accepted and supported. It is the only standardized and language independent way for XML processing. Constant XML fragments can be programmed in a pure object-oriented manner, which is rather tedious, or by parsing an XML fragment into the object structure, which requires runtime validation. Low-level bindings ensure well-formedness of dynamically generated documents at compile time, but defer validation until runtime.

Partial compile time validation. A series of proposals [Bou02], called high-level bindings, have been presented. With Sun's JAXB, Microsoft's .Net Framework, Exolab's Castor, Delphi's Data Binding Wizard, Oracle's XML Class Generator [Sun01, Mic01, Exo01, Bor01, Ora01] we only mention the better known products. These approaches assume that all processed documents follow a given schema. This description is used to map the docu-

ment structure onto language types or classes reproducing directly the semantics intended by the schema. Like low-level binding, high-level binding provides no facilities to cope with constant XML fragments. Therefore the formulation of constant XML fragments has to be done by nested constructor or method calls, or by parsing of fixed documents, called unmarshalling. The first procedure is tedious for the programmer, the second one needs validation at run-time. High-level bindings ensure well-formedness of dynamically generated documents at compile time. Validity is only supported to a limited extent depending on the selected language mapping.

Full compile time validation. The XDuce language [HVP00, HP03] is a special functional language developed as an XML processing language. It introduces so called regular expression types the values of which are comparable to our XML objects. Elements are created by specific constructors and the content can be accessed through pattern matching. XDuce supports type inference for patterns and variables and performs a subtyping analysis to ensure validity of regular expression type instances at compile time. The subtyping algorithm is implemented on the basis of regular tree automata. It operates on an additional internal representation for regular expression types which can be a source of inefficiency. This internal representation is avoided in our approach. Furthermore, because XOBE is an extension of Java, it is easier to couple it with other components like database systems. Recently, the Xtatic project [GP03] was founded as the successor of XDuce. The main purpose of Xtatic is to couple the concepts of XDuce with the object oriented programming language C#. Xtatic has similar goals as XOBE, however, Xtatic is still in an early stage and, in contrast to XOBE, there seams to be no running prototype available.

BigWig [BMS01, BMS02] is a special programming language for developing interactive web services. JWig [CMS03] is the successor of BigWig. The main purpose of JWig is to integrate the XML specific parts of BigWig in Java. In this respect, JWig is quite close to XOBE. The main difference in JWig is that there is only one XML type. Typed XML document templates with gaps are introduced. In order to generate XML documents dynamically, gaps can be substituted at runtime by other templates or strings. For these templates JWig validates all possibly dynamically computed documents according to a given abstract DTD. This is done by two data flow analyses constructing a graph which summarizes all possible documents. This graph is analyzed to determine validity of those documents. In comparison to our approach templates can be seen as methods returning XML objects. The arguments of the methods correspond to the gaps of the templates. Because there is only one XML type and because JWig's type checking algorithm is based on data flow analyses it differs totally from ours. Jwig's data flow analysis is very time consuming. In contrast, in practice XOBE's type checking algorithm has linear running time, see section 4.4. Moreover, we believe that XOBE's type system is more expressive because we can incorporate XML Schema's extension and restriction mechanisms quite naturally into the subtyping algorithm. This seems to be difficult in JWig. In the Xact project [KMS04] JWig's validation algorithm is extended to the problem of static analysis of XML transformations in Java. As in XOBE, XPath is used for expressing XML transformations. The efficiency problems are inherited from JWig.

XJ [HRS+03] is a new project pursued by IBM research on the integration of XML into Java concentrating on traversing XML structures by using XPath. The distinguishing char-

acteristic of XJ is its support for inplace updates.

Xen [MSB03] is an integration of XML into popular object-oriented programming languages such as C# or Java currently under development at Microsoft Company. Xen uses XML constructors similar as XOBE's XML object constructors.

The upcoming standard of an XML query language XQuery [W3C02] has to support validity as well. Based on XQuery another challenging approach is presented by the XML programming language XL [FGK02]. XL is a programming language for the implementation of web services. It provides high-level and declarative constructs adopting XQuery. Additionally imperative language statements are introduced making XL a combination of an imperative and a declarative programming language. In contrast to XL, which is a stand-alone programming language, XOBE is defined as an extension of Java. Java is an already established programming language for web services. Thus XOBE can significantly benefit by using already developed Java code.

3 XML Objects in XOBE

In this section we briefly introduce the syntax and semantics of XML Objects in an informal manner. A more detailed introduction can be found in [KL02, KL03, Kem03]. XOBE extends the object-oriented programming language Java by language constructs to process XML fragments. Due to space limitations, we introduce only the construction of XML objects. Traversing XML objects by XPATH [W3C99] is not needed for describing the type system of XOBE. Details can be found in [KL03, Kem03].

In XOBE, we represent XML fragments, i.e. trees corresponding to a given schema or a DTD, by XML objects. Therefore, XML objects are first-class data values that may be used like any other data value in Java. The given schema or DTD is used to type different XML objects.

We use the following XML schema describing a bookstore as the basis for our example. According to this schema, a bookstore contains several books. Each book contains several authors and one title.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="bookstore">
    <xsd:complexType>
      <xsd:sequence>
        <xsd:element name="book"  minOccurs="0" maxOccurs="unbounded">
          <xsd:complexType>
            <xsd:sequence>
              <xsd:element name="author" minOccurs="0" maxoccurs="unbounded"
                                                    type="xsd:string"/>
              <xsd:element name="title" type="xsd:string"/>
            </xsd:sequence>
          </xsd:complexType>
        </xsd:element>
      </xsd:sequence>
    </xsd:complexType>
  </xsd:element>
</xsd:schema>
```

Listing 1: XML schema for `bookstore`

The following program shows a method which reads books with their authors and titles. It subsequently constructs an XML object for these books. The class declaring `collectBooks` uses our bookstore schema by importing the corresponding file by a clause like `import bookstore.xsd`.

```
1  bookstore collectBooks(){
2      XML<book*> books = <>;
3      int countBooks = keyboard.readint();
4      for (int i=1; i<=countBooks; i++){
5          XML<author*> authors = <>;
6          int countAuthors = keyboard.readint();
7          for (int j=1; j<=countAuthors; j++) {
8              String author = keyboard.readString();
9              authors = authors+
10                 <author> {author} </author>;
11         }
12         String title = keyboard.readString();
13         book b = <book>
14                     {authors}
15                     <title>{title}</title>
16                 </book>;
17         books = books + b;
18     }
19     return
20         <bookstore>
21             {books}
22         </bookstore>;
23 } //collectBooks
```

Listing 2: Method `collectBooks`

Line 2 declares a variable `books` for XML objects. A corresponding value is a list of books. `books` is initialized by the empty list. Line 5 declares a variable `authors` for lists of authors. Lines 9 and 10 append a newly created author element to the list of authors. The author element is constructed in line 10 by a so called **XML object constructor**. An XML object constructor is an XML fragment where values of other XML objects can be inserted in places where the corresponding XML element is allowed according to the underlying XML schema. In line 10, the string content of variable author is inserted. This conforms with the underlying XML schema because the content of an author element is a string. In line 13, the declaration `book b` is an abbreviation for `XML<book> b`. Lines 13-16 use an XML object constructor for constructing a book out of the previously generated author list and the title.

This finishes our short introduction to XOBE. Due to space limitations, we did not cover the important aspect of using XPath [W3C99] within XOBE for type safe decompositon of XML values. More details and more examples can be found in [Kem03, KL02, KL03, Kra02, LK02].

4 The XOBE Type System

This section is the major part of this paper presenting the XOBE type system. Type analysis is done in two steps. First, the types of the XML object expressions are determined using *type inference*. Second, the subtype relationship of the inferred types is checked by a *subtyping algorithm*.

4.1 Formalizing XML

For a precise definition of our subtyping algorithm we need a formalization of XML. We use *regular hedge expressions* [BKMW01] and *regular hedge grammars* describing sets of trees.

Definition 4.1 Let B and E be a finite sets of simple types and element names respectively with $B \cap E = \emptyset$. A *set of all hedges* T^* over a set of terminal symbols $T = B \uplus E$ ($\uplus$ denotes disjoint union) is defined inductively as follows:

$\epsilon \in T^*$ is the empty hedge,
$b \in T^*$ with $b \in B$ is a hedge,
$e[v] \in T^*$ with $e \in E$ and the hedge $v \in T^*$ is a hedge and
$vw \in T^*$ with the hedges $v \in T^*$ and $w \in T^*$ is a hedge. □

Definition 4.2 The set of *regular hedge expressions Reg* over a set of terminal symbols $T = B \uplus E$ and a set N of nonterminal symbols (names of groups and complex types, $N \cap T = \emptyset$) is defined recursively by:

$\emptyset$	$\in$ *Reg*	(empty set),	ϵ	$\in$ *Reg*	(empty hedge),
b	$\in$ *Reg*	(simple type),	n	$\in$ *Reg*	(complex type),
$e[r]$	$\in$ *Reg*	(element),	$(r\|s)$	$\in$ *Reg*	(regular union),
$(r;s)$	$\in$ *Reg*	(regular concatenation)	$(r)*$	$\in$ *Reg*	(Kleene star)

for all $b \in B$, $n \in N$, $e \in E$ and $r, s \in$ *Reg*. □

For simplifying parentheses, we assume operator precedence in the decreasing order $*$; $|$.

We define $r+ = r; r*$, $r? = r|\epsilon$ and for $I = \{i_1, ..., i_n\}$: $\Big|_{i \in I} r_i = r_{i_1} \mid ... \mid r_{i_n}$.

Definition 4.3 The *hedge language* $L(r)$ over a regular hedge expression $r \in$ *Reg* for a given set of production rules P of the form $n \to r$ with $n \in N$, $r \in Reg$ and $n \to r \in P$, $m \to r \in P \Longrightarrow n \neq m$, is defined by:

$L(\emptyset)$	$=$	$\{\}$	$L(\epsilon)$	$=$	$\{\epsilon\}$
$L(b)$	$=$	$\{b\}$	$L(n)$	$=$	$L(r)$ with $n \to r \in P$
$L(e[r])$	$=$	$\{e[u] \mid u \in L(r)\}$	$L(r\|s)$	$=$	$L(r) \cup L(s)$
$L(r;s)$	$=$	$\{uv \mid u \in L(r), v \in L(s)\}$	$L(r*)$	$=$	$\{\epsilon\} \cup L(r; r*)$

for all $b \in B$, $n \in N$, $e \in E$ and $r, s \in$ *Reg*. ϵ denotes the empty hedge. □

The predicate $isnullable? : Reg \to \{$**true**,**false**$\}$ decides $\epsilon \in L(r)$ for $r \in Reg$. Details can be found in [Kem03].

Definition 4.4 A *regular hedge grammar* is defined by $G = (T, N, s, P)$ with T, N, P as defined in definitions 4.2, 4.3. $s \in$ *Reg* is a start expression. Each rule $n \to r \in P$ has to fulfill the following two conditions guaranteeing regularity of the grammar:

1. If the nonterminal symbol n is defined recursively, the recursive application has to be in the last position of the regular expression r.

2. If a nonterminal symbol n is defined recursively, the expression s in front of the recursive application has to fulfill $\neg$*isNullable?*(s).

□

Definition 4.5 A *regular inequality* of two regular hedge expressions r and s is defined by:

$$r \leq s \Leftrightarrow L(r) \subseteq L(s)$$

□

Example 4.1 The following regular hedge grammar corresponds to the XML schema in listing 1. Element names and simple types are **`boldfaced`**, nonterminal symbols are *italic*. The start expression s is *bookstore*.

$$\begin{array}{rlrl} \textit{bookstore} & \rightarrow \textbf{\texttt{bookstore}}[\textit{book}*] & \textit{book} & \rightarrow \textbf{book}[\textit{author}* \,;\, \textit{title}] \\ \textit{author} & \rightarrow \textbf{\texttt{author}}[\textbf{\texttt{string}}] & \textit{title} & \rightarrow \textbf{\texttt{title}}[\textbf{\texttt{string}}] \end{array}$$

The following regular inequalities hold:

$$\begin{aligned} \textbf{book}[\textit{author}* \,;\, \textit{title}] &\leq \textbf{book}[\textit{author}* \,;\, \textit{title}] \\ \textbf{book}[\textit{author} \,;\, \textit{author}* \,;\, \textit{author} \,;\, \textit{title}] &\leq \textbf{book}[\textit{author}* \,;\, \textit{title}] \end{aligned}$$

4.2 Type Inference

The type corresponding to an XML object constructor is given by a regular hedge expression. Type inference for XML object constructors is quite simple because all variables have to be declared in XOBE.

Example 4.2 The type of the left hand side of the assignment in lines 13-16 in listing 2 is *book*. The type of the right hand side of the assignment is **book**[*author*$*$; **`title`**[**`string`**]]

The subtyping algorithm which is described in the next chapter has to check the regular inequality **book**[*author*$*$; **`title`**[**`string`**]] $\leq$ *book*

4.3 Subtyping Algorithm

After inferring the types of the XML objects in an XOBE program, the type system checks the correctness of the concerned statements using the subtyping algorithm. For this we adopt Antimirov's algorithm [Ant94] for checking inequalities of regular expressions and extend it to the hedge grammar case. The idea of the algorithm is that for every invalid regular inequality there exists at least one reduced inequality which is *trivially inconsistent*, a notion which is defined by the function *inc* as follows.

Definition 4.6 A regular inequality $r \leq s$ is called *trivially inconsistent* if

$$inc(r \leq s) = (isNullable?(r) \wedge \neg isNullable?(s))$$

holds. □

Definition 4.7 For $r \in \mathit{Reg}$ the set of leading terminal symbols is defined by $term(r) = \{t \mid t \in T, t \cdot \sigma \in L(r)\ for\ a\ \sigma\}$. More details are given in [Kem03] □

The reduction of regular inequalities is expressed by partial derivatives *der* of regular hedge expressions. They formalize the set of hedges which can follow after an already recognized terminal symbol. A partial derivative consists of a pair of expressions. The first component stands for the element content of the reduced terminal symbol, i.e. the child dimension. The second component represents the expression part after the reduced terminal symbol, i.e. the sibling dimension. The exact definition is given in [Kem03]. Due to space limitations, we present only some examples in this paper..

In the following example the regular expression $\mathbf{author}[\mathbf{string}]; \mathit{author}*$ is reduced by the given terminal symbol **author**:

$$\mathit{der}_{\mathbf{author}}(\mathbf{author}[\mathbf{string}]; \mathit{author}*) = \{(\mathbf{string}, \mathit{author}*)\}$$

The result is a set of regular hedge expression pairs because we can receive multiple pairs as the following example shows:

$$\mathit{der}_{\mathbf{book}}(\mathbf{book}[\mathit{title}]|\mathbf{book}[\mathit{author}; \mathit{title}]) = \{(\mathit{title}, \epsilon), (\mathit{author}; \mathit{title}, \epsilon)\}$$

Please notice that in the case of Kleene star operations the length of the resulting expressions can increase:

$$\mathit{der}_{\mathbf{author}}((\mathbf{author}[\mathbf{string}]; \mathit{title})*) = \{(\mathbf{string}, \mathit{title}; (\mathbf{author}[\mathbf{string}]; \mathit{title})*)\}$$

Based on the partial derivatives of regular expressions we can define partial derivatives of regular inequalities. This definition is quite complex in the hedge grammar case because of the two dimensions. For the definition we adopt a set-theoretic observation from [HVP00]. Examples will follow in example 4.3

Definition 4.8 A *partial derivative* of a regular inequality $r \leq s$ with $r, s \in \mathit{Reg}$ with respect to a terminal symbol $x \in T$ is defined by

$$\begin{aligned}\mathit{part}_x(r \leq s) = \{(c_r \leq \bigm|_{i \in I} c_s^i) \ \vee\ (r_r \leq \bigm|_{i \in \overline{I}} r_s^i) \mid \\ (c_r, r_r) \in \mathit{der}_x(r) \wedge \\ \mathit{der}_x(s) = \{(c_s^1, r_s^1), \ldots, (c_s^n, r_s^n)\} \text{ with} \\ I \in \mathcal{P}(\{1, \ldots, n\}) \text{ and } \overline{I} = \{1, \ldots, n\} \backslash I\}\end{aligned}$$

with $c_r, r_r, c_s^i, r_s^i \in \mathit{Reg}$ and $\mathcal{P}(I) = \{J \mid J \subseteq I\}$. □

The subtyping algorithm is defined by two *subtyping judgements* $\Gamma \vdash r \leq s \Rightarrow \Gamma'$ and $\Gamma \vdash^* r \leq s \Rightarrow \Gamma'$ with a set Γ of regular inequalities of type $t \leq u$. Both judgements have to be interpreted as: "The algorihm proves $r \leq s$ and all inequalities $t \leq u$ in Γ are not trivially inconsistent. All results are returned in the set Γ' with all regular inequalities of the partial derivatives of $r \leq s$. "

Definition 4.9 Given a set of regular inequalities Γ, the following two *subtyping judgements* have to be distinguished:

$$\Gamma \vdash r \leq s \Rightarrow \Gamma' \qquad r \leq s \text{ is a valid inequality in } \Gamma$$
$$\Gamma \vdash^* r \leq s \Rightarrow \Gamma' \qquad r \leq s \text{ is a valid inequality in } \Gamma$$

with $r, s \in Reg$. Γ' is the resulting set of regular inequalities. □

Because the production rules of the hedge grammar can be defined recursively it can happen that an already calculated inequality appears during the algorithm later on. To ensure termination in that case we save all already seen inequalities in the set Γ. For this reason we have to introduce the two subtyping judgements. With judgement $\vdash^*$ we indicate that an inequality has been added to Γ.

Definition 4.10 The *subtyping algorithm* is defined by the following rules:

$$\frac{r \leq s \in \Gamma}{\Gamma \vdash r \leq s \Rightarrow \Gamma} \qquad \text{(HYP)}$$

$$\frac{\begin{array}{c} r \leq s \notin \Gamma, \\ \Gamma \cup \{r \leq s\} \vdash^* r \leq s \Rightarrow \Gamma' \end{array}}{\Gamma \vdash r \leq s \Rightarrow \Gamma'} \qquad \text{(ASSUM)}$$

$$\frac{\begin{array}{c} \neg \mathit{inc}(r \leq s), \\ \text{For all } x \in \mathit{term}(r) \text{ and for all } i \in \{1, \ldots, k\} \\ \text{with } \mathit{part}_x(r \leq s) = \{(c_{r,1} \leq c_{s,1} \vee r_{r,1} \leq r_{s,1}), ..., (c_{r,k} \leq c_{s,k} \vee r_{r,k} \leq r_{s,k})\} \text{ is} \\ \Gamma_{i-1} \vdash c_{r,i} \leq c_{s,i} \Rightarrow \Gamma_i \vee \Gamma_{i-1} \vdash r_{r,i} \leq r_{s,i} \Rightarrow \Gamma_i \end{array}}{\Gamma_0 \vdash^* r \leq s \Rightarrow \Gamma_k} \qquad \text{(REC)}$$

□

With rule HYP we test if the inequality in question is already in the set of all calculated inequalities Γ terminating that recursion branch. Rule ASSUM switches between the two judgements $\vdash$ and $\vdash^*$ adding the inequality to Γ. The rule REC is applicable if the inequality is not trivially inconsistent. With operation *part* all partial derivatives are calculated and checked recursively.

Example 4.3 Consider the regular inequality $\mathit{author}*; \mathit{title} \leq \mathit{author}; \mathit{author}*; \mathit{title}|\mathit{title}$. We start with $\Gamma_0 = \emptyset$. The computations and the derivation tree of the execution are given in figure 1. The inequality is accepted finally.

Due to the regularity of the production rules the subtyping algorithm is guaranteed to terminate. For a detailed correctness proof we refer to [Kem03].

$$term(\mathit{author}*; \mathit{title}) = \{\texttt{author}, \texttt{title}\}$$

$$der_{\texttt{author}}(\mathit{author}*; \mathit{title}) = \{(\texttt{string}, \mathit{author}*; \mathit{title}), (\texttt{string}, \mathit{title})\}$$

$$der_{\texttt{author}}(\mathit{author}; \mathit{author}*; \mathit{title}|title) = \{(\texttt{string}, \mathit{author}*; \mathit{title})\}$$

$$der_{\texttt{title}}(\mathit{author}*; \mathit{title}) = \{(\texttt{string}, \epsilon)\}$$

$$der_{\texttt{title}}(\mathit{author}; \mathit{author}*; \mathit{title}|title) = \{(\texttt{string}, \epsilon)\}$$

$$part_{\texttt{author}}(\mathit{author}*; \mathit{title} \leq \mathit{author}; \mathit{author}*; \mathit{title}|\mathit{title}) =$$

$$\{ \quad (\texttt{string} \leq \texttt{string} \vee \mathit{author}*; \mathit{title} \leq \emptyset), \tag{1}$$

$$\quad (\texttt{string} \leq \emptyset \vee \mathit{author}*; \mathit{title} \leq \mathit{author}*; \mathit{title}) \ \} \tag{2}$$

$$part_{\texttt{title}}(\mathit{author}*; \mathit{title} \leq \mathit{author}; \mathit{author}*; \mathit{title}|\mathit{title}) =$$

$$\{ \quad (\texttt{string} \leq \texttt{string} \vee \epsilon \leq \emptyset), \tag{3}$$

$$\quad (\texttt{string} \leq \emptyset \vee \epsilon \leq \epsilon) \ \} \tag{4}$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{\Gamma_7 \vdash^* \epsilon \leq \epsilon \Rightarrow \Gamma_3}\,\textsc{Rec}}{\Gamma_6 \vdash \epsilon \leq \epsilon \Rightarrow \Gamma_3} \vee \dfrac{Error}{\Gamma_6 \vdash \epsilon \leq \emptyset},}{\Gamma_6 \vdash^* \texttt{string} \leq \texttt{string} \Rightarrow \Gamma_3} \vee \dfrac{\dfrac{Error}{\Gamma_3 \vdash \epsilon \leq \emptyset} \vee \dfrac{}{\Gamma_3 \vdash \epsilon \leq \epsilon \Rightarrow \Gamma_3}\,\textsc{Hyp}}{\Gamma_1 \vdash \texttt{string} \leq \texttt{string} \Rightarrow \Gamma_3} \vee \dfrac{\begin{array}{c}Error\\ \vdots \\ \Gamma_1 \vdash \mathit{author}*; \mathit{title} \leq \emptyset\end{array}}{}, \tag{1}$$

$$\dfrac{Error}{\Gamma_3 \vdash \texttt{string} \leq \emptyset} \vee \dfrac{\begin{array}{c}\textsc{Hyp}\\ \vdots \\ \Gamma_3 \vdash \mathit{author}*; \mathit{title} \leq \mathit{author}*; \mathit{title} \Rightarrow \Gamma_5\end{array}}{}, \tag{2}$$

$$\dfrac{\dfrac{}{\Gamma_5 \vdash \texttt{string} \leq \texttt{string} \Rightarrow \Gamma_5}\,\textsc{Hyp}}{} \vee \dfrac{Error}{\Gamma_5 \vdash \epsilon \leq \emptyset}, \tag{3}$$

$$\dfrac{\dfrac{Error}{\Gamma_5 \vdash \texttt{string} \leq \emptyset} \vee \dfrac{}{\Gamma_5 \vdash \epsilon \leq \epsilon \Rightarrow \Gamma_5}\,\textsc{Hyp}}{} \tag{4}$$

$$\dfrac{\Gamma_1 \vdash^* \mathit{author}*; \mathit{title} \leq \mathit{author}; \mathit{author}*; \mathit{title}|\mathit{title} \Rightarrow \Gamma_5}{\Gamma_0 \vdash \mathit{author}*; \mathit{title} \leq \mathit{author}; \mathit{author}*; \mathit{title}|\mathit{title} \Rightarrow \Gamma_5}$$

Figure 1: Proving inequality $\mathit{author}*; \mathit{title} \leq \mathit{author}; \mathit{author}*; \mathit{title}|\mathit{title}$

4.4 Complexity and Extensions

The complexity of the subtyping algorithm is EXPTIME complete as shown in [Sei90]. This means that in the worst case the number of checked inequalities depends exponentially on the length of the given inequality. Nevertheless, in contrast to the classic procedure using a tree automaton the algorithm works more efficiently in some cases. In the classic procedure both automata representing the regular hedge expression of both sides of the inequality in question have to be made deterministic. In our algorithm the right hand side of the inequality has to be made deterministic lazily, i.e. only as much as needed.

In XML Schema and DTDs restrictions to general hedge grammars are assumed. First all element types with the same element name in one content model have to have the same content model. As shown in [Kem03] this simplifies the subtyping algorithm to a PSPACE complete complexity, which is the same as comparing regular string expressions [Ant94].

The second restriction is that the content models have to be one-unambiguous. This leads to a linear subtyping algorithm.

The subtyping algorithm described so far deals with the *structural typing* in XML. However, in XML Schema additional concepts like substitution groups, type extensions and type restrictions sometimes called *named typing* exist. This requires an extension of our regular hedge grammars. Therefore we introduce a reflexive and transitive *substitution group relation SubGr* holding the relations of element names as defined as substitution groups in the schema. Additionally the *named type relation Inh* is defined were the nonterminal types of the hedge grammar are in relation corresponding to the specified type extensions and type restrictions. Further the strategy of the subtyping algorithm has to be more sophisticated as well, because during the calulations we have to take care of the two relations. The idea is to reduce the regular inequality by appropriate nonterminal symbols also instead of reducing only by terminal symbols. This allows to take the relations *SubGr* and *Inh* into account. This extended algorithm is described in detail in [Kem03].

5 Implementation

XOBE is realized as a Java preprocessor [Kra02]. The general architecture is shown in Figure 2. In our implementation we use the Java compiler compiler JavaCC [Web02]

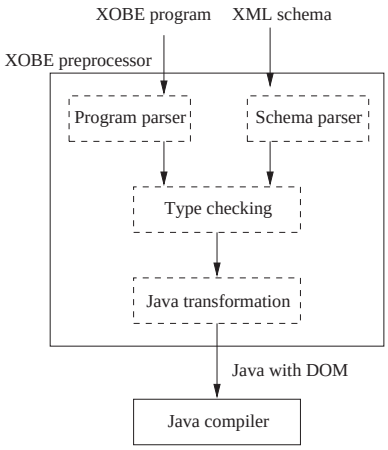

Figure 2: Architecture of the XOBE precompiler

to generate the XOBE parser. Additionally we use the XML parser Xerces [Apa01] to recognize the used schemas. The internal representation of the processed XOBE program is done with the Java tree builder JTB [TWP00].

XML objects are internally stored by using the standard representation of the Document Object Model (DOM) [W3C98a], recommended by the W3Consortium. Please note that

even though DOM is an untyped XML implementation not guaranteeing validity at compile time, the transformed XML objects in the XOBE program are valid. This holds because our type checking algorithm guarantees this property.

The XOBE system including the type checking system was, among others, successfully used in a web based real estate broker system [Kra02] and in a web based academic exercise administration system [Spi04].

6 Concluding Remarks, Outlook for Future Work

This paper presented a short overview over the XOBE project and concentrated on the type system of XOBE. XOBE is an extension of the programming language Java, addressing the programming purposes of web applications and web services. The language extension combines Java with XML by introducing XML objects which represent XML fragments. XML objects are created using XML object constructors. In XML object constructors, previously generated XML objects can be inserted in places which are allowed according to the declared schema. The validity of all XML objects within a program is checked by the XOBE type system at compile time.

In the future we plan to extend XOBE by language constructs allowing to modify XML objects. Additionally we will use XOBE in various application areas. One area will be the media archive software developed in Lübeck [Beh00]. The media archive implementation in its present state primarily uses Java Server Pages for generating HTML and XML. Other application areas will be looked at also. Moreover, we are currently working on the integration of XQuery [W3C02] into the language. Allowing XML objects to be persistent results then in an XML-based database programming language [SL04].

References

[ABS00] Serge Abiteboul, Peter Buneman, and Dan Suciu. *Data on the Web, From Relations to Semistructured Data and XML*. Morgan Kaufmann Publishers, San Francisco, California, 2000.

[AG98] Ken Arnold and James Gosling. *The Java Programming Language*. The Java Series. Addison Wesley Longman, Inc., 2. edition, 1998.

[Ant94] Valentin Antimirov. Rewriting Regular Inequalities. In Reichel, editor, *Fundamentals of Computation Theory*, volume 965 of *Lecture Notes in Computer Science (LNCS)*, pages 116–125, Berlin Heidelberg New York, 1994. Springer-Verlag.

[Apa01] The Apache XML Project. Xerces Java Parser. `http://xml.apache.org/xerces-j/index.html`, 15. November 2001. Version 1.4.4.

[Beh00] Ralf Behrens. MONTANA: Towards a Web-based infrastructure to improve lecture and research in a university environment. In *Proceedings of the 2nd Int. Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS 2000), Milpitas, California*, pages 58–66. IEEE Computer Society, June 2000.

[BKMW01] Anne Brüggemann-Klein, Makoto Murata, and Derick Wood. Regular Tree and Regular Hedge Languages over Unranked Alphabets: Version 1. Technical Report HKUST-TCSC-2001-05, Hong Kong University of Science & Technology, April 3 2001. Theoretical Computer Science Center.

[BMS01] Claus Brabrand, Anders Møller, and Michael I. Schwartzbach. Static Validation of Dynamically Generated HTML. In *Proceedings of Workshop on Program Analysis for Software Tools and Engineering (PASTE 2001), June 18-19, Snowbird, Utah, USA*, pages 38–45. ACM, 2001.

[BMS02] Claus Brabrand, Anders Møller, and Michael I. Schwartzbach. The BIGWIG project. In *ACM Transactions on Internet Technology*, volume 2(2), pages 79–114. ACM, 2002.

[Bor01] Borland. *XML Application Developer's Guide, JBuilder*. Borland Software Corporation, Scotts Valley, CA, 1997,2001. Version 5.

[Bou02] Ronald Bourret. XML Data Binding Resources. web document, `http://www.rpbourret.com/xml/XMLDataBinding.htm`, 28. July 2002.

[CMS03] Aske Simon Christensen, Anders Møller, and Michael I. Schwartzbach. Extending Java for High-Level Web Service Construction. In *ACM Transactions on Programming Languages and Systems*, volume 25(6), pages 814–875. ACM, 2003.

[Exo01] ExoLab Group. Castor. ExoLab Group, `http://castor.exolab.org/`, 2001.

[FGK02] Daniela Florescu, Andreas Grünhagen, and Donald Kossmann. XL: An XML Programming Language for Web Service Specification and Composition. In *Proceedings of International World Wide Web Conference (WWW 2002), May 7-11, Honolulu, Hawaii, USA*, pages 65–76. ACM, 2002. ISBN 1-880672-20-0.

[FK00] Duane K. Fields and Mark A. Kolb. *Web Development with Java Server Pages, A practical guide for designing and building dynamic web services*. Manning Publications Co., 32 Lafayette Place, Greenwich, CT 06830, 2000.

[Gai95] M. Gaither. *Foundations of WWW-Programming with HTML and CGI*. IDG-Books Worldwide Inc., Foster City, California, USA, 1995.

[GP03] Vladimir Gapayev and Benjamin C. Pierce. Regular Object Types. In *ECOOP 2003, Lecture Notes in Computer Science 2743*, pages 151–175. Springer-Verlag, 2003.

[HP03] Haruo Hosoya and Benjamin C. Pierce. XDuce: A Statically Typed XML Processing Language. In *ACM Transactions on Internet Technology*, volume 3(2), pages 117–148. ACM, 2003.

[HRS+03] Matthew Harren, Mukund Raghavachari, Oded Shmueli, Michael Burke, Vivek Sarkar, and Rajesh Bordawekar. XJ: Integration of XML Processing into Java. *IBM Research Report RC23007 (W0311-138)*, November 18, 2003.

[HVP00] Haruo Hosoya, Jérôme Vouillon, and Benjamin C. Pierce. Regular Expression Types for XML. In *Proceedings of the Fifth ACM SIGPLAN International Conference on Functional Programming (ICFP '00), Montreal, Canada*, volume 35(9) of *SIGPLAN Notices*, pages 11–22. ACM, September 18-21 2000. ISBN 1-58113-202-6.

[JDO] JDOM Project. JDOM FAQ. `http://www.jdom.org/docs/faq.html`.

[Kem03] Martin Kempa. *Programmierung von XML-basierten Anwendungen unter Berücksichtigung der Sprachbeschreibung*. PhD thesis, Institut für Informationssysteme, Universität zu Lübeck, 2003. Aka Verlag, Berlin, (in German).

[KL02] Martin Kempa and Volker Linnemann. VDOM and P-XML – Towards A Valid Programming Of XML-based Applications. *Information and Software Technology, Elsevier Science B. V.*, pages 229–236, 2002. Special Issue on Objects, XML and Databases.

[KL03] Martin Kempa and Volker Linnemann. Type Checking in XOBE. In Gerhard Weikum, Harald Schöning, and Erhard Rahm, editors, *Proceedings of Datenbanksysteme für Business, Technologie und Web (BTW), 10. GI-Fachtagung,*, volume P-26 of *Lecture Notes in Informatics*, pages 227–246. Gesellschaft für Informatik, 26.-28. Februar 2003.

[KMS04] Christian Kirkegaard, Anders Møller, and Michael I. Schwartzbach. Static Analysis of XML Transformations in Java. *IEEE Transactions on Software Engineering*, 30(3):181–192, March 2004.

[Kra02] Jens-Christian Kramer. Erzeugung garantiert gültiger Server-Seiten für Dokumente der Extensible Markup Language XML. Master's thesis, Institut für Informationssysteme, Universität zu Lübeck, 2002. (in German).

[LK02] Volker Linnemann and Martin Kempa. Sprachen und Werkzeuge zur Generierung von HTML- und XML-Dokumenten. *Informatik Spektrum, Springer-Verlag Heidelberg*, 25(5):349–358, 2002. (in German).

[Mic01] Microsoft Corporation. .NET Framework Developer's Guide. web document, `http://msdn.microsoft.com/library/default.asp`, 2001.

[MSB03] Erik Meijer, Wolfram Schulte, and Gavin Biermann. Programming with Circles, Triangles and Rectangles. `http://www.cl.cam.ac.uk/~gmb/Papers/vanilla-xml2003.html`, 2003.

[Ora01] Oracle Corporation. *Oracle9i, Application Developer's Guide – XML, Release 1 (9.0.1)*. Redwood City, CA 94065, USA, June 2001. Shelley Higgins, Part Number A88894-01.

[PLC99] Eduardo Pelegrí-Llopart and Larry Cable. JavaServer Pages Specification, Version 1.1. Java Software, Sun Microsystems, `http://java.sun.com/products/jsp/download.html`, 30. November 1999.

[Sei90] Helmut Seidl. Deciding equivalence of finite tree automata. *SIAM Journal of Computing*, 19(3):424–437, June 1990.

[SL04] Henrike Schuhart and Volker Linnemann. XOBE-DB: A statically typed XML Database Programming Language Based on JAVA and XQUERY, in preparation, 2004.

[Spi04] Torben Spiegler. Übungsdatenverwaltungssystem mit XOBE. Master's thesis, Institut für Informationssysteme, Universität zu Lübeck, 2004. (in German).

[Sun01] Sun Microsystems, Inc. Java 2 Platform, Standard Edition, v 1.3.1, API Specification. `http://java.sun.com/j2se/1.3/docs/api/index.html`, December 2001.

[TWP00] Kevin Tao, Wanjun Wang, and Dr. Jens Palsberg. Java Tree Builder JTB. `http://www.cs.purdue.edu/jtb/`, 15. May 2000. Version 1.2.2.

[W3C98a] W3Consortium. Document Object Model (DOM) Level 1 Specification, Version 1.0. Recommendation, `http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001/`, 1. October 1998.

[W3C98b] W3Consortium. Extensible Markup Language (XML) 1.0. Recommendation, `http://www.w3.org/TR/1998/REC-xml-19980210/`, 10. February 1998.

[W3C99] W3Consortium. XML Path Language (XPath), Version 1.0. Recommendation, `http://www.w3.org/TR/xpath`, 16. November 1999.

[W3C01] W3Consortium. XML Schema Part 0: Primer. Recommendation, `http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/`, 2. May 2001.

[W3C02] W3Consortium. XQuery 1.0: An XML Query Language. Working Draft, `http://www.w3.org/TR/2002/WD-xquery-20021115/`, 15. November 2002.

[Web02] WebGain. Java Compiler Compiler (JavaCC) – The Java Parser Generator. `http://www.webgain.com/products/java_cc/`, 2002. Version 2.1.

[Wil99] A. R. Williamson. *Java Servlets by Example*. Manning Publications Co., Greenwich, 1999.

[WS92] L. Wall and R. L. Schwartz. *Programming Perl*. O'Reilly & Associates, Inc., Sebastipol, California, 1992.

Appendix: Performance Measurements

Our interest concerning the performance of the XOBE implementation is twofold. First, we want to know the time which is spent for precompiling XOBE programs and especially for the type checking algorithm. Second, we measure the evaluation time of our resulting DOM-based servlets which we compare to a standard non-DOM servlet implementation. The programs are executed on a Sun Blade 1000 with two Ultra Sparc 3 (600 Mhz) processors running Solaris 8 (SunOS 5.8).

The program Estate generates XHTML web pages out of the data of a plain XML file following a small non-standard schema. UDV (**U**ebungs**D**aten**V**erwaltung) realizes a web-based academic exercise administration system based on XHTML servlets. The application communicates over JDBC with an Informix database management system. A WML connection to a medical media archive is realized by the servlet-based web application MobileArchive (MoArch). The program which accesses the archive through a given API allows navigation in the archive structure, searching for objects, and viewing of media objects in specific media formats. UDV and MobileArchive have been migrated from a standard Java servlet implementation.

Application	code	schema	compilation	standard	XOBE
Estate	158	1231	2.16	-	0.9
UDV	195	1196	4.61	0.01	0.01
MoArch	1045	355	4.49	0.03	0.04

In the table the number of lines of the whole XOBE programs and the number of lines of the imported schemas are presented in the first two columns. The third column of the table shows the time (in s) spent for precompiling the XOBE program. The last two columns give an impression of how the performance of the servlets is affected by the DOM-based implementation. The column "standard" shows the time (in s) which has elapsed evaluating the standard non-XOBE servlet implementation. The last column gives the running time (in s) of the XOBE program. All the times are averages determined by multiple runs.

As indicated by the table our precompiler runs at acceptable speed for these applications. Even the applications which use quite large types from the XHTML or WML schema are compiled in encouraging time. The execution speed of our DOM-based servlet implementations is slower than the standard servlets as expected but still in a convenient range. The reason for this is that the standard servlets neither guarantee well-formedness nor validity at compile time for the generated XML fragments.