Web Programming in Scheme with LAML

KURT NØRMARK
Department of Computer Science
Aalborg University
Denmark
(*e-mail:* `normark@cs.auc.dk`)

Abstract

Functional programming fits well with the use of descriptive markup in HTML and XML. There is also a good fit between S-expressions in Lisp and the XML data set. These similarities are exploited in LAML which is a software package for Scheme. LAML supports exact mirrors of the three variants of XHTML 1.0, SVG 1.0, and a number of more specialized XML languages. The mirrors are all synthesized from document type definitions (DTDs). Each element in a mirror is represented by a named function in Scheme. The mirror functions validate the XML document while it is generated. The validation is based on finite state automata automatically derived from the DTD.

1 Introduction

In this paper we discuss the use of Scheme (Kelsey *et al.*, 1998) in the domain of web programming and web authoring. Our primary concern is the modeling of HTML and XML in Scheme. The notion of web programming is limited to programmatic contributions in static web documents and to transformations of web documents. Server-side web programming is not directly addressed in this paper, although most aspects of the paper are relevant for server-side programming as well.

LAML stands for *Lisp Abstracted Markup Language*. The key idea of LAML is to make existing and major markup languages such as XHTML and SVG available as a set of Scheme functions. LAML supports the generation of Scheme mirror functions of any XML language defined by a DTD. The Scheme mirror functions reflect the properties and constraints of elements and attributes in the markup language.

LAML documents are written as Scheme programs. The textual content is represented as string constants. Internally, a document is represented as an abstract syntax tree (AST) in which the textual content, subtrees of the AST, white space markers, and character references can be distinguished by their dynamic types. A character reference refers to a specific character in a character set. HTML and XML document fragments are written as Scheme expressions which call the mirror functions. In a LAML source document there are no lexical nor syntactical traces left of HTML and XML. The validity of a LAML web document is checked at document generation time, corresponding to type checking at run-time. LAML uses a standard Scheme reader, in contrast to some similar Scheme-based systems

```
(load (string-append laml-dir "laml.scm"))
(laml-style "simple-xhtml1.0-transitional-validating")

(write-html '(raw prolog)
  (html 'xmlns "http://www.w3.org/1999/xhtml"
    (head (title "Hello World"))
    (body (p "Hello" (a 'href "http://www.w3c.org/" "W3C")))))

(end-laml)
```

Fig. 1: *A LAML "Hello World" document.*

(BRL and Scribe) which use an extended Scheme syntax. LAML can be used with any R4RS or R5RS Scheme system which implements a small collection of well-defined, operating system related procedures and functions (such as `delete-file` and `file-exists?`.)

The primary goal of LAML is to support the creation of complex web material in Scheme. In many ways complex web material resemble non-trivial programs. The need of abstraction is a primary concern. At the fine grained level, abstraction can be supported by definition of functions that encapsulate a number of document details. At a more coarse grained level, linguistic abstraction is supported in LAML by the generation of exact mirrors of XML languages, as defined by DTDs. Programmatic means of expressions, as reflected by selection and iteration, is also important when we deal with complex web documents. As a particular aspect, LAML has been designed to make good use of higher-order list functions. This will be illustrated in section 4.

The source of a LAML web document is a Scheme program which uses the LAML libraries, most importantly the set of HTML mirror functions and mirrors of other XML languages. Working on this basis, the Scheme programming language is available at any location in a web document, and at any time during the authoring process. As a pragmatic consequence, many problem solving aspects can be handled inside the document—expressed in Scheme—as opposed to handling by external XML tools and processors. Due to this we use the term *programmatic authoring* for the approach (Nørmark, 2002).

The main contribution of this work is the mirroring scheme that makes HTML elements, and elements from other XML languages, available as Scheme functions. The integrated validation of the documents, at document generation time, is an important part of the approach. The fitting of the framework to support a natural organization of document data in lists is also important.

In section 2 we discuss a couple of simple examples of complete LAML documents, beginning at the "Hello World" level. In section 3 the XHTML mirror functions are explained and discussed. Section 4 contains additional examples, primarily illustrating the use of higher-order functions together with LAML. In section 5 the XML framework in LAML is covered. The work on LAML is related to similar work in section 6. Conclusions are drawn in section 7.

2 Initial examples

Figure 1 shows a "Hello World" example to illustrate the composition of a complete LAML document. The first line loads the fundamental LAML software; the second line loads the XHTML 1.0 transitional mirror library; then follows a `write-html` clause, which contains a `(html ...)` expression. The expression uses the XHTML mirror functions `html`, `head`, `title`, `body`, `p`, and `a` which correspond to the similarly named elements in XHTML. The first parameter of `write-html` controls the format of the textual rendering and the document prologue. The rendering may be raw or pretty printed, and the document prologue consists of an XML declaration and a document type definition.

Most LAML documents introduce a number of document abstractions. Even in relative simple web documents there are many good uses of functional abstractions. This is illustrated in Figure 2 by elaborating the example from Figure 1. Many useful abstractions are related to attribute values, such as the function `w3c-url` that abstracts the prefix part of the W3C URL. Others are related to sets of attributes, such as `html-props` and `body-props`.

It is also useful to introduce content-related abstractions. As an example, the document in Figure 2 implements and uses the function `indent-pixels` for textual indentation. The function can be implemented with use of an HTML table. The function `author-signature`, which is intended to be defined in a personal LAML startup file, returns the author's name, affiliation, and email address.

Based on the examples we make two observations about content-related abstractions in LAML. First, ordinary positional parameters do not fit well with the parameter conventions of the HTML mirror functions. Therefore it is attractive to use a parameter profile of `indent-pixels` which is similar to the conventions of the HTML mirror functions. We show how this can be done in section 4. Second, if a coherent collection of content abstractions is necessary, it is useful to implement this collection as a new XML language in LAML. We discuss this in section 5.

3 XHTML mirror functions

The XHTML mirror functions are designed with the goal that element instances in web documents should have straightforward and easily recognizable counterparts in Scheme. As examples, the XML clauses

```
<tag1 a1="v1" ... am="vm">contents</tag1>
<tag2 a1="v1" ... am="vm" />
```

correspond to the Scheme expressions

```
(tag1 'a1 "v1" ... 'am "vm" "contents")
(tag2 'a1 "v1" ... 'am "vm")
```

Expressions like these are evaluated to instances of ASTs, which in turn are represented as tagged list structures. An AST node holds information about the element name, the element contents, the attributes, the XML language being used, and whether the node represents an empty XML element. An AST is typically

```
(load (string-append laml-dir "laml.scm"))
(laml-style "simple-xhtml1.0-transitional-validating")

(define html-props (list 'xmlns "http://www.w3.org/1999/xhtml"))

(define body-props
  (list 'bgcolor (rgb-color-encoding white) 'text (rgb-color-encoding black)
        'link (rgb-color-encoding blue) 'vlink (rgb-color-encoding blue)))

(define (w3c-url suffix) (string-append "http://www.w3c.org/" suffix))

(define (indent-pixels p indented-form)
  (table 'border "0"
    (tr (td 'width (as-string p))
        (td 'width "*" indented-form))))

(write-html '(raw prolog)
  (let ((ttl "A simple page"))
    (html html-props
      (head (title ttl))
      (body  body-props
        (h1 ttl)
        (indent-pixels 50
          (p "The" (a 'href (w3c-url "") "W3C")
              "web site has information about"
              (a 'href (w3c-url "MarkUp/") "HTML") _ ","
              (a 'href (w3c-url "XML/") "XML") _ ","
              "and many other web technologies."))
        (author-signature)   ))))

(end-laml)
```

Fig. 2: *A simple LAML web document with a number of abstractions.*

transformed to a lower level representation (such as an HTML AST), rendered as a string, or rendered directly to an open output port. In both rendering situations we avoid excessive string concatenation to reduce the amount of garbage collection of string parts.

A mirror function accepts AST values, character reference values (tagged lists like the ASTs) as well as strings, symbols, booleans, and lists. At run time, the types of the actual parameter values are used to control the interpretation of the parameters, prior to the building of an AST. The possibility of passing an arbitrary long list of parameters of different dynamic types is crucial for our approach. The mirror functions obey the following *LAML parameter passing rules*:

- ***Rule 1.*** A symbol represents an attribute name. Symbols of the form `css:`*a* refer to the *a* attribute in CSS. A symbol must be followed by a string that plays the role of the attribute value.
- ***Rule 2.*** A string which does not succeed a symbol is an *element content item*. Character reference values as well as AST values returned by mirror functions are also element content items.

- ***Rule 3.*** Element content items are implicitly separated by white space. An explicit boolean *false* value between element content items suppresses white space. The boolean *false* value is bound to the underscore variable.
- ***Rule 4.*** A list of symbols, strings, booleans, character references, and ASTs is processed recursively, and the resulting element content items, attributes and white space markers are spliced with the surrounding list of parameters.

LAML can also be used in a relaxed mode where numbers and characters serve as element content items and attribute values. The following LAML expression illustrates the parameter passing rules.

```
(p "The" (a 'href "www.w3c.org" "WWW") "Consortium" _ ".")
```

The value of the expression is the following slightly abbreviated AST:

```
(ast "p"
 ("The" #t (ast "a" ("WWW") (href "www.w3c.org") ...) #t "Consortium" ".")
 () double xhtml10-transitional)
```

A boolean *true* value in the AST represents white space. The AST can be rendered as the following HTML clause:

```
<p>The <a href = "www.w3c.org">WWW</a> Consortium.</p>"
```

The mutual order of the element content items and the attributes is arbitrary as long as Rule 1 is adhered to. Thus, `(a 'href "URL" "A" "B" "C")` and `(a "A" 'href "URL" "B" "C")` are equivalent expressions. The rationale behind Rule 3 is to support white space in between element content items (the typical case) without use of additional and explicit markup.

In Lisp it is often convenient to represent document fragments as nested lists. This is the rationale behind Rule 4. As an example, the expression

```
(let ((attributes (list 'start "3" 'compact "compact"))
      (contents (map li (list "one" "two" "three"))))
  (ol 'id "demo" contents (li "final") attributes))
```

which can be rendered as

```
<ol id="demo" start="3" compact="compact">
  <li>one</li> <li>two</li> <li>three</li> <li>final</li>
</ol>
```

shows that both an attribute list and a list of element content items can be passed to the `ol` mirror function.

The XHTML mirror functions validate the generated document at the time the LAML expressions are evaluated. The validation is done relative to the underlying DTD. Both the document composition and the attributes are checked. The document composition must be in accord with the element content models, which taken together represent a context free grammar of the XML language. The attributes are checked for attribute existence, presence of required attributes, and avoidance of attribute duplication. In case of validation problems, warnings are issued. If the author prefers, a validation failure may alternatively lead to a fatal error. Additional details of the document validation framework is discussed in section 5.

The validation of a document against the DTD would be in vain if the textual content or an attribute value of a document is allowed to contain the characters '<', '>', or '&' (or a double quote character in an attribute value). Instead of prohibiting these characters in LAML source documents we translate them to their similar HTML character references, such as <. The translation is carried out by means of a systematic mapping of every character in the textual contents and in attribute values. We also use the mapping to translate national characters, such as the Danish 'æ', 'ø', and 'å', to the corresponding HTML character references.

4 Examples with higher-order functions

There are many good uses of higher-order functions in relation to the XHTML mirror functions. As the first application, we will see how an HTML table can be made by combining the table, tr, and td mirror functions. In many contexts we find it natural to represent tables as list of rows, where each row is a list of elements:

```
(define sample-table '(("Row" "no." "1") ("Row" "no." "2")))
```

The following expression generates an XHTML table of sample-table

```
(table (map (compose tr (map td)) sample-table) 'border "1")
```

The table is rendered as

```
<table border="1"> <tr><td>Row</td> <td>no.</td> <td>1</td></tr>
                   <tr><td>Row</td> <td>no.</td> <td>2</td></tr>
</table>
```

Above, it is assumed that map is curried (done by the LAML function curry-generalized). The function compose composes two or more functions to a single function.

The LAML higher-order function xml-modify-element can bind attributes and content items to fixed values in a mirror function. As an example, the following expression returns a specialized a (anchor) function in which the target and the title attributes have fixed values:

```
(xml-modify-element a 'target "main" 'title "Goes to the main window")
```

As pointed out in section 2, it is very useful to abstract commonly used patterns of content items and attributes in XML documents. In many situations it is convenient and natural to use the LAML parameter passing rules from section 3 for these abstractions. We will now show how to adapt the existing function indent-pixels from Figure 2 to make use of LAML parameter passing. Instead of the expression

```
(indent-pixels 50 (div (p "First par.") (p "Second par.")))
```

we introduce attributes and content items, such as in

```
(new-indent-pixels 'indentation "50" (p "First par.") (p "Second par."))
```

With the new parameter profile we can pass an arbitrary number of content items to new-indent-pixels without aggregating them with div. The function xml-in-laml-parametrization generates the new version of the indentation function from the existing one:

```
(define new-indent-pixels
 (xml-in-laml-parametrization indent-pixels
  (lambda (contents attributes)
    (list (get-prop 'indentation attributes) (div contents)))
  (required-implied-attributes '(indentation) '())))
```

The second parameter of `xml-in-laml-parametrization` is a function which must return the parameter list to `indent-pixels`. The third parameter of `xml-in-laml-parametrization` is supposed to validate the contents and the attributes. Above, we use the function `required-implied-attributes` which returns a predicate that ensures the presence of the `indentation` attribute, and that no other attributes are passed. A function similar to `xml-in-laml-parametrization` allows us to make ad hoc abstractions on top of existing XML mirror functions.

5 Synthesis of XML mirror functions

The mirror functions of an XML language can be synthesized from the XML document type definition (DTD) of the language. LAML supports a DTD parser which delivers a list representation of the DTD, in which all entity instances (textual macro applications) are unfolded. The list representation of the DTD is used as input to the LAML mirror generation tool, which creates a Scheme source file with the mirror functions of the XML elements. The XHTML mirrors described in section 3, as well as a mirror of SVG, have been produced by these tools.

As an important aspect, the mirror functions validate XML documents at document generation time. The validation of the attributes has already been explained in section 3. LAML defines a validation procedure for each XML element. The validation procedure checks the context free correctness of a construct relative to the content specifications in the XML DTD. In case of validation problems, an error message is issued. We have emphasized the production of straightforward and easily understandable error messages. The content specifications are regular expressions. As examples of content specifications, the `table` and the `body` elements in XHTML 1.0 are constrained by the following (slightly abbreviated) regular expressions:

```
(caption?, (col*|colgroup*), thead?, tfoot?, (tbody+|tr+))
(#PCDATA | a | abbr | acronym | address | applet | b | basefont | ...)*
```

The LAML mirror synthesizer generates deterministic finite state automata for the elements that have *element content* (such as `table`) whereas the elements with *mixed content* (such as `body`) are validated by simpler means. The automata are implemented by use of Algorithm 3.5 of Aho, Sethi, and Ullman (1986). The automata are represented as lists and vectors in Scheme, and they are embedded directly and compactly in the validation procedures. Automaton compactness is important to keep down the size of the Scheme source files and thereby the software loading times. The automaton validation functions are fast due to use of binary search for the transitions.

In the XHTML 1.0 strict mirror the average number of transitions per automaton is 189, the maximum number is 1039, and the median is 14. In the SVG 1.0 mirror the average number of transitions is 290, the maximum number is 1600, and the median is 31. The largest SVG automaton occupies approximately 16 Kbytes in the Scheme mirror source file. The largest automata occur for element-contents models of the shape `(e`$_1$ `| e`$_2$ `| ... | e`$_n$`)*`. The size of the Scheme source file of the XHTML 1.0 strict mirror is 236 Kbytes, and that of the SVG mirror is 527 Kbytes.

In addition to the mirrors of XHTML and SVG we have defined a number of other XML languages in LAML. The mirrors of these XML languages can be seen as linguistic abstractions rather than sets of individual functions. As a central aspect, a set of library functions are shared among all XML languages in LAML. This shared library supports the internal AST document format, higher-order AST traversal and transformation functions, textual rendering functions, and common content and attribute validation functions.

When two different XML languages in LAML have identically named elements, there will be a clash of mirror function names in Scheme. XML solves this problem by means of *name spaces* which disambiguate the two names by means of a unique prefix. In LAML, we have introduced the concept of a *language map*. A language map for a given XML language maps an element name to the corresponding Scheme mirror function. As an example take the following:

`(xhtml10-strict 'title)` $\Rightarrow$ *the* `title` *mirror function in XHTML 1.0 strict*

If no ambiguity is present, a mirror function can be accessed via a simple name. In case of ambiguity, a warning is issued at document generation time, and the mirror function should be accessed via the language map. At mirror generation time, we check that no mirror function clashes with names of R4RS Scheme functions.

6 Related work

The description of related work is structured relative to the programming language being used. We first discuss related work in Scheme, then work related to other functional programming languages, and finally work in other paradigms. Throughout the discussion we focus on the representation of the web documents in relation to the base programming language. In addition we focus on the web document validation approaches being used.

DSSSL (ISO/IEC 10179, 1996) is a collection of languages for processing of SGML documents. DSSSL uses a subset of Scheme as the underlying programming language. The processing which is involved covers document querying, transformation, and formatting.

Scribe (Serrano & Gallesio, 2002) is a Scheme-based system for authoring of web pages, in particular technical documents. Document fragments are represented as Scheme expressions in both Scribe and LAML. Scribe is based on a non-standard Scheme reader, which introduces a new bracketed syntax for (lists of) strings with inspiration from Scheme quasiquotation. The string `[a ,(bold "bold") text]` serves as an example. Scribe supports a particular document language (in the style of LaTeX) and because of that Scribe can generate output in different formats, such as HTML, Postscript, and PDF. In contrast, LAML is limited to produce

XML documents. Document validation seems not to be an issue in Scribe. Like LAML, Scribe uses an internal AST document representation. Scribe relies on a novel multi-step evaluation process which allows document introspection, such as needed for generation of a table of contents. In LAML, the similar effect is obtained by first establishing the full AST of an XML source document followed by two or more traversals of the source AST (one for extraction of structural information, for instance, and one for a detailed rendering of the document in the target XML language).

BRL (Lewis, 2003) is a language designed for server-side, database connected web applications. BRL represents a document in HTML, with Scheme program fragments embedded in square bracket notation. Alternatively, a BRL document can be seen as a non-standard Scheme program in which strings are surrounded by "reverse square brackets," such as `]a string[`. In contrast, a LAML document is a standard Scheme program which avoids the mixing of XML markup and Scheme fragments. Validation of the resulting HTML documents is not an issue in BRL.

Kiselyov (2002) defines an XML format in Scheme called SXML. An SXML clause is represented as an S-expression (a list data structure), whereas a LAML clause is a Scheme expression which refers to named XML mirror functions. Both formats are intended for authoring purposes. Conceptually, however, the SXML format is similar to LAML ASTs. Validation of SXML documents is intended to be done by an SXML producing tool, such as a parser.

Latte (Glickstein, 1999) is a mixture of the LaTeX text formatting system and Scheme, at least at the conceptual level. In Latte, the author uses a LaTeX-like markup style. Most interesting, however, Latte supports a Scheme-like language in TEX syntax. Like Scribe, Latte can produce output in several formats, such as HTML, TEX, and plain text.

XEXPR (Nicol, 2000) is a scripting language, inspired by Lisp and Scheme, that uses XML syntax. XEXPR can be seen as a mirror of some Lisp dialect in XML. LAML goes in the opposite direction, by creating a mirror of XML languages in Scheme.

XQuery (Boag *et al.*, 2003) and XSLT (Clark, 1999) are both based on the concepts of functional programming languages, and they are both developed by the World Wide Web Consortium. XQuery is a proposal for an XML query language, and XSLT is an XML transformation language based on pattern matching and replacement. As an alternative to XSLT, the PLT Scheme group has developed XT3D for XML transformation by example (Krishnamurthi *et al.*, 2000) with inspiration from the R5RS Scheme macro facility. As part of this work it is possible to generate Scheme builder functions from XML Schemas. In LAML the similar mirror functions are generated from XML DTDs. Internally, the PLT tools represent XML documents as list structures, which are called x-expressions. These are similar to the AST structures used in LAML.

In a recent paper, Kiselyov and Krishnamurthi (2003) describe a Scheme counterpart to the W3C XSLT transformation framework, which they call SXSLT. SXSLT works on SXML structures, which have been described above. It is argued that SXSLT is superior to XSLT, and that it is more adequate for power users than

XT3D. Transformation of XML documents in LAML relies on a few AST traversal functions, and it relies on the precondition that every source document of the transformation is valid. In addition, the LAML transformation framework is backed up by statically derived information about the possible presence of certain elements in documents rooted by other elements. The details can be found in (Nørmark, 2003b).

Wallace and Runciman (1999) discuss two different representations of XML documents in Haskell. One is based on a generic tree representation of XML documents; the other is based on typed document fragments, where the DTD gives rise to a number of algebraic type definitions in Haskell. The driving force behind the second approach is full validation of XML documents via static type checking of the Haskell XML programs. The ideas in this work are embodied in the system called HaXml (HaXml, 2003).

Meijer and colleagues have in a number of papers dealt with aspects of web programming using Haskell. In the first of these a Haskell framework for CGI programming is presented (Meijer, 2000). In a second paper, Meijer and Shields (2000) define a new language called XMλ which is intended for generation of dynamic XML documents. The main focus of this work is to ensure creation of valid XML documents by use of static type checking of XMλ programs. Like in ASP, PHP, and JSP, an XMλ expression `e` is escaped as `<%= e %>` in an XML document. In LAML, there is no need for escaping of Scheme expressions, but the textual contents can be seen as escaped in string quotes.

Thiemann describes a modeling of XML and HTML in Haskell (Thiemann, 2002). The ideas are implemented in the system called WASH/HTML (WASH, 2003). Thiemann can synthesize Haskell combinator libraries from XML and HTML DTDs. The synthesized libraries validate Haskell XML and HTML expressions statically by means of Haskell type checking. Several levels of validations are provided. Thiemann argues that full validation is not practical for real life web programs and that weak validity is sufficient for practical purposes. In the work with LAML we have found that a comprehensive validation of XML documents is both important and worthwhile.

XDuce (Hosoya & Pierce, 2003) is a special purpose, statically typed functional programming language for XML processing. XDuce uses regular expression types, which are comparable to the element content model of XML DTDs. XDuce relies on regular expression pattern matching for decomposition of XML documents. As a practical incompleteness, XDuce does not support the handling of XML attributes.

Hanus (2001) describes a functional/logical web programming framework for the language called Curry. This work is based on a straightforward modeling of HTML as Curry data structures.

Curl (Hostetter *et al.*, 1997) is an object-oriented programming language specifically designed for web authoring and web programming purposes. Curl represents documents as plain text with escaped program fragments in curly brackets. In its foundation Curl is statically typed, but it is also possible to use dynamic types.

`<bigwig>` (Brabrand *et al.*, 2002) is a C-like web programming language. `<bigwig>` checks the validity of the involved HTML documents at compile time. In addition,

Table 1. *A summary of related systems in terms of the programming languages being used, the web document representations, and the validation approaches.*

System	**Language**	**Representation**	**Validation approach**
LAML	Scheme	Scheme expressions	Dynamic type check
Scribe	Scheme	Scheme expressions[a]	None
BRL	Scheme	HTML with escape to Scheme expressions	None
SXML	Scheme	S-expression data structures	By external tools
Latte	Scheme-inspired	LaTeX-like with Scheme-inspired expressions	None
XEXPR	Lisp-inspired	XML	By external tools
HaXml[b]	Haskell	Haskell expressions	Static type check
XMλ	Haskell-like	XML with escape to the programming language	Static type check
WASH/HTML	Haskell	Haskell expressions	Static type check
XDuce	Xduce language[c]	Expressions in Xduce	Static type check
Curry web scripting	Curry	Curry data structures	None
Curl	Curl language[d]	Plain text with escape to Curl fragments	Static/dynamic
Bigwig	Bigwig language[e]	Bigwig fragments with escape to HTML templates	Static type check
Jwig	Java-like	Java program with escape to XML templates	Static type check

[a] with special string syntax
[b] type based
[c] special purpose functional language
[d] special purpose object-oriented language
[e] special purpose C-like language

`<bigwig>` supports a session concept and a client-side form validation framework. More recently, the work on `<bigwig>` has been generalized and adopted to Java in the JWIG system (Christensen *et al.*, 2003).

Table 1 summarizes the description of the related work. For each related system, which we have discussed above, the table shows the programming language being used, the representation of the web document, and the document validation approach.

7 Conclusions

LAML is designed for authoring of complex web pages and web sites. LAML provides mirror libraries of existing XML markup languages, such as XHTML and SVG, as well as a number of more specialized XML languages. A LAML document is a Scheme program which uses a normal Scheme reader. LAML documents can be processed by use of most Scheme systems. The parameter passing rules of the XML mirror functions have been developed through a number of LAML generations.

We have found that programmatic authoring using LAML is convenient and powerful for Scheme programmers. The possibility of introducing abstractions in web documents is of great importance. Linguistic abstraction, in terms of mirrors of new XML languages, deals with the overall needs. It is, however, our experience that there will always be a need for special, ad hoc abstractions, implemented as functions. This is especially the case when we create documents and programs in XML languages not controlled by the author, such as HTML and SVG. In this context it is important to support parameter passing rules of ad hoc abstractions that are similar to the LAML parameter passing rules.

We have used LAML extensively over the last five years, both for processing of LAML documents and for server-side CGI programming. The comprehensive validation of static LAML documents is seen as a valuable asset, because any deviation from the document standard is identified when the web documents are generated. Run-time validation of server-side web programs is less attractive. The fully automatic generation of the validation procedures is a major step forward, compared to earlier versions of LAML which required some manual programming efforts to produce the validation procedures.

This paper is relative to version 23 of LAML. LAML is available as free software from the LAML homepage (Nørmark, 2003a).

Acknowledgements

Thanks to Lone Leth Thomsen for help with the final version of this paper. Thanks are also due to the reviewers. Their extensive review reports helped improve this paper considerably.

References

Aho, A. V., Sethi, R., & Ullman, J. D. (1986). *Compilers - principles, techniques and tools*. Addison-Wesley.

Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J., & Siméon, J. 2003 (August). *XQuery 1.0: An XML query language.* W3C Working Draft. `http://www.w3.org/TR/xquery`.

Brabrand, C., Møller, A., & Schwartzbach, M. I. (2002). The <bigwig> project. *ACM transactions on internet technology*, **2**(2), 79–114.

Christensen, A. S., Møller, A., & Schwartzbach, M. I. (2003). *Extending Java for high-level web service construction.* `http://www.brics.dk/~mis/jwig.pdf`. To appear in ACM Transactions on Programming Languages and Systemes.

Clark, J. 1999 (November). *XSL transformations (XSLT) version 1.0*. W3C Recommendation. `http://www.w3.org/TR/xslt`.

Glickstein, B. (1999). *Latte the language for transforming text*. `http://www.latte.org/-latte.html`.

Hanus, M. (2001). High-level server side web scripting in Curry. *Pages 76–92 of:* Ramakrishnan, I.V. (ed), *PADL 2001*. Lecture notes in computer science 1990. Springer Verlag.

HaXml. (2003). *Haxml*. `http://www.cs.york.ac.uk/fp/HaXml/`.

Hosoya, H., & Pierce, B. C. (2003). Xduce: A statically typed XML processing language. *ACM transactions on internet technology*, **3**(2), 117–148.

Hostetter, M., Kranz, D., Seed, C., Terman, C., & Ward, S. (1997). Curl: A gentle slope language for the web. *World wide web journal*, **2**(2). `http://www.w3j.com/6/s3.-kranz.html`.

ISO/IEC 10179. (1996). *Information technology - processing languages - document style semantics and specification language (DSSSL)*. `http://www.oasis-open.org/cover/-dsssl.html`.

Kelsey, R., Clinger, W., & Rees, J. (1998). Revised^5 report on the algorithmic language Scheme. *Higher-order and symbolic computation*, **11**(1), 7–105.

Kiselyov, O. 2002 (August). *SXML*. `http://okmij.org/ftp/Scheme/SXML.html`.

Kiselyov, O., & Krishnamurthi, S. (2003). SXSLT: Manipulation language for XML. *Pages 256–272 of:* Dahl, V., & Wadler, P. (eds), *PADL 2003*. Lecture notes in computer science 2562. Springer Verlag.

Krishnamurthi, S., Gray, K. E., & Graunke, P. T. (2000). Transformation-by-example for XML. *Pages 249–262 of:* Pontelli, E., & Costa, V. S. (eds), *PADL 2000*. Lecture notes in computer science 1753. Springer Verlag.

Lewis, B. R. 2003 (June). *BRL—a database-oriented language to embed in HTML and other markup*. `http://brl.sourceforge.net/brl.pdf`.

Meijer, E. (2000). Server side web scripting in Haskell. *Journal of functional programming*, **10**(1), 1–18.

Meijer, E., & Sheilds, M. (2000). *Xmλ - a functional language for constructing and manipulating XML documents*. Submitted to USENIX Annual Technical Conference 2000. `http://www.cse.ogi.edu/~mbs/pub/xmlambda/`.

Nicol, G. T. 2000 (November). *XEXPR - a scripting language for XML*. W3C Note. `http://www.w3.org/TR/xexpr/`.

Nørmark, K. 2002 (May). Programmatic WWW authoring using Scheme and LAML. *The proceedings of the eleventh international world wide web conference - the web engineering track*. ISBN 1-880672-20-0. `http://www2002.org/CDROM/alternate/296/`.

Nørmark, K. (2003a). *The LAML home page*. `http://www.cs.auc.dk/~normark/laml/`.

Nørmark, K. 2003b (October). XML transformations in Scheme with LAML - a minimalistic approach. *The proceedings of the international Lisp conference, ILC 2003*. Association of Lisp Users. `http://www.cs.auc.dk/~normark/laml/papers/-xml-transformations.pdf`.

Serrano, M., & Gallesio, E. 2002 (October). *This is Scribe!* Presented at the *Third Workshop on Scheme and Functional Programming*. `http://www-sop.inria.fr/mimosa/-fp/Scribe/doc/scribe.html`.

Thiemann, P. (2002). A typed representation for HTML and XML documents in Haskell. *Journal of functional programming*, **12**(5), 435–468.

Wallace, M., & Runciman, C. (1999). Haskell and XML: generic combinators or type-based translation? *Pages 148–159 of: Proceedings of the fourth ACM SIGPLAN international conference on functional programming.* ACM Press.

WASH. (2003). *Web authoring system Haskell.* `http://www.informatik.uni-freiburg.-de/~thiemann/WASH/`.