# Typing XHTML Web Applications in ML

**Martin Elsman**

**joined work with Ken Friis Larsen**

**IT University of Copenhagen**

**PADL'04 Talk, June 19, 2004**

# Practical Motivation

## SMLserver:

- An efficient multi-threaded Web server platform for Standard ML programs

## Features of SMLserver (presented at PADL'03):

- A rich interface for writing Web applications, including:
  - Easy access to form data and other request information
  - Access to RDBMSs through an efficient generic interface
  - Support for type safe data caching, fetching of foreign Web pages, filtering and scheduling of script execution
- Programs are compiled into bytecode files, which are loaded only once but may be executed many times
- SMLserver is used for a series of administrative systems at the IT University of Copenhagen

# Why Standard ML?

### Static typing:

- Web applications are exposed to users early
- Web applications change rapidly
- Static typing can ease maintenance and improve robustness

### Higher-order functions:

- High degree of code reuse and shallow interfaces
- Good for separation of development tasks (design vs. application logic)

### A rich modules system:

- Name space management
- High degree of code reuse

### Based on the ML Kit Compiler:

- No reference tracing garbage collection and no tags, but region based memory management

# The Problem

- Traditionally, generated XHTML code is not guaranteed to *conform* to a particular XHTML specification.
- Also, a particular use of form data is not guaranteed to be consistent with the corresponding forms.

## A Solution:

- Provide a set of *type safe* combinators for constructing conforming XHTML documents.
- Use *phantom types* to express combinator conformity requirements:
  - validity
  - element prohibitions
  - linearity constraints on attributes
- Use phantom types for enforcing consistency between use of form data and corresponding form definitions.

# Conformity of XHTML 1.0 Documents

### Well-formedness:

All start-tags must have a corresponding closing-tag, all elements must be properly nested, and no attribute name may appear more than once in the same start-tag.

### Validity:

A valid XHTML document must be derivable from the context free grammar specified by the XHTML DTD.

### Six Element Prohibitions:

For example, to all depth of nesting, a `<big>` element must not be contained in a `<pre>` element.

# An Unsafe Combinator Library

## Simplified Library:

```
signature XHTML = sig
 type elt
 val $ : string->elt
 val & : elt*elt->elt
 val td    : elt->elt
 val tr    : elt->elt
 val table : elt->elt
end
structure XHtml :> XHTML =
 struct ... end
```

## Example Application Code:

```
open XHtml
fun concat es =
  foldr & ($"") es
fun toCols xs =
  concat (map (td o $) xs)
fun toRows xs =
  concat (map (tr o toCols) xs)
fun toTable strings =
  table(toRows strings)
```

## Comments:

- The library partly guarantees well-formedness of generated documents.
- The library does not guarantee validity; consider applying the `toTable` function to the empty list.

# A Simplified XHTML 1.0 DTD

## The DTD:

```
<!ENTITY  %block  "p|table">
<!ENTITY  %inline "#PCDATA">
<!ENTITY  %flow   "%block|%inline">
<!ENTITY  %td     "td">
<!ENTITY  %tr     "tr">
<!ELEMENT p     (%inline)*>
<!ELEMENT td    (%flow)*>
<!ELEMENT tr    (%td)+>
<!ELEMENT table (%tr)+>
```

## The DTD captures:

- the distinction between `inline`, `block`, `flow`, `td`, and `tr` entities
- two notions of sequencing (* and +)

# A Type Safe Simplified XHTML Combinator Library

```
signature XHTML = sig
 type ('b,'i)flw and tr
  and td and blk and inl
  and 'e elt and NOT
 val emp : unit
      -> ('b,'i)flw elt
 val $ : string
      -> ('b,inl)flw elt
 val p : (NOT,inl)flw elt
      -> (blk,'i)flw elt
 val table : tr elt
      -> (blk,'i)flw elt
 val tr : td elt -> tr elt
 val td : (blk,inl)flw elt
      -> td elt
 val & : 'e elt * 'e elt->'e elt
end
```

- The empty element must be of entity `flow`, `inline`, or `block`.
- Text ($) results in an `inline` entity.
- A <p> element results in a `block` entity, but consumes an `inline` entity.
- A <table> element results in a `block` entity and consumes a `tr` entity.
- A <tr> element results in a `tr` entity and consumes a `td` entity.
- A <td> element results in a `td` entity and consumes a `flow` entity (i.e., either a `block` entity or an `inline` entity.

# Implementation of the Type Safe Combinator Library

```
structure XHtml :> XHTML =
struct infix &
 datatype e = Elt of string*e
  | S of string | Seq of e*e
 type 'e elt = e
  and blk=unit and inl=unit
  and NOT=unit
  and ('b,'i)flw = unit
  and tr=unit and td=unit
 fun $ s = S s
 fun p e = Elt("p",e)
 fun td e = Elt("td",e)
 fun tr e = Elt("tr",e)
 fun table e = Elt("table",e)
 fun emp() = S ""
 fun e & e' = Seq(e,e')
end
```

- All element types share the same generic datatype implementation.
- Type Safety: XHTML documents, constructed using the combinator library, conform to the XHTML 1.0 specification.

# Example Application Code Revisited

## Modified Application Code:

```
open XHtml
fun concat xs = case xs of
    [] => raise List.Empty
  | [x] => x
  | x::xs => x & concat xs
fun toCols xs =
  concat (map (td o $) xs)
fun toRows xs =
  concat (map (tr o toCols) xs)
fun toTable strings =
  table(toRows strings)
```

- By replacing the earlier declaration of the `concat` function, generated XHTML code is guaranteed to comform to the XHTML 1.0 DTD.

# A Simplified XHTML Attribute Library

- Simplified abstract attribute library that statically guarantees the linearity well-formedness condition on attributes:

```
signature XHTML_ATTR = sig
  type ('a0,'a,'b0,'b)attr
  type na and width and height
  val width  : int -> (na,width,'b,'b)attr
  val height : int -> ('a,'a,na,height)attr
  val % : ('a0,'a,'b0,'b)attr * ('a,'a1,'b,'b1)attr
       -> ('a0,'a1,'b0,'b1)attr
end
```

- The `attr` type records "incoming" and "outgoing" linearity information (present or `na`) for each possible attribute.

# Attributes Continued

## Examples:

- The expression: `width 100 % height 30`
  has type `(na,width,na,height)attr`.
- The expression: `width 100 % height 30 % width 10`
  cannot be given a type.
- The order of attributes is insignificant.

## Attributes in XHTML 1.0:

- No element supports more than a dozen attributes.
- Type parameter reuse can reduce the number of parameters for the `attr` type.
- An appropriate attribute accepting combinator is introduced for each XHTML element.

# Scriptlets and Form Consistency

## An Example Scriptlet For Computing a Body Mass Index:

```
functor bmi (F : sig val h : int Form.var
                     val w : int Form.var
                 end) : SCRIPTLET =
  struct open XHtml infix &
    val response = case (Form.get F.h, Form.get F.w) of
       (Form.Ok h, Form.Ok w) =>
         let val bmi = Int.div(w * 10000, h * h)
             val txt = if bmi > 25 then "too high!"
                       else if bmi < 20 then "too low!"
                       else "normal"
         in Page.page "Body Mass Index"
             (p ($ ("Your BMI is " ^ txt)))
         end
       | _ => Page.page "Form Error" (p($"Go Back"))
  end
```

# Enforcing Form Consistency

## Compile-time Code Generation in SMLserver:

- Based on the scriptlet functor arguments, code is generated at compile time for instantiating the scriptlet upon client requests.
- Also based on the scriptlet functor arguments, due to the lack of recursive modules in Standard ML, an abstract scriptlet interface is generated for each scriptlet.

## Example Generated Abstract Scriptlet Interface:

```
structure bmi : sig
  type h and w
  val h : (h,int)name
  val w : (w,int)name
  val form : (h->w->nil,nil)elt -> (nil,nil)elt
end
```

- The abstract scriptlet interface introduces the concept of *type lists* — i.e., lists at the type level, with constructors `nil` and `->`.

# Enforcing Form Consistency — Continued

## Simplified XHTML Interface with Input Controls:

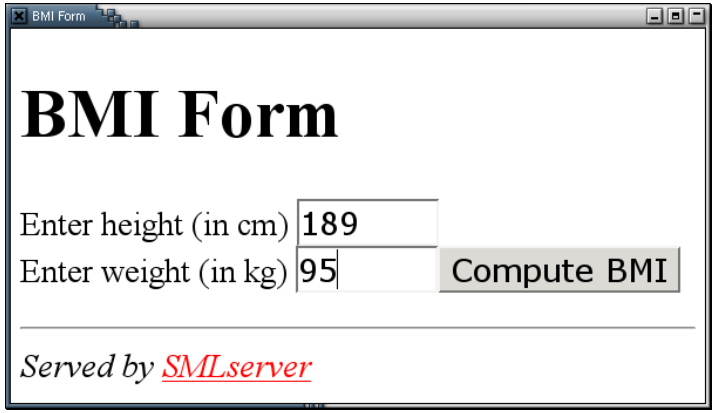
```
signature XHTML = sig
  type ('a,'b)elt and nil and ('n,'t)name
  val inputText : ('n,'t)name -> ('n->'a,'a)elt
  val inputSubmit : string -> ('n->'a,'a)elt
  val $ : string -> ('a,'a)elt
  val & : ('a,'b)elt * ('b,'c)elt -> ('a,'c)elt
end
```

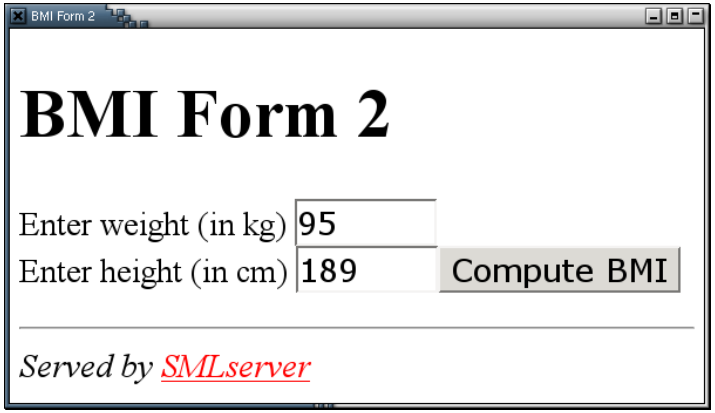
## Scriptlet Implementing Example Body Mass Index Form:

```
functor bmiform () : SCRIPTLET =
  struct open XHtml infix &
    val response = Page.page "BMI Form" (bmi.form
      (p( $"Enter height (in cm)" & inputText bmi.h & br()
        & $"Enter weight (in kg)" & inputText bmi.w
        & inputSubmit "Compute BMI")))
  end
```

# Type Indexed Functions for Input Control Reordering

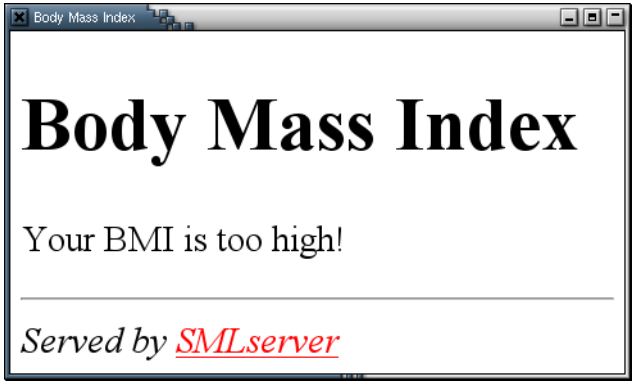
## Two forms can target the same scriptlet:




## Type Indexed Swapping:

```
type ('old,'new)idx
val One  : unit ->
  ('a->'b->'x,'b->'a->'x)idx
val Succ : ('a->'x,'b->'y)idx
  -> ('a->'c->'x,'b->'c->'y)idx
val swap : ('x,'xx)idx ->
  ('x,'y)elt -> ('xx,'y)elt
```



# Type Indexed Reordering Continued

## Alternative Body Mass Index Form:

```
functor bmiform2 () : SCRIPTLET =
  struct open XHtml infix &
    val response = Page.page "BMI Form" (bmi.form
      (swap (One())
      (p( $"Enter weight (in kg)" & inputText bmi.w & br()
        & $"Enter height (in cm)" & inputText bmi.h
        & inputSubmit "Compute BMI"))))
  end
```

- The expression "`swap (One())`" is given the type

```
(w->h->nil,nil)elt -> (h->w->nil,nil)elt
```

# Putting it all Together

- The XHTML conformity condition and the form consistency condition are orthogonal conditions, each expressible using separate type parameters in the `elt` type constructor.
- Thus, from the simple XHTML interfaces presented here, it is straightforward to construct the XHTML interface present in SMLserver.
- The `elt` type constructor in the SMLserver XHTML library has six type parameters.
- The approach extends to check boxes, radio buttons, drop-down menues, etc. — see companion technical report for details.

# Related Work

- The Haskell WASH/CGI library by Thiemann. PADL'02, JFP'02, WFLP'02.
- The JWIG/Bigwig projects at BRICS (Brabrand, Møller, and Schwartzbach).
- Other work on phantom types for embedded languages and type-indexed functions. Fluet & Pucella 2002; Leijen & Meijer 2000; Fridlender & Indrika 2000; McBride 2002; Yang 1998.
- Type safe language bindings for XML; e.g., XStatic, XDuce, XMLambda, UUXML.