Web Programming with SMLserver

Martin Elsman[⋆] mael@it.edu
Niels Hallenberg nh@it.edu

IT University of Copenhagen.
Glentevej 67, DK-2400 Copenhagen NV, Denmark

Abstract. SMLserver is an efficient multi-threaded Web server platform for Standard ML programs. It provides access to a variety of different Relational Database Management Systems (RDBMSs), including Oracle, MySQL, and PostgreSQL. We describe the execution model and the region-based memory model of SMLserver and explain our solutions to the design issues we were confronted with in the development. We also describe our experience with programming and maintaining Web applications using Standard ML, which provides higher-order functions, static typing, and a rich module system. Through experiments based on user scenarios for some common Web tasks, the paper demonstrates the efficiency of SMLserver, both with respect to script execution and database connectivity.

1 Introduction

Higher-order functions and a modules language for exposing the functionality of composable components are promising features for Web application development, where code reuse and separation of programming tasks (layout from implementation) are of primary concern.

The rapid change and development of Web applications combined with the way that Web applications are exposed to users also suggests that Web applications should be particularly robust to changes and easy to maintain. This observation is in contrast to how most Web applications are built—namely with scripting languages that have only limited support for finding errors in the program before it is exposed to users. A powerful static type system, on the other hand, enforces many programming errors to be found and fixed at compile time, although with the cost of an imposed compilation step in the development cycle.

SMLserver [7] is a Web server platform for Standard ML [14], a programming language which provides the features requested above, namely higher-order functions, a rich module system, and a powerful static type system. SMLserver builds on a bytecode backend and interpreter for the ML Kit [22], a compiler for the full Standard ML programming language. The interpreter, called the *Kit Abstract Machine* (KAM) [6], is embedded in a module for AOLserver, an open source Web server provided by America Online.[1] The KAM supports caching of

[⋆] Part time at Royal Veterinary and Agricultural University of Denmark.
[1] A port of SMLserver to the open source Web server Apache is ongoing.

المنارة للاستشارات
www.manaraa.com

loaded code, multi-threaded execution, and other features, including database interoperability.

The focus of this work is two-fold. We first demonstrate that programming Web applications with Standard ML provides many useful programming idioms, based on higher-order functions, static typing, and the rich Standard ML module system. Second, we present evidence that Web server support for high-level functional programming languages, such as Standard ML, can be as efficient as the use of highly tuned scripting languages, such as TCL and PHP.

1.1 Background

The ideas behind SMLserver came to mind in 1999 when the first author was attending a talk by Philip Greenspun, the author of the book "Philip and Alex's Guide to Web Publishing" [11]. Philip and his coworkers had been writing an astonishing 250,000 lines of dynamically typed TCL code to implement a community system that they planned to maintain, extend, and even customize for different Web sites. Although Philip and his coworkers were very successful with their community system, the dynamic typing of TCL makes such a large system difficult to maintain and extend, not to mention customize.

The SMLserver project was initiated at the end of 2000 by the construction of an embeddable runtime system and a bytecode backend for the ML Kit. Once the bytecode backend and the embeddable runtime system was in place, the KAM was embedded in an AOLserver module in such a way that requests for files with extension `.sml` and `.msp` (also called *scripts*) cause the corresponding compiled bytecode files to be loaded and executed. In April 2001, the basic system was running, but more work was necessary to support caching of loaded code, multi-threaded execution, and other features, such as database interoperability and a type safe caching interface. SMLserver is open source and distributed under the GNU General Public License (GPL).

1.2 Outline of the Paper

The paper proceeds as follows. In Sect. 2, we describe how SMLserver serves requests by loading and executing compiled scripts. In Sect. 3, we demonstrate the use of higher-order functions and type polymorphism for providing a type safe caching (i.e., memoization) interface for SMLserver Web scripts. In Sect. 4, we describe how SMLserver scripts may interface to an RDBMS through a generic interface, which makes extensive use of higher-order functions and type polymorphism for convenient access and manipulation of data in an RDBMS.

In Sect. 5, we describe how the region based memory model scales to a multi-threaded environment where programs run shortly, but are executed often. In Sect. 6, we demonstrate the efficiency of SMLserver, both with respect to script execution and database connectivity, by comparing the number of requests SMLserver may serve each second with numbers for other Web server platforms. We also measure the effect that some of the design decisions we were confronted

المنارة للاستشارات
www.manaraa.com

with in the development have on script execution time. Finally, we describe related and future work and conclude.

2 Serving Pages to Users

We shall now see how to create a small Web service for presenting the time-of-day to a user. The example uses the `Time.now` function from the Standard ML Basis Library to obtain the present time of day. HTML code to send to the user's browser is constructed using Standard ML string primitives:

```
val time_of_day = Date.fmt "%H.%M.%S" (Date.fromTimeLocal(Time.now()))
val _ = Ns.Conn.return
   "<html><head><title>Time of day</title></head> \
   \  <body bgcolor=white><h2>Time of day</h2> \
   \    The time of day is " ^ time_of_day ^ ".<hr></i> \
   \    Served by <a href=http://www.smlserver.org>SMLserver</a></i> \
   \</body></html>"
```

The result of a user requesting the file `time_of_day.sml` from the Web server is shown in Fig. 1. The script uses the function `Ns.Conn.return` to send an HTTP response with HTTP status code 200 (Page found) and MIME type `text/html` to the browser along with HTML code passed in the argument string.

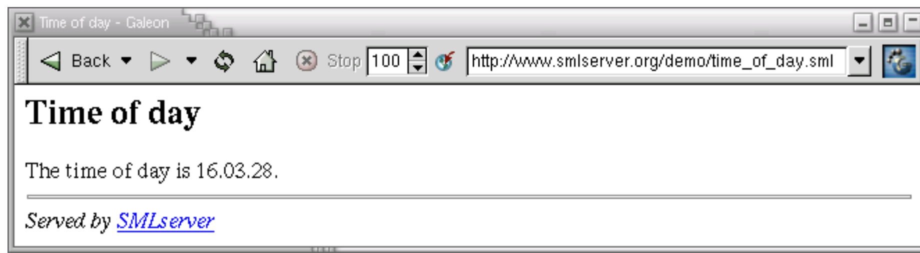

Fig. 1. The result of requesting the script `time_of_day.sml`.

In Sect. 2.2 we shall see how support for quotations may be used to embed HTML code in Web applications somewhat more elegantly than using Standard ML string literals. SMLserver also supports an alternative to quotations and strings in the form of an abstract combinator library for constructing HTML code. Although the use of the combinator library does not guarantee the validity of the generated HTML code, it may help eliminate certain types of errors at compile time. In addition, SMLserver has support for ML Server Pages, which provides a notation for embedding Standard ML code in HTML code, similar to PHP and Microsoft's Active Server Pages (ASP). ML Server Pages are stored in files with extension `.msp`.

www.manaraa.com

2.1 Loading and Serving Pages

SMLserver is implemented as a module `nssml.so`, which is loaded into the AOLserver Web server when the Web server starts. At this time, future requests for *scripts* (i.e., `.sml`-files and `.msp`-files) are served by interpreting the bytecode file that is the result of compiling the requested script. Compilation of scripts into bytecode files is done by the user explicitly invoking the SMLserver compiler `smlserverc`. The SMLserver compiler takes as argument a *project file*, which lists the scripts that a client may request along with Standard ML library code to be used by the scripts.

The first time a script is requested, SMLserver executes initialization code for each library file and caches the resulting initial heap, which can then be used for execution of the requested script and future requests. To serve a script, SMLserver first loads the requested script and caches the result (if it is not already in the cache), after which the script is executed. After execution, the heap is restored and made available for future requests.

Thus, SMLserver initiates execution in identical initial heaps each time a request is served, which means that it is not possible to maintain state implicitly in Web applications using Standard ML references or arrays. Instead, state must be maintained explicitly using a Relational Database Management System (RDBMS) or the cache primitives supported by SMLserver. Another possibility is to emulate state behavior by capturing state in form variables or cookies.

At first, this limitation may seem like a major drawback. However, the limitation has several important advantages:

- Good memory reuse. When a request has been served, memory used for serving the request may be reused for serving other requests.
- Support for a threaded execution model. Requests may be served simultaneously by interpreters running in different threads without the need for maintaining complex locks.
- Good scalability properties. For high volume Web sites, the serving of requests may be distributed to several different machines that communicate with a single database server. Serving many simultaneous requests from multiple clients is exactly what an RDBMS is good at.
- Good durability properties. Upon Web server and hardware failures, data stored in Web server memory is lost, whereas data stored in an RDBMS may be restored using the durability features of the RDBMS.

The limitation does not suggest that session support is impossible; sessions with timeout semantics can be encoded using SMLserver's caching features.

2.2 Quotations for HTML Embedding

Although SMLserver supports generation of HTML code through HTML combinators, it is sometimes more convenient to write HTML code directly. In this section we introduce the notion of *quotations* [19], an elegant extension to Standard ML, which eases readability and maintainability of embedded object language fragments (e.g., HTML code) within Standard ML programs. Although

المنارة للاستشارات
www.manaraa.com

quotations are not officially Standard ML [14], many compilers provide support for quotations, including Moscow ML, SML/NJ, and the ML Kit. Here is a small quotation example that demonstrates the basics of quotations:

```
val text = "fun"
val ulist : string frag list =
  `<ul><li>Web programming is ^text
   </ul>`
```

The program declares a variable `text` of type `string`, a variable `ulist` of type `string frag list`, and indirectly makes use of the constructors of this predeclared datatype:

```
datatype 'a frag = QUOTE of string | ANTIQUOTE of 'a
```

What happens is that the quotation bound to `ulist` evaluates to the list:

```
[QUOTE "<ul><li>Web programming is ", ANTIQUOTE "fun", QUOTE "\n</ul>"]
```

Using the `Quot.flatten` function, which has type `string frag list->string`, the value bound to `ulist` may be turned into a string (which can then be sent to a browser.)

To be precise, a quotation is a particular kind of expression that consists of a non-empty sequence of (possibly empty) fragments surrounded by back-quotes:

exp	::=	`` ` ``*frags*`` ` ``	quotation
frags	::=	*charseq*	character sequence
	|	*charseq* `^`*id frags*	anti-quotation id
	|	*charseq* `^(`*exp*`)` *frags*	anti-quotation exp

A *character sequence*, written *charseq*, is a possibly empty sequence of printable characters or spaces or tabs or newlines, with the exception that the characters `^` and `` ` `` must be escaped using the notation `^^` and ``^` ``, respectively.

A quotation evaluates to a value of type `ty frag list`, where `ty` is the type of all anti-quotation variables and anti-quotation expressions in the quotation. A character sequence fragment *charseq* evaluates to `QUOTE "`*charseq*`"`. An anti-quotation fragment `^`*id* or `^(`*exp*`)` evaluates to `ANTIQUOTE` *value*, where *value* is the value of the variable *id* or the expression *exp*, respectively.

To ease programming with quotations, the type constructor `quot` is declared at top-level as an abbreviation for the type `string frag list`. Moreover, the symbolic identifier `^^` is declared as an infix identifier with type `quot * quot -> quot` and associativity similar to `@`.

2.3 Obtaining Data from Users

The following example demonstrates the use of quotations for embedding HTML code and the use of the SMLserver Library structure `FormVar` for accessing and validating user input and so-called "hidden" form variables for emulating state in a Web application. The example that we present is a simple Web game, which, by use of the functionality in the structure `Random`, asks the user to guess a number between zero and 100:

المنارة للاستشارات
www.manaraa.com

```
fun returnPage title pic body = Ns.return
  `<html><head><title>^title</title></head>
    <body bgcolor=white> <center>
    <h2>^title</h2> <img src=^pic> <p>
     ^(Quot.toString body) <p> <i>Served by <a
        href=http://www.smlserver.org>SMLserver</a>
    </i> </center> </body>
   </html>`

fun mk_form (n:int) =
  `<form action=guess.sml method=post>
     <input type=hidden name=n value=^(Int.toString n)>
     <input type=text name=guess>
     <input type=submit value=Guess>
   </form>`

fun processGuess n =
  case FormVar.wrapOpt FormVar.getNat "guess"
    of NONE => returnPage "You must type a number - try again"
                          "bill_guess.jpg" (mk_form n)
     | SOME g => if g > n then
                    returnPage "Your guess is too big - try again"
                               "bill_large.jpg" (mk_form n)
                 else if g < n then
                    returnPage "Your guess is too small - try again"
                               "bill_small.jpg" (mk_form n)
                 else
                    returnPage "Congratulations!" "bill_yes.jpg"
                      `You guessed the number ^(Int.toString n) <p>
                       <a href=guess.sml>Play again?</a>`
val _ =
  case FormVar.wrapOpt FormVar.getNat "n"
    of NONE => let (* generate new random number *)
                   val n = Random.range(0,100) (Random.newgen())
               in returnPage "Guess a number between 0 and 100"
                             "bill_guess.jpg" (mk_form n)
               end
     | SOME n => processGuess n
```

The functions `returnPage` and `mk_form` use quotations for embedding HTML code. The function `Ns.return`, which takes a value of type `quot` as argument, returns the argument to the client.

The expression `FormVar.wrapOpt FormVar.getNat` results in a function of type `string -> int option`. The function takes the name of a form variable as argument and returns `SOME(`i`)`, where i is an integer obtained from the string value associated with the form variable. If the form variable does not occur in the query data, is not a well-formed natural number, or its value does not fit in 32 bits, the function returns `NONE`. The argument given to `FormVar.wrapOpt`, namely `FormVar.getNat`, is a function with type `string -> int` and the prop-

www.manaraa.com

erty that it raises an exception if its argument is not a proper natural number. The use of higher-order functions for form variable validation is necessary to obtain a shallow interface and gain a high degree of code reuse. In particular, the `FormVar` structure provides wrapper functions that make it possible to report multiple error messages to the user concerning invalid form content.

In the case that no form variable `n` exists, a new random number is generated and the game is started by presenting an introduction line to the player along with a form for entering the first guess. The game then proceeds by returning different pages to the user depending on whether the user's `guess` is greater than, smaller than, or equal to the random number `n`.

Notice that the game uses the HTTP request method `POST`, so that the random number that the user is to guess is not shown in the browser's location field. It is left as an exercise to the reader to find out how—with some help from the Web browser—it is possible to "guess" the number using only one guess. Figure 2 shows three different pages served by the "Guess a Number" game.

Fig. 2. Three different pages served by the "Guess a Number" game.

3 Caching Support

SMLserver has a simple type safe caching interface that can be used to cache data so that information computed by some script invocation can be used by subsequent script invocations. The cache functionality is implemented as a structure `Cache`, which matches the signature `CACHE` listed in Fig. 3.

A cache of type (α,β) `cache` maps keys of type α `Type` to values of type β `Type`. The cache interface defines a set of *base types* (e.g., `Int`, `Real` and `String`) and a set of type constructors to build new types (e.g., `Pair`, `List`, and

www.manaraa.com

```
signature CACHE =
  sig
    datatype kind = WhileUsed of int | TimeOut of int | Size of int
    type ('a,'b) cache
    type 'a Type
    type name = string

    val get      : 'a Type * 'b Type * name * kind -> ('a,'b) cache
    val memoize  : ('a,'b) cache -> ('a -> 'b) -> 'a -> 'b

    val Int      : int Type
    val Real     : real Type
    val String   : string Type
    val Pair     : 'a Type -> 'b Type -> ('a*'b) Type
    val Option   : 'a Type -> 'a option Type
    val List     : 'a Type -> 'a list Type
    ...
  end
```

Fig. 3. The cache interface.

`Option`). A cache has a *cache name*, which is represented by a Standard ML string. SMLserver supports three *kinds* of caches:

- Size caches. Entries in caches of kind `Size(`n`)` expire when there is not enough room for a new entry (maximum cache size is n bytes). Oldest entries expire first.
- Timeout caches. For caches of kind `TimeOut(`t`)`, an entry expires t seconds after it is inserted. This kind of cache guarantees that the cache is updated with freshly computed information, even if the cache is accessed constantly.
- Keep-while-used caches. An entry in a cache of kind `WhileUsed(`t`)` expires when it has not been accessed in t seconds. This kind of cache is useful for caching authentication information, such as passwords, so as to lower the pressure on the RDBMS.

The function `get` obtains a cache given a domain type, a range type, a cache name, and a kind. The first time `get` is called with a particular domain type, a particular range type, and a particular cache name, a new cache is constructed. Conceptually one can think of the function `get` as having the constrained (or bounded) polymorphic type [8]

$$\forall \alpha \leq \texttt{Type},\ \beta \leq \texttt{Type}\ .\ \texttt{name * kind -> } (\alpha, \beta)\ \texttt{cache}$$

where `Type` denotes the set of types supported by the cache interface. As an example, the following expression constructs a cache named `mycache`, which maps pairs of integers to lists of reals:

```
get (Pair Int Int, List Real, "mycache", Size (9*1024))
```

المنارة للاستشارات
www.manaraa.com

The function `memoize` adds caching functionality (i.e., memoization) to a function. Assuming that the function f has type `int -> string * real` and c is an appropriately typed cache, the expression `memoize` c f returns a new function f', which caches the results of evaluating the function f. Subsequent calls to f' with the same argument results in cached pairs of strings and reals, except when a result no longer lives in the cache, in which case f is evaluated again.

The cache interface also provides functions for flushing caches, adding entries, and deleting entries (not shown in the signature above).

We now present a currency exchange rate service that uses the function `memoize` to cache an exchange rate obtained from a foreign Web site. The Web service is implemented as a single file `exchange.sml`:

```
structure C = Cache
val c = C.get (C.String, C.Option C.Real, "currency", C.TimeOut 300)
val form = `<form method=post action=exchange.sml>
              <b>Dollar amount</b><br><input type=text name=a>
              <input type=submit value="Value in Danish Kroner">
            </form>`
fun fetchRate url =
  case Ns.fetchUrl url of
    NONE => NONE
  | SOME pg => let val pattern = RegExp.fromString
                     ".+USDDKK.+<td>([0-9]+).([0-9]+)</td>.+"
                in case RegExp.extract pattern pg
                     of SOME [r1,r2] => Real.fromString (r1^"."^r2)
                      | _ => NONE
                end
val fetch = C.memoize c fetchRate
val url = "http://se.finance.yahoo.com/m5?s=USD&t=DKK"
val body = case FormVar.wrapOpt FormVar.getReal "a"
             of NONE => form
              | SOME a =>
                case fetch url
                  of NONE  => `The service is currently not available`
                   | SOME rate =>
                      `^(Real.toString a) USD gives
                       ^(Real.fmt (StringCvt.FIX(SOME 2)) (a*rate)) DKK.
                       <p>` ^^ form
val _ = Page.return "Currency Exchange Service" body
```

The program creates a cache `c` that maps strings (base type `String`) to optional reals (constructed type `Option Real`). The cache kind `TimeOut` is used to limit the pressure on the foreign site and to make sure that the currency rate is updated every five minutes.

The exchange rate (American dollars to Danish kroner) is obtained by fetching a Web page using the function `Ns.fetchUrl`, which takes an URL as argument and returns the contents of the page as a string. Once the page is received,

المنارة للاستشارات
www.manaraa.com

support for regular expressions is used to extract the appropriate information (the currency exchange rate) from the Web page.

The function `Page.return` is used to return HTML code to the client; the function takes two arguments, a string denoting a title for the page and a body for the page in terms of a value of type `quot`.

4 Interfacing with an RDBMS

In this section we present an interface for connecting to an RDBMS from within Web scripts written with SMLserver. We shall not argue here that it is a good idea to use an RDBMS for keeping state on a Web server, but just mention that a true RDBMS provides data guarantees that are difficult to obtain using other means. RDBMS vendors have also solved the problem of serving simultaneous users, which make RDBMSs ideal for Web purposes.

The language used to communicate with the RDBMS is the standardized Structured Query Language (SQL). Although each RDBMS has its own extensions to the language, to some extent, it is possible with SMLserver to write Web services that are indifferent to the RDBMS of choice. SMLserver scripts may access and manipulate data in an RDBMS through the use of a structure that matches the `NS_DB` signature:

```
signature NS_DB =
  sig
    structure Handle : ...
    val dml          : quot -> unit
    val foldr        : ((string->string)*'a->'a)->'a->quot->'a
    val qq           : string -> string
    val qqq          : string -> string
    ...
  end
```

Because SMLserver supports the Oracle RDBMS, the PostgreSQL RDBMS, and MySQL, there are three structures in the `Ns` structure that matches the `NS_DB` signature, namely `Ns.DbOra`, `Ns.DbPg`, and `Ns.DbMySQL`. The example Web server project file includes a file `Db.sml`, which binds a top-level structure `Db` to the structure `Ns.DbPg`; thus, in what follows, we shall use the structure `Db` to access the PostgreSQL RDBMS.

A *database handle* identifies a connection to an RDBMS and a *pool* is a set of database handles. When the Web server is started, a configurable number of pools are created. At any time, a database handle is owned by at most one script. Moreover, the database handles owned by a script at any one time belong to different pools. The database functions request database handles from the initialized pools and release the database handles again in such a way that deadlocks are avoided; with the use of only one pool with two database handles, say, a simple form of deadlock would appear if two scripts executing simultaneously each had obtained a database handle from the pool and were both requesting a second database handle.

المنارة للاستشارات
www.manaraa.com

The NS_DB function dml with type quot->unit is used to execute SQL *data manipulation language* statements (i.e., insert and update statements) in the RDBMS. On error, the function raises the top-level exception Fail.

The function foldr, is used to access data in the database. A select statement is passed as an argument to the function. The function is similar to the Basis Library function List.foldr. An application foldr *f b sql* executes the SQL statement given by the quotation *sql* and folds over the result set, similarly to how List.foldr folds over a list. The function *f* is the function used in the folding with base *b*. The first argument to *f* is a function of type string->string that maps column names into values for the row. Because the number of database handles owned by a script at any one time is limited to the number of initialized pools, nesting of applications of database access functions (such as foldr) is limited by the number of initialized pools. On error, the function raises the top-level exception Fail and all involved database handles are released appropriately.

The function qq, which has type string->string, returns the argument string in which every occurrence of a quote (') is replaced with a double occurrence (''), which is how quotes are escaped in SQL string literals. The function qqq is similar to the qq function with the extra functionality that the result is encapsulated in quotes ('...').

We now show a tiny "Guest Book" example, which demonstrates the database interface. The example consists of the file guest.sml, which presents guest book entries and a form for entering new entries, and a file guest_add.sml, which processes a submitted guest book entry. The data model, which is the basis for the guest book service, consists of a simple SQL table:

```
create table guest (
  email    varchar(100),
  name     varchar(100),
  comment  varchar(2000)
);
```

The table guest contains the three columns email, name, and comment. A row in the table corresponds to a form entry submitted by a user; initially, the table contains no rows. The file guest.sml includes the following code:

```
val form = `<form method=post action=guest_add.sml><table>
              <tr><td valign=top colspan=3>New comment<br>
                    <textarea name=c cols=65 rows=3
                     wrap=virtual>Fill in...</textarea></tr>
              <tr><td>Name<br><input type=text size=25 name=n>
                  <td>Email<br><input type=text size=25 name=e>
                  <td><br><input type=submit value="Add">
            </tr></table></form>`

fun layoutRow (f,acc) =
  `<li> <i>^(f "comment")</i>
   -- <a href="mailto:^(f "email")">^(f "name")</a><p>` ^^ acc
```

المنارة للاستشارات
www.manaraa.com

```
val rows = Db.foldr layoutRow ``
  `select email,name,comment from guest order by name`

val _ = Page.return "Guest Book" (`<ul>` ^^ rows ^^ `</ul>` ^^ form)
```

The function `Db.foldr` is used to query the database for rows in the table; the function `layoutRow`, which has type `(string->string)*quot->quot` is used to format each row appropriately. The first argument passed to this function is a function, which returns the contents of the given column in the row. Notice also that quotations are used to embed SQL statements in the code. Figure 4 shows the result of requesting the file `guest.sml`. The file `guest_add.sml`, which we

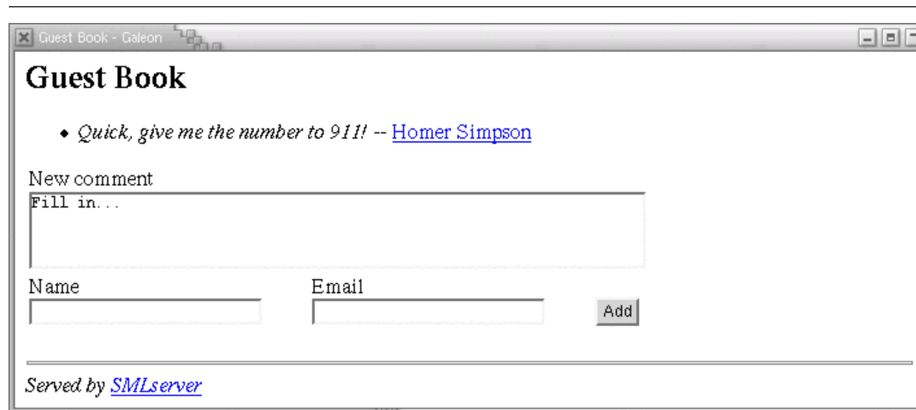

Fig. 4. The result of requesting the file `guest.sml`.

shall not list here, uses the `FormVar` functionality for extracting form variables and the function `Db.dml` to add an entry in the `guest` table.

For databases that support transactions, SMLserver supports transactions through the use of the `Handle` structure.

5 The Execution Model

Before we describe how SMLserver caches loaded bytecode to gain efficiency and how a multi-threaded execution model makes it possible for SMLserver to serve multiple requests simultaneously, we describe the region-based memory management scheme used by SMLserver.

5.1 Region-based Memory Management

The memory management system used in SMLserver is based on region inference [23], but extended appropriately to deal correctly with multi-threaded program

www.manaraa.com

execution. Region inference inserts allocation and deallocation directives in the program at compile time; no pointer-tracing garbage collection is used at runtime.

In the region memory model, the store consists of a stack of regions. Region inference turns all value producing expressions e in the program into e `at` ρ, where ρ is a region variable, which denotes a region in the store at runtime. Moreover, when e is an expression in the source program, region inference may turn e into the target expression `letregion` ρ `in` e' `end`, where e' is the target of analyzing sub-expressions in e and ρ is a region variable. At runtime, first an empty region is pushed on the stack and bound to ρ. Then, the sub-expression e' is evaluated, perhaps using ρ for allocation. Finally, upon reaching `end`, the region is deallocated from the stack. Safety of region inference guarantees that a region is not freed until after the last use of a value located in that region [23]. Functions in the target language can be declared to take regions as arguments and may thus, depending on the actual regions that are passed to the function, produce values in different regions for each call.

After region inference, the region-annotated program is compiled into bytecode for the KAM through a series of compilation phases [5, 1, 6]. Dynamically, a region is represented as a linked list of constant-sized region pages, which are chunks of memory allocated from the operating system. When a region is deallocated, region pages in the region are stored in a *free list*, also from which region pages are obtained when more memory is requested for allocation.

A consequence of region-based memory management is that no tags are needed at runtime to distinguish between different types of values, as are usually necessary for pointer tracing garbage collection.

For all the programs that we have developed using SMLserver, region inference has proven to recycle memory sufficiently without using a combination of region inference and garbage collection [12] or enforcing the programmer to write region-friendly programs.

5.2 Multi-Threaded Execution

SMLserver supports multi-threaded execution of scripts with a shared free list of region pages. The memory model allows two threads executing simultaneously to own the same region page at two different points in time. This property, which can potentially reduce the overall memory usage, is obtained by protecting the free list with mutual exclusion locks (i.e., mutex's).

SMLserver also maintains a mutex-protected *pool of initial heaps*, which makes it possible to eliminate the overhead of library initialization in the presence of multi-threaded execution. Before a script is executed, an initial heap is obtained from the pool. After execution, the heap is recovered before it is given back to the pool. For type safety, the process of recovering the pool involves restoring the initial heap to ensure that mutable data (e.g., references) in the initial heap are reinitialized.

For the Standard ML Library, approximately 18kb of region pages, containing mostly closures, are copied each time a heap is recovered. By storing mutable

المنارة للاستشارات
www.manaraa.com

data (i.e., references and arrays) in distinct regions, most of the copying can be avoided, which may further improve the efficiency of SMLserver.

6 Measurements

In this section, we measure the performance of SMLserver with respect to script execution time and compare it to a CGI-based ML Server Pages implementation, TCL on AOLserver, and PHP on Apache (Apache 1.3.22). We also measure the effect that caching of compiled scripts has on performance. Finally, we measure the overhead of interpreting initialization code for libraries for each request.

All measurements are performed on an 850Mhz Pentium 3 Linux box, equipped with 384Mb RAM. The program we use for benchmarking is ApacheBench, Version 1.3d.

The benchmark scripts include eight different scripts. The `hello` script returns a small constant HTML document. The `date` script uses a library function to show the current date. The script `db` connects to a database and executes a simple query. The script `guest` returns three guest list entries from the database. The script `calendar` returns 13 formatted calendar months. The script `mul` returns a simple multiplication table. The script `table` returns a 500 lines HTML table. The script `log` returns a 500 lines HTML table with database content.

The use of higher-order functions, such as `List.foldl` and `List.map`, in the MSP version of the `calendar` script are translated into explicit `while` loops in the TCL and PHP versions of the script.

Performance figures for SMLserver on the eight benchmark scripts are shown in the fourth column of Table 1. The column shows, for each benchmark, the number of requests that SMLserver serves each second when ApacheBench is instructed to have eight threads issue requests simultaneously for 60 seconds. Measurements for the Web server platforms MosML/MSP, AOLserver/TCL, and Apache/PHP are shown in the first three columns. There are two observations to point out:

1. For all scripts, SMLserver performs better than any of the other Web server platforms.
2. The MosML/MSP platform performs worse than any of the other three platforms on any of the benchmark scripts, most probably due to the CGI approach used by MosML/MSP.

The fifth column of Table 1 shows the efficiency of SMLserver with caching of script bytecode disabled (caching of library bytecode is still enabled). The measurements demonstrate that caching of script bytecode improves performance between 3 and 53 percent with an average of 37 percent.

The sixth column of Table 1 shows the efficiency of SMLserver with library execution enabled on all requests (library and script code is cached). Execution of library code on each request degrades performance between 10 and 74 percent with an average of 44 percent. The performance degrade is highest for less involved scripts. The four scripts `hello`, `date`, `db`, and `guest` use more time on library execution than executing the script itself.

المنارة للاستشارات
www.manaraa.com

Program	Requests / second					
	MosML MSP	AOLserver TCL	Apache PHP	SMLserver MSP	No script caching	With library execution
`hello`	55	724	489	**1326**	916	349
`date`	54	855	495	**1113**	744	337
`db`	27	558	331	**689**	516	275
`guest`	25	382	274	**543**	356	249
`calendar`	36	27	37	**101**	69	80
`mul`	50	185	214	**455**	300	241
`table`	21	59	0.7	**93**	84	75
`log`	8	12	0.4	**31**	30	28

Table 1. The first four columns compares script execution times for SMLserver with three other Web server platforms. Caching of loaded script bytecode improves performance between 3 and 53 percent (column five). Column six shows that execution of library code on each request degrades performance between 10 and 74 percent.

7 Related Work

Related work fall into several categories. First, there is much related work on improving the efficiency of CGI programs [15], in particular by embedding interpreters within Web servers [20], which may drastically decrease script initialization time. In particular, expensive script forking and script loading may be avoided and a *pool of database connections* can be maintained by the Web server, so that scripts need not establish individual connections to a database.

Second, there is a large body of related work on using functional languages for Web programming. Meijer's library for writing CGI scripts in Haskell [13] provides low-level functionality for accessing CGI parameters and sending responses to clients. Thiemann extends Meijer's work by providing a library WASH/CGI [21], which supports sessions and typing of forms and HTML using combinators. The `mod_haskell` project [4] takes the approach of embedding the Hugs Haskell interpreter as a module for the Apache Web server. Also, Peter Sestoft's ML Server Pages implementation for Moscow ML [17] provides good support for Web programming, although it is based on CGI and thus does not provide high efficiency (see Table 1).

Graunke et al. [10] demonstrate that programming a Web server infrastructure in a high-level functional language can be as efficient as utilizing an existing Web server infrastructure. Their work does not suggest, however, how multi-threaded execution of scripts can be supported in the context of server state. Using an existing Web server infrastructure, such as Apache or AOLserver, also has the advantage of pluggable modules for providing SSL (Secure Socket Layer) support and efficient pool-based database drivers for a variety of database systems.

Queinnec [16] suggests using continuations to implement the interaction between clients and Web servers. In a separate paper, Graunke et al. [9] demon-

المنارة للاستشارات
www.manaraa.com

strate how Web programs can be written in a traditional direct style and transformed into CGI scripts using CPS conversion and lambda lifting. In contrast to Queinnec, their approach uses the client for storing state information (i.e, continuation environments) between requests. It would be interesting to investigate if this approach works for statically typed languages, such as Standard ML.

Finally, `<bigwig>` [18, 2] provides a type system, which guarantees that Web applications return proper HTML to clients. To support typing of forms and sessions (to ensure type safety), `<bigwig>` programs are written in a special domain-specific language. Also, the session support provided by `<bigwig>` raises the question of when session state stored on the Web server should be garbage collected.

8 Future Directions

There are many directions for future work. One ongoing direction is the development of an SMLserver Community Suite (SCS), which already contains composable modules for user authentication, multi-lingual Web sites, and much more. SMLserver and SCS is used at the IT University of Copenhagen for running a course evaluation system and other administrative systems, which amounts to approximately 30,000 lines of Standard ML (excluding the Basis Library).

Not surprisingly, we have experienced that the static type system of Standard ML eases development and maintenance of Web applications. However, there are three aspects of Web application development with SMLserver where further work may give us better static guarantees:

1. Embedded HTML code is untyped. Data sent to a browser is not guaranteed to be valid HTML. Use of HTML combinators for constructing HTML code could increase faith in our code, but to completely ensure validity of HTML code requires dynamic tests for text embedded in HTML code.
2. Form variables are untyped. The correspondence between form variables expected by a script and the form variables provided by a request is not modeled by the Standard ML type system. A solution to this problem and the problem that HTML code is untyped has been proposed in the `<bigwig>` project [18, 2], but the solution builds on a new language tailored specifically to Web applications.
3. Embedded SQL queries are untyped. An extension to the Standard ML type system to support embedding of SQL queries has been proposed [3], but it requires a drastic departure from the Standard ML language with the addition of extensible records and variant types. Another possibility is to separate database queries from the program logic and have a tool generate type safe query functions from query specifications. In this way, queries that are invalid with respect to the underlying data model can be rejected at compile time.

www.manaraa.com

9 Conclusion

In this paper, we have presented SMLserver, a multi-threaded Web server platform for executing Web applications written in Standard ML. Making use of the advanced language features of Standard ML provides many advantages for Web programming:

- Higher-order functions combined with the rich module language of Standard ML, provide mechanisms to gain a high degree of code reuse and means for constructing shallow interfaces; examples include modules for form variable validation, database interaction, and data caching.
- The static type system of Standard ML provides very good maintenance properties, which is particularly important for Web programming where, often, program modifications are exposed to users early. Experience with writing large Web applications (+30,000 lines of code) with SMLserver demonstrates the importance of the maintenance properties and that SMLserver scales to the construction of large systems.

Measurements demonstrate that Web applications written with SMLserver perform better than Web applications written with often used scripting languages, both with respect to script execution time and database connectivity.

Finally, we have shown that the region-based memory model scales to a multi-threaded environment where programs run shortly but are executed often. More information about SMLserver is available from `http://www.smlserver.org`.

Acknowledgments

We would like to thank Lars Birkedal, Ken Friis Larsen, Peter Sestoft, and Mads Tofte for many fruitful discussions about this work.

References

1. Lars Birkedal, Mads Tofte, and Magnus Vejlstrup. From region inference to von Neumann machines via region representation inference. In *Proceedings of the 23rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 171–183. ACM Press, January 1996.
2. Claus Brabrand, Anders Møller, and Michael I. Schwartzbach. The `<Bigwig>` project. *ACM Transactions on Internet Technology*, 2(2), May 2002.
3. Peter Buneman and Atsushi Ohori. Polymorphism and type inference in database programming. *ACM Transactions on Database Systems*, 21(1):30–76, 1996.
4. Eelco Dolstra and Armijn Hemel. *mod_haskell*, January 2000. `http://losser.st-lab.cs.uu.nl/mod_haskell`.
5. Martin Elsman. Static interpretation of modules. In *Procedings of Fourth International Conference on Functional Programming (ICFP'99)*, pages 208–219. ACM Press, September 1999.
6. Martin Elsman and Niels Hallenberg. A region-based abstract machine for the ML Kit. Technical Report TR-2002-18, IT University of Copenhagen, August 2002.

المنارة للاستشارات

www.manaraa.com

7. Martin Elsman and Niels Hallenberg. *SMLserver—A Functional Approach to Web Publishing*. The IT University of Copenhagen, February 2002. (154 pages). Available via `http://www.smlserver.org`.
8. Matthew Fluet and Riccardo Pucella. Phantom types and subtyping. In *Second IFIP International Conference on Theoretical Computer Science (TCS'02)*, pages 448–460, August 2002.
9. Paul Graunke, Shriram Krishnamurthi, Robert Bruce Findler, and Matthias Felleisen. Automatically restructuring programs for the web. In *17th IEEE International Conference on Automated Software Engineering (ASE'01)*, September 2001.
10. Paul Graunke, Shriram Krishnamurti, Steve Van Der Hoeven, and Matthias Felleisen. Programming the web with high-level programming languages. In *European Symposium On Programming (ESOP'01)*, April 2001.
11. Philip Greenspun. *Philip and Alex's Guide to Web Publishing*. Morgan Kaufmann, May 1999. 596 pages. ISBN: 1558605347.
12. Niels Hallenberg, Martin Elsman, and Mads Tofte. Combining region inference and garbage collection. In *ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'02)*. ACM Press, June 2002. Berlin, Germany.
13. Erik Meijer. Server side Web scripting in Haskell. *Journal of Functional Programming*, 10(1):1–18, January 2000.
14. Robin Milner, Mads Tofte, Robert Harper, and David MacQueen. *The Definition of Standard ML (Revised)*. MIT Press, 1997.
15. Open Market, Inc. *FastCGI: A High-Performance Web Server Interface*, April 1996. Technical white paper. Available from `http://www.fastcgi.com`.
16. Christian Queinnec. The influence of browsers on evaluators or, continuations to program web servers. In *Fifth International Conference on Functional Programming (ICFP'00)*, September 2000.
17. Sergei Romanenko, Claudio Russo, and Peter Sestoft. *Moscow ML Owner's Manual*, June 2000. For version 2.00. 35 pages.
18. Anders Sandholm and Michael I. Schwartzbach. A type system for dynamic web documents. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL)*. ACM Press, January 2000.
19. Konrad Slind. Object language embedding in Standard ML of New Jersey. In *Proceedings of the Second ML Workshop, CMU SCS Technical Report*. Carnegie Mellon University, Pittsburgh, Pennsylvania, November 1991.
20. Lincoln Stein and Doug MacEachern. *Writing Apache Modules with Perl and C*. O'Reilly & Associates, April 1999. ISBN 1-56592-567-X.
21. Peter Thiemann. Wash/CGI: Server-side Web scripting with sessions and typed, compositional forms. In *Procedings of Practical Aspects of Declarative Languages (PADL'02)*. Springer-Verlag, January 2002. Portland, Oregon.
22. Mads Tofte, Lars Birkedal, Martin Elsman, Niels Hallenberg, Tommy Højfeld Olesen, and Peter Sestoft. Programming with regions in the ML Kit (for version 4). Technical Report TR-2001-07, IT University of Copenhagen, October 2001.
23. Mads Tofte and Jean-Pierre Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, 1997.

المنارة للاستشارات

www.manaraa.com