Ur/Web Programing Language: a brief overview

Article · June 2016

CITATIONS
0

READS
469

2 authors, including:

Asoke Nath
St. Xavier's College, Kolkata
267 PUBLICATIONS **1,889** CITATIONS

SEE PROFILE

International Journal of Innovative Research in Advanced Engineering (IJIRAE) ISSN: 2349-2763
Issue 05, Volume 3 (May 2016) www.ijirae.com

Ur/Web Programing Language: a brief overview

Sweta Chakraborti*
Department of Computer Science
St. Xavier's College(Autonomous)
30 Park Street, Kolkata, India

Asoke Nath
Department of Computer Science
St. Xavier's College(Autonomous)
30 Park Street, Kolkata, India

Abstract— This papert defines different components of Ur/Web programming language. Web programming has gradually evolved from a document delivery platform to architecture for distributed programming. Ur/Web, a domain-specific, statically typed functional programming language with a much simpler model for programming modern Web applications. Ur/Web's model is unified, where programs in a single programming language are compiled to other "Web standards" languages as needed; supports novel kinds of encapsulation of Web-specific state; and exposes simple concurrency, where programmers can reason about distributed, multithreaded applications.

Keywords— Web Porgramming Language;HTML-SQL; encapsulation; transactions;remote procedure calls;

I. INTRODUCTION

Ur is a programming language has introduced richer type system features into functional programming in the traditional form of ML and Haskell. Ur is functional, statically typed. Ur supports a meta-programming based on type-level computation with type-level records. Ur/Web is Ur plus a special standard library and associated rules for optimized result. Ur/Web supports construction of dynamic web applications backed by SQL databases. The signature of the standard library is such that well-typed Ur/Web programs is error free in a very broad sense. In process of particular page generations, it does not suffer from any kinds of code-mismatch, dead intra-application links and does not return to invalid HTML and SQL queries. It sets out in an orderly or improper manner in communication with SQL databases or between browsers and web servers and this considered to be basic foundation of the Ur/Web methodology. It is also possible to use meta-programming to build significant application pieces by analysis of type structure. The type system guarantees that the admin interface sub-application that remains be free of the bugs irrespective of well-typed table description input. These compiled programs will often be even more efficient than what most programmers would bother to write in C. The compiler also generates JavaScript versions of client-side code, with no need to write those parts of applications in a different language.[1]

II. NOVELTY AND GENERAL OVERVIEW OF UR/WEB

Ur/Web delivers a simple programming model retaining the essence of the Web as an application platform, from the standpoints of security and performance.

An application is a program in one language (Ur/Web) that runs on one server and many clients, with automatic compilation of parts of programs into the languages appropriate to the different nodes (e.g., JavaScript). Clients may often deviate arbitrarily from provided code to run while the server is under the programmer's control. For reasons of performance scaling or reliability, multiple physical machines might be used to implement server functionality. [2]

All objects passed between parts of the application are strongly typed. Applications may be written with no explicit marshaling or other conversion of data between formats. Where snippets of code appear as first-class values, they are presented as abstract syntax trees, ruling out flaws like code injection vulnerabilities that rely on surprising consequences of concatenating code-as-strings. The only persistent state in the server sits in an SQL database, accessed through a strongly typed SQL interface. The server exposes a set of typed functions that are callable remotely. Interaction begins with the application in a new browser tab by making a remote procedure call to one of these functions, with arbitrary correctly typed arguments. The server runs the associated function atomically, with no opportunity to observe interference by any other concurrent operations, generating an HTML page that the client displays. The HTML page in a client may contain traditional links to other pages, which are represented as suspended calls to other remotely callable functions, to be forced when a link is followed, to generate the new HTML page to display.

HTML page may also contain embedded Ur/Web code that runs in the client. Such code may produces as many client side threads at its convenience and the threads obey a cooperative multithreading semantics, where one thread runs at a time, and threads may be switched during well-defined blocking operations. Threads may modify the GUI shown to the user, via a functional-reactive programming system that mixes dataflow programming with imperative callbacks.[2][3]

Client-side thread code may also make blocking remote procedure calls treated similarly to those for regular links. Such a call may return a value of any function-free type, not just HTML pages; and the thread receiving the function result may compute with it to change the visible GUI programmatically, rather than by loading a completely new page. As before, every remote procedure call appears to execute atomically on the server.

Server code may allocate typed message-passing channels, which may be both stored in the database and returned to clients via remote function calls. The server may send values to a channel, and a client that has been passed the channel may receive those values asynchronously. Channel sends are included in the guarantee of atomicity for remote calls on the server; all sends within a single call execution appear to transfer their messages to the associated channels atomically[2][3].

The fundamental procedure and basic components are discussed here briefly

HTML AND SQL

Mainstream modern Web applications manipulate code in many different languages and protocols. Ur/Web hides most of them within a unified programming model. Two languages explicitly exposed i.e. HTML, for describing the structure of Web pages as trees, and SQL, for accessing a persistent relational database on the server. In contrast to mainstream practice, Ur/Web represents code fragments in these languages as first-class, strongly typed values. Type system of Ur to define rich syntax tree types, where the generic type system is sufficient to enforce the typing rules of the embedded languages, HTML and SQL.

Programmers may benefit from that style of more precise type-checking. On the other hand, more complex static checking of XML may be more difficult for programmers to understand. An additional benefit of Ur/Web's approach is that XML checking need not be built into the language but is instead encoded as a library using Ur's rich but general-purpose type system.

Here is an example of HTML and SQL respectively.[1]

```
funmain () = return <xml>
<head>
<title>Hello world!</title>
</head>

<body>
<h1>Hello world!</h1>
</body>
</xml>

Tablet : { A : int, B : float, C : string, D : bool }
PRIMARYKEYA

funlist () =
rows<- queryX (SELECT * FROM t)
        (fn row =><xml><tr>
<td>{[row.T.A]}</td><td>{[row.T.B]}</td><td>{[row.T.C]}</td><td>{[row.T.D]}</td>
<td><form><submit action={delete row.T.A} value="Delete"/></form></td>
</tr></xml>);
return<xml>
<table border=1>
<tr><th>A</th><th>B</th><th>C</th><th>D</th></tr>
{rows}
</table>

<br/><hr/><br/>
```

```
<form>
<table>
<tr><th>A:</th><td><textbox{#A}/></td></tr>
<tr><th>B:</th><td><textbox{#B}/></td></tr>
<tr><th>C:</th><td><textbox{#C}/></td></tr>
<tr><th>D:</th><td><checkbox{#D}/></td></tr>
<tr><th/><td><submit action={add} value="Add Row"/></td></tr>
</table>
</form>
</xml>

andadd r =
dml (INSERTINTO t (A, B, C, D)
VALUES ({[readErrorr.A]}, {[readErrorr.B]}, {[r.C]}, {[r.D]}));
xml<- list ();
return<xml><body>
<p>Row added.</p>

{xml}
</body></xml>

Anddelete a () =
dml (DELETEFROM t
WHEREt.A = {[a]});
xml<- list ();
return<xml><body>
<p>Row deleted.</p>

{xml}
</body></xml>
funmain () =
xml<- list ();
return<xml><body>
{xml}
</body></xml>
```

B. Addition of more Encapsulation

Lack of encapsulation in a large traditional application is not appreciated so functionality should be modularized, e.g. into classes implementing data structures. The general model is that the SQL database is a preexisting resource, and any part of the application may create an interface to any part of the database. Therefore it is analogize in such a scheme to an object-oriented language where all class fields are public; it forecloses on some very useful styles of modular reasoning.

C. Client-Side GUI Scripting

Extension takes advantage of client-side scripting to make applications more responsive, without the need to load a completely fresh page after every user action. Mainstream Web applications are scripted with JavaScript, but, as in Links and similar languages, Ur/Web scripting is done in the language itself, which is compiled to JavaScript as needed. Following is an example of client–side GUI.

```
sequenceseq
fun increment () = nextvalseq
fun main () =
src<- source 0;
return<xml><body>
<dyn signal={n <- signal src; return <xml>{[n]}</xml>}/>
<button value="Update" onclick={fn _ => n <- rpc (increment ()); set src n}/>
</body></xml>
```

D.Reactive GUIs

Ur/Web GUI programming follows the functional-reactive style. The visible page is described via dataflow, as a pure function over some primitive streams of values. Ur/Web adopts a less pure style, where the event callbacks of imperative programming is retained. These callbacks modify data sources, which are a special kind of mutable reference cells. The only primitive streams are effectively the sequences of values that data sources take on, where new entries are pushed into the streams mostly via imperative code in callbacks.

E. Remote Procedure Calls

To write an application that runs within a single page another way for this application is to contact the server, to trigger state modifications and receive updated information. Ur/Web's first solution to that problem is remote procedure calls (RPCs), allowing client code to run particular function calls as if they were executing on the server, with access to shared database state. Client code only needs to wrap such calls to remotely callable functions within the RPC keyword, and the Ur/Web implementation takes care of all network communication and marshaling. Following is an example of RPC.

```
structureRoom = Broadcast.Make(struct
typet = string
end)

sequence s
table t : { Id : int, Title : string, Room : Room.topic }
PRIMARYKEY Id

fun chat id () =
   r <- oneRow (SELECTt.Title, t.RoomFROM t WHEREt.Id = {[id]});
ch<- Room.subscriber.T.Room;

newLine<- source "";
buf<- Buffer.create;

let
funonload () =
let
fun listener () =
                s <- recvch;
Buffer.writebuf s;
listener ()
in
listener ()
end

fungetRoom () =
        r <- oneRow (SELECTt.RoomFROM t WHEREt.Id = {[id]});
returnr.T.Room

fun speak line =
room<- getRoom ();
Room.send room line

fundoSpeak () =
line<- get newLine;
setnewLine "";
rpc (speak line)
in
return<xml><bodyonload={onload ()}>
<h1>{[r.T.Title]}</h1>

<button value="Send:" onclick={fn _ =>doSpeak ()}/><ctextbox source={newLine}/>
```

```
<h2>Messages</h2>

<dyn signal={Buffer.renderbuf}/>

</body></xml>
end

fun list () =
queryX' (SELECT * FROM t)
   (fn r =>
count<- Room.subscribersr.T.Room;
return<xml><tr>
<td>{[r.T.Id]}</td>
<td>{[r.T.Title]}</td>
<td>{[count]}</td>
<td><form><submit action={chat r.T.Id} value="Enter"/></form></td>
<td><form><submit action={delete r.T.Id} value="Delete"/></form></td>
</tr></xml>)

and delete id () =
dml (DELETEFROM t WHERE Id = {[id]});
main ()

and main () =
let
fun create r =
id<- nextval s;
room<- Room.create;
dml (INSERTINTO t (Id, Title, Room) VALUES ({[id]}, {[r.Title]}, {[room]}));
main ()
in
ls<- list ();
return<xml><body>
<h1>Current Channels</h1>

<table>
<tr><th>ID</th><th>Title</th><th>#Subscribers</th></tr>
        {ls}
</table>

<h1>New Channel</h1>

<form>
        Title: <textbox{#Title}/><br/>
<submit action={create}/>
</form>
</body></xml>
end
```

F. Message-Passing from Server to Client

Web browsers make it natural for clients to contact servers via HTTP requests, but the other communication direction may also be useful. Real-world applications often use a technique called long polling, where a client opens a connection and is willing to wait an arbitrary period of time for the server's response. The server can hold all of these long-poll connections open until there is a new event to distribute. The mechanics are standardized in recent browsers with the WebSockets protocol, providing an abstraction of bidirectional streams between clients and servers. Ur/Web presents an alternative abstraction (implemented with long polling) where servers are able to send typed messages directly to clients.

III. CONCLUSION AND FUTURE SCOPE

A fundamental tension in the design of programming languages is between the convenience of high-level abstractions and the performance of low-level code. Optimizing compilers help bring the best of both worlds. It has been shown how an optimizing compiler can provide very good server-side performance for dynamic Web applications compiled from a very high-level functional language, Ur/Web, based on dependent type theory. Some of optimization techniques are domain-agnostic, as in compile-time elimination of higher-order features. Other crucial techniques are domain-specific, as in understanding generation of HTML pages, database queries, and their effect interactions. With all these features combined, the Ur/Web compiler produces servers are efficient and that outperform all of the most popular Web frameworks. Ur/Web has also been used successfully in deployed applications. The techniques suggested are simple enough to re-implement routinely for a variety of related domain-specific functional languages [3].

REFERENCES

[1] http://impredicative.com/ur/
[2] Chlipala, Adam. The Ur/Web Manual. 2016
[3] Chlipala, Adam. "Ur/Web: A Simple Model for programming the Web." The 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15), January 15-17, 2015, Mumbai, India.