XQuery in the Browser

Ghislain Fourny
Systems Group
ETH Zurich
gfourny@inf.ethz.ch

Donald Kossmann
Systems Group
ETH Zurich
donaldk@inf.ethz.ch

Tim Kraska
Systems Group
ETH Zurich
tim.kraska@inf.ethz.ch

Markus Pilman
Systems Group
ETH Zurich
mpilman@student.ethz.ch

Daniela Florescu
Oracle
USA
dana.florescu@oracle.com

ABSTRACT

Over the years, the browser has become a complete runtime environment for client-side programs. The main scripting language used towards this purpose is JavaScript, which was designed so as to program the browser. A lot of extensions and new layers have been built on top of it to allow e.g. DOM navigation and manipulation. However, JavaScript has become a victim of its own success and is used way beyond its possibilities, leading to increased code complexity. We suggest to reduce programming complexity by proposing XQuery as a client-side programming language. We wrote an extension for Microsoft Internet Explorer, based on the Zorba XQuery engine, which allows execution of XQuery scripts in the browser. An extension for Firefox is on the way as well. This paper demonstrates how client-side applications in XQuery look like and what they can do within a very small amount of code.

Categories and Subject Descriptors

D.3.3 [**Software**]: Programming languages - Language Constructs and Features; H.4.0 [**Information Systems**]: Information Systems Applications - General

General Terms

Design, Languages

Keywords

XML, XQuery, XQueryP, Browser, Script, JavaScript, DOM, Update, Programming, HTML, Event handling

1. INTRODUCTION

Today, the browser is no longer just a rendering engine: it is a complete programming platform. There are several different approaches to programming the browser, the most popular being JavaScript. Other approaches are Flash, Java applets, the Google Web Toolkit, etc. We follow the trend to use a server-side programming language for client programming, but rather than using Java, we propose to use the XQuery family, which has been tailored to navigate and manipulate XML, and hence is very well suited for HTML navigation and manipulation.

XQuery has been a recommendation since 2007 [1]. An extension is on the way which provides update features [3], and several working drafts have been proposed to give it some imperative features, e.g. XQueryP [2]. There is a big industry push for using XQuery for database querying, as well as business-logic programming. We would like to show that XQuery can also run on the client-side, and that it actually runs well at the client: it offers a higher level of abstraction to the programmer, a flexible and powerful declarative syntax. Significant actions can be performed on the HTML document with few lines of code. Furthermore, using XQuery as a client-side language solves some important issues. a) XQuery brings modularity, allowing bigger applications. b) It removes the impedance mismatch with XML navigation and manipulation because XML is handled natively. c) The code is portable between the browser, as well as between the client and the server. Nowadays there can be as many as five different programming languages (e.g., Java, JSP, HTML, SQL, JavaScript) in a single file on the server. With the XQuery family, there only remains a single language to do database querying, business logic, webpage server-sided generation and client-sided manipulation.

The XQuery standard only misses a few features to allow client-side programming and support full fledged AJAX application development. We made proposals (e.g. event-handling, asynchronous calls) to fill in this gap and make XQuery a full-featured client-side programming language [4].

The remainder of this paper is organized as follows. Section 2 gives an overview of the architecture. Section 3 gives an example showing how to manipulate the DOM and handle events. Section 4 gives an example of an asynchronous web service call. Section 5 gives an overview of our demonstration.
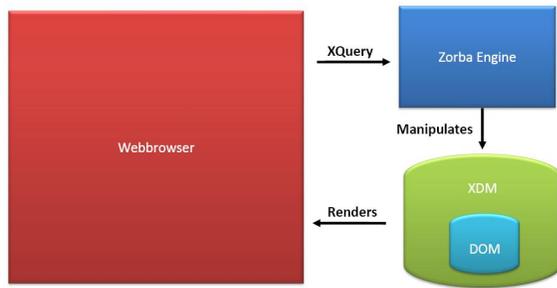
Figure 1: Interaction with the XDM and the DOM

2. THE ARCHITECTURE

We chose to build an extension for the Microsoft Internet Explorer. An implementation for Firefox is under development. Figure 1 shows the overall architecture. First, the browser receives and parses an XHTML document. It then generates the DOM, renders the webpage and initializes our extension. The XQuery engine receives the XQuery code. In our work, we used Zorba, which is developed in C++ by the FLWOR Foundation. It is open-source code under an Apache 2.0 license. The XQuery engine implements the XDM (XML Data Model) on top of the DOM: hence, Zorba modifies the XDM, which modifies the Microsoft Internet Explorer DOM as well.

We extended the XQuery syntax so as to be able to register listeners for events, to handle webservices calls, and to access the styles of HTML elements. These extensions are described in [4].

3. EXAMPLE 1: TABLES

The first example shows how the DOM can be navigated and manipulated, as well as how event-handling is performed with XQuery in the browser. The following HTML webpage, which includes an XQuery script, generates dynamically an addition, substraction, multiplication or division table.

```
<html><head>
  <title>Table Generation:
            Demonstration</title>
  <script type="text/xquery">
    declare function generateTable($event) {
      let $operation := $event/target//@value
      return
      replace node //table with
      <table>{
        for $i in 0 to
          //textbox[@name="nbrows"]/@value - 1
        return <tr>{
          for $j in 0 to
            //textbox[@name="nbcolumns"]
              /value - 1
          return <td>{
            if($operation="+") then $i+$j
            else if ($operation="-") then $i-$j
            else if ($operation="*") then $i*$j
            else $i/$j
          }</td>
        }</tr>
      }</table>
    };
```


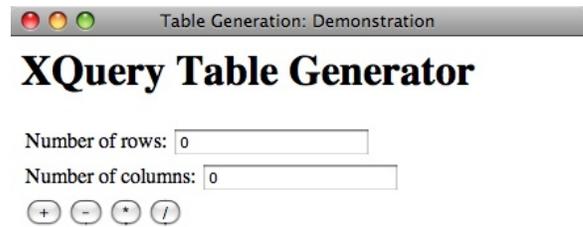Figure 2: Original Webpage of Example 1

```
  on event "onclick"
    at //input[@type="button"]
    attach listener generateTable
  </script>
</head><body>
  <h1>XQuery Table Generator</h1>
  <table>
    <tr><td>Number of rows:
      <input type="textbox"
            name="nbrows" value="0"/>
    </td></tr>
    <tr><td>Number of columns:
      <input type="textbox"
            name="nbcolumns" value="0"/>
    </td></tr>
    <tr><td>
      <input type="button" value="+"/>
      <input type="button" value="-"/>
      <input type="button" value="*"/>
      <input type="button" value="/"/>
    </td></tr>
  </table>
</body></html>
```

The body of the webpage originally contains two textboxes, in which the user can input the number of rows and columns he wishes for his output table, as well as four buttons (+, -, * and /), on which the user can click to build the table for the corresponding operation. The body is pure HTML. Figure 2 shows a preview of the original webpage.

The header contains a script tag which contains XQuery code, with the type property set to "text/xquery".

The XQuery code first defines a new function generateTable which is used as an event listener. Furthermore, the script registers this listener for onclick events. The listener takes an event node as argument. This event node is an XML node containing information about the event, for example if the user clicked on the "+" button:

```
<ev:event>
  <ev:target>
    <input type="button" value="+">
  </ev:target>
  ...
</ev:event>
```

The function first uses this event node to access the value of the button which was clicked (in this case, "+") and stores it in a variable $operation. Then the table which was originally in the body is replaced with a new table, dynamically.
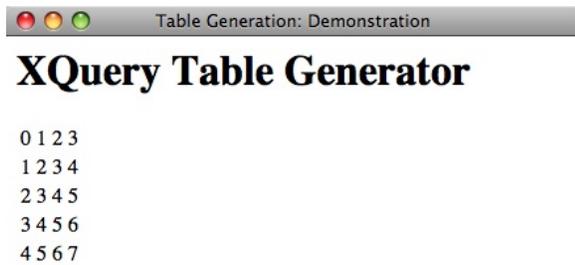
Figure 3: Result of Example 1, 5 Rows and 4 Columns, + Operation

First, the rows are generated. The number of rows is read in the textbox field. Then for each row, there is a similar iteration on the columns, which produces cells. For each cell, depending on $operation, the relevant operation is performed with the cell coordinates and written in the cell.

Finally, the function generateTable is bound to the event "onClick" occuring at any button in the HTML webpage. This is one of the syntax extensions we proposed for XQuery in the browser [4].

Figure 3 shows the resulting addition table with 5 rows and 4 columns.

4. EXAMPLE 2: AJAX WEATHER FORE-CAST

As in AJAX, XQuery in the browser also allows the programmer to asynchronously receive information from a server whilst the user is viewing the page. This is done with webservices, the details of which are hidden to the programmer, providing a good level of abstraction. The following HTML webpage, which includes an XQuery script, generates dynamically a weather forecast with data received from the server when the user presses the button.

```
<html><head>
  <title>Weather Forecast:
            Demonstration</title>
  <script type="text/xquery">
    import service namespace
      weather = "http://example.com"
      from "http://www.example.com/wsdl";

    declare function getWeather($event) {
      on event "stateChanged"
        behind weather:getWeather(
          //input[@name="city"]/@value
        )
        attach listener onWeatherResponse
    };

    declare function onWeatherResponse($event) {
      if($event/readyStatus=4) then
        let $weatherType :=
              $event/response//weathertype,
            $temperature :=
              $event/response//temperature
        return {
          if($temperature) then
```
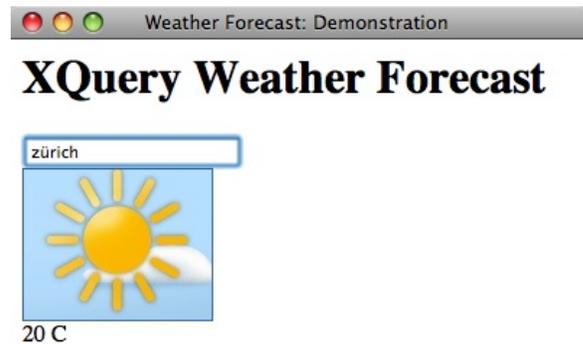

Figure 4: Result of example 2

```
        replace value of
          node //div[id="temperature"]
          with $temperature;
        if($weatherType)
        replace value of
          node //div[id="weather"]
          with <img src="{$weatherType}.jpg"/>
      };
    on event "onKeyUp"
      at //input[@type="textbox"]
      attach listener getWeather
  </script>
</head><body>
  <h1>XQuery Weather Forecast</h1>
  <input type="textbox" name="city"/>
  <div id="weather"/>
  <div id="temperature"/>
</body></html>
```

The HTML body contains a textbox in which the user can enter a city. Two placeholders are also here (weather and temperature) to display the information received from the server.

As in the previous example, the XQuery code is contained in a script tag. First, the webservice namespace weather is declared. A first function, getWeather, is called whenever the user presses a key. This function asynchronously performs a webservice call, and binds the function onWeather-Response with a status change of this call. This is also a syntax extension that we proposed in [4].

OnWeatherResponse is called when the results are back from the server. It extracts the weather type and the temperature from the response in a straightforward way (this is all XML, which XQuery handles natively) and then displays the temperature in the temperature field, as well as an image representing the weather in the weather field (we assume that the corresponding jpg files are in the same directory).

5. DEMO AND CONCLUSION

We will perform a live demonstration of the two examples with our implementation of the Internet Explorer extension and show that the XQuery family is a good candidate for client-side programming, because if handles XML natively and is a full-featured programming environment. It can potentially do whatever JavaScript can do, just with less code and at a higher level of abstraction.

6. REFERENCES

[1] S. Boag, D. Chamberlin, M. Fernandez, D. Florescu, J. Robbie, and J. Siméon. XQuery 1.0: An XML Query Language. `http://www.w3.org/TR/xquery/`, jan 2007.

[2] D. Chamberlin, M. Carey, D. Florescu, D. Kossmann, and J. Robbie. XQueryP: Programming with XQuery. In *XIME-P, Chicago*, 2006.

[3] D. Chamberlin, D. Florescu, and J. Robbie. XQuery Update Facility. `http://www.w3.org/TR/xqupdate/`.

[4] G. Fourny, D. Kossmann, T. Kraska, M. Pilman, and D. Florescu. XQuery in the Browser. Technical report, ETH Zurich, November 2007.