AOJS: Aspect-Oriented JavaScript Programming Framework for Web Development

Hironori Washizaki,Atsuto Kubo,Tomohiko Mizumachi,Kazuki Eguchi,Yoshiaki Fukazawa
Waseda University, 3-4-1, Okubo, Shinjuku-ku, Tokyo, Japan
washizaki@waseda.jp, {a.kubo, t.mizu, wirbelwind, fukazawa}@fuka.info.waseda.ac.jp

Nobukazu Yoshioka, Hideyuki Kanuka, Toshihiro Kodaka
National Institute of Infomatics, Hitachi, Ltd., Fujitsu Laboratories ltd.
nobukazu@nii.ac.jp, hideyuki.kanuka.dv@hitachi.com, tkodaka@jp.fujitsu.com

Nobuhide Sugimoto, Yoichi Nagai, Rieko Yamamoto
Toshiba Solutions Corporation, NEC Corporation, Fujitsu Laboratories ltd.
sugimoto.nobuhide@toshiba-sol.co.jp, y-nagai@bc.jp.nec.com, r.yamamoto@jp.fujitsu.com

ABSTRACT

JavaScript is a popular scripting language that is particularly useful for client-side programming together with HTML /XML on the Web. As JavaScript programs become more complex and large, separation of concerns at the implementation level is a significant challenge. Aspect orientation has been a well known concept to realize improved separation; however, existing mechanisms require modifications in the target modules for aspect weaving in JavaScript (i.e., not "complete" separation). In this paper, we propose an Aspect-Oriented JavaScript framework, named "AOJS", which realizes the complete separation of aspects and other core modules in JavaScript. AOJS can specify function executions, variable assignments and file initializations in JavaScript programs as the joinpoints of aspects. Moreover, AOJS guarantees the complete separation of aspects and core program modules by adopting a proxy-based architecture for aspect weaving. By utilizing these features, we confirmed that AOJS offers improved modifiability and extendability for JavaScript programming.

Categories and Subject Descriptors

D.2.10 [Software Engineering]: Design; D.3.3 [Programming Languages]: Language Constructs and Features

General Terms

Design, Languages

Keywords

Aspect-Oriented Programming, JavaScript, Web Development, Separation of Concerns, AOJS

1. INTRODUCTION

Web servers and corresponding application servers are the common infrastructures for business services, government/public services and community services. To enhance the usability of the service applications or realize the rich-client applications mainly on the web, the client-side scripts have been widely accepted because the scripts are processed on each client (usually web browser) independently from the corresponding web servers.

JavaScript (ECMAScript[1]) is a popular scripting language that is particularly useful for client-side programming together with HTML/XML on the Web. In JavaScript programming, there are many concerns that cannot be encapsulated and separated into independent modules due to the limitation of JavaScript's modularization mechanism. Such crosscutting concerns in JavaScript include both of conventional (language independent) concerns such as logging, and language specific ones such as Ajax (asynchronous JavaScript and XML)-based functions.

For example, when conducting a beta-test (informal acceptance test) of a typical web application with JavaScript program, it might be necessary to log all value changes of specific variables in script files used on multiple web pages, and send each log to remote sever at runtime by asynchronous communications because of variety of web client environments such as browsers and OSs.

Embedding remote-logging function codes into each location where variable assignments will take place is the traditional way for adding such logging to the original program. However since such additional code scatters on many locations and tangle with other concerns' codes, the maintainability of the program will decrease significantly.

To solve the scattering and tangling problems in JavaScript programming for the web client development, we propose an Aspect-Oriented JavaScript programming framework, named "AOJS", which realizes the complete separation of aspects and other core modules in JavaScript. Below, we first introduce several conventional frameworks and their problems. Next we explain our new framework and how it solves these problems. Finally, we report a result of experimental evaluations and draw a conclusion from the result.

2. PROBLEMS IN CONVENTIONAL FRAMEWORKS

To encapsulate and separate realizations of such crosscutting concerns into independent modules, there are several Aspect-Oriented Programming (AOP[2]) frameworks for JavaScript.

Conventional AOP frameworks for JavaScript can be divided into two types: (a) weaving JavaScript codes as HTML handlers to HTML elements that have specific id/name attributes (such as Aspectjs[3] and Cerny.js[4]), and (b) weaving JavaScript codes to JavaScript codes by using the JavaScript language mechanism (such as Dojo[5], Ext JS[6], Yahoo! UI library[7] and Google Ajaxpect[8]).

However there are several problems in conventional frameworks:

- P_1: Regarding all of conventional frameworks, it is necessary to modify the target program to include extended library and/or describe aspects in itself. It leads to incomplete separation between aspects (additional codes and rules for weaving) and the target programs.

- P_2: None of conventional frameworks can ensure the consistency between original programs and the ones with aspects weaved. In other words, conventional frameworks allow programmers to modify original JavaScript programs after the programmers weave aspects into the programs and deploy them on a Web server. Such possibility of incompatibleness leads to the maintainability and reliability problems.

- P_3: None of conventional frameworks can specify the location where variable assignments take place as joinpoints for weaving JavaScript codes.

3. AOJS: ASPECT-ORIENTED JAVASCRIPT PROGRAMMING FRAMEWORK

To solve the above-mentioned problems $P_1 \sim P_3$ we propose AOJS in below. AOJS is based on the proxy architecture for weaving aspects and the joinpoint model[9] for specifying program locations where the aspects will be weaved.

3.1 Proxy-based runtime weaving

AOJS realize the complete separation of aspects and target programs by the proxy-based runtime weaving. This design solves the problem P_2 in addition to P_1; AOJS always ensures the consistency between programs and the ones with aspects weaved by the proxy. Moreover the client does not have to consider AOP nor the weaving process when requesting web pages with JavaScript programs; it is only necessary to know the proxy's URL.

Figure 1 shows the architecture of AOJS. AOJS is realized as a server that communicates with web clients that request web pages via HTTP and web servers that store/provide original web pages and JavaScript programs. AOJS server consists of two parts: the reverse proxy for judging necessity of weaving and redirecting requests/outputs, and the weaver for weaving. We implemented the reverse proxy by using Perl because of its simplicity for implementing such proxy tool, and the weaver by using Java with JavaCC[10] for parsing and modifying JavaScript programs.

Runtime behavior of AOJS is as follows:

1. The client requests web pages with JavaScript programs to the server via HTTP.

2. The reverse proxy fetches the target web pages with JavaScript programs from the existing web server.

3. The reverse proxy passes the JavaScript programs to the weaver. The weaver weaves JavaScript code fragments (advices) into the appropriate locations (joinpoints) in original JavaScript programs, and returns the results to the reverse proxy. These advices and joinpoints are specified in the aspect file in the XML format.

4. The reverse proxy returns the web pages and the JavaScript programs with aspects to the client.

5. The client runs the JavaScript programs in the web pages.

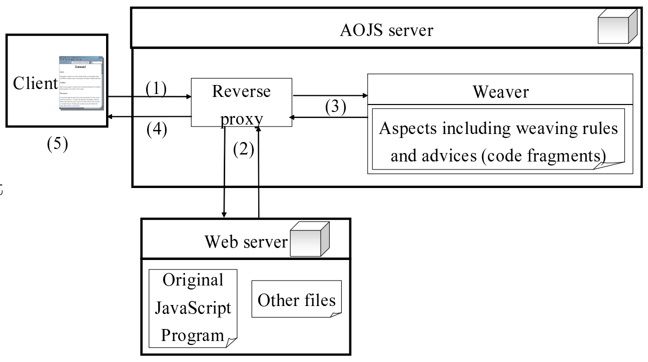

AOJS server
Client
(1)
(4)
(5)
Reverse proxy
(3)
Weaver
Aspects including weaving rules and advices (code fragments)
(2)
Web server
Original JavaScript Program
Other files

Figure 1: Architecture of AOJS

3.2 Practical pointcuts

To solve the remaining problem P_3, we design AOJS to deal with the following pointcuts including the variable assignment:

- Variable assignments: AOJS allows programmers to specify any variable assignment as a joinpoint by using the pointcut <var>.

  Figure 2 shows an example of the aspect file using the variable assignment pointcut targeting the Fibonacci program shown in Figure 3. The fourth line (<var varname="/fib_gen_1/ret">) in Figure 2 specifies all of assignment locations for the variable ret defined in the function fib_gen_1 (the fourth line in Figure 3). In AOJS, target variables can be specified with their scope hierarchies by using '/'. Since JavaScript is a dynamic and weakly typed language, we designed the variable assignment pointcut to specify target variables by variable names with the context hierarchy (NOT using types).

- Function executions: AOJS allows programmers to specify any function execution as a joinpoint by using the pointcut <function>.

  For example, the 16th line in Figure 2 specifies the execution location of the function fib_gen_2 (the execution raised by the 24th line in Figure 3).

```
<?xml version="1.0" ?>
<aspectsetting>
 <initializeFile>init.js</initializeFile>
 <var varname="/fib_gen_1/ret">
  <before><![CDATA[window.alert("/ret@before: " + ret
     + "<br />");]]></before>
  <after><![CDATA[window.alert("/ret@after: " + ret
     + "<br />");]]></after>
 </var>
 <var varname= "/y">
  <before><![CDATA[document.write("/y@before: " + y
     + "<br />");]]></before>
  <after><![CDATA[document.write("/y@after: " + y
     + "<br />");]]></after>
 </var>
 <function functionname = "/fib_gen_2">
 <before><![CDATA[var beg = (new Date()).getTime();]]>
  </before>
  <after><![CDATA[var end=(new Date()).getTime();
 sendLog( __retvalue__ + ", " + (end - beg) + "ms");]]>
  </after>
 </function>
</aspectsetting>
```

Figure 2: Aspect file in XML format

```
var x = 0; var y = 1;
function fib_gen_1() {
  var dummy, ret;
  ret = x + y; x = y;
  dummy = y = ret;
  return ret;
}
var z = 1;
function fib_gen_2() {
  function fib_2(x) {
    if(x <= 0) { return 1; }
    else if(x == 1) { return 1; }
    else { return fib_2(x-1)+fib_2(x-2); }
  }
  return fib_2(z++);
}
function fib_1() {
  var ret; ret = 100; y = 200;
  for(var i = 0; i < 30; ++i)
   document.form1.result.value = fib_gen_1();
}
function fib_2() {
  for(var i = 0; i < 30; ++i)
   document.form2.result.value = fib_gen_2();
}
```

Figure 3: Fibonacci calculation program in JavaScript (excerpt)

- Initializations: AOJS allows programmers to specify the beginning part of the target JavaScript file as a joinpoint by using the pointcut <initializeFile>.

3.3 Aspects

AOJS allows programmers to write the before-advices (specified by <before> in Figure 2) and the after-advices (<after>) for any variable assignment and any function execution. Moreover, it is possible to weave any additional codes into the location where the initialization pointcuts specify.

- The before-advice means the additional codes that will run just before its corresponding joinpoint. For example, the fifth line in Figure 2 specifies that the following statement ("window.alert(...)") will run before any variable assignment of the variable ret.

- Similarly, the after-advice means the additional codes that will run just after its corresponding joinpoint.

Finally, in AOJS, all of described pointcuts and advices are encapsulated into an aspect file in the form of XML, like shown in Figure 2.

3.4 Weaving process

AOJS's weaver uses a code template (shown in Figure 4) for code replacement. The template conducts the following four steps:

1. Executes <before>

2. Executes <target>, and stores the return value of the <target> expression into the temporal variable __retvalue__

3. Executes <after>

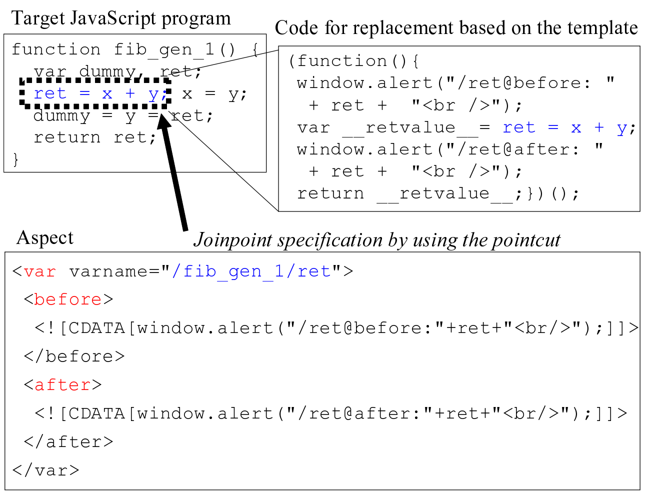
4. Returns the value stored in __retvalue__

By using the code template, the weaving process consists of the following two steps: Firstly, the weaver recognizes all of program locations where the described pointcuts specify by parsing the target JavaScript program files and aspect files. Secondly, for each location (joinpoint), the weaver generates the code for replacement based on the template, and replaces the original code to the code for replacement.

Figure 5 shows the example of weaving when a variable assignment has been specified as a joinpoint, and Figure 6 shows that when a function execution has been specified. In each figure, the code portion surrounded by the dashed line will be replaced by the code for replacement based on the template. These replacements add new behavior into the target program while keeping the original functionality.

```
(function(){
  <before>
  var __retvalue__= <target>;
  <after>
  return __retvalue__;
}) ();
```

Figure 4: Code template

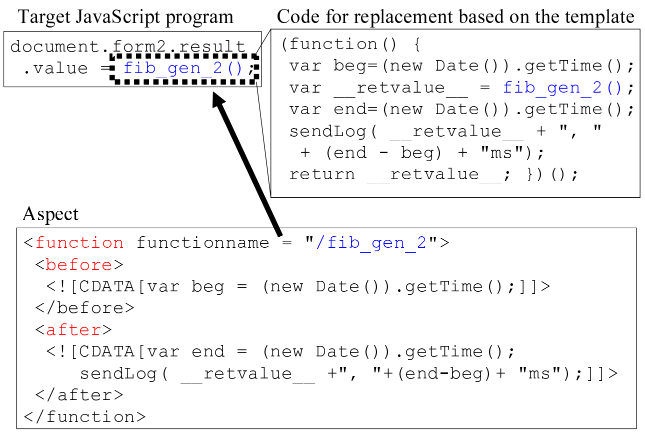

Target JavaScript program
Code for replacement based on the template
Aspect
Joinpoint specification by using the pointcut

```
function fib_gen_1() {
  var dummy, ret;
  ret = x + y; x = y;
  dummy = y = ret;
  return ret;
}
```

```
(function(){
 window.alert("/ret@before: "
  + ret +  "<br />");
 var __retvalue__= ret = x + y;
 window.alert("/ret@after: "
  + ret +  "<br />");
 return __retvalue__;})();
```

```
<var varname="/fib_gen_1/ret">
 <before>
  <![CDATA[window.alert("/ret@before:"+ret+"<br/>");]]>
 </before>
 <after>
  <![CDATA[window.alert("/ret@after:"+ret+"<br/>");]]>
 </after>
</var>
```

Figure 5: Weaving mechanism for variable assignment


Target JavaScript program
Code for replacement based on the template
Aspect

```
document.form2.result
 .value = fib_gen_2();
```

```
(function() {
 var beg=(new Date()).getTime();
 var __retvalue__ = fib_gen_2();
 var end=(new Date()).getTime();
 sendLog( __retvalue__ + ", "
  + (end - beg) + "ms");
 return __retvalue__; })();
```

```
<function functionname = "/fib_gen_2">
 <before>
  <![CDATA[var beg = (new Date()).getTime();]]>
 </before>
 <after>
  <![CDATA[var end = (new Date()).getTime();
     sendLog( __retvalue__ +", "+(end-beg)+ "ms");]]>
 </after>
</function>
```

Figure 6: Weaving mechanism for function execution

4. EXPERIMENTAL EVALUATIONS

We conducted some experimental evaluations regarding the functionality and runtime performance of AOJS, under the following environments:

- AOJS server: AMD Athlon 3500+, 4GB Memory, Debian/GNU Linux 4.0r3, Apache 2.2.3, Squid 2.6
- Client: Intel Core2Duo U7500, 2GB Memory, Ubuntu Desktop 8.04

4.1 Functionality

To confirm the functionality of AOJS, we conducted a simple experiment. We tried to add a remote logging functionality to the Fibonacci program (shown in Figure 3) without any modification on the original program.

This addition has been realized by only writing a small aspect shown in Figure 2, deploying the aspect on the AOJS server and accessing to the AOJS server (instead of the original web server). Figure 7 shows the time log file obtained remotely and asynchronously from a web client by accessing to the AOJS server.

From this experiment, we confirmed that AOJS enables the complete separation between aspects and target original JavaScript programs because the functionality addition has been realized without any modification on the target Fibonacci program. Moreover, we confirmed that AOJS has enough ability to specify joinpoints including variable assignments.

```
[Fri Jun 27 09:34:25 2008] [133.9.74.xx] 196418, 161ms
[Fri Jun 27 09:34:25 2008] [133.9.74.xx] 317811, 266ms
[Fri Jun 27 09:34:25 2008] [133.9.74.xx] 514229, 431ms
[Fri Jun 27 09:34:26 2008] [133.9.74.xx] 832040, 694ms
[Fri Jun 27 09:34:27 2008] [133.9.74.xx] 1346269, 1086ms
```

Figure 7: The time log file obtained by executing the Fibonacci calculation program with the logging aspect (excerpt)

4.2 Runtime performance and the revised architecture

From the viewpoint of the practicality, the runtime performance of the AOJS server is crucial because (a) good response time is often required for typical web applications, and (b) JavaScript programs are sometimes originally used for improving the usability of the target applications; the AOJS server should not decrease the runtime performance significantly.

To confirm the runtime performance of AOJS, we measured the throughput and response time when obtaining the web page containing the Fibonacci calculation program with/without aspects. Table 1 shows the obtained performance results when the Apache Benchmark Tool as the web client requested the same page totally 10^5 times.

In Table 1, we confirmed that the throughput and response time of the case where the aspect has been weaved decreased significantly compared with the case where no aspect has been weaved. It is because the weaving process will take place at each request time, and the process consumed 550ms for weaving the aspect to the Fibonacci program.

To solve this performance problem, we re-designed the architecture of AOJS (shown in Figure 8) by adding a cache proxy in front of the reverse proxy, and modifying the reverse proxy to notify the recent update of aspects/pages/programs to the cache proxy. We implemented the cache proxy by using Squid[11]. By using the cache proxy, the weaving process takes place only at the first request time while the target web pages, included JavaScript programs and aspects remain unchanged.

As shown in Table 1, the runtime performance of AOJS with the cache proxy improved significantly; the throughput and response time (median value) of the case where the aspect has been weaved with the usage of the cache proxy became almost the same values as those of the case where no aspect has been weaved. Therefore it is confirmed that AOJS with the cache proxy can be used practically from the viewpoint of the runtime performance. As our future work, we will try to integrate the weaver to the web server and compare its performance with the current proxy-based one.

5. RELATED WORK

Regarding JavaScript, there are several existing AOP frameworks as mentioned in section 2, such as Aspectjs and Cerny.js. However none of these frameworks can realize the complete

Table 1: Performance measurement results

Measurement	No aspects and no cache proxy	Aspect and no cashe proxy	Aspect and the cash proxy
Throughput [times/sec]	1232	1.77	1327
Response time [median; msec]	0	560	1

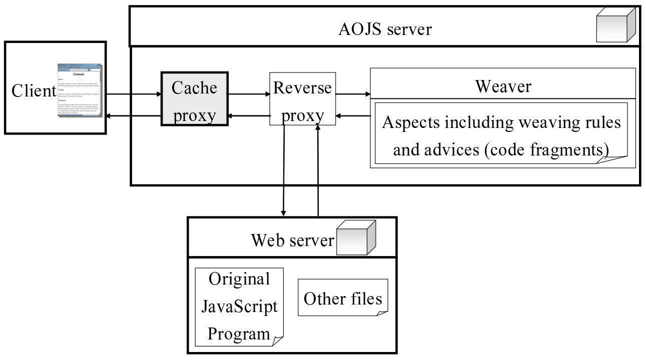

AOJS server
Client
Cache proxy
Reverse proxy
Weaver
Aspects including weaving rules and advices (code fragments)
Web server
Original JavaScript Program
Other files

Figure 8: Revised architecture with the cache proxy

separation, and specify the location where variable assignments take place as joinpoints.

AjaxScope is a proxy-based approach for performing on-the-fly instrumentation of JavaScript code[12]. It can rewrite any JavaScript abstract syntax tree nodes by new instrumentation code such as the performance profiling. Its fundamental mechanism is very similar to our framework. However it is not an AOP framework; it is unclear how easy to define a new aspect (called "policy" in AjaxScope) corresponding to programmers' concerns.

Regarding other languages for web development, Stamey et al proposed an AOP environment for PHP language, called AOPHP[13]. Although its mechanism of weaving aspect at the request time is similar to our framework, its language target and possible joinpoints are different from our framework.

6. CONCLUSION AND FUTURE WORK

We built a new AOP framework for JavaScript programming, called AOJS. AOJS are useful to modularize crosscutting concerns in JavaScript programs, such as logging and Ajax mechanisms. By adapting proxy-based architecture for aspect weaving, AOJS guarantees the complete separation of aspects and target programs; it leads to the consistency between programs and the ones with aspects weaved by AOJS. It is also easy to weave/remove aspects at runtime by only changing the URL for accessing. These features have not been achieved by any conventional framework/research in JavaScript.

As the result of experimental evaluations, we confirmed that AOJS enables the complete separation of aspects and target programs, and has enough ability to specify joinpoints including variable assignments. Moreover, it is possible to improve runtime performance by adding the cash proxy.

As our future works, we have a plan to extend AOJS to cover other joinpoints (such as function calls), and to deal with the around advice[9]. Moreover we will evaluate the usefulness of AOJS for larger-scale web-based applications.

Acknowledgement

This research is partially supported by Joint Forum for Strategic Software Research (SSR) of International Information Science Foundation (IISF) and GRACE Center of National Institute of Informatics (NII). Our thanks go to the anonymous reviewers for their valuable comments.

7. REFERENCES

[1] ISO/IEC 16262:2002, Information technology - ECMAScript language specification, 2002.
[2] Gregor Kiczales, John Lamping, Anurag Menhdhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier and John Irwin: Aspect-oriented programming, Proc. European Conference on Object-Oriented Programming (ECOOP), pp.220–242, 1997.
[3] LECACHEUR Sebastien: Aspectjs, http://zer0.free.fr/aspectjs/
[4] Robert Cerny: Cerny.js, http://www.cerny-online.com/cerny.js
[5] The Dojo Foundation: The Dojo Toolkit, http://dojotoolkit.org/
[6] Ext, LLC: Ext JS: Cross-Browser Rich Internet Application Framework, http://extjs.com/
[7] Yahoo! Inc.: The Yahoo! User Interface Library (YUI), http://developer.yahoo.com/yui/
[8] Google: Ajaxpect: Aspect-Oriented Programming for Ajax, http://code.google.com/p/ajaxpect/
[9] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm and William G. Griswold: An Overview of AspectJ, Proc. European Conference on Object-Oriented Programming (ECOOP), pp.327–353, 2001.
[10] Java Compiler Compiler (JavaCC) - The Java Parser Generator, https://javacc.dev.java.net/
[11] Squid: Optimising Web Delivery, http://www.squid-cache.org/
[12] Emre Kiciman and Benjamin Livshits: AjaxScope: a platform for remotely monitoring the client-side behavior of web 2.0 applications, Proc. 21st ACM SIGOPS Symposium on Operating Systems Principles, pp.17–30, 2007.
[13] John Stamey, Bryan Saunders and Simon Blanchard: The aspect-oriented web, Proc. 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information, pp.89–95, 2005.