DESIGNING A CLIENT SIDE DATA SCRAPPER

Valeriu IONESCU
University of Pitesti, Romania
valeriu.ionescu@upit.ro

Keywords: Java Applet, self-signing applet, client side processing, data scrapper.

Abstract: *Leveraging the server load for small business applications can be a crucial decision that can alleviate the implementation and development costs. This can be performed by passing some of the processing to the client. In this paper I will present considerations on the implementation of client sided applications in signed Java Applets and will present a sample implementation for data scrapping.*

1. INTRODUCTION

The internet is an immense source of information that can be extracted of specific information to create content that targets a specific area. In the present state of the development of the internet applications, web services and web applications are getting more attention then ever because they create a dynamic information space based on internet standards such as XAML for data encoding [1]

Most of the processing is done server side however clients have processing power that can be used to improve network traffic and server load.

While services do not offer the user a GUI the information provided can be very valuable so they limit the number of accesses to their services per hour, per day or per week for a user. For example Yahoo! Web Search Web Services are limited to 5000 queries per IP per day per API) [2] and Google Search is limited to 1000 searches per day (however Google has changed to limitless use if AJAX Search API is used). In a similar manner frequent calls to a server from a single location can be interpreted as an attack.

Of course there is the solution of using a web cache but this is only useful for information that doesn't change very often. For example implementing a plagiarism detection system based on search engine queries can be helped by this system because the data sources on the web do not change very often.

However, sites that update their content many times a day, that perform diverse searches or the ones that require constant monitoring of data from certain sources are not suitable for web caching.

2. DATA SCRAPPING

Data scrapping is the process of human readable data retrieval, parsing and re-display from a third party data source (that offers no convenient API), in a new form, in order to serve a new purpose (such as easy data interrogation). Data scrapping is a form of data mining.

Data scrapping can be used for selecting the information sources in many domains by sites that want to offer their users a consistent web browsing experience; therefore they try to integrate the external data as well as possible.

While data scrapping from offline software is easy to do, performing the same process from a web interface is problematic.

If users want to embed in their pages data that has a limited number of accesses per day/hour and that data will be accessed by a large number of users, they will be very fast in the impossibility to offer the service at all.

Also there are costs associated to the data traffic. If the embedded data is multimedia and the requests from the users will be sent on their behalf, then the traffic will be prohibitive to implement due to the costs involved. Therefore

the web administrator can offer the service if the users themselves access the off-server content.

There is also the question of user policy that these sites present and the users have to adhere to in order to use their services. In most cases these limit the use to non-commercial applications.

However the agreement does not always concern the use of the web data content itself but the use of a service that offers the information in an easy to use manner. In this case a web scrapper would not breach the term of use.

These are some of the problems and limitations that exist for the queries that are server-based.

3. ALTERNATIVES FOR DATA RETRIEVAL

One of the alternatives for data retrieval (fig. 1) would be to use a remote querying service like Yahoo Query Language (YQL) and use those servers to access to parse the remote sites for data. These return data resulting from the parsing of web sites formatted in XML or JSON (JavaScript Object Notation) format. This would reduce the traffic from our server because we can find the relevant data easier as a result of the queries, however these services themselves have usage limitations (in the case of YQL 100,000 calls per day per application and 1,000 calls per hour per IP without an application key).

There is also another problem: many sites offer hard linking protection and generate the links/content based on the user input or on the user connection data. Therefore even if the data received in JOSN form can be correctly used for the server that generated the query, any links passed to the client will be unusable because the connection data has changed.

The other alternative for data retrieval is the implementation of a query that is client generated instead of server generated. This approach has its advantages and disadvantages.

The first would be that for a server the generation of 1000 queries per hour is a normal thing, for a user however this is an extremely rare case (probably due to a virus).

The second advantage would be that the heavy traffic is now linked directly from the site we need the data from to the client, therefore it reduces the local costs because data is no longer passing though our servers to get to the client.

The limitations of this approach are also significant.

The server side data generation makes very little assumptions on the hardware the client is running, on the connection speed or the limitations of their internet connection. The server only needs a certain technology to be available on the client and that a browser capable of supporting the service is used.

The client approach on the other hand needs the server to make many such assumptions, and some of them may not be accurate.

One of these assumptions is that the function executed remotely (on the client) has enough security rights to be executed correctly. This may not be correct as there are strict limits on accessing local user resources such as hard-disks and there are also restrictions on the generation of cross-server queries. These are necessary for protecting the client and the client has full control over them irrelevant on the server execution necessities.

There are many ways to implement client side logic:

-JavaScript (very spread, however it interpreted differently from browser to browser),

-Silverlight (less used but easy to program),

-Java Applet (very used and much faster then Javascript)

-Flash (very spread but the programming support arrived only recently), however only some are able to pass the requirements for the intended purpose.

Speed may also be an issue because we do not know what kind of processing power the user has. Therefore the fastest should be chosen. A test of compared implementations (for the computation for prime numbers below 500,000) showed that between the common methods: Javascript, Silverlight, Java Applet and Flash, the applet offers the greatest speed [3, 4].

Other advantages include:

- that they are supported by most browsers with little compatibility problems because of the politic of Java program compatibility between different platform versions;

- they can be cached (therefore re-accessing the same page leads to faster load times);

a signed applet can access unrestrictedly the client data on the local machine and can make web requests on behalf of the user.

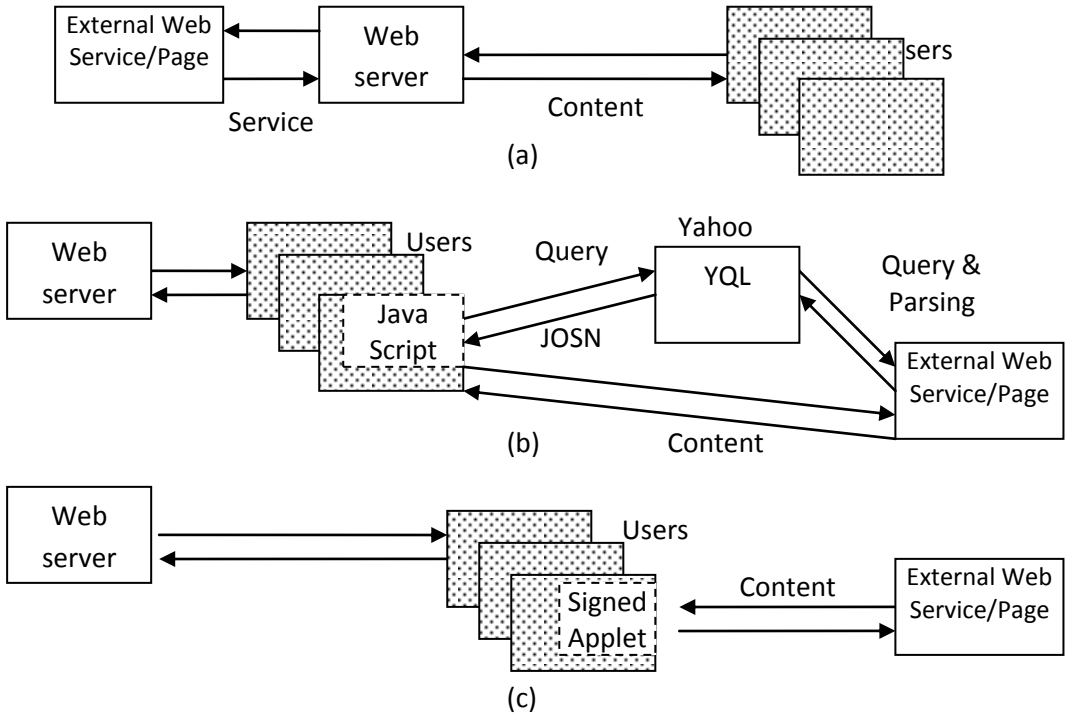

Fig. 1 Three methods for data retrieval: (a) classic method server sided (b) using the YQL querying language (c) using signed applets

4. THE APPLET IMPLEMENTATION

Applet security is a major issue that needs to be taken into account. There are two types of applets: signed and unsigned. An unsigned Applet has strict security restrictions and can not create any back connection or no other socket connections (like URLConnection). For this type of connections a signed applet is necessary.

The applets that offer the cross-server access capability are the signed ones. They can be signed with a security certify that can be verified by an independent authority or can be signed by the developers (without an independent confirmation). In the latter case the user will be warned accordingly (fig. 2).

When the user approves the execution of a signed applet it gets the rights to be executed as a normal local application, allowing any calls to remote servers.

For the development of this application I have used Java Netbeans 7.0. The implementation makes a javascript call the function from the Applet only when the user has confirmed the execution of the Applet; otherwise a warning page can be displayed.

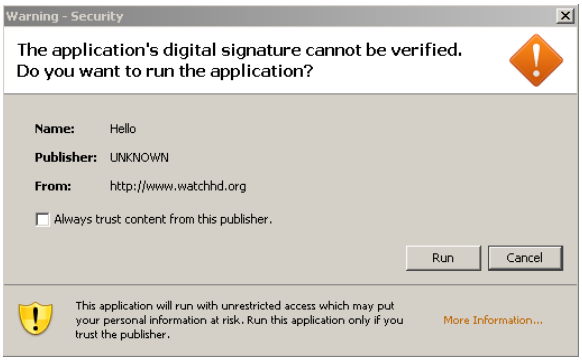

Fig. 2 Certificate warning for the self-signed Applet

The Javascript communication with the applet is very simple and consists of simply calling the function name. The following example presents a Javascript which shows a pop-up that calls the function *fctLink* from the *Hello* applet:

```
alert("Link\r\n:" +document.Hello.fctLink());
```

Parameters can be sent to be used by the applet using: <PARAM name=*Name* value=*Value*>.

In order to sign an applet there are few basic tools that must be used. The *keytool* application (found in %JAVA_HOME%/bin/) is provided within Java Runtime Environment for

creating self signing certificates. The parameters of *keytool* are:

-*keystore* is the name/full-path of the certificate file that we want to create.

-*keyalg* is the algorithm used for generating the certificate. The options are "DSA" with the default signature algorithm is "SHA1withDSA", and "RSA" with the default signature algorithm is "MD5withRSA". For this implementation I have used RSA. [5]

-*genkey* generates a new keystore.

-*alias* is the keystore entry and is case insensitive.

If the certificate is verified by a certificate authority server (VeriSign is one of them but there are many alternatives) the resulted *cer* file can be imported with the same *keytool*.

After the keystore is created the *jarsigner* tool will be used to sign the Java Archive (JAR) (fig. 3).

The applet implementation uses `HttpURLConnection` to connect to a web page, `getHeaderField` to read the header in order to monitor if it is the right page and `InputStreamReader` in order to read the reminder of the page and parse it accordingly.

Field	Value
Version	V3
Serial Number	[1300565848]
Signature Algorithm	[SHA1withRSA]
Issuer	CN=valeriu, OU=upit, O=upit, L=pit, ST=a...
Validity	[From: Sat Mar 19 22:17:28 EET 2011, To: ...
Subject	CN=valeriu, OU=upit, O=upit, L=pit, ST=a...
Signature	0000: 54 35 CE 87 16 47 4B 8D 34 EE 2F 9...
MD5 Fingerprint	FB:2E:8D:CB:E3:67:3E:DD:C5:EE:55:FB:6A...
SHA1 Fingerprint	E7:CD:28:F9:49:19:BD:8C:B6:6A:DC:A9:D...

Fig. 3 Details of the created self-signed JAR

5. RESULTS

The page was implemented on a web site that provides external multimedia links. the site has a traffic of approximately 10000 users/month and was tested for several weeks. The external content was tested on both hard-link protected sites and sites with no protection.

The hard-link protection was in most cases the introduction of a random generated string in the file link based on the user connection data. Because the request was generated on behalf of the user, this had no effect on the final result and the applet execution. The other method of hard-link protection was the generation of CAPTCHA images. Many of the generated images, however, had no letter distortion and had only background modifications that would prove easy to read if dedicated cod is implemented (fig. 4).

Fig. 4 CAPTCHA easy to recognize in OCR software

The result was that more than 70% of the people visiting the site executed the applet, despite being self-signed and presenting the security warning. The percent can be improved if third-party certificate authority server is used.

6. CONCLUSIONS

The application developed and tested was a self-signed applet used for client side data scrapping. The applet was successfully executed by most users that had accessed the site proving that the security warning was no deterrent. There was a variety of browsers used for data access. This also show that if the site has a user base that trusts it can execute any external service/web interrogation and complex parsing, while still maintaining the web server load to a manageable level. The development of this kind of processing architecture would also have the advantage that the development process

7. REFERENCES

[1] Tim Berners-Lee, “Web Services”, 2009/08/27, World Wide Web Consortium Accessed:26.05.2011,http://www.w3.org/DesignIssues/WebServices.html

[2] Yahoo! Inc., “Rate Limiting for Yahoo! Search Web Services”, Accessed: 26.05.2011, http://developer.yahoo.com/search/rate.html

[3] Roozz.com, “Roozz Technology Comparison”, Accessed: 26.05.2011 http://www.roozz.com/node/4

[4] Oracle Corporation, “J2SE 5.0 Performance White Paper“, Accessed: 26.05.2011, http://java.sun.com/performance/reference/whitepapers/5.0_performance.html

[5] Oracle Corporation, “keytool - Key and Certificate Management Tool” http://download.oracle.com/javase/6/docs/technotes/tools/windows/keytool.htm