HTML5 Web Workers

T-111.5502 Seminar on Media Technology B P

Final Report

Hannu Järvinen
Department of Media Technology
Aalto University, P.O. Box 15400, 00076 Aalto
hannu.jarvinen@aalto.fi

Abstract

HTML5 is currently a working draft at the W3C. It defines numerous new functionalities and thus offers interesting possibilities for the web developers to create rich web applications. One of the new features in HTML5 is the support for Web workers. They can be used to implement thread kind of behaviour in web browser tabs and windows. When different code sections can be executed simultaneously and are able to communicate with each other, both easier and more flexible web development is possible and new innovative applications can be created.

Introduction

The development of HTML has not been straightforward. The first version of HTML was created in 1990 and the current version, HTML 4.0, was standardized by the World Wide Web Consortium (W3C) in 1997. However, this language was not based directly on eXtensible Markup Language (XML), but on more flexible Standard Generalized Markup Language (SGML). W3C started to work on the well-formed, XML based version of the language and came up with the XHTML in 2000. The work was continued by developing the following XHTML 2.0 language.

When W3C was working on the XHTML 2.0, the Web Hypertext Application Technology Working Group (WHATWG) began to develop the new SGML based version of the HTML in 2004. Finally, in 2009 W3C discontinued the work on XHTML 2.0 and joined the WHATWG on
the effort of creating the HTML5. Now, the organizations are working together towards developing the standard.

In addition to the markup, HTML5 describes a number of Application Programming Interfaces (APIs) and many of the new functionalities are to be used with JavaScript. Thus, it actually seems, that part of the new standard is implemented as JavaScript libraries integrated in browsers and these functionalities can be used only with JavaScript. This can be interpreted as the final acceptance of the JavaScript as an essential part of the web programming by the standardization organizations.

One of the new features in HTML5 is the support for Web workers [1]. They can be used to create thread kind of behaviour in web browser tabs and windows. This offers a way for different sections of the code to be executed simultaneously, and a way for these sections to communicate with each other. This report gives a short introduction to the Web workers, describes how they can be used, presents some existing implementations, and envisions possible future applications.

**Threading in browsers**

While the web browser itself can be implemented using one or more threads, that does not directly affect the behaviour that the user experiences while browsing the web. Another issue is if new threads are assigned to different tabs or windows that the browser is executing. Usually the situation where user perceives the difference is when one of the tabs or windows gets stuck with executing a heavy-weight script. If threading is not well implemented, one tab or window can cause the whole browser to freeze. In some cases, the browser is smart enough to ask the user if she wants to interrupt the execution of such non-responding script.

When considering an individual web page, one tab or a window, it can also be implemented using one or more threads by the browser. However, while browser can use multiple threads for executing it, web programmer can not access the underlying thread functions. Instead, the programmers have been forced to manage their application with one thread and to overcome this weakness by utilizing the eventing functionalities. When evaluating the usability of JavaScript as a real programming language, Mikkonen and Taivalsaari noted that event-oriented programming actually works surprisingly well [2]. However, others report that using eventing in large-scale can cause control-flow difficulties and unexpected behaviour [3].

As a solution for the lack of threads in web programming, HTML5 presents the Web workers. However, Web workers differ from the traditional threads used in programming languages. For example, while in programming languages it is possible to spawn a thread to execute a certain function, in HTML5, the code executed by the Web worker has to be defined in a separate file. Also, the communication between the main thread and the worker is done in a special way called web messaging. The worker is also unable to access the Document Object Model (DOM) tree and the execution is done in its own sandbox. Both web messaging and Web workers are described in more detail in the following sections.

While Web workers are not exactly like threads in traditional programming languages, for the rest of this report the words Web worker, worker and thread are utilized equally when referring to Web workers as defined in HTML5.
Messaging API

To understand the functioning of the Web workers, one should first be familiar with the Messaging API, also known as Web messaging in HTML5. Different parts of the HTML5 seem to nicely support each other, and similarly the Messaging API offers the basis for both Web workers and Web sockets.

For security and privacy reasons, web browsers have earlier prevented documents from different domains from affecting each other. Now, the Messaging API is supposed to offer cross-document messaging in a secure way. With HTML5, the communication between the documents from different domains is thus possible.

Messaging API offers the functionality for communication between different sections of the Javascript code, and also between different documents. For the Web workers, it means having two-way communication for the spawned threads with the main thread or with each other. In practice, the Messaging API offers a postMessage method, which can be used to send plain text messages. Of course this limits the communication, as only text can be sent, but JSON.stringify() function luckily helps to send more complicated structures as well. A code example of how the Messaging API is used is given in the next section.

Web Workers

As described earlier, Web workers differ from the traditional threads in a couple of ways. Example JavaScript code listings are given here to describe the basic functionality. First, to create a new Web worker, a file name of the corresponding JavaScript file is given as an attribute in the constructor (see Listing 1). A new Web worker is created, which starts to execute the JavaScript in the worker.js file. The worker is executing in its own sandbox and cannot access the DOM tree. Example code listings of creating and communicating with a simple Web worker, called dedicated worker, are given below.

```javascript
var worker = new Worker('worker.js');
Listing 1: Creating a new Web worker
```

Listing 2 describes how to send a message to the worker. A postMessage method is offered by the Messaging API.

```javascript
worker.postMessage("Hello Worker!");
Listing 2: Messages can be sent to the worker using postMessage method defined by the Messaging API.
```

In a similar way, Listing 3 describes a way for the main thread (one that created the worker) to receive messages from the worker using the onmessage method.

```javascript
worker.onmessage = function (event) {
    alert("Message from the worker: " + event.data);
};
Listing 3: Messages can be received from the worker using the onmessage method.
```
The worker (worker.js) can also receive and sent back messages to the main thread. Listing 4 describes how to send back messages and Listing 5 how to receive them. The functionality is exactly the same, but there is no need to reference the target.

```javascript
postMessage("Hello main thread!");
Listing 4: Web worker can simply call postMessage method to send messages back to the main thread.
```

```javascript
onmessage = function (event) {
    alert("Message from the main thread: " + event.data);
};
Listing 5: Web worker can use onmessage method to receive messages from the main thread.
```

In addition to the dedicated Web workers, also Shared Web workers are described in HTML5. Shared workers enable the communication for a worker with multiple documents. In practice, this is implemented using ports. Also, when shared workers are created giving a same file name as an attribute, all these workers refer to the same instance. Usage of shared workers is very similar to dedicated workers. In basic situations the only difference is the additional port reference is function calls. Actually, dedicated workers are also shared workers with port functionality hidden behind the scenes. [4]

**Implementations**

Before HTML5 offered Web workers, Maki and Iwasaki [3] were not satisfied with the situation where Ajax programmers needed to write complex asynchronous programs on a single thread. The programs could be implemented, but only with events, which causes control-flow difficulties. They presented a library for common browsers, that supports multithreaded style.

In practice, their multithreaded style is converted into many “normal” style blocks, which are then run using thread-scheduler (a trampolined style program). This naturally causes some overhead, but not as much as communication delay of Ajax applications.

One application using Web workers is Ravan [5], a JavaScript based distributed computing system that can be used for cracking. It utilizes the computing power of several distributed browsers to perform brute force attacks on salted hashes. The main idea and the difference to earlier solutions is the ease of use, as one can simply submit a hash to be cracked or offer their computing power by clicking on a web page.

A project by Ryza and Wall considered a JavaScript based MapReduce Framework for Web Browsers (MRJS). In their implementation, the idea is simply that the main JavaScript thread starts workers to execute distributed computing on the background. Workers independently communicate over HTTP to get their tasks [6].

Another project by Okamoto and Kohana [7] studied using Web workers for load distribution in a real-time web application. They considered a Web based Multiplayer Online Role-Playing Game (MORPG), where workload on the server usually increases when more users are connected to it. They used Web workers to share the workload to the client machines and managed to decrease the CPU usage of the server, but at the same time, increased the communication
load. They also pointed out a problem with the reliability of the data calculated at the server side. In shooting games, where fairness is the most important factor, measures have to be adopted to overcome this drawback.

When considering what Web workers offer to the web programming when comparing to the traditional Ajax programming, the conceptual difference is that the documents are able to be aware of each other, interact and function simultaneously. Thus, they offer a possibility to implement any kind of applications that need these kind of features. The obvious applications include global background computing tasks, for example, grid computing similar to distributed.net project by NASA, and local background computing tasks such as applying a filter to an image. However, when threading and web messaging offer new possibilities for programmers, they also takes some freedom away from the users. For example, when documents are able to be aware of each other, a commercial web site might not let user to enter a section on the site if the advertisements are prevented by a pop-up blocker.

Web workers and Messaging API also enable innovative new integrated applications to be created. Similarly to how, for example, Google and Facebook have integrated their services with other web sites, web messaging and shared workers offer a way for different web applications to communicate with each other in a local environment and share common task executors. Now, just as different Facebook apps can access and utilize the personal user data inside the Facebook, it is possible for completely different web documents to do the same, also on the background without user even noticing it. Behind the scenes, multiple JavaScript agents can perform their tasks and interact with each other and the web.

**Support**

Support for Messaging API and Web workers vary between different browsers. In general the support is quite good at the moment, but unfortunately not all the browsers support even the Messaging API completely [8].

![Cross-document messaging - Working Draft](image)

*Figure 1: Browser support for Messaging API*
As seen in the Figure 1, the support for the Messaging API is quite good. Even the mobile browsers are already supporting it. Unfortunately, the Internet Explorer has only a partial support, and the same trend seems to continue in the near and farther future versions.

![Figure 2: Browser support for Web workers](image)

The Figure 2 illustrates the support for Web workers in different browsers. All the main browsers already have the support, apart from Internet Explorer, which will however have it in the version 10.0. The situation is comparatively good as some of the mobile browsers are also already supporting Web workers. For the Shared Web workers the situation is not as good. As seen in the Figure 3, while some of the main browsers and mobile browsers already have support, Internet Explorer and Firefox do not have support nor is it known when they will.

![Figure 3: Browser support for Shared Web workers](image)
Conclusions

Web workers offer interesting possibilities for web developers to create new kind of innovative applications. They are successfully doing their part in bringing the web user experience closer to the native desktop application user experience. HTML5 has lots of really great new features, and Web workers are definitely not the least significant one.

The usage of Web workers is straightforward and simple. They do not offer exactly the same functionality as threads in traditional programming languages, but a more constrained version of multitasking support. Shared workers complement the feature set offering multicast interaction support.

Applications exploiting Web workers have already been implemented and they are basically utilizing the background processing possibility to execute heavy computing tasks. Future application might be more innovative creating basis for new kind of services on the Web.

Based on the current and planned support for the Web workers in browsers, it seems that the developers can be quite optimistic when developing new application. Only downside is the dragging support in Internet Explorer, as the users of that browser are less likely to regularly upgrade to next versions.

References


