Open Access

Open Journal of Web Technologies (OJWT)
Volume 1, Issue 2, 2014

http://www.ronpub.com/journals/ojwt
ISSN 2199-188X

A Comparative Evaluation of Current HTML5 Web Video Implementations

Martin Hoernig, Andreas Bigontina, Bernd Radig

Image Understanding and Knowledge-Based Systems, Technische Universität München,
Boltzmannstr. 3, 85748 Garching, Germany, {hoernig,bigontia,radig}@in.tum.de

Abstract

HTML5 video is the upcoming standard for playing videos on the World Wide Web. Although its specification has not been fully adopted yet, all major browsers provide the HTML5 video element and web developers already rely on its functionality. But there are differences between implementations and inaccuracies that trouble the web developer community. To help to improve the current situation we draw a comparison between the most important web browsers. We focus on the event mechanism, since it is essential for interacting with the video element. Furthermore, we compare the seeking accuracy, which is relevant for more specialized applications. Our tests reveal varieties of differences between browser interfaces and show that even simple software solutions may still need third-party plugins in today's browsers.

Type of Paper and Keywords

Short communication: *HTML5 video, events, seeking, user agents, browser, evaluation*

1 Introduction

After the Web Hypertext Application Technology Working Group (WHATWG) was founded in 2004, the advancement of the Hypertext Markup Language (HTML) [15] soon was pooled mainly in this group. Here, HTML is advanced version-less as a *living standard* [3]. In terms of this group, HTML5 has become the wrong name suggesting a complete language [1]. As some parts of the specification are more mature than others (and have a more reliable browser support), they argue that a version schema has become impractical for HTML. However, the "official" specification of the W3C [9] aims at a stable snapshot of this living standard. In this paper, we will only focus on the W3C version of HTML and refer to it as HTML5, in the same way the W3C does.

HTML5 [9] introduces a standard for the integration of videos or movies into web pages: HTML5 video. Before HTML5, browser plugins (like Adobe Flash [4] or Microsoft Silverlight [21]) were usually used. Now, given a standard compliant browser, a web developer should be able to achieve comparable or even better results with native techniques. But are current browsers standard compliant regarding HTML5 video? To examine the situation, we focused our experiments on the most popular browsers and operating systems, which have a market share of over 95% (see, for instance, [2] for a browser usage statistic). Hence, Microsoft Internet Explorer [20], Mozilla Firefox [23] and Google Chrome [13] were tested on Microsoft Windows [22] and Apple Safari [6] on Apple OS X [5] (in each case the newest version was used). The HTML5 standard does not specify a video format. Therefore, the supported video formats vary. We decided to perform our experiments with h264 AVC [29] within MP4 containers [16], which is the only video format supported by all user agents within our test set. Furthermore, it is a widely used industry standard for applications like broadcast TV or Blu-ray video.

```
<html>
<head>
  <title>Non-stop video?</title>
</head>
<body>
  <video width="320" height="240"
    controls="controls" id="vid">
    <source src="sample.mp4" type="video/mp4">
    <!-- Fallback: --> No browser support.
  </video>
  <script>
    var v = document.getElementById("vid");
    // if pause is clicked
    v.addEventListener('pause', function() {
      // resume
      v.play();
    });
  </script>
</body>
</html>
```

Listing 1: Basic example revealing the first problem

Listing 1 shows a simple HTML code with JavaScript [7] that loads a video ("sample.mp4") with standard browser video controls and waits for the video element to receive the *pause* event. If the video is started and paused later on by the user, this event is fired, causing the playback to continue immediately. According to the specification [9], the *pause* event is also fired when the video ends. Hence, the JavaScript code should also cause an endless loop of the video. Some browsers, however, violate the standard and miss the *pause* event when the video reaches the end. Therefore, the behavior is browser dependent even in this simple example.

More on events and how the implementations handle them in contrast to the specification is given in Section 2. Section 3 addresses the seeking capabilities. In particular, we check whether the shown frame changes appropriately, when the *seeked* event occurs, by examining the frame number and measuring any deviation from the expected one.

Of course the results of both sections may change with newer browser versions and our paper only shows the state of the development on the day of its release, but as browsers are often in use for a longer period of time, the issues of current browsers remain the problems of tomorrow's web developers. Internet Explorer 8, for instance, a browser released in 2009 and not supporting the video element, has five years later still a market share of 21.4% according to [2].

1.1 Related Work

Besides the specifications [9, 15] themselves, recent guidebooks to HTML5 video and its possibilities in general are [18, 26, 12]. With HTML5 enabling a new generation of web applications, a wider range of functionalities within a common environment became possible. In earlier times the integration of plugins was necessary to create rich and interactive user experiences. While ActiveX [19, 10] played a decisive part, browser specific plugins became more and more important [14]. Although some plugins are available for a broad range of browsers, like Adobe Flash, other plugins are specific to a restricted range of platforms or browsers. To overcome this limitation and other plugin restrictions (a survey to different plugins, their strengths and weaknesses, is given in [17]), a more advanced language specification, HTML5, was introduced. It was examined regarding its video accessibility [25] and peer-to-peer video abilities [24, 8]. To the best of our knowledge, a detailed examination of the most relevant implementations of the video element has not been done yet.

2 Events

Events are an important link between HTML and JavaScript and are part of the HTML specification. They are used to communicate all kinds of user input and user agent (respectively browser) occurrences to the running web application. In the case of the video element, events are used to treat incidents occurring during the playback of a video. A collection of events important in this section and their appearance is shown in Table 1.

To check the different browsers against a wrong event behavior, we created a semi-automatic test setup consisting of video playback with buffer underrun and seeking. Because we had to force the buffer underrun via a reduction of the network bandwidth with an external application [27], we tested this part manually. Our test procedure contains the following commands (1, 2, 5) and conditions (3, 4):

1. **load**: Load a test video, assure that the network bit rate is high enough for interruption-free playback. Note that *autoplay* is disabled.
2. **start**: Start playback (via JavaScript) and play the first seconds.
3. **underrun**: Force a buffer underrun by slowing down the connection while the user-agent tries to receive more data.
4. **reset**: Reestablish a connection with sufficient bandwidth for interruption-free playback. (The playback should start automatically.)
5. **seek**: Seek forward to an unbuffered position. (The playback should start as data is forthcoming.)
6. **end**: The video reaches the end.

Table 1: Summary of events, cited from [9, 4.7.10.16 Event summary]

Event	Fired when...
progress	The user agent is fetching media data.
suspend	The user agent is intentionally not currently fetching media data.
stalled	The user agent is trying to fetch media data, but data is unexpectedly not forthcoming.
loaded-metadata	The user agent has just determined the duration and dimensions of the media resource and the text tracks are ready.
loaded-data	The user agent can render the media data at the current playback position for the first time.
canplay	The user agent can resume playback of the media data, but estimates that if playback were to be started now, the media resource could not be rendered at the current playback rate up to its end without having to stop for further buffering of content.
canplay-through	The user agent estimates that if playback were to be started now, the media resource could be rendered at the current playback rate all the way to its end without having to stop for further buffering.
playing	Playback is ready to start after having been paused or delayed due to lack of media data.
waiting	Playback has stopped because the next frame is not available, but the user agent expects that frame to become available in due course.
seeking	The seeking IDL attribute changed to true, and the user agent has started seeking to a new position.
seeked	The seeking IDL attribute changed to false after the current playback position was changed.
Others:	*loadstart*, *abort*, *error*, *emptied*, *durationchange*, *play*, *pause*, *ended*, *timeupdate*, *resize*, *ratechange*, *volumechange*

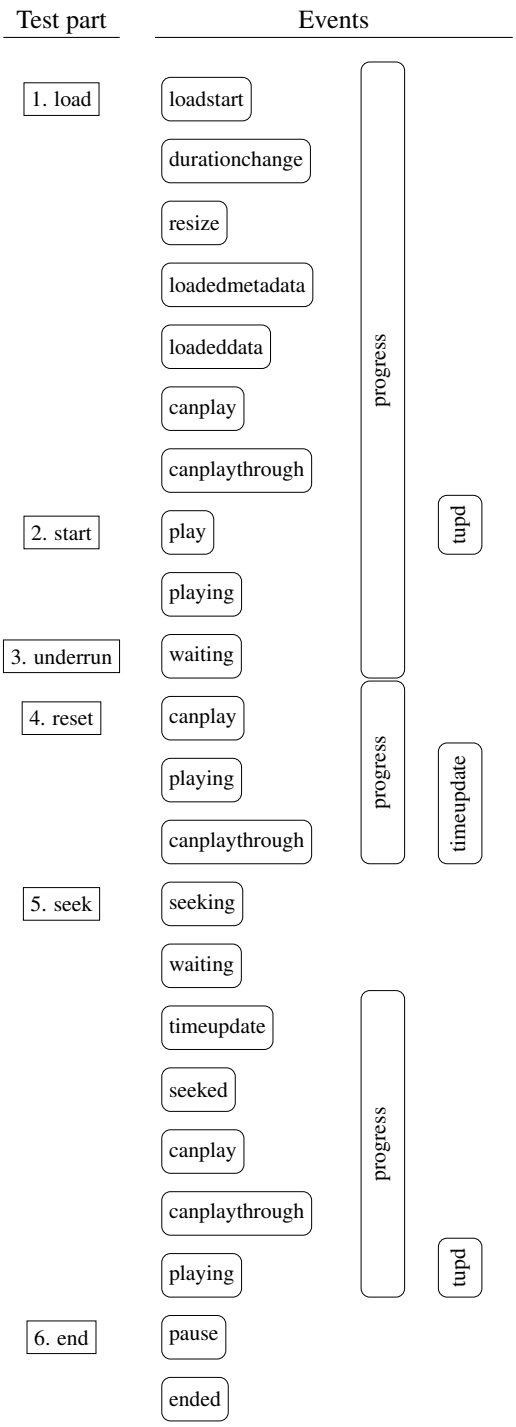

Figure 1: Specification conform event sequence. Fired during loading (1), playing (2), buffer underrun start (3) and end (4), seeking (5), and video end (6). Horizontally shown events are fired once in the given order, vertical entries represent events fired repeatedly.

A video processed in a hypothetical standard conform user agent would fire an event sequence according to Figure 1. This is not the only valid sequence as some options exist, e.g. a user agent can suspend the loading step after it has loaded the meta data to reduce bandwidth and fire the *suspend* event. A *stall* event could

also be fired if the buffer underrun (4) did last too long (user agent specific timeout) and "data is unexpectedly not forthcoming." [9, 4.7.10.16 Event summary]. However, the pretended buffer underrun (4) is defined to be short enough not to evoke a *stall* event. Besides such exceptions, the events are mandatory and have to occur in the given sequence. Since we know the network bit rate and the video bit rate, we can take the *canplaythrough* event as mandatory as well.

The behavior of the test browsers according to the points (1) to (6) is evaluated with the event sequence a [9]-conform user agent evokes. A compilation of all events is shown in Figure 1. Our primary test video is "Big Buck Bunny" from The Peach Open Movie Project [28], which is also used as a test video at W3C.

Unfortunately, not a single browser passes our test setup (see Table 2). Only Chrome fires *resize* on load as required, but also sends a *timeupdate* event, which should not be present. While the video start is handled correctly, only Firefox reliably fires the *waiting* event if the playback has stopped. Safari does it sometimes. We denoted such behavior as unpredictably missing (um) as we have not discovered the pattern underneath. The playback resume is done right again by Firefox (except for the *canplaythrough* event, see the following list of observations for details). Then again seeking is handled right on Chrome. The most important events in this context, *seeking* and *seeked*, are present all across the test set. In Internet Explorer a *pause* event is missing when the video ends.

Observations:

- Internet Explorer fires *timeupdate* events in waiting state (when no update has to be committed). These events must be fired every 15 to 250 ms during the "time marches on" steps [9, 4.7.10.8 Playing the media resource], which apply if the current playback position changes, what is not the case in waiting state.
- Firefox fires the *canplaythrough* event after buffering is completed or halted instead of a bandwidth depending solution. In earlier versions of Firefox (e.g. version 28.0) this was handled correctly. Now, this event seems to be misunderstood.
- Chrome uses an optional substep within the resource fetching algorithm [9, 4.7.10.5 Loading the media resource], which is designed for a conservative download strategy, if the user does not request the resource actively (e.g. in a preload step). This is not true in our case. The resource fetching fires a lot of *suspend* events to the video element, which is in this sense a wrong behavior.
- Safari shows some unpredictable behavior if buffer underruns occur, for example:
  - No audio is played after the playback starts again.
  - The specification states that a user agent must show the last rendered frame in the waiting state [9, 4.7.6 The video element], but Safari unpredictably shows a loop of the last frames.
  - The *canplaythrough* event is fired together with a *waiting* event, but *canplaythrough* should only be fired if the new internal ready state is $HAVE\_ENOUGH\_DATA$ [9, 4.7.10.7 Ready states]. It is a strange behavior if the ready state alternates between $HAVE\_ENOUGH\_DATA$ and something else without reason, particularly within a buffer underrun.

Considering today's conditions, a web application cannot rely on a unified user agent behavior. While video loading and end detection are no noteworthy problems, the *resize* event on load (1) is handled correctly only in Chrome.

Since the presence of most of the events is insignificant for the example from Listing 1, most browsers will show the same result. Internet Explorer, however, will not restart the video after it reaches the end. If a buffer underrun would occur, maybe during a broadcast on a mobile connection, the behavior in the event system is in general unpredictable. Most of the tested browser showed problems with the handling of the *waiting* event to signal that "playback has stopped because the next frame is not available, but the user agent expects that frame to become available in due course." [9, 4.8.10.15 Event summary]. In this situation, a proper buffer underrun handling is not possible. A web-based player is not able to communicate the situation to the user. Moreover instabilities during the playback were observed during buffer underruns that made a page reload necessary.

Nevertheless, the events *seeking* and *seeked* were present across all tested user agents. In the next section, we are going to examine how reliable they are.

3 SEEKING

In this section we will examine the seeking capabilities of the selected user agents. An implementation compatible with the standard will set the *seeking* attribute to true and fire a *seeking* event when the *currentTime* attribute is changed. When the video data at the requested time is available, *seeking* has to be set to false and a *seeked* event must be fired.

Table 2: Event test results. The user agents undergo our tests with some discrepancies. The candidates showed the absence of expected events (missing m, unpredictably missing um) and unexpected events (wrongly fired wf).

User agent, version, OS	1. load	2. start	3. underrun	4. reset	5. seek	6. end
Internet Explorer 11.0.9600.16521 Windows	m: *resize*	✓	m: *waiting*	m: *canplay*, *playing*, *canplaythrough*	m: *canplay*, *canplaythrough*	m: *pause*
Firefox 31.0 Windows	m: *resize*, *canplaythrough*	✓	✓	m: *canplaythrough*	wrong order; wf: *waiting*	✓
Chrome 36.0.1985.125 m Windows	wf: *timeupdate*	✓	m: *waiting*	m: *canplay*, *playing*, *canplaythrough*	✓	✓
Safari 7.0 (9537.71) OS X	m: *resize*	✓	um: *waiting*	um: *canplay* *playing*	m: *waiting*, *canplay*, *canplaythrough* *playing*	✓

Table 3: *seeking* and *seeked* events are fired appropriately in all tested user agents.

User agent, version	Operating system	*seeking* event	*seeked* event
Internet Explorer 11.0.9600.16521	Windows	100%	100%
Firefox 31.0	Windows	100%	100%
Chrome 36.0.1985.125 m	Windows	100%	100%
Safari 7.0 (9537.71)	OS X	100%	100%

In a first series of tests this basic behavior is examined. We use a video of ten minutes length and jump to 1000 positions in this video. We repeat this procedure ten times with different positions. These positions were chosen at random at a preparation step, but are the same for all user agents. Results are shown in Table 3.

seeking and *seeked* are crucial events, and indeed, as mentioned in Section 2, all tested user agents pass this test. Though, a different question is if firing these events correlates with the changes of the displayed segment of the video. Is the requested frame already available when the *seeked* event is fired?

Listing 2 shows how the *seeked* event could be used to draw a "snapshot" of the video onto a canvas element after the user sought to a position in the video. Assuming the video is paused, will the user see the same frame in canvas and video element? Similar code could be used to create preview pictures to certain positions in the video.

```
<html>
<head>
  <title>Snapshot on seeked?</title>
</head>
<body>
  <video width="640" height="480"
    controls="controls" id="vid">
    <source src="sample.mp4" type="video/mp4">
    <!-- Fallback: --> No browser support.
  </video>
  <canvas width="640" height="480" id="can">
  </canvas>
  <script>
    var v = document.getElementById("vid");
    var c = document
              .getElementById("can")
              .getContext("2d");
    // if seeking procedure completed
    v.addEventListener('seeked', function() {
      // draw video on canvas
      c.drawImage(v, 0, 0, 640, 480);
    });
  </script>
</body>
</html>
```

Listing 2: Example usage of the seeked event

A different aspect is the accuracy of the seeking implementation. In some scenarios the frame rate is known to the application and it intents to step through the video in a frame-by-frame manner. Another application might store the position in a video to return to it later. Will the user be able to see exactly the same frame again, or will there be a deviation?

To answer those questions, we created a video that stores the frame number in each image. This is simply done by a binary coding of this number and drawing it as black and white blocks onto the image. Some additional checksum bits ensure the correctness of the encoding. When playing this video with a user agent, we can retrieve the frame number by drawing the video onto a canvas element and examining the pixel data.

As before we seek to 1000 positions in the video and check the frame number when the *seeked* event is fired. This procedure is repeated ten times with different positions. Various issues have been observed, depending on the user agent. The results are summarized in Table 4.

Table 4: We examine whether a *seeked* event actually indicates a change of the displayed frame and observe several issues.

User agent, version	***seeked* event fired without change of the image? (e.g. too early)**
Internet Explorer 11.0.9600.16521	Happens in 86.17% of all cases.
Firefox 31.0	Never.
Chrome 36.0.1985.125 m	Never.
Safari 7.0 (9537.71)	Happens when seeking as reaction to a *loadedmetadata* event.

Using Internet Explorer, in 86.17% of all cases it was not possible to retrieve the expected frame number from a video when the *seeked* event was fired. However, when checking again later a change in the frame number could be observed (the latencies differed depending on video and test system). It is thereby likely that the *seeked* event is fired while the video element is still decoding or rendering the requested frame. Under these conditions, taking the snapshot after seeking, as in Listing 2, is not reliably possible. It is unknown which position in the video actually corresponds to the resulting snapshot.

With Safari we observe a too early fired *seeked* event, when we seek within a callback of the *loadedmetadata* event. From the statement of the specification that says about the *HAVE_METADATA* ready state that "The API will no longer throw an exception when seeking." [9, 4.7.10.7 Ready states] we conclude that seeking must be possible with the occurrence of the *loadedmetadata* event.

Except for this special case Safari fires the *seeked* event as required. Chrome and Firefox also perform these tests without problems. Next, we analyze the accuracy of the seeking implementation. We give every user agent enough time to actually finish seeking before we check the frame number. The results can be found in Table 5.

We find that Internet Explorer has a constant deviation from the expected frame by two frames. Looking at this problem in detail we can see that the first frame of the video is displayed three times as long as the others. This leads to the observed offset of two frames. Chrome fails to seek to the correct frame in 1.45% of all cases. When the wrong frame is shown, it is one, two or three frames off in our experiments. This might be close enough for some applications, but might trouble others. Finally, Firefox and Safari both seek to the expected frames without deviations.

Altogether, the experiments show that the *seeked* event is not very reliable. What is the benefit of a *seeked* event, if we cannot be sure that a new frame is shown when it is fired? Furthermore, we have shown small deviations from the expected frame. These are usually not important and most users won't even take notice of this issue. However, it prevents the emergence of application that rely on frame-precise seeking.

4 IMPROVEMENTS TO HTML5 VIDEO

As we are interested in a powerful HTML5 video platform to supersede third party plugins as the leading internet video technology, the portfolio of possible applications should not suffer from a transition to HTML5. Examples for possible media-oriented tasks are:

- video post-processing (edit the pixel data manually, e.g. to change the contrast),
- audio equalization, and
- audio spectrum visualization.

Unfortunately, these applications are currently not possible using pure HTML5. Though the video buffer can be obtained and drawn to a canvas (timer-based), it is not possible to ensure this for every frame, since the frame rate is unknown. The audio data cannot be accessed anyway. We suggest the following improvements to HTML5:

- Querying detailed meta data information (e.g. frame rate, bit rate, etc.)
- Frame-wise stepping, querying frame number
- Audio buffer access and manipulation
- Manipulating pixel data of the video (e.g. introducing a new *newframe* event to control the manipulation on a canvas, fired after the availability of a new frame buffer)

Table 5: The seeking accuracy is tested by measuring any deviation from the expected frame number.

User agent, version	Operating system	Wrong frames	Average absolute deviation	Maximal absolute deviation
Internet Explorer 11.0.9600.16521	Windows	100%	2	2
Firefox 31.0	Windows	0%	0	0
Chrome 36.0.1985.125 m	Windows	1.45%	1.8966	3
Safari 7.0 (9537.71)	OS X	0%	0	0

5 Conclusion

HTML5 video is an important step towards the goal of having a unified way of displaying videos on the web without requiring third party plugins. However, when trying to interact with the element via JavaScript and create a user agent independent solution, several issues become apparent. We took a close look at the event system of the browsers and compared their behavior. The comparison between user agents and the specification showed missing events and differences of implementations. As a consequence among other issues, there is no reliable way of handling an underrun.

The analysis of seeking implementations reveals further problems. The *seeked* event that is fired too early in Internet Explorer prohibits web applications, for instance, to seek to a position and take a snapshot of the video. The inaccuracies in seeking in Chrome prevent meaningful frame by frame stepping through the video. Those and other bugs force web developers to find browser-dependent workarounds. As this is a contradiction to the paradigm of feature detection-based web development [11], HTML5 video needs a more consistent application interface in current browsers. If special applications like web video cutting are planned, we have to recommend (based on today's level of knowledge) the usage of third party plugins as seeking accuracy and completion are in general no trustworthy information.

We are offering user agent tests, test cases, and test data to developers to evaluate and improve new implementations. We hope the next versions of the tested browsers stand our tests. With a growing distribution of an unified and standard conform video interface, we will see more advanced online multimedia applications and user experiences.

Acknowledgments

This work was supported by the German Research Foundation (DFG) and the Technische Universität München within the funding programme Open Access Publishing.

References

[1] "HTML is the new HTML5," http://blog.whatwg.org/html-is-the-new-html5, 2011, Accessed: 2014-09-15.

[2] "NetApplications," http://www.netmarketshare.com/report.aspx?qprid=0&qptimeframe=M&qpsp=184&qpcustomd=0, 2014, Accessed: 2014-06-10.

[3] "What does "Living Standard" mean?" http://wiki.whatwg.org/wiki/FAQ#What_does_.22Living_Standard.22_mean.3F, 2014, Accessed: 2014-09-15.

[4] Adobe Systems Inc., "Adobe Flash Player," http://www.adobe.com/products/flashplayer, Initial release: 1996.

[5] Apple Inc., "OS X," http://www.apple.com/osx, Initial release: 2001.

[6] ——, "Safari," http://apple.com/safari, Initial release: 2003.

[7] B. Eich, Ecma International, "ECMAScript Language Specification, 5th Edition," Ecma International, Geneva, Switzerland, Tech. Rep., 2009, ECMA-262.

[8] A. Bakker, R. Petrocco, M. Dale, J. Gerber, V. Grishchenko, D. Rabaioli, and J. Pouwelse, "Online video using bittorrent and html5 applied to wikipedia," in *Peer-to-Peer Computing (P2P), 2010 IEEE Tenth International Conference on*. IEEE, 2010, pp. 1–2.

[9] R. Berjon, S. Faulkner, T. Leithead, E. D. Navara, E. O'Connor, S. Pfeiffer, and I. Hickson, "HTML5," W3C, Candidate Recommendation 29 April 2014, Apr. 2014, http://www.w3.org/TR/html5/.

[10] D. A. Chappell, *Understanding ActiveX and OLE*. Microsoft Press, 1996.

[11] S. P. Corti, "HTML5: Browser and Feature Detection," http://msdn.microsoft.com/en-us/magazine/hh475813.aspx, October 2011, Accessed: 2014-09-15.

[12] B. Frain, *Responsive web design with HTML5 and CSS3*. Packt Publishing Ltd, 2012.

[13] Google Inc., "Google Chrome," http://www.google.com/chrome, Initial release: 2008.

[14] C. Grier, S. T. King, and D. S. Wallach, "How I learned to stop worrying and love plugins," in *In Web 2.0 Security and Privacy*. Citeseer, 2009.

[15] I. Hickson, "HTML," WHATWG, Tech. Rep., http://whatwg.org/html.

[16] ISO, "Information technology — Coding of audio-visual objects — Part 14: MP4 file format," International Organization for Standardization, Geneva, Switzerland, Tech. Rep., 2003, ISO 14496-14:2003.

[17] T. Lammarsch, W. Aigner, A. Bertone, S. Miksch, T. Turic, and J. Gartner, "A comparison of programming platforms for interactive visualization in web browser based applications," in *Information Visualisation, 2008. IV'08. 12th International Conference*. IEEE, 2008, pp. 194–199.

[18] M. MacDonald, *HTML5: The Missing Manual*. O'Reilly Media, Inc., 2013.

[19] Microsoft Corporation, "ActiveX," Initial release: 1996.

[20] ——, "Internet Explorer," http://microsoft.com/ie, Initial release: 1995.

[21] ——, "Microsoft Silverlight," http://www.microsoft.com/silverlight, Initial release: 2007.

[22] ——, "Windows," http://windows.microsoft.com, Initial release: 1985.

[23] Mozilla Foundation and contributors Mozilla Corporation, "Mozilla Firefox," http://mozilla.org/firefox, Initial release: 2002.

[24] J. K. Nurminen, A. J. Meyn, E. Jalonen, Y. Raivio, and R. G. Marrero, "P2P media streaming with HTML5 and WebRTC," in *IEEE International Conference on Computer Communications. IEEE*, 2013.

[25] S. Pfeiffer and C. Parker, "Accessibility for the HTML5 <video> element," in *Proceedings of the 2009 International Cross-Disciplinary Conference on Web Accessibililty (W4A)*. ACM, 2009, pp. 98–100.

[26] M. Pilgrim, *HTML5: up and running*. O'Reilly Media, Inc., 2010.

[27] SoftPerfect Research, "Bandwidth Manager," https://www.softperfect.com/, PO Box 140, Fortitude Valley QLD 4006, Australia.

[28] Sun Microsystems, "Sun's Network.com Renders Computer-Animated Movie "Big Buck Bunny"," Press Release, June 2008.

[29] T. Wiegand, G. J. Sullivan, G. Bjontegaard, and A. Luthra, "Overview of the H. 264/AVC video coding standard," *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 13, no. 7, pp. 560–576, 2003.

AUTHOR BIOGRAPHIES

Martin Hoernig studied Mathematics at the Leipzig University of Applied Sciences. He received his master's degree in 2011. In the same year he worked as a scientist at the Max Planck Institute for Human Cognitive and Brain Sciences Leipzig. Today, he is a research assistant in the TU Munich Image Understanding and Knowledge-Based Systems Group headed by Bernd Radig. His primary areas of research are: image analysis and understanding, learning and decision making, as well as web technologies.

Andreas Bigontina received his bachelor's degree in Computer Science in 2012 and is currently pursuing his master's degree in Computer Science, both at the Technische Universität München. He worked as research assistant as part of the Image Understanding and Knowledge-Based Systems Group, where he mainly was concerned with web technologies. His current research interests include computer vision and machine learning.

Bernd Radig received his diploma degree in Physics in 1972 from the University of Bonn and the doctor degree in Computer Science in 1978 from the University of Hamburg. In the University of Hamburg he also got his venia legendi and finished his habilitation dissertation in 1982. He was Assistant and Associate Professor in Hamburg (1982-1986) and full professor, chair of Image Understanding and Knowledge-Based Systems, Fakultät für Informatik, Technische Universität München (1986-2009). He is a member of the *Emeriti of Excellence* programme. He was chairman and founder of the Association of Bavarian Research Cooperations (1993-2007), a unique network of scientists, specialising in challenging disciplines in accordance with Bavarian enterprises. In 1988 he founded the Bavarian Research Centre for Knowledge Based Systems (FOR-WISS), an institute common to the three universities TU München, Erlangen and Passau. He was the general chairman of the annual symposium of the German Association for Pattern Recognition in 1981, 1991 and 2001. He was also the general chairman of the European Conference on Artificial Intelligence (ECAI) in 1988. He is active as organizer and programme committee member of the German-Russian Workshop on Pattern Recognition. He holds the German Order of Merit (1992) and the award *Pro Meritis Scientiae et Litterarum* of the State of Bavaria for outstanding contributions to science and art (2002). His current research activities are in real-time image sequence understanding for applications in robotics, sports or driver assistance systems.