# The Text 2.0 Framework

## Writing Web-Based Gaze-Controlled Realtime Applications Quickly and Easily

**Ralf Biedert, Georg Buscher, Sven Schwarz, Manuel Möller, Andreas Dengel**
German Research Center for Artificial Intelligence (DFKI)
Kaiserslautern, Germany
firstname.lastname@dfki.de

**Thomas Lottermann**
University of Kaiserslautern
Kaiserslautern, Germany
tomlottermann@googlemail.com

### ABSTRACT

We created a simple-to-use framework to construct gaze-responsive applications using web technology focussing on text. A plugin enables any compatible browser to interpret a new set of gaze handlers that behave similar to existing HTML and JavaScript mouse and keyboard event facilities. Keywords like `onFixation`, `onGazeOver`, and `onRead` can be attached to parts of the DOM tree and are triggered on the corresponding viewing behavior. The plugin is part of a distributed architecture featuring a remote gaze provider and a number of assisting services and tools. Using this framework we implemented a number of applications providing help on comprehension difficulties.

### INTRODUCTION

Text 2.0 is text that knows that it's being read[1]. Thus its ingredients are eye tracking, text being read and *realtime* interaction. Text 2.0 is capable to adapt to the mode the reader is in. It offers several benefits to him or her because it reduces information where they are not needed, and puts in more information where it is helpful.

The domain of Text 2.0 holds lots of new user-text interactions paradigms where the existing prototypes and demos have merely scratched the surface yet. Applications falling into this domain are, for example, the eyeBook[2], iDict[4], *the reading assistant*[6], AdeLE[5], and a number of prototypes we developed like intelligent footnotes using DBPedia[2], segmentations and more. While some of the individual demos and use cases are not new, we think the combination denoted as Text 2.0 is.

Due to the rather straightforward simplicity of programmatically handling text we think a more thorough study of its combination with realtime eye tracking is worth investigation and opens new perspectives in various fields. We see these fields not only in the entertainment domain but also in the fields of knowledge management, language learning, and HCI.

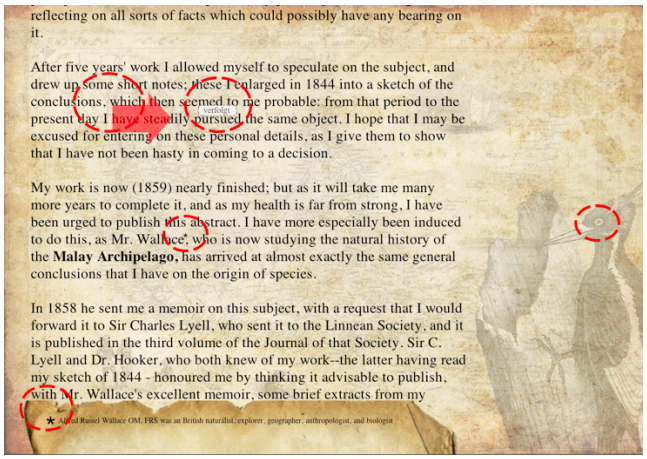


**Figure 1. Sample application *The Origin of Species* depicting a selection of features like intelligent footnotes, translations, image effects and an automatic bookmark. The application is displayed inside a web browser and created using the Text 2.0 Engine. The red circles mark gaze-responsive effects and areas.**

However, in order to explore these possibilies properly they have to be implemented. Current eye tracking application frameworks do not allow for a simple creation of interactive textual applications since many related aspects, like rendering, layout, dynamic modification of content and of course the support of graphics and animation are time-consuming to implement.

We introduce a framework taking care of these issues. Therefore we propose using HTML and web browsers as the foundation for further developments. There have been almost 20 years of hypertext research and most of the problems mentioned have already been solved. Furthermore HTML, JavaScript and CSS are simple to use and there is a plethora of existing literature covering almost any aspect of these technologies. That way it is also easier to properly polish an application to get an even more realistic use case than with only partially written prototypes.

Also, the usage of common languages and concepts significantly lowers the barrier for beginners to write eye tracking applications. Masses of web designers are now easily ca-

[1] `http://text20.net`
[2] `http://dbpedia.org`

Presented at International IUI 2010 Workshop on Eye Gaze in Intelligent Human Machine Interaction.

```
"It is true that on that you can't have come from very far away ..."<br>
And  he sank into a reverie, which lasted a long time. Then, taking my sheep
out of <span onRead="eventHandler.playAudio('papercrumple.wav');">his pocket</span>,
the contemplation of his treasure.<br>  You can imagine how my curiosity was aroused by
"My little man, where do you come from? What is this 'where I live,' of which you speak?
```

**Figure 2. Gaze responsive annotations can be embedded directly into a web page. In this case the handler playAudio is executed when reading behavior is detected on the specified span region.**

pable of writing gaze-responsive applications. Lastly, we think that a framework based on web technologies eases the creation not only of textual eye tracker applications, but of many other interactive programs as well.

## THE DEVELOPER'S INTERFACE

The foremost goal in the design of the API was to create a natural *look and feel* that blends into existing event handling and programming interfaces of browser environments.

While most dynamic HTML is done using JavaScript, there are two different ways of associating user-defined code with input events based on mouse or in this case gaze data: Event handlers specified inside HTML tags are called upon the occurrence of events below the DOM-node they are declared on. The other mechanism is to hook event callbacks on a global level. The Text 2.0 engine allows for both.

We added a number of new attributes definable on most HTML tags, handling fixation and reading events similar to mouse events, compare Figure 2:

- `onFixation` specifies a corresponding handler that should be executed when the user starts or ends a fixation on this element. If another fixation occurs within the same element, the handler is executed again.

- `onGazeOver` and `onGazeOut` work similar to the pre-existing `onMouseOver` and `onMouseOut` handlers. As soon as the user's gaze enters an element, its `onGazeOver` handler is being called. In contrast to `onFixation` further fixations inside the element do not create additional events. When the gaze leaves the element, the handler specified in `onGazeOut` is executed. The handling also works with nested child nodes, i.e., nodes below the tagged node.

- `onPerusal` and `onRead` are called when the plugin detects reading or skimming behavior. In that case, the corresponding handler is called periodically on every new fixation with an approximate reading speed (characters per second). See below for a discussion of the reading detection. This handler is usually put on `div` and `span` tags to execute eyeBook-like events that should be protected from premature execution by occasional fixations. The handler `onRead` is currently an alias of `onPerusal` but will be based on a calibrated reading speed in the future.

In addition, there are a number of functions exposed to the JavaScript engine directly. In contrast to the event-based mechanisms above they are dedicated to global event handling and setup, not bound to any element of the DOM tree and, thus, not specific to any region on the page. They include:

- setup functions - used to define the location of the eye tracking server, setup of automatic network discoveries, logging and recording levels (ranging from no logging to a complete recording of the session including gaze data, page geometry and system events), paths to additional gaze evaluations modules and extensions, as well as various gaze parameters.

- global listeners - can be used to listen to eye tracking events on an application wide level. They allow access to raw gaze data, filtered data, and fixation data.

- extensions - the browser plugin is modular and plugin-based itself. It comes with a number of standard extensions, loads user extensions specified in the setup phase and can dynamically access network services. These extensions can choose to expand the internal functionality of the plugin (e.g., replace algorithms), but they can also expose themselves to the JavaScript side.

## ARCHITECTURE

The framework has been implemented using a distributed architecture (see Figure 3).

Its core consists of a browser plugin providing the aforementioned services and a remote eye tracking server. This separation helps to improve the initialization time during page changes.

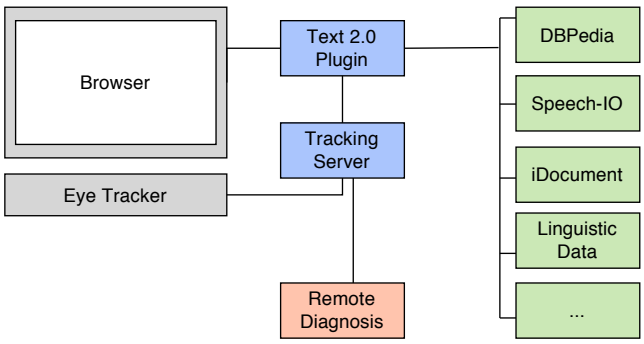


**Figure 3. The Text 2.0 plugin provides eye tracking data to the browser and mediates access to supporting services.**

### The Core Plugin

The core plugin accepts raw gaze information from the remote eye tracking server, performs various processing steps, and provides the results to the currently displayed web page in real time. It also allows for an automatic mouse fallback in the case no eye tracking server is found.

In normal operation, the internal data flow is cascaded and starts with the filtering of incoming tracking events. An attached fixation detection step builds the foundation for the perusal detector, which gives realtime information on reading and skimming behavior[3].

However, the greatest obstacle in the development of the plugin had been a general lack of access to the layout and no obvious API to access individual HTML elements. Plugins typically need to execute JavaScript to obtain the rendering coordinates and size of the page's content which can induce a significant performance bottleneck if the page is complex. Also, the current APIs do not provide access to the pixel locations of the individual words within blocks of text. In order to enable the plugin (and not only the web page) for reasoning on the displayed content on the screen, both problems needed to be overcome and addressed by the framework:

Updates of the page's general geometry are observed by a JavaScript function that tags modified DOM elements, sorts them into buckets, rechecks these buckets periodically, and batch-transmits changes back to the plugin. The problem of word access is handled by a process we call *spanification* that segments text nodes consisting of multiple words into a set of span nodes containing one word each. Using this technique we are able to obtain the bounding boxes for every word on the web page. The obtained and transmitted word locations are stored inside a data structure called *pseudorenderer* on the plugin side. This pseudorenderer resembles a virtual screen device and allows for fast reasoning outside the JavaScript thread, which frees its resources for handling the remaining user interface tasks.

Besides the realtime functionality, the plugin also keeps session logs. These logs contain the tracking data and layout information and can be used to replay recorded gaze interactions. Due to the modular nature of the plugin any replay of recorded sessions is done with the same plugin again. This means that the same algorithms can be used on the original data again, which is helpful for analysis and debugging. It also means, that new algorithms implemented on experimental data can be used without any modification during realtime sessions.

The browser plugin itself is based on a plugin architecture[3]. With the help of this component framework the plugin's functionality can be extended during runtime. Exemplary addons we created within our project are interfaces for speech recognition and synthesis that enable the plugin to give verbal feedback on its status, like, for example whether the mouse or eye tracking device are currently being used or if the tracking data flow has been interrupted unexpectedly.

### Supporting Services

Many applications on realtime text also require assisting datasources, e.g., the intelligent footnote depicted in Figure 1. The services implemented so far provide linguistic, statistic and linked data to the plugin and the web page. While these data sources are not necessarily required for operating the core plugin, all of them are needed by a number of our prototypes and, beyond that, are naturally related to text, which justifies their close association with the architecture of Text 2.0.

From a technical perspective these services are usually distributed across the network and can discovered automatically on the plugin's startup. This is especially required in the case of DBPedia or the currently being integrated iDocument[1] as they tend to be memory-intensive and bring along lengthy initializations times, making them unfeasible for any embedding attempts.

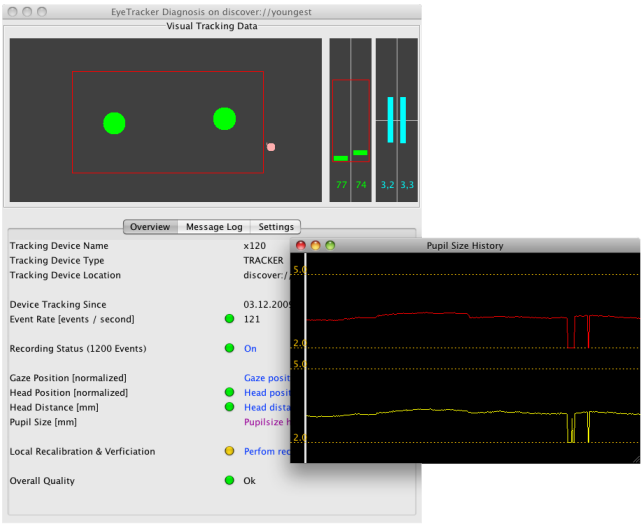


**Figure 4. The remote diagnosis remotely displays a live feed of the eye tracking data and computes a number of quality measures.**

The diagnosis interface is an additional central support facility, although it requires manual observation. Since the eye tracking data provider is a network service on its own, it can be accessed from a remote machine. We implemented a visualization component which takes the measured eye tracking data, displays it as realtime or line charts and graphically summarizes the quality of the data using a number of quality indicators for the head and eye positions as well as data flow and fixation metrics. The tools helps during experiments to ensure a proper recording quality in realtime and take corrective steps in time. A screenshot of the tool is given in Figure 4.

## OUTLOOK

We stated that Text 2.0 is a combination of eye tracking, text and *realtime* interaction. While any two-fold combination of these three features have seen intense interest, all of them taken together offer some unique features worth more investigation: Eye tracking is probably the best non-invasive measure for attention we have. It allows the computer to see what kind of visual information is currently entering the mind of the user. At the same time, text is a very rewarding object for analysis. It is likewise comprehensible to humans as well as to computers. This contrasts to arbitrary scene perception, where neither the gaze paths nor the actual content of the scene convey information which could be used easily for reliable reasoning. This of course does not mean that gaze patterns during reading have been fully understood, however the clear and abstract structure of text simplifies the process of correlating eye tracking data with presented stimuli and hypothesized cognitive processes.

[3] `http://code.google.com/p/jspf`

While most of our applications and prototypes implemented so far react on gaze within a few hundred milliseconds, the framework is also capable of tracking reading and interaction behavior on a longer scale. It allows us to construct assistants using aggregated reading data of several sessions that adapt better to the user's profile. This paves the way towards attentive documents.

We created a number of prototypes on our own. The most notable prototype is a Text 2.0 version of the introduction chapter of Darwin's *The Origin of Species* and contains several assistances. These include automatic translations presented as glosses (compare [4]) when comprehension difficulties are detected on difficult words. Intelligent footnotes are presented for terms which have no translation in the speaker's native language and for technical terms. In that case a small asterisk besides the word signalizes that more information is available in a dynamic footnote at the lower part of the screen. This box contains an automatic excerpt from DBPedia (the semantic web version of Wikipedia). A dialog system is integrated as well and helps reducing ambiguities. Questions like "how do you pronounce that", "can you explain this" and "what's the translation" can be spoken by the user and a verbal answer is given, the demonstrative is resolved using the current point of regard. Other features like a segmentation of long words, automated bookmarks upon return and visual feedback while reading are implemented as well. Most parts of the application's interface and reactions have been written by students familiar with web design, but not with eye tracking. We think this fact nicely illustrates the potential of our easy-to-use eye tracking framework.

The Text 2.0 framework's predecessor eyeBook[2] was focussed on entertainment only. In contrast, this architecture allows for general purpose text-based eye tracking applications. We are planning to use it to construct applications in the areas of knowledge management and language learning.

From a technical perspective we are preparing for discussions with browser vendors in order to address the mentioned issues of textual and geometrical access to the DOM elements. Also, while the plugin works well with WebKit[4] and Safari[5] on Mac and Windows, we are planning to extend its browser support. Safety and security concerns are also on our roadmap but we assume them to be manageable, as security models of comparable technologies like web cameras or microphones have shown. Furthermore we are considering to eventually release an open source version of the Text 2.0 framework.

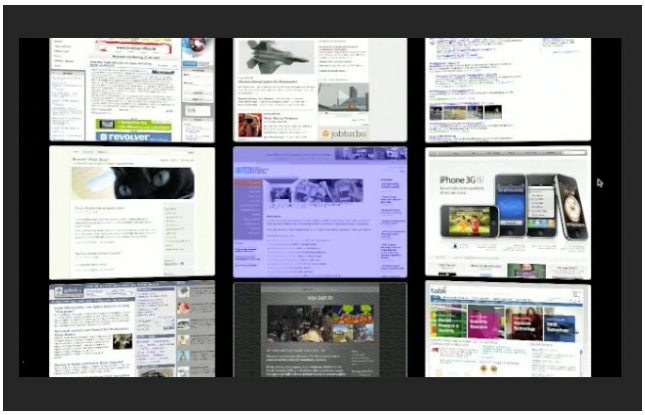

**Figure 5. Integrating eye tracking into browsers helps us to participate on their progress. Only a few days after WebKit released their first version of 3D CSS transforms we implemented a gaze responsive 3D tab exposé within only four hours of work.**

## CONCLUSION

We presented an easily usable framework that mixes eye tracking data with web technology. The API can can be understood by any web designer with a bit of JavaScript experience intuitively. Simple gaze-responsive applications can be implemented with almost no programming while complex projects benefit from a clear separation of structure (HTML), logic (JavaScript) and design (CSS). We think the principal choice of using a browser plugin offers fundamental benefits in the long term. The speed of the development progress in the web community is high and new exciting features are constantly integrated into browsers, enhancing the possibilities of gaze-responsive applications almost for free, as we showed with the implementation of a gaze enhanced tab exposé (see Figure 5) which we developed in only a few hours.

## ACKNOWLEDGMENTS

We would like to thank Eugen Massini, Andreas Buhl and Tim Biedert for their contributions to the engine. Parts of this work have been funded by the Stiftung für Innovation Rheinland-Pfalz.

## REFERENCES

1. B. Adrian, J. Hees, L. van Elst, and A. Dengel. idocument: Using ontologies for extracting and annotating information from unstructured text. *KI 2009: Advances in Artificial Intelligence*, 5803/2009:249–256, 2009.

2. R. Biedert, G. Buscher, and A. Dengel. The eyebook – using eye tracking to enhance the reading experience. *Informatik-Spektrum*, (In Publication), Sep 2009.

3. G. Buscher, A. Dengel, and L. van Elst. Eye movements as implicit relevance feedback. *CHI '08 extended abstracts on Human factors in computing systems*, pages 2991–2996, 2008.

4. A. Hyrskykari. Eyes in attentive interfaces: Experiences from creating idict, a gaze-aware reading aid. *acta.uta.fi*, Jan 2006.

5. F. Mödritscher, V. García-Barrios, C. Gütl, and D. Helic. The first adele prototype at a glance. *Proceedings of World Conference on Educational Multimedia, Hypermedia and Telecommunications*, pages 791–798, Jan 2006.

6. J. Sibert, M. Gokturk, and R. Lavine. The reading assistant: eye gaze triggered auditory prompting for reading remediation. *Proceedings of the 13th annual ACM symposium on User interface software and technology*, pages 101 – 107, Jan 2000.

[4] `http://webkit.org`
[5] `http://www.apple.com/safari`