Media Technology

Development of an easy-to-use marker API for the free JavaScript web map library khtml.maplib

First Bachelor Thesis

Completed by

Ewald Wieser
mt091110

From the St. Pölten University of Applied Sciences
Media Technology degree course

Under the supervision of
Bernhard Zwischenbrugger

St. Pölten, on October 18th 2011

(Signature Author) (Signature Advisor)

Fachhochschule St. Pölten GmbH, Matthias Corvinus-Straße 15, 3100 St. Pölten, T: +43 (2742) 313 228, F: +43 (2742) 313 228-339, E: office@fhstp.ac.at, I: www.fhstp.ac.at

Declaration

- I declare, that the attached research paper is my own, original work undertaken in partial fulfillment of my degree.

- I have made no use of sources, materials or assistance other than those, which have been openly and fully acknowledged in the text.

- If any part of another person's work has been quoted, this either appears in inverted commas or (if beyond a few lines) is indented.

- Any direct quotation or source of ideas has been identified in the text by author, date, and page number(s) immediately after such an item, and full details are provided in a reference list at the end of the text.

- I understand that any breach of the fair practice regulations may result in a mark of zero for this research paper and that it could also involve other repercussions. I understand also that too great a reliance on the work of others may lead to a low mark.

St. Pölten, on October 18th 2011

(Signature Author)

Abstract

This thesis shows the research and development process of a marker interface for the JavaScript web map library khtml.maplib. The aim is to get an application programming interface to place markers on the web map that are rich in features and have a nice look and feel, but yet are easy to implement for a standard user.
Therefore, the marker interfaces of some of the most popular web map services are investigated for their features, their generated DOM-structure, and the necessary source code to create a marker. Additionally the info window interface is examined as well, for the possibility to provide more information about a tagged place. The different web maps are then compared and subsequently an own interface for the khtml.maplib library is developed. Standard marker images and info window images are produced as well as the source code is implemented. A main focus lays on the proper functioning on different web browsers and different devices. To achieve this goal several workarounds and specific functions have to be implemented.
The result is not a perfect, but yet highly featured marker interface with a nice standard marker image that is easy to implement in existing web pages by users, who are not so experienced in programming.

Table of contents

1 Introduction

Web maps are interactive maps on websites (cf. Mitchell 2008, p. 9). Compared to static maps they provide some kind of interaction, which allows the visitor to change the map section and the level of detail via mouse, keyboard or – on modern devices – via touch-gestures. With every interaction new data has to be loaded from a map server. Web maps are used for route planning, position tracking or simply to mark a specific position. The biggest advantage of web maps is that they can be implemented in normal websites and so be accessed via a normal web browser with no need for special software or knowledge.

Markers on web maps are usually displayed as overlays on the map. They are used to flag points-of-interest with small images to mark their position. Therefore, the coordinates of the position on the map are stored with the marker. Usually markers provide some further information for the user about the place they flag in form of an info window that opens when the user drags the mouse cursor over the marker or clicks on it.

There are many popular web map services in the internet such as Google Maps, BING Maps or Yahoo Maps, just to name a few of them. They all provide a sophisticated API (application programming interface) with a lot of features, not only for JavaScript but also Flash and other programming languages. But they are all limited to private and non-commercial use (cf. Google 2011a; Microsoft 2011a; Yahoo 2011a).
The best-known open source alternative is OpenStreetMap with its JavaScript API OpenLayers. OpenStreetMap (OSM) is a geographic database of the world, where everybody can contribute and help to improve the information it contains (cf. Bennett 2010, pp. 8-10). Its data and code is free to use under the Creative Commons Attribution-ShareAlike 2.0 (CC-BY-SA) license, as long as the result is also distributed under this license (cf. OpenStreetMap 2011a). OpenLayers is the most popular open source, client side JavaScript web map library and is licensed under the 2-clause BSD license (cf. OpenLayers 2011a). This license allows the commercial use of the library and any modification without publishing the original source code. In the current version 2.10 it does not support multitouch on modern smartphones (cf. Hazzard 2011, pp. 8-9).

This is where khtml.maplib comes into play. It is a free JavaScript web map library for all kind of map data with a lot of features, such as bitmap-overlays and vector drawing, and it has full multitouch support. Khtml.maplib is licensed under the GNU Lesser General Public License (LGPL) (c.f. KHTML 2011a). This license allows the commercial use of the software distributed under LGPL as long as it is only used as an external library, the original code is not modified and the original author is named (cf. Free Software Foundation 2007). Any modification of the original software has to be licensed under LGPL again and only be distributed together with the source code. As the khtml.maplib library is still in beta-status and its features need improvement and testing, this was chosen as the general aim of the work.

The topic of this thesis is, first, to analyse the marker interfaces of the most popular web maps for their features, differences and how easy it is to integrate them into a website.

The second task is to develop and implement an own, highly featured and yet fast and easy to integrate marker interface for khtml.maplib. A high priority lays also on the speed of the interface.

All the development is done without copying any source code or images to avoid the conflict with copyright laws.

2 Comparison of popular web maps

To find out the features and differences of the marker interfaces of the most popular web maps, their JavaScript developer API documentations are compared and the structure of the generated Document Object Model (DOM) trees are analysed. Therefore, web development tools like Firebug for Mozilla Firefox, Google Chrome's built-in developer tools and Microsoft Internet Explorer's developer toolbar are used. Additionally the necessary syntax to create a marker and an info window together with the look and feel of the result are investigated.

2.1 Google Maps

Google Maps is by far the most popular web map application in the Internet. According to BuiltWith 95% of the websites are using Google Maps to implement a map under the top million websites (cf. BuiltWith 2011).

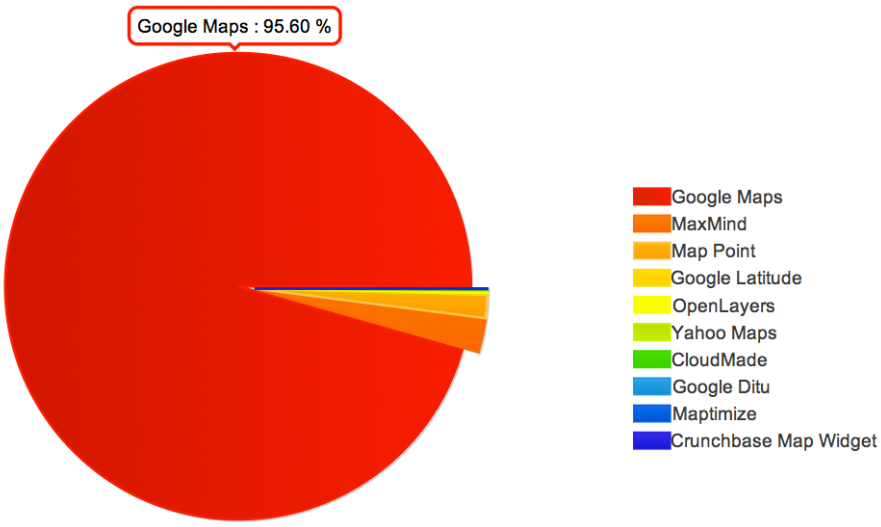

Figure 1: Mapping distribution in the top million websites (cf. BuiltWith 2011)

2.1.1 API & necessary syntax

The application programming interface (API) for Google markers and info windows is very extensive and provides a lot of features and options. They use six different classes with 28 properties in total to make their markers and info windows as customizable as possible (cf. Google 2011b).

When creating a marker, properties like the icon, a title, whether it can be dragged or not, and many more are defined in the separate *MarkerOptions* class. The *Marker* class itself has 26 different methods to manipulate

the properties during runtime. Additionally it provides 22 events to react on user inputs, like a click on the marker, drag & drop of a marker and so on.
The *MarkerImage* class is used to define the options for a custom marker image and a custom marker shadow. For a custom marker a special *MarkerShape* class defines the outlines where the marker will react on mouse events (click, mouseover, …).

To display more information about a place that is tagged with a marker, Google uses the *InfoWindow* class that provides methods and events for manipulation at runtime, together with the *InfoWindowOptions* class that is again used to define the properties when creating an info window.

The necessary syntax to create a marker on a map and open an info window on a click on the marker is quite simple. First the marker object is created and attached to the map at the same step, then the info window object is created with its content and at last the event handler is attached to the marker to open the info window.

```
// Create a marker object and display on the map
var marker = new google.maps.Marker({
      position: new google.maps.LatLng(-25.363882,131.044922),
      map: map,
      title: 'Uluru (Ayers Rock)'
});

// Create an info window object
var infowindow = new google.maps.InfoWindow({
      content: "This is an infowindow with some text!"
});

// Open the info window on a click onto the marker
google.maps.event.addListener(marker, 'click', function() {
      infowindow.open(map,marker);
});
```

Table 1: Source code for generating a marker and info window on Google Maps (cf. Google 2011f)

As it can be seen, the syntax is easy to understand and most of the options can be defined when creating the object without using a separate function, which makes it comfortable to implement.

2.1.2 Look and Feel

The above code creates a simple standard marker with a nice shadow on the map that changes the mouse cursor and shows a title when the mouse is moved over it.

Figure 2: Google map with standard marker and title (cf. Google 2011f)

With a mouse click on the marker an info window opens with a neat shadow as well. The map is moved down a little bit so that the info window is fully visible. The size of the info window and the shadow depends on the content and also on the size of the map. On maps with different sizes and also on different screens, like smartphones, the size of the info window is always adjusted to its full visibility. If the content exceeds the size of the info window, a scrollbar is added, so that the user can read the whole content.

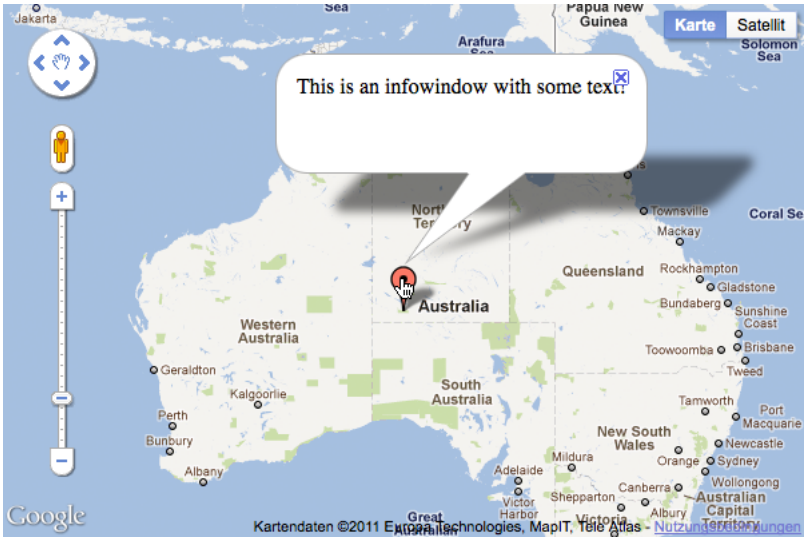

Figure 3: Google map with standard marker and info window (cf. Google 2011f)

A draggable Google marker even provides a nice drag & drop-animation that lifts the marker and its shadow up from the map and shows a little drag cross for the position to be placed on. When the marker is dragged to an edge of the map, the map is moved in the opposite direction.

Figure 4: Google map with standard marker and drag & drop animation (cf. Google 2011g)

2.1.3 DOM-tree

Every marker consists of a combination of 3 layers, each on a different level on the map. The upper layer contains the marker image together with an image map that outlines the border of the marker. This layer is invisible but used for mouse cursor change and for clicking and grabbing the marker. The middle layer also contains the same marker image and is used to visualize the marker. The lower layer contains the shadow of the marker.

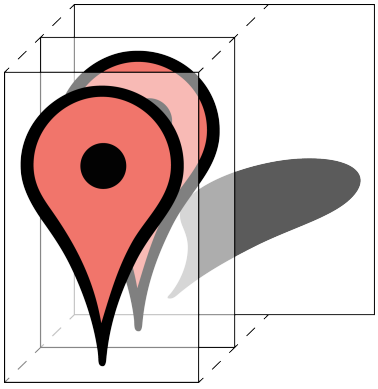

Figure 5: Google Maps standard marker structure (cf. Google 2011c)

The info window is divided into 19 separate parts, each containing a piece of the same image file. Thus the size of the info window can easily be adjusted to the content without changing the radius of the borders and the size of the pointer by just resizing the straight parts in between. The content and other accessories, like the close button, sit on top of these images in separate divisions.

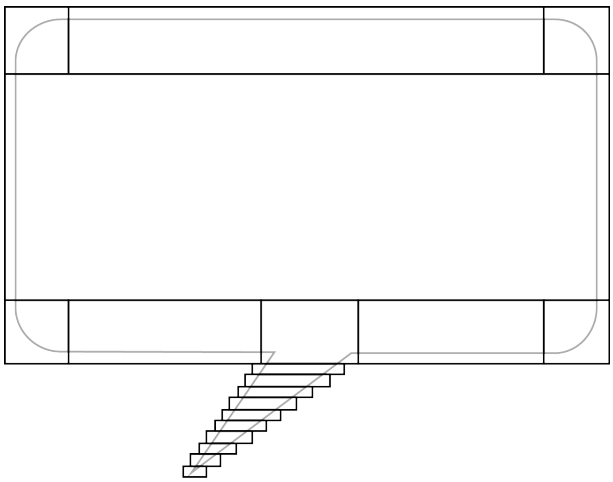

Figure 6: Google Maps info window structure (cf. Google 2011d)

The underlying shadow of the info window is also divided into 11 parts that contain fractions of the same image file again, and it can so be resized according to the info window.

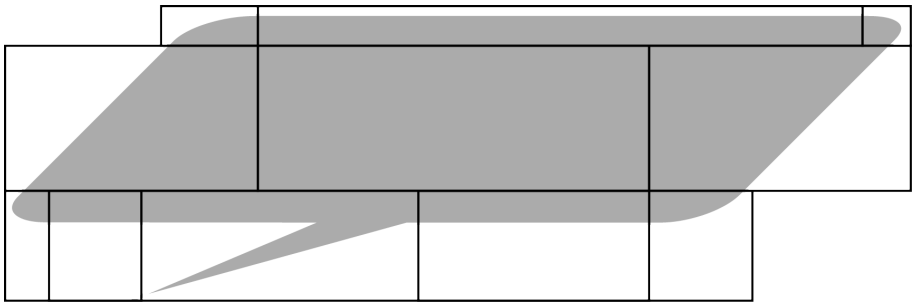

Figure 7: Google Maps info window shadow structure (cf. Google 2011e)

2.2 Microsoft Maps

2.2.1 API & necessary syntax

Microsoft's *PushPin* and *InfoBox* classes are not as sophisticated as Google's, but with 24 properties in total nearly as customizable. They also use a separate class called *PushPinOptions* to define the options for their *PushPin* and it provides 13 methods to manipulate it (move, show, hide, label, makedraggable, …) and triggers 11 events to react on user inputs (click, drag, changed, …). The properties for the *InfoBox* are defined in the *InfoBoxOptions* class and can be manipulated via 20 methods and 4 events.

The necessary syntax to create a *PushPin* with an *InfoBox* is a little bit more complex than Google's. Both objects have to be added to the map with a separate command, which makes it more complicated for the user.

```
// Create the pushpin object
var pin = new Microsoft.Maps.Pushpin(
        new Microsoft.Maps.Location(40, -120),
        {text: '1'}
);

// Add the pushpin to the map
map.entities.push(pin);

// Create the info box for the pushpin
pinInfobox = new Microsoft.Maps.Infobox(
        pin.getLocation(),
        {title: 'My Pushpin',
        description: 'This is the Location.',
        visible: false}
);

// Add a handler for the pushpin click event
Microsoft.Maps.Events.addHandler(pin, 'click', function () {
        pinInfobox.setOptions({ visible:true });
});

// Add the info box to the map
map.entities.push(pinInfobox);
```

Table 2: Source code for generating a marker and info window on Microsoft Map (cf. Microsoft 2011e)

2.2.2 Look and feel

This is what the Microsoft *PushPin* looks like on the map. It's a nice image but compared to Google's marker it lacks some features like a shadow, the change of the mouse cursor or a title when the mouse cursor is over it.

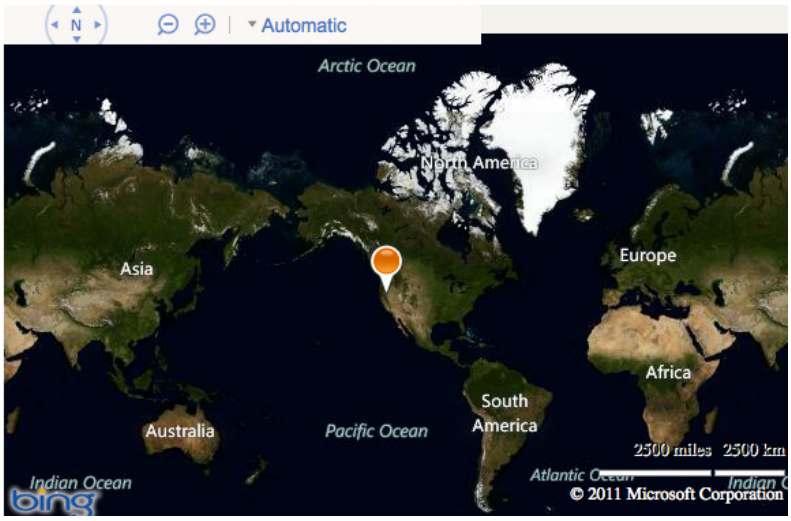

Figure 8: Microsoft map with standard marker (cf. Microsoft 2011e)

When the *PushPin* is clicked, the *InfoBox* opens right on the same location, covers the *PushPin* and does not even move the map to be fully visible.

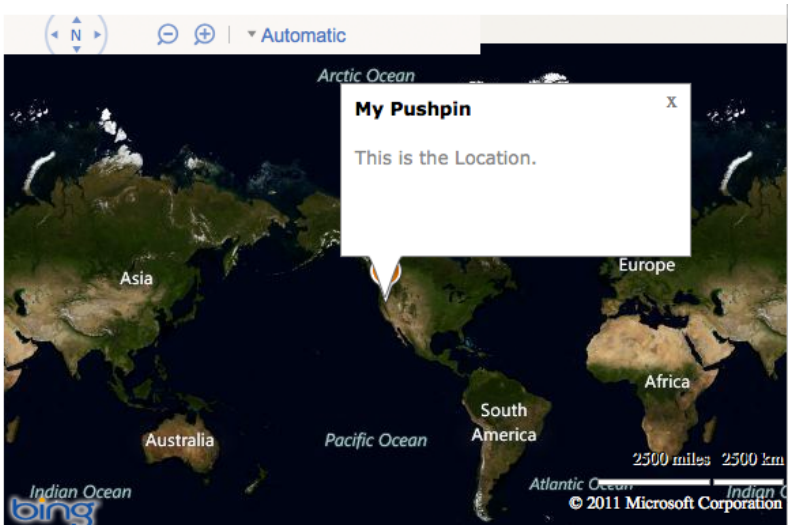

Figure 9: Microsoft map with standard marker and info window (cf. Microsoft 2011e)

2.2.3 DOM-tree

The structure in the DOM-tree of Microsoft's *PushPins* is quite simple, since it does neither provide a title on mouse over, nor a cursor change and not even a shadow. It just consists of an image in a single div-container.

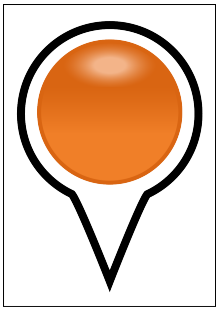

Figure 10: Microsoft Maps standard marker structure (cf. Microsoft 2011c)

The *InfoBox* is also quite simply structured. It's just one div-container with background colour and border colour for the body and one div-container with an image for the pointer.
With its sharp corners it looks a little clumsy and it does neither provide auto-resize according to the content nor a scroll bar to read the whole text. Instead the text is cut off when it exceeds the size has to be defined manually with the width- and height-property of the *InfoBoxOptions* class.

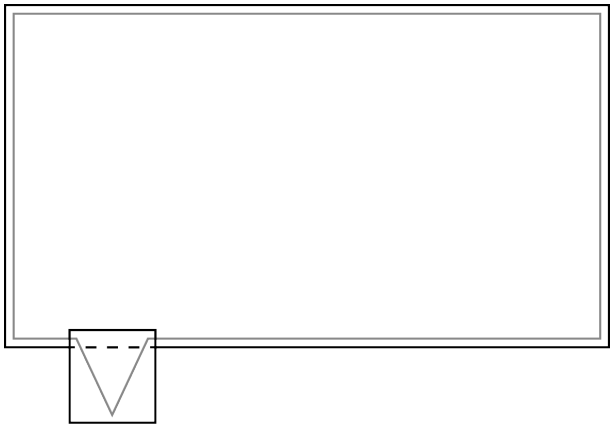

Figure 11: Microsoft Maps info window structure (cf. Microsoft 2011d)

2.3 Yahoo! Maps

2.3.1 API & necessary syntax

Yahoo's marker is a combination of only the two classes *YMarker* and *YImage*. They provide 9 properties in total, which is not much. The *SmartWindow* is not handled as a separate object but as a property of the marker. So the 17 methods and 10 events are used to manipulate the *YMarker* (show, hide, move, changeImage, …) and the *SmartWindow* (open, close, update).

The necessary syntax to create a standard marker with an info window seems to be much simpler but is only the result of the lack of features.

```
// Create the marker object
var marker = new YMarker(new YGeoPoint(40,-120));

// Add the marker to the map
map.addOverlay(marker);

// Add an event to open a SmartWindow
YEvent.Capture(marker, EventsList.MouseClick, function (){
      marker.openSmartWindow("This is a SmartWindow!");
});
```

Table 3: Source code for generating a marker and info window on Yahoo Map (cf. Yahoo 2011e)

2.3.2 Look and feel

The marker itself looks nice with its little shadow, but the info window looks a little bit clumsy.

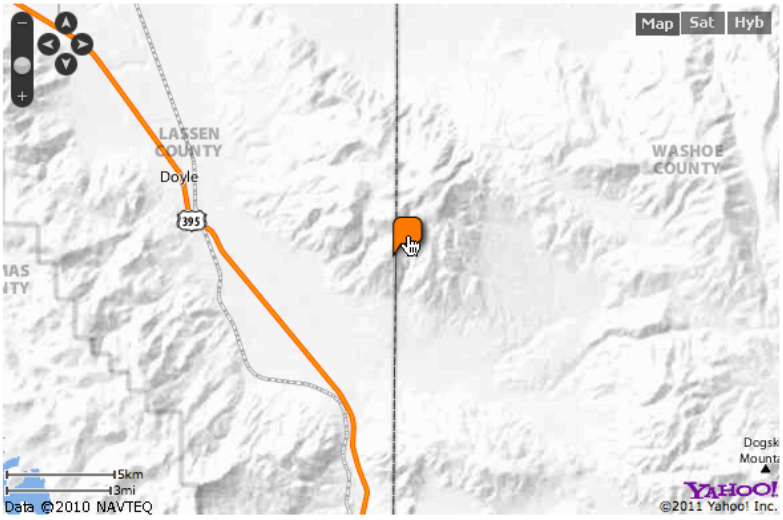

Figure 12: Yahoo map with standard marker (cf. Yahoo 2011e)

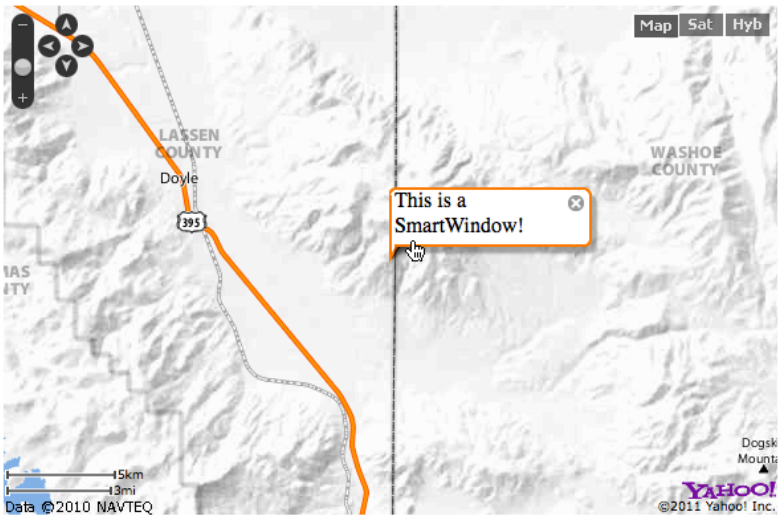

Figure 13: Yahoo map with standard marker and info window (cf. Yahoo 2011e)

2.3.3 DOM-tree

The structure of the *YMarker* is quite simple. The standard marker image has a little shadow that looks nice but is all in one image file.

Figure 14: Yahoo! Maps standard marker structure (cf. Yahoo 2011c)

The *SmartWindow* is not as simple as Microsoft's *InfoBox* but not as sophisticated as the Google's *InfoWindow*. It's combined of 9 divisions, each containing a separate image file, which makes it easily expandable according to the content. It also has a simple but nice shadow.

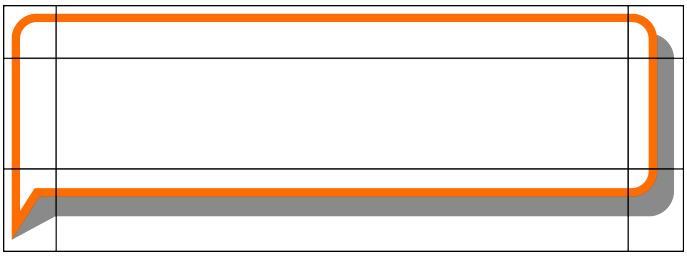

Figure 15: Yahoo! Maps info window structure (cf. Yahoo 2011d)

2.4 OpenLayers

2.4.1 API & necessary syntax

OpenLayers uses a *Marker* class together with an *Icon* class to define custom icons for the marker. Together they provide 10 properties to define it and 21 methods to modify it. The biggest difference to the other maps is that they use a special layer to place the markers on and this layer has to be created first.
For an info window they provide several different classes (*Popup*, *Anchored*, *AnchoredBubble*, *Framed* and *FramedCloud*), each extending the previous one. Together the 5 classes provide 15 properties and 33 methods to manipulate the info window.

But with that many features, it also takes more code to create a simple marker and a popup. First a marker layer has to be created and added to the map. Then a marker object can be created and added to this layer. After that the popup object can be defined with a lot of parameters that are necessary. And at last the event handler is attached to the marker that opens the info window.

```
// Create a new layer for markers
var markers = new OpenLayers.Layer.Markers( "Markers" );

// Add the layer to the map
map.addLayer(markers);

// Create a new marker object
var marker = new OpenLayers.Marker(new OpenLayers.LonLat(0,0));
// Add the marker to the map
markers.addMarker(marker);

// Create a new popup
popup = new OpenLayers.Popup.FramedCloud("chicken",
       new OpenLayers.LonLat(0,0),
       new OpenLayers.Size(200,200),
       "This is an example popup.",
       marker.icon,
       true,
       function() {
             map.removePopup(popup);
       }
);

// Open the popup at a click on the marker
marker.events.register("click", marker, function (){
      map.addPopup(popup);
}, false);
```

Table 4: Source code for generating a marker and info window on OpenLayers (cf. OpenLayers 2011d)

2.4.2 Look and feel

This is what the OpenLayers standard marker looks like. It is not much of an artwork but it's functional. It's a single image with neither a title nor a cursor change when the mouse is moved over it.

Figure 16: OpenLayers map with standard marker (cf. OpenLayers 2011d)

The *FramedCloud* looks quite nice with its round corners and the curved pointer. It also has a nice function, which always anchors it to the corner of the marker that has the largest distance to the edge of the map.

Figure 17: OpenLayers map with standard marker and info window (cf. OpenLayers 2011d)

2.4.3 DOM-tree

The structure of the standard marker is not very complex. It's a single image without any shadow and it looks like a cheap copy of the Google marker image.

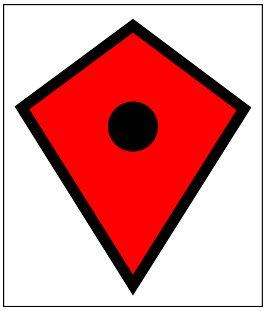

Figure 18: OpenLayers standard marker structure (cf. OpenLayers 2011b)

As opposed to the marker, the *FramedCloud* has a higher complexity. It allows it to be stretched to the content but does not make it too sophisticated.

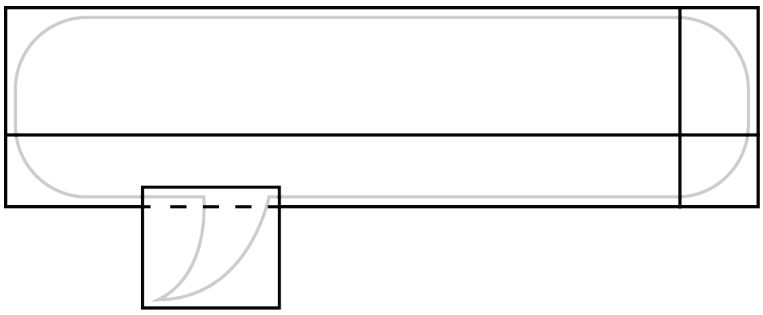

Figure 19: OpenLayers info window structure (cf. OpenLayers 2011c)

3 Marker interface of khtml.maplib

3.1 Existing marker interface

The existing marker interface of the khtml.maplib library is quite rudimental. It has only 3 parameters to be defined when creating the object, only 5 methods to manipulate the marker during runtime and 3 events to react on user inputs.

Parameter	Description
point	{<khtml.maplib.LatLng>} the position of this marker
el	{<DOM-element>} to display at the position. Can be an image or any other HTML-element.
options: {dx, dy}	{pixels} offset from the upper-left corner of the element, where to put the anchor point for the map.

Table 5: Parameters for the khtml.maplib marker class constructor (cf. KHTML 2011b)

Property	Description
el	The appended DOM-element.
marker	Parent-div of the appended DOM-element.
point	{< khtml.maplib.LatLng>} actual position of this marker
style	CSS-styling for the marker.
options	Options of the marker.

Table 6: Properties of the khtml.maplib marker class (cf. KHTML 2011b)

Method	Description
init	Called by the map when creating the marker object.
render	Draws the marker on it's position on the map.
position (pos)	Changes the position of the marker. {<khtml.maplib.LatLng>} Returns {<khtml.maplib.LatLng>} the actual position of the marker.
clear	Removes the marker from the map.
Moveable (enabled)	Makes the marker moveable / not moveable with the mouse. {Boolean}

Table 7: Methods of the khtml.maplib marker class (cf. KHTML 2011b)

Event	Description
down (evt)	Called when pressing a mouse button on the marker
up (evt)	Called when releasing a mouse button on the marker
move (evt)	Called when moving the mouse cursor over the marker

Table 8: Events of the khtml.maplib marker class (cf. KHTML 2011b)

There is no standard marker image implemented and the user has to provide the interface with a complete DOM-element to get a marker on the map. But it already has the possibility to define an offset for the anchor point of the marker element and to make the marker moveable with the mouse.

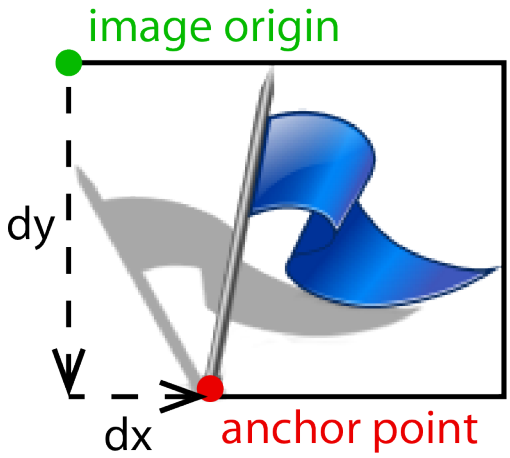

Figure 20: Pixel offset from image origin to anchor point
(cf. KHTML 2011d)

The syntax to create a marker might not be as clear for a standard user as it is for a practised programmer. First a DOM-element has to be created that contains the image. Then the image source has to be defined. After that a marker object is created with the position, the DOM-element and the distances for the offset of the anchor point to the image origin. At last the marker can be added to the map.

```
// Create a DOM-element for the marker
var img = document.createElement("img");
img.setAttribute("src","images/flag.png");

// Create a new marker with the element
var marker = new khtml.maplib.marker(
       new khtml.maplib.LatLng(30,20),
       img,
       {dx: "-10px",
        dy: "-32px"}
);

// Add the marker to the map
map.addOverlay(marker);
```

Table 9: Source code for generating a marker in old khtml.maplib API
(cf. KHTML 2011b)

3.2 New interface

Based on the knowledge gained from the research of the marker interfaces of the most popular web maps, the first attempt was to use the existing marker interface and increase it with as many features as possible. This worked out pretty well, at least for some of the features.
The option for a shadow was implemented together with the possibility to define a custom width and height for both, the marker image and the shadow. Also the nice drag & drop-animation seen at Google's marker was planned. But when it came to the implementation of a title and a cursor change when moving the mouse cursor over the marker, an option for the shape with the outlines of the marker had to be defined and that made the existing interface quite complex.

khtml.maplib.Marker (				
	point:			khtml.maplib.Point
	obj:			DOM-element or string (path to image)
	options {			
		dx:		int
		dy:		int
		width:		int
		height:		int
		shadow {		
			el:	DOM-element or string (path to image)
			dx:	int
			dy:	int
			width:	int
			height:	int
		}		
		draggable:		true/(false)
		animatedDrag:		true/(false)
		title:		string
	}			
);				

Table 10: First draft of the new khtml.maplib Marker class constructor

So the whole interfaced was revised to make it easier to understand. Google's marker interface was used as a template to make the code created with custom marker generators also usable with khtml.maplib. Google's features were increased by the possibility to add a normal DOM-element instead of an URL for a marker image to be able to display normal text on the map.

khtml.maplib.Marker (					
	khtml.maplib.overlay.MarkerOptions{				
		position			khtml.maplib.LatLng
		map			khtml.maplib.map
		icon {			khtml.maplib.overlay.MarkerImage
			url:		string (path to image) or DOM-element
			size {		
				width:	int
				height:	int
			}		
			origin {		
				x:	Int

				y:	int
			}		
			anchor {		
				x:	int
				y:	int
			}		
		}			
		shadow {			khtml.maplib.overlay.MarkerImage
			url:		string (path to image) or DOM-element
			size {		
				width:	int
				height:	int
			}		
			origin {		
				x:	int
				y:	int
			}		
			anchor {		
				x:	int
				y:	int
			}		
		}			
		draggable:			true/(false)
		raiseOnDrag:			true/(false)
		shape{			
			shape:		circle, rect, poly
			coords:		x,y,r for circle
					x1,y1,x2,y2 for rect
					x1,y1,x2,y1,...,xn,yn for poly
		}			
		title:			string
	}				
);					

Table 11: Final draft of the new khtml.maplib Marker class constructor (cf. KHTML 2011c)

For the info window the interface is much simpler. At the time of object creation, only the content has to be provided. The position is optional. The *InfoWindow* class has only 3 methods (open, close and render). When applying the open method on the info window, the map to place it on has to be supplied and, if no position is provided, an anchor object can be defined, the info window sticks to.

khtml.maplib.overlay.InfoWindow (			
	khtml.maplib.overlay.InfoWindowOptions{		
		content:	string (HTML-code)
		position:	khtml.maplib.LatLng where to place the InfoWindow when it's not attached to an anchor
	}		
);			

Table 12: New khtml.maplib InfoWindow class constructor (cf. KHTML 2011c)

This way the marker and info window interface increased to 5 classes (*Marker*, *MarkerOptions*, *MarkerImage*, *InfoWindow*, *InfoWindowOptions*) with 13 properties and 12 methods in total (cf. KHTML 2011c).

3.3 Implementation

The implementation of the new marker interface into the khtml.maplib library together with the testing was done in a Google Summer of Code project in May – August 2011.
The API was increased by a standard marker image including a shadow and a shape, the possibility to add a custom marker with custom shadow and custom shape, a title option, a nice drag & drop-animation as seen at Google's marker and the possibility to add the marker to the map at creation, without a separate command. A map move function was also added, when a marker is dragged to the edge of the map, as well as an auto re-center function when an info window is opened that is out of the map.

3.3.1 Necessary syntax

The necessary syntax to create a standard marker and an info window is quite easy and understandable. First a marker object is created with all the options, then an info window object is created and at last the event handler is attached to the marker to open the info window on a click on it.

```
// create a marker object
var marker = new khtml.maplib.overlay.Marker({
       position: new mr.LatLng(myLatlng),
       map: map,
       title: 'Uluru (Ayers Rock)',
       draggable:true
});

// Create an infowindow object
var infowindow = new khtml.maplib.overlay.InfoWindow({
       content: "This is an example infowindow"
});

// Open the infowindow on a click on the marker
marker.attachEvent( 'click', function() {
       infowindow.open(map, this);
});
```

Table 13: Source code for generating a marker and info window in new khtml.maplib API (cf. KHTML 2011c)

3.3.2 Look and feel

This is what the newly implemented standard marker looks like. It's a neat image, has a nice shadow, a title and the mouse cursor changes when the mouse is moved over it. (All of the following maps with markers are generated with the source code produced in this project.)

Figure 21: Khtml.maplib map with new standard marker and title

The info window also has a nice shadow, the size is automatically adjusted to the content and the map is moved to display the whole info window.

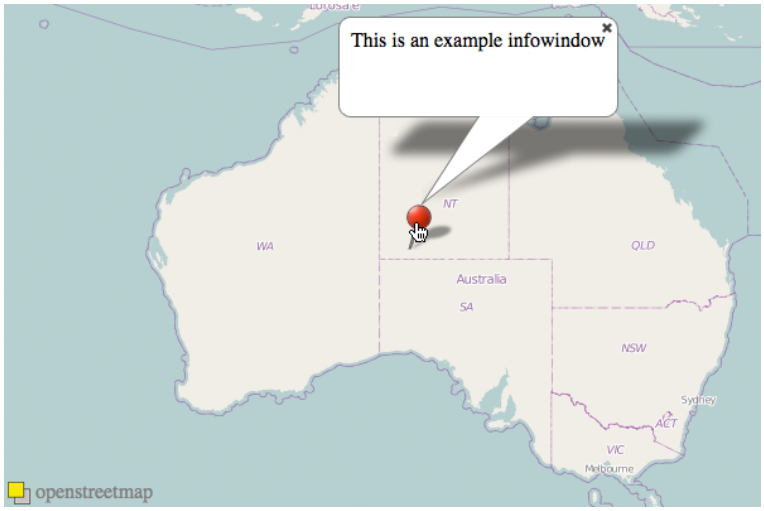

Figure 22: Khtml.maplib map with standard marker and info window

A nice drag & drop-animation, which smoothly lifts the marker and its shadow up from the map and shows a drag cross, is also included. Also a move map function is implemented that moves the map to the opposite directions when you drag a marker to an edge.

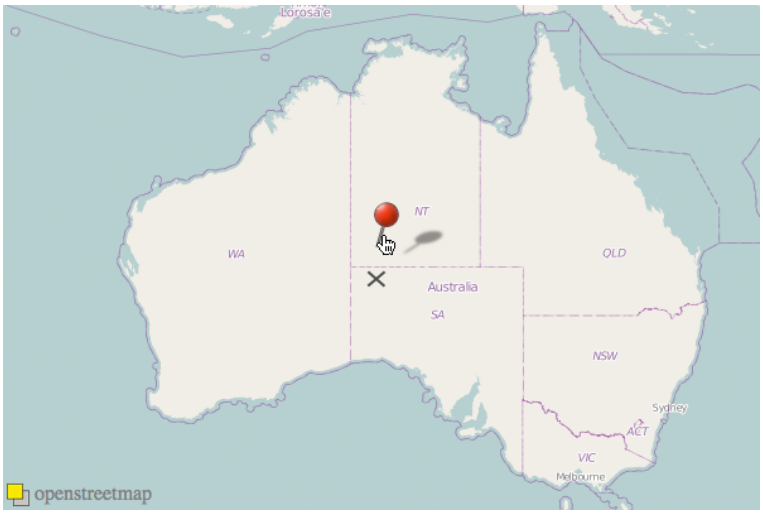

Figure 23: Khtml.maplib map with standard marker and drag & drop animation

3.3.3 DOM-tree

The structure of the marker is also inspired from Google's marker. It consists of two separate divisions, one containing the marker image, the other containing the shadow image. They are both on different layers on the map and get different *z-Index* attributes to make the marker always overlap the shadow, especially when there are many markers on one map.

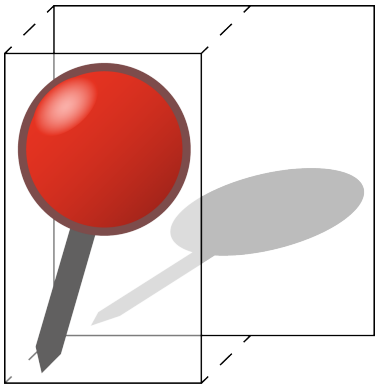

Figure 24: Khtml.maplib standard marker structure

The structure of the info window and its shadow are also constructed after the template of Google. The info window consists of 21 different divisions. Each of the 4 corners contains one quarter of the corner image file and the pointer parts all contain a part of the same pointer image file. The straight parts are constructed each by a division with the CSS-attributes *background-color* and *border-top/-right/-bottom/-left* set according to its position.

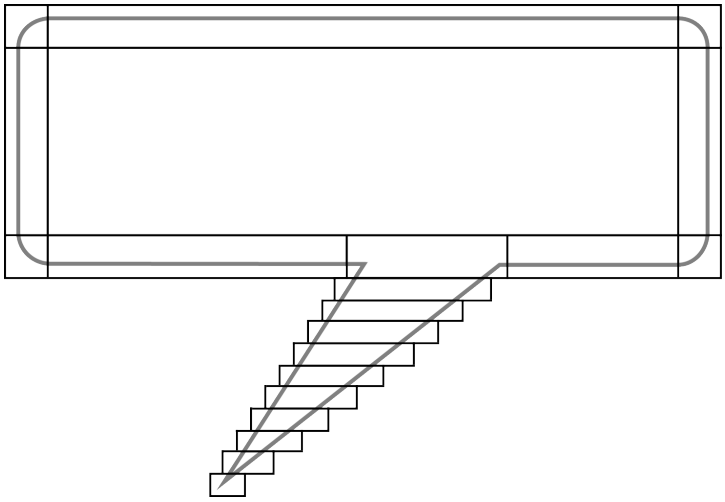

Figure 25: Khtml.maplib info window structure

The shadow consists of 11 divisions, all of them containing a part of one big image. So the size of both, the info window and its shadow can easily be adjusted to the content by just enlarging the straight parts.

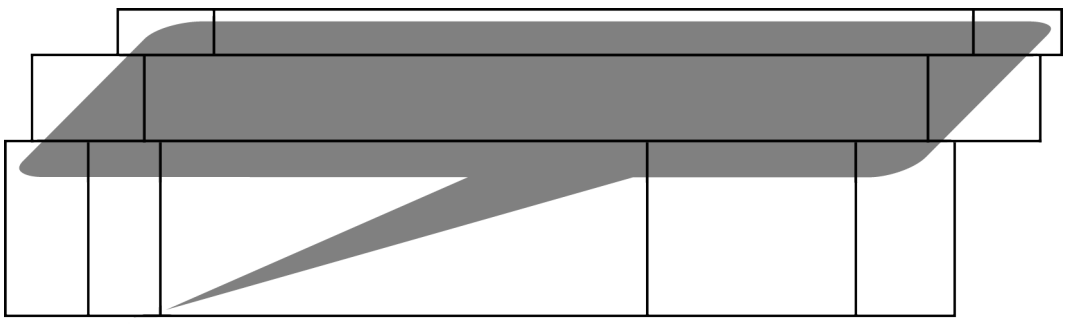

Figure 26: Khtml.maplib info window shadow structure

3.3.4 Embedded images

The following images are for the standard marker, its shadow, the drag cross, and the info window. The marker and the drag cross were drawn in Adobe Photoshop and the shadow for the marker was created with an online custom marker tool, where a custom marker can be uploaded and the website automatically creates the shadow and the outlines of the marker.

Figure 27: Khtml.maplib embedded standard marker image

Figure 28: Khtml.maplib embedded standard marker shadow

Figure 29: Khtml.maplib embedded drag cross image

The four corners of the info window are cut out of one image file to save memory. Each of them displays one quarter of this circle. This method was chosen to make the info window work on older browsers not supporting HTML5.

Figure 30: Khtml.maplib embedded info window corners image

The info window pointer is also one image file that is cut in parts and stitched together to make a cursor change on the image happen as near to the outlines of the pointer as possible.

Figure 31: Khtml.maplib embedded info window pointer image

The shadow of the info window is one large image file. Every info window gets its specific shadow that is stitched together from small parts of this large image. This shadow was also created with the custom marker tool.

Figure 32: Khtml.maplib embedded info window shadow image

The close button is a simple image that is displayed in a separate division on top of the info window and its content, and it gets an event handler attached that closes the info window on a click on it.

Figure 33: Khtml.maplib embedded close button image

All of these images are embedded into the source code of the library using the data-URI method with base64 encoding to be able to download the whole library in one file to minimize the number of HTTP requests. But on older browsers, for example Microsoft Internet Explorer 6 and 7 (IE6/7), embedded images are not supported. So all the images additionally have to be made available from a global source, to load them on demand.

3.4 Specific workarounds and performance tests

One requirement is that both, the marker and the info window, should work on older browsers and mobile devices as well. To achieve this goal quite a few workarounds are needed to differ between mobile clients and desktop computers and for different browser version. On mobile browsers, for example, a workaround is needed to make moveable markers grabbable with the touch of a finger. Because of the same reason, a different size for the close button of the info window is applied.

Figure 34: Khtml.maplib map with standard marker on Apple iPhone

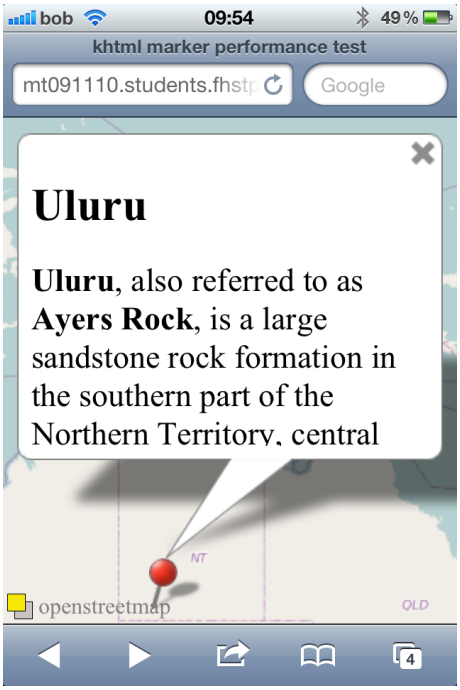

Figure 35: Khtml.maplib map with standard marker and info window on Apple iPhone

Another problem is the transparency of marker images. Therefore, in IE6 a so-called alpha-image-hack has to be applied. Otherwise all the transparent parts of the marker image would become coloured. Older browsers also do not support custom cursors by default, so the necessary cursor images for grabbing and dragging a marker have to be drawn an implemented.

Figure 36: Khtml.maplib hand cursor image

Figure 37: Khtml.maplib fist cursor image

Also a main focus is the performance when adding many markers to one map. Therefore, a test site was created, where many markers are placed on one map and the time for creating all the markers is measured. Also the frames per second drawn when the map is moved are measured. A lot of testing and comparing to the other web maps was necessary to reach this performance.

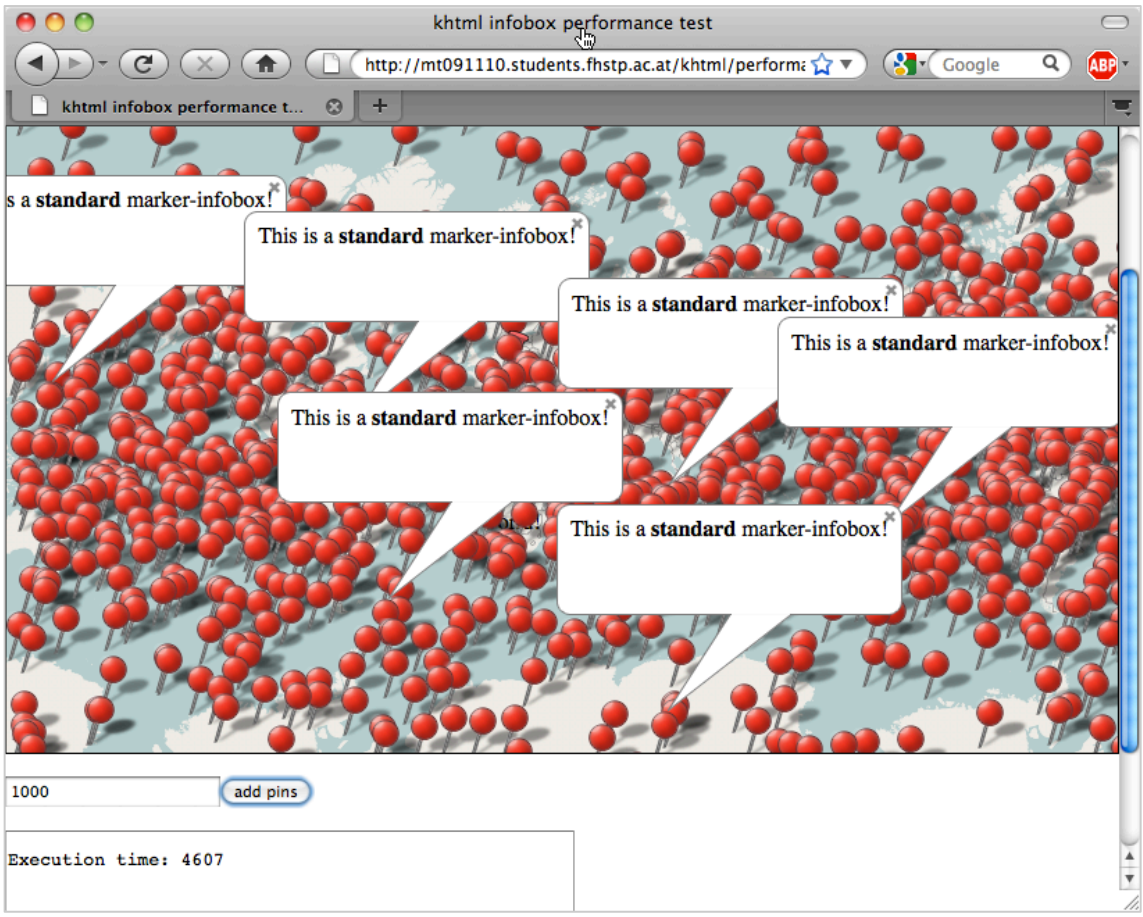

Figure 38: Khtml.maplib test website for performance and speed

Conclusion

All of the different marker-APIs have their benefits but also disadvantages. Google provides a very sophisticated API with a lot of features for a marker, many methods to manipulate them and a mass of events to react on user inputs. Yet it still remains easy to understand and use for someone who only wants to integrate a simple marker in a web app and not read pages of developer documentation to do so.
Microsoft, for example, provides a text property for their marker that is directly displayed on top of the marker. This makes it easy to number markers if you have more than one on a map, without needing a separate image for every marker. OpenLayers provides even more functionality than Google but lacks the nice shadow and drag & drop-animation. For the *FramedCloud* the user can even define a custom popup image. But all the options make it more complicated for the standard user to implement in an own website just to show one marker. Google provides enough functionality for the experienced user with their custom icon and shadow and the drag & drop-animation, but is still easy enough to implement for a standard user with a great look and feel.

The developed marker API for khtml.maplib tries to achieve this richness in functionality while keeping the simplicity to implement. In total, more than 2000 lines of source code were written for the marker and info window interface during the 4 months of the Google Summer of Code project. But so far only just a few parts of all the possibilities of a marker interface are achieved. There is still more improvement and testing needed to come close to the level that Google is on with its marker API.
Further improvements may imply more events for both, the marker and info window class, to react on user inputs, an even higher performance for many static markers for example by placing them on a HTML5 canvas element instead of directly on the map, or the clustering of markers when there are many of them in the same range.

As khtml.maplib is completely an open source project, everybody may contribute to it. The source code is freely available at github (*https://github.com/robotnic/khtml.maplib*). For news about khtml.maplib or support with problems with the library, a mailing list was created (*khtml.maplib@freelists.org*).

References

Bennett, J. (2010). OpenStreetMap - Be your own Cartographer. Birmingham: Packt Publishing Ltd. Ebook-version.

BuiltWith. (2011). Mapping Usage Statistics. [http://trends.builtwith.com/mapping (accessed on Sept. 28th 2011)]

Free Software Foundation. (2007). GNU Lesser General Public License. [http://www.gnu.org/licenses/lgpl.html (accessed on Sept. 26th 2011)]

Google. (2011). a. Nutzungsbedingungen für Google Maps. [http://www.google.com/intl/de_ALL/help/terms_maps.html (accessed on Sept. 24th 2011)]

Google. (2011). b. Google Maps Javascript API V3 Reference. [http://code.google.com/intl/en/apis/maps/documentation/javascript/reference.html (accessed on Sept. 28th 2011)]

Google. (2011). c. Google standard marker image. [http://maps.gstatic.com/mapfiles/markers/marker_sprite.png (accessed on Oct. 16th 2011)]

Google. (2011). d. Google info window image. [http://maps.gstatic.com/mapfiles/iw3.png (accessed on Oct. 16th 2011)]

Google. (2011). e. Google info window shadow image. [http://maps.gstatic.com/mapfiles/iws3.png (accessed on Oct. 16th 2011)]

Google. (2011). f. Google info window example. [http://code.google.com/intl/en/apis/maps/documentation/javascript/examples/infowindow-simple.html (accessed on Oct. 18th 2011)]

Google. (2011). g. Google marker animations example. [http://code.google.com/intl/en/apis/maps/documentation/javascript/examples/marker-animations.html (accessed on Oct. 18th 2011)]

Hazzard, E. (2011). OpenLayers 2.10 Beginner's Guide. Birmingham: Packt Publishing Ltd. Ebook-version.

KHTML. (2011). a. khtml.org Map. [http://www.khtml.org (accessed on Sept. 26th 2011)]

KHTML. (2011). b. Khtml.maplib developer API V0.97 Marker class. [http://www.khtml.org/osm/v0.97/doc/api/symbols/khtml.maplib.overlay.Marker.html (accessed on Oct. 11th 2011)]

KHTML. (2011). c. Khtml.maplib developer API V0.98.5. [http://maplib.khtml.org/maplib/v0.98.5/doc/api/ (accessed on Oct. 11th 2011)]

KHTML. (2011). d. Khtml marker pixel offset. [http://www.khtml.org/osm/v0.83/images/topleft.png (accessed on Oct. 16th 2011)]

Microsoft. (2011). a. Microsoft® Bing™ Maps platform API's Terms of Use. [https://www.microsoft.com/maps/product/terms.html (accessed on Sept. 24th 2011)]

Microsoft. (2011). b. Microsoft.Maps API Reference. [http://msdn.microsoft.com/en-us/library/gg427611.aspx (accessed on Oct. 6th 2011)]

Microsoft. (2011). c. Microsoft.Maps point of interest image.
[http://ecn.dev.virtualearth.net/mapcontrol/v7.0/i/poi_search.png (accessed on Oct. 16th 2011)]

Microsoft. (2011). d. Microsoft.Maps info window pointer image.
[http://ecn.dev.virtualearth.net/mapcontrol/v7.0/i/pointer_shadow.png (accessed on Oct. 16th 2011)]

Microsoft. (2011). e. Microsoft.Maps PushPin and InfoBox example source code.
[http://msdn.microsoft.com/en-us/library/gg508987.aspx (accessed on Oct. 18th 2011)]

Mitchell, T. (2008). Web-Mapping mit Open Source-GIS-Tools. Köln: O'Reilly Verlag GmbH & Co. KG. Ebook-version.

OpenStreetmap. (2011). a. Copyright & License.
[http://www.openstreetmap.org/copyright?copyright_locale=en (accessed on Sept. 26th 2011)]

OpenLayers. (2011). a. Free Maps for the Web. [http://www.openlayers.org (accessed on Sept. 27th 2011)]

OpenLayers. (2011). b. OpenLayers standard marker image. [http://openlayers.org/dev/img/marker.png (accessed on Oct. 16th 2011)]

OpenLayers. (2011). c. OpenLayers standard marker image. [http://openlayers.org/dev/img/cloud-popup-relative.png (accessed on Oct. 16th 2011)]

OpenLayers. (2011). d. OpenLayers marker example. [http://openlayers.org/dev/examples/markers.html (accessed on Oct. 18th 2011)]

Yahoo. (2011). a. Yahoo! Maps Terms Of Use. [http://info.yahoo.com/legal/us/yahoo/maps/mapstou/mapstou-278.html (accessed on Sept. 24th 2011)]

Yahoo. (2011). b. Maps AJAX API Reference Manual – Version 3.8.
[http://developer.yahoo.com/maps/ajax/V3.8/index.html (accessed on Oct. 7th 2011)]

Yahoo. (2011). c. Yahoo standard marker image. [http://l.yimg.com/a/i/us/map/aj/markerf19824.png (accessed on Oct. 16th 2011)]

Yahoo. (2011). d. Yahoo info window images. [http://l.yimg.com/a/i/us/map/aj/org_e.png; http://l.yimg.com/a/i/us/map/aj/org_n.png; http://l.yimg.com/a/i/us/map/aj/org_ne.png; http://l.yimg.com/a/i/us/map/aj/org_nw.png; http://l.yimg.com/a/i/us/map/aj/org_s.png; http://l.yimg.com/a/i/us/map/aj/org_se.png; http://l.yimg.com/a/i/us/map/aj/org_sw.png; http://l.yimg.com/a/i/us/map/aj/org_w.png (all accessed on Oct. 16th 2011)]

Yahoo. (2011). e. Yahoo marker example source code. [http://developer.yahoo.com/maps/ajax/#ex4 (accessed on Oct. 18th 2011)]

List of figures

List of tables