# Supporting Dynamic Labeling in Web Map Applications

Thomas Brinkhoff
Jade University of Applied Sciences
Institute for Applied Photogrammetry and
Geoinformatics (IAPG)
Ofener Str. 16/19
26121 Oldenburg, Germany
thomas.brinkhoff@jade-hs.de

**Abstract**

Web mapping based on client-side application programming interfaces (APIs) is very popular, especially for mashups. However, we can observe that most of these applications use no labeling. Thus, fast comprehensibility of such maps is hindered. In this paper, we investigate the reasons for this situation. Based on this analysis, a specification is presented that fosters the use of dynamic point labels in web map applications. This concept is abstract, simple to use, flexible and can be combined with a grid-based management of symbols. We demonstrate the applicability of this approach by a prototype that performs a browser-based dynamic labeling.

*Keywords:* Web cartography, dynamic map labeling, mashup, application programming interface (API)

## 1 Introduction

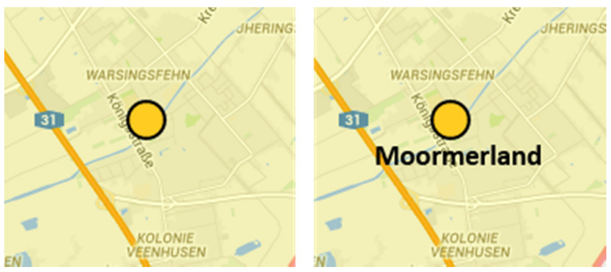
An essential task for designing meaningful and appealing maps is an adequate labeling of relevant map features. Map labeling is an established research area. Many algorithms have been developed and evaluated in the last decades. Thus, standard GIS packages provide (more or less) sophisticated tools for map labeling (Brewer 2005).

In web cartography, the situation is slightly different. First, we have to distinguish between server-side and client-side map generation. On the server side two cases typically occur: In the first case, map tiles are pre-processed. Then, the labeling can be extensively pre-computed for each zoom level. Google Maps (Figure 2a), Microsoft Maps and the OGC Web Map Tile Service (WMTS) are prominent examples for such procedure. From user and application perspective, however, such labels are fixed; it is not possible to define a different alignment or density suitable for specific applications. In the second case, the map is computed by the server dynamically according to the user request. The ISO/OGC Web Map Service (WMS) works like this. In principle, this approach allows an application- and user-defined styling and labeling by using Styled Layer Descriptions (Lupp 2007).

Map-based mashups combine application-specific georeferenced features requested from a spatial database with a base map (Brinkhoff 2007). A web browser performs the combination generally by using JavaScript-based APIs like the Google Maps JavaScript API or OpenLayers API. Former mashups worked often with symbols drawn independently from the current zoom level. The result were illegible maps showing too many symbols in smaller scales (Figure 2b). The situation has altered in the meantime: current mashups change the visibility of symbols depending on the current scale; often they combine clustered features by depicting a group symbol (together with the number of features) in smaller scales (Figure 2c).

If we consider the labeling of application-specific features, we observe following situation: Mostly no labeling is performed or the symbols show numbers referring to a list of objects depicted elsewhere (Figure 2d). What are the reasons? An obvious answer would be that we do not need a labeling anymore (a) because the labeling in base map is sufficient and/or (b) because a tooltip or an information box are provided after hovering over or clicking on a symbol. Answer (a) is clearly wrong because labels in the base map are independent from application objects. Thus, the base map provides orientation but allows no object identification. In many cases, the labels in the base map lead to misleading identifications like it is illustrated in Figure 1.

Figure 1. The symbol does not refer to the settlement “Warsingsfehn” but to the municipality “Moormerland”.



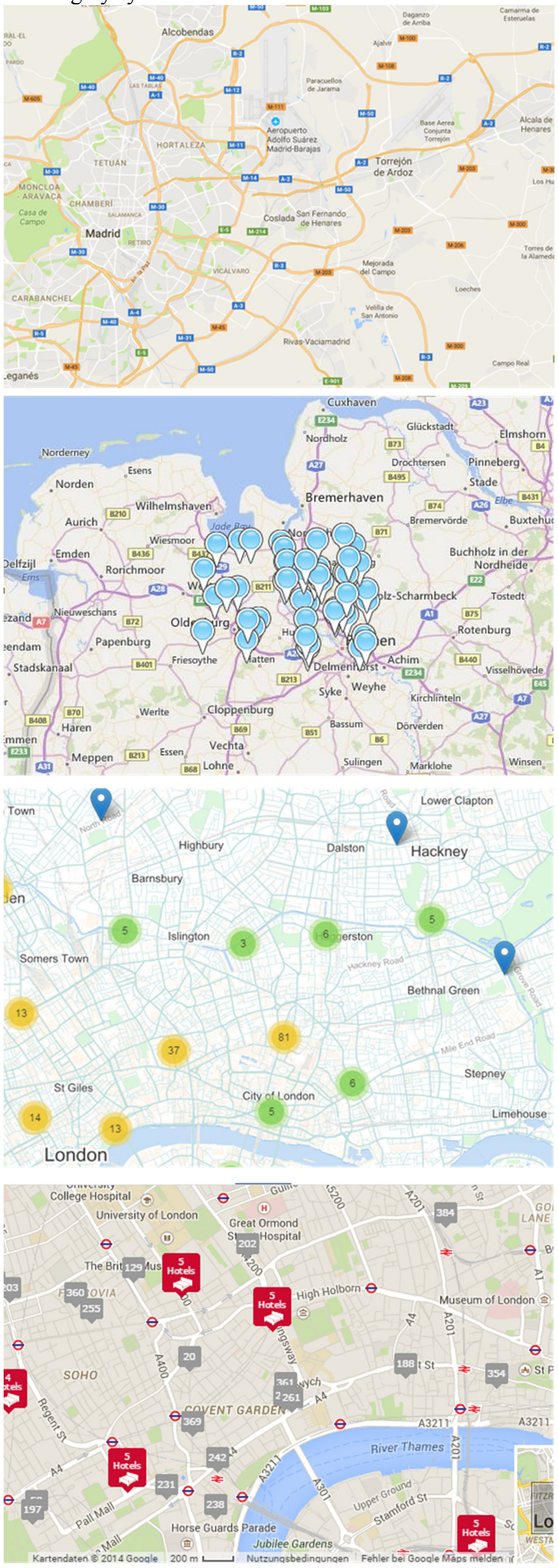


Figure 2. (a) Base map with labels, (b) map overloaded with symbols, (c) map with combined symbols, and (d) map with grey symbols that contain numbers as labels.

Tooltips and information boxes require (at least) moving the mouse cursor to a map symbol. Such an approach takes considerable time and does not provide an immediate overview like labels. Thus, there exist other – more technical – reasons for missing labels in mashups. One technical restriction is that the labeling of application-specific objects must be computed dynamically by a web client because a user can typically filter the objects that should be displayed (e.g., all hotels in the price span of 60€ to 120€ per night). Therefore, a pre-processed labeling is not helpful.

The rest of the paper is organized as follows: After presenting related work, specifications and APIs in the next section, the obstacles for labeling maps by web clients are discussed in detail. Section 4 presents extensions to existing specifications and APIs that would eliminate these hurdles for point labels. A corresponding prototype is illustrated in section 5. The paper concludes with a summary and an outlook to future work**.**

## 2 Related Work

### 2.1 Map Labeling

Positioning labels on a map is a traditional cartographic problem (Imhof 1975) (Yoeli 1972). Algorithmic labeling is topic in the field of cartography as well as in the field of algorithmic geometry. Formann & Wagner (1991) showed that this problem is NP-hard. Thus, work on map labeling tries to find good approximations but not optimal solutions. For web applications that compute labels by the client, we strive for fast algorithms. In return, the resulting labeling may have some faults, e.g., collisions or less labels than possible. Several feasible algorithms have been developed, e.g., (Wagner & Wolff 1997) (Petzold et al. 2003) (Been et al. 2006). There are also proposals with special focus on web and mobile mapping, e.g., by Bereuter & Weibel (2012) and by Zhang & Harrie (2006).

According to Been et al. (2006) we have three base operations for a fast labeling ("in interactive speed"): (1) a filtering operation that reduces the large set of labels to a manageable (i.e. much smaller) set of labels, (2) a selection operation that determines the actually displayed labels, and (3) the placement of the labels. These operations can be intertwined.

### 2.2 OGC Specifications

The OGC introduced with the Styled Layer Descriptor Implementation Specification (SLD 1.0) an XML specification for WMS styles (Lalonde 2002). It can be used for selecting predefined ("named") styles or for defining user styles for existing layers ("named layer") or user-defined geometries ("user layer"). Styling rules can be defined by scale ranges or more freely by using OGC/ISO Filter Encoding. "Symbolizers" define the styling of a concrete geometry type – text labels are supported by "TextSymbolizers" (see Section 3.1). For broadening its application range, the OGC spilt SLD 1.0 some years later into two specifications: Symbology Encoding (SE) for pure style definition (Müller 2006) and Styled Layer Descriptor Profile of the WMS (WMS-SLD 1.1) for its coupling with web map services (Lupp 2007). SE provides more styling capabilities than SLD 1.0.

### 2.3 Web Map Servers

Web map servers need style definitions especially for computing WMS raster maps on the fly. In contrast to client-side web applications, they underlie no restrictions with respect to the executing engine (web browser) or the programming language (JavaScript).

GeoServer supports SLD for defining styles. For solving label conflicts, GeoServer supports a proprietary "Priority" element. The numeric value can be constant for a layer or it can be retrieved from the feature or be calculated. "If the Priority element is not present, or if a group of labels all have the same priority, then standard SLD label conflict resolution is used. Under this strategy, the label to display out of a group of conflicting labels is chosen essentially at random." (GeoServer 2016)

The MapServer defines styles by proprietary "mapfiles". The main properties for the discussion in this paper are the "PRIORITY" parameter, an integer value that can be constant for a layer or be taken from a feature attribute, and the "POSITION" parameter that defines the position of the label relative to an anchor point (e.g., "ul" means upper left). Alternatively, an "Auto" value tells to calculate a label position automatically not interfering with other labels. If all possible positions cause a conflict, then a label is not drawn. "POSITION" can only be defined for layers (MapServer 2017).

## 3 Obstacles for a Dynamic Map Labeling

Several impediments hinder developers of web applications to perform a labeling on web clients.

### 3.1 Symbology Encoding

As discussed before, SE (Müller 2006) is the current specification for defining styles in general and text labels in particular for geospatial web services. A "TextSymbolizer" includes a "Label" element, which specifies the text of the label by an attribute name or an expression, and a "LabelPlacement" element, which is defined either by a "PointPlacement" or a "LinePlacement" element. A "PointPlacement" is suitable for labeling point symbols and allows the specification of an anchor point, a displacement and a rotation. An anchor point is relatively defined with respect to the bounding box of the label by "AnchorPointX" and "AnchorPointY" elements. "Displacement" elements allow defining a distance of the label to a symbol in a defined unit of measure. In case of a "LinePlacement", a perpendicular offset to the line, a repeat flag, a horizontal alignment, gaps and a generalization flag can be specified. The last property allows simplifying geometries.

We can summarize that SE does not provide an abstract, simple-to-use specification for visibility and text alignment that can be used for dynamic labeling. The handling of conflicts is shifted to an implementing "system".

### 3.2 Client-side Web Map APIs

There exist several APIs for programming client-side web map applications.

The *Google Maps JavaScript API* (Google 2016) does not directly support labels. Text elements can be added by user-defined overlays that consist of HTML block elements with text. The *Google Maps Utility Library* (Google 2014) provides a "Map Label" class with a fixed label per symbol.

The *ArcGIS API for JavaScript* allows specifying the labeling by using the "setLabelingInfo" method of a "FeatureLayer" (Esri 2016). This method gets "LabelClass" objects with following relevant attributes as argument:

- "labelExpression" (adjusts the formatting of labels),
- "minScale"/"maxScale" (number),
- "labelPlacement" (single (!) value of above-left/center-left/below-left/…),
- "sizeInfo" (defines the symbol size changes),
- "where" (selects labeled features by a SQL clause).

The *OpenLayers 3 API* (OpenLayers 2016) supports text labels for vector features by a "style.Text" class. Beside pure styling properties, it consists of:

- "text": the label text,
- "offsetX" / "offsetY": horizontal / vertical offset in pixels,
- "rotation",
- "scale" (number),
- "textAlign" (left/right/...) and
- "textBaseline" (bottom/top/…).

We can summarize that current client-side web map APIs provide a restricted, heterogeneous support for dynamic labeling. There exist no simple-to-use, abstract interface that allows the user to define the behavior of the labeling.

## 4 Supporting Dynamic Labeling

Web application developers should easily use and configure dynamic labeling. An individual pre-computing of the visibility range or of the alignment of each label is not a feasible solution. Dynamic labeling has to address a wide range of web map applications. Therefore, it should be as flexible as possible. On the other hand, web browsers on different types of devices (desktop PC as well as mobile devices) must handle dynamic labeling. Thus, the algorithms used should be as simple as possible; suboptimal solutions for the label placement are acceptable.

For extending an API or a document specification (like SE), we have to define a set of properties with respect to a feature, a layer or the map. Considering the previous requirements and obstacles as well as the experiences with several web applications, we propose the following properties for the case of features represented by point symbols with labels. Most of these properties are defined as functions in order to allow returning a stored attribute value as well as the result of a computation. Pure styling properties (e.g., symbol size, font family, text decoration, color, etc.) are not considered here.

***EnableLabeling (Map Property and Layer Property as Boolean)***

These properties enable or disable the labeling for the complete map or for single layers. The labeling will only be applied for a layer, if the labeling is enabled for both, the map and the layer.

***LabelText (Feature Function returns String)***

The label text may be an attribute value or is computed.

***Priority (Feature Function returns Float)***
The priority is the most abstract form to regulate the visibility of an object. A feature function provides the priority as a numerical value. This gives highest flexibility and simplifies the use by a single point of access. The function may return an attribute value, may compute the priority individually or may call a function of the layer the feature belongs to. In case of using objects from different layers, the application designer has to normalize the priorities. A default priority should exist.

***MinZoomLevelViz, MaxZoomLevelViz (Feature Functions return Integer)***
In addition, it may be reasonable to guarantee the visibility of an object. Two optional parameters allow defining an open or closed range of zoom levels. If the range is set, features will be always depicted if the current zoom level is within the range (also in the case of collisions!).

***MinZoomLevelNoViz, MaxZoomLevelNoViz (Feature Functions return Integer)***
Furthermore, it may be reasonable to restrict the visibility of an object. Two optional parameters allow defining an open or closed range of zoom levels. If the range is set, features will be never depicted if the current zoom level is outside the range.

***SymbolVisibilityBehavior (Feature Function returns String)***
A label will generally not be drawn, if the symbol is not visible. In opposite direction, the situation is not straightforward: Symbols may be depicted with or without label. Thus, we have to define this behavior. Three values are reasonable: “SymbolAlwaysWithLabel”, “SymbolMayWithoutLabel” or “SymbolWithoutLabel”. In the first case, the label of a symbol is nonetheless drawn, even if it is in an unsolvable conflict with other symbols or labels. In the second case, only the symbol would be drawn is such situations. The “SymbolWithoutLabel” can be used in cases like an opened information box. The default behavior is “SymbolAlwaysWithLabel”.

***ScaleFactor(CurrentZoomLevel as Integer) (Feature Function returns Float)***
It is often reasonable to adapt the base size of symbol and label depending on the current zoom level. This optional function returns a scaling factor for a given zoom level. Typically, the layer an object is belonging to would provide such a function. However, for some selected features a different scaling may be appropriate. A reasonable default value is 1.

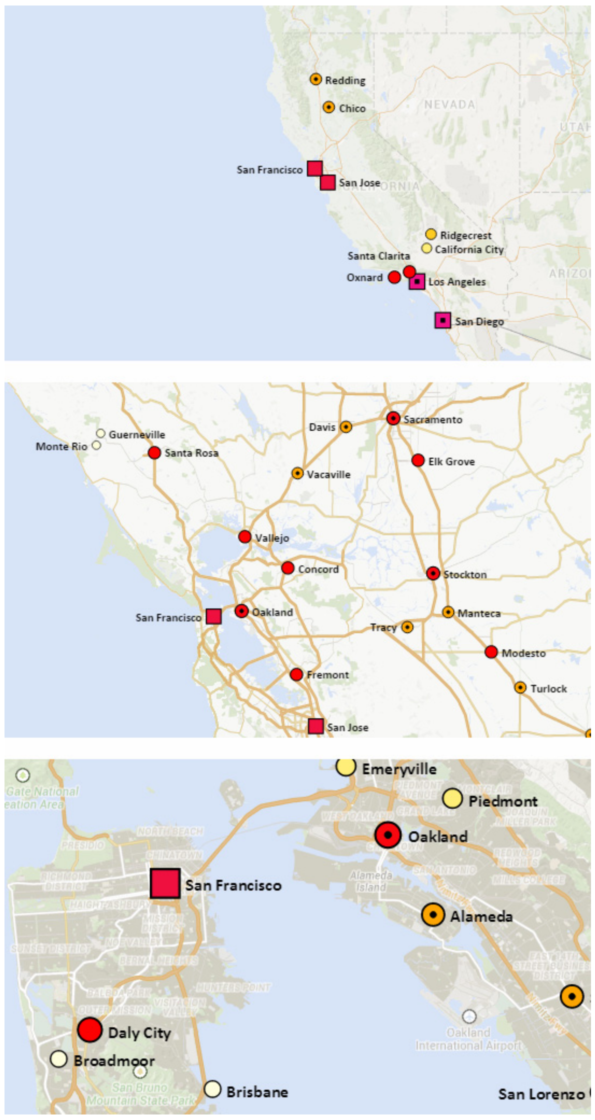
***Alignments (Feature Function returns List<Alignment>)***
MapServer distinguishes between a given alignment and an automatic alignment. That approach seems to be too restricted. If an alignment is computed (especially in larger zoom levels), there may exist several possible alignments. Some of these alignments may be preferred for special feature types or applications. Therefore, it should be possible to define a list of alignments starting with to most preferable alignment and ending with the most undesirable alignment. Alignments that are not contained in this list are not applied. A default list of alignments should be provided.

## 5 Prototype

We developed a prototype on top of the Google Maps JavaScript API for investigating the feasibility of the presented approach.

In the following examples, cities were depicted using a symbol according to their population class and a label of their name. The priority was set to the population attribute of the cities. The visibility behavior corresponds to “SymbolAlwaysWithLabel”. The scale factor depends on the zoom level. Eight different alignments were allowed in the following order: “center-east”, “center-west”, “north-center”, “south-center”, “north-east”, “north-west”, “south-east”, and “south-west”.

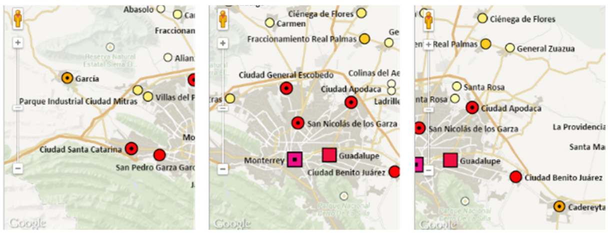
Figure 3. Map in different zoom levels.



We implemented a simple grid-based algorithm for a combined symbol selection and dynamic labeling in JavaScript. The filtering step (see Section 2.1) is performed by using the bounds of the current viewport, the selection step is done by the grid and the feature priority (i.e., *n* features with the highest priority per grid cell are selected), and finally the alignment is determined for labels of selected features by testing neighbored features for collision. Because the scale function adapts symbol and label sizes, the grid size becomes smaller with decreasing zoom levels (Figure 3).

We tested the prototype with maps consisting of several thousand points. Even in such cases, dynamic labeling leads to no noticeable delay. Thus, it is also triggered every second while a pan operation is performed (Figure 4).

Figure 4. Map panned from west to east.



## 6 Conclusions

Several technical obstacles for missing support of dynamic labels have been identified and discussed in this paper. We need especially simple-to-use APIs that support dynamic labeling for web applications. A concept for introducing labeling into web map APIs for the case of point symbols has been presented. Its main advantages are:

- The concept is sufficiently abstract – visibility is primarily controlled by a priority function and not by individual scale ranges.
- The concept is simple to use by a defined default behavior. The only required steps are to define a label text function and to enable the labeling.
- The concept is flexible; it allows the definition of additional properties for improving the labeling or for handling special situations.
- The concept can be combined with a grid-based management of symbols.

Future work consists of extending the proposed concept to line and area features. In addition, a prototype for the OpenLayers API is planned. Finally, performance investigations with respect to JavaScript environments are of interest.

## References

Been K., Daiches E. & Yap C. (2006) Dynamic Map Labeling. IEEE Transactions on Visualization and Computer Graphics, 12(6): 773–780.

Bereuter P. & Weibel R. (2012) Real-time generalization of point data in mobile and web mapping using quadtrees. Cartography and Geographic Information Science 40(4): 271–281.

Brewer C.A. (2005) Comparison of GIS and Graphics Software for Advanced Cartographic Symbolization and Labeling: Five GIS Projects. Proceedings of the International Cartographic Association Conference.

Brinkhoff T. (2007) Increasing the Fitness of OGC-Compliant Web Map Services for the Web 2.0. 10th AGILE International Conference on Geographic Information Science. In: The European Information Society, Lecture Notes in Geoinformation and Cartography, 247–264.

Esri Inc. (2016) ArcGIS API for JavaScript Reference 3.19. https://developers.arcgis.com/javascript/jsapi/. [Accessed 4 February 2017].

Formann M. & Wagner F. (1991) A packing problem with applications to lettering of maps. Proceedings 7th Annual ACM Symposium on Computational Geometry, 281–288.

GeoServer (2016) GeoServer 2.10.x User Manual – SLD Styling. http://docs.geoserver.org/stable/en/user/styling/sld/index.html. [Accessed 4 February 2017].

Google Inc. (2014) Map Label – A Google Maps JavaScript API utility library. https://github.com/googlemaps/js-map-label. [Accessed 4 February 2017].

Google Inc. (2016) Google Maps Javascript API V3 Reference. https://developers.google.com/maps/documentation/javascript/reference. [Accessed 4 February 2017].

Imhof E. (1975) Positioning Names on Maps. The American Cartographer, 2(2):128–144.

Lalonde W. (ed.) (2002) OGC Styled Layer Descriptor Implementation Specification, Version 1.0.0, OGC 02-070.

Lupp M. (ed.) (2007) OGC Styled Layer Descriptor profile of the Web Map Service Implementation Specification, Version 1.1.0 (revision 4), OGC 05-078r4.

MapServer (2017) MapServer 7.0.4 Documentation – Map File – LABEL. http://mapserver.org/mapfile/label.html. [Accessed 4 February 2017].

Müller M. (ed.) (2006) OGC Symbology Encoding Implementation Specification, Version 1.0.0 (revision 4), OGC 05-077r4.

OpenLayers (2016) OpenLayers 3 API Documentation. http://openlayers.org/en/latest/apidoc/. [Accessed 4 February 2017].

Petzold I., Gröger G. & Plümer L. (2003) Fast Screen Map Labeling – Data Structures and Algorithms. Proceedings 23rd International Cartographic Conference, 288–298.

Wagner F. & Wolff A. (1997) A practical map labeling algorithm. Computational Geometry 7:387–404.

Yoeli P. (1972) The logic of automated map lettering. The Cartographic Journal, 9(2):99–108.

Zhang Q. & Harrie L. (2006) Real-Time Map Labelling for Mobile Applications. Computers, Environment and Urban Systems, 30(6): 773–783.