

deCarta's Open Architecture for AJAX Draggable Maps

Geoffrey Hendrey, *Senior Software Architect*, Brent Hamby, *Senior Software Engineer*

deCarta, 4 North Second Street, Suite 950, San Jose, CA 95113

Abstract—AJAX mapping applications are rapidly replacing traditional “click-and-wait” applications for web-based map, local search, driving direction and other map-centric applications. The technical details of AJAX mapping applications have been hard to find as most well-known implementations use obfuscated JavaScript for both performance and security reasons. This article describes deCarta's open architecture for browser-based mapping applications. The cornerstones of deCarta's architecture are XML web services and open-source JavaScript. deCarta's Drill Down Server (DDS) Web Services is available as a standard “click-to-run” software delivery. A commercial hosted option is also available including a free “Developer Zone” that allows developers to prototype and customize their application. deCarta's DDS Web Services builds on the trusted, highly-scalable technology of DDS and provides the industry's first off-the-shelf solution for custom draggable maps.

Index Terms—AJAX, OpenLS, Web Services, XML, Maps, Tiling

I. INTRODUCTION

THIS document¹ assumes a basic familiarity with web services, XML, and JavaScript. Collectively, these technologies are used to create so-called “AJAX” (Asynchronous JavaScript and XML) applications. The characteristics of an AJAX application are responsiveness and interactivity. The terms “tiling” and “draggable map” have become synonyms for an AJAX-style mapping and driving directions application. Such applications are rapidly phasing out traditional web-based mapping applications.

Developers tasked with building an AJAX mapping application are faced with many challenges, such as:

1. Developing JavaScript that is portable across browsers
2. Managing asynchronous XML communication with their server
3. Designing client architecture for off-screen map loading

Geoffrey Hendrey has been a Senior Software Architect at deCarta since 2001. e-mail: ghendrey@deCarta.com

Brent Hamby has been a Senior Software Engineer with deCarta since 2002. e-mail: bhamby@deCarta.com

¹ The content of this document is the subject of a pending patent.

4. Designing the client architecture for managing a rotating grid of tiles with the JavaScript event model
5. Managing map tiles in the context of spatial operations such as panning and zooming to create a fluid user experience
6. Architecting a scalable server environment capable of handling massive load differentials compared to a static map application
7. Designing a map that is differentiated and branded in terms of fonts, colors, and feature densities
8. Overlaying route geometry and points of interest (POIs)
9. Robustly handling sloppy user address input including typos, wrong postal codes, and non-existent street numbers

II. BUILDING ON WEB SERVICES

deCarta's Drill Down Server® (DDS) is a high-performance geospatial server that has been recognized for years for its ability to handle high volumes of maps and routes. More recently, deCarta has developed a standards-compliant XML-based web services layer that runs on top of DDS. Called “DDS Web Services”, it is available in both a hosted and “host-your-own” model. This document focuses on the explicit TileGrid support provided by DDS Web Services. This document is not a replacement for the *DDS Web Services Application Developer's Guide* available to DDS licensees and deCarta Developer Zone users. All of the standard features provided by DDS Web Services are fully compatible with the TileGrid, including:

- Directory service (POI)
- Routing service
- Geocoding and Reverse Geocoding service
- Traffic Reporting (upcoming)

It is suggested that readers log onto <http://developer.deCarta.com>, download the DDS Web Services client distribution from the deCarta Developer Zone and familiarize themselves with the general concepts of programming to DDS Web Services. These concepts include:

- The OpenLS XML for Location Services standard

(XLS)

- Optional Java client API
- Web-based XML development and debugging tools
- Sample applications and sample code

III. TILE GRID CONCEPT

The centerpiece of the JavaScript mapping application is the grid of map tiles. First we will explain the grid conceptually, and then describe its implementation in JavaScript.

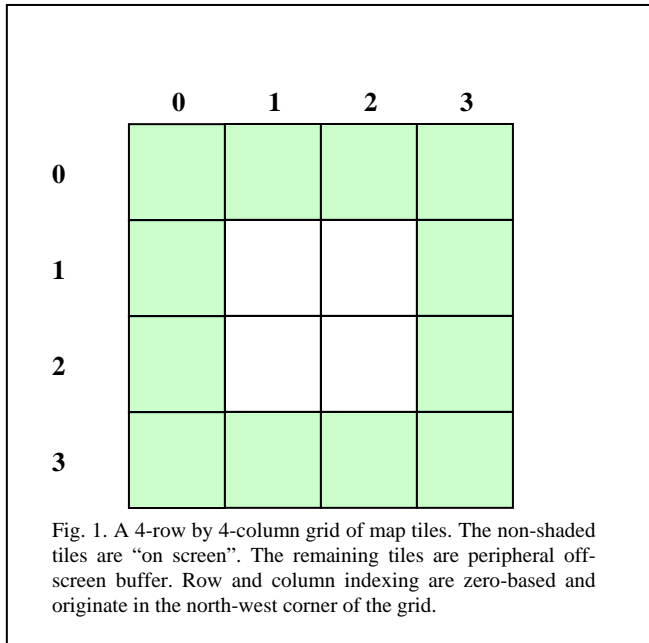


Fig. 1. A 4-row by 4-column grid of map tiles. The non-shaded tiles are “on screen”. The remaining tiles are peripheral off-screen buffer. Row and column indexing are zero-based and originate in the north-west corner of the grid.

The map is comprised of a grid of M rows and N columns. The application may choose any value for M and N. Each tile is an image representing a portion of the overall map. The image width and height must be the same for each image in the grid, but the application can choose any pixel width and height it deems sensible. Practice has shown that a 4 by 4 grid comprised of 300-pixel by 300-pixel image tiles is a robust combination.

In addition to the grid dimensions and image pixel dimensions, the application designer may choose to use some of the tiles as an off-screen peripheral buffer. Such a buffer is desirable because the peripheral images will be loaded before the user drags them on screen. This causes new “offmap” tiles to be loaded into the off-screen buffer for typical mouse drags, which minimizes the visibility of tile loading and increases the user’s perception that the map is a single, continuous sheet, as opposed to a grid of tiles.

Using DDS Web Services, the URL for each tile in the grid can be retrieved using a single XML query. The XML query is identical to a PortrayMapRequest XML query, with the addition of a single element called TileGrid.

```

1 <PortrayMapRequest>
2   <Output width="300" height="300" format="GIF">
3     <CenterAddress>
4       <Radius unit="KM">1</Radius>
5       <Address countryCode="US">
6         <freeFormAddress>
7           4 n 2nd st, san jose ca
8         </freeFormAddress>
9       </Address>
10    </CenterAddress>
11    <TileGrid rows="4" columns="4"/>
12  </Output>
13 </PortrayMapRequest>

```

Listing. 1. A PortrayMapRequest with a TileGrid requested in the output. The server returns a PortrayMapResponse containing a TileGrid element populated with individual Tile elements.

Listing 1 shows a DDS Web Services PortrayMapRequest. Line 11 of the listing shows the addition of a TileGrid element to the request. All other details of the PortrayMapRequest are identical to the non-tiled version of the request. Line 2 shows the pixel width and height attributes. When the TileGrid element is present, the pixel width and height apply to EACH TILE in the grid. Therefore, for the example shown in Listing 1, each image tile will be a 300 × 300 GIF image.

IV. TILE GRID SPATIAL POSITIONING

Unlike other web-based map tiling systems that pre-render, store and cache every tile at every zoom level for the entire geographic coverage, DDS Web Services supports on-the-fly rendering of tiles. This allows the grid to be positioned in an arbitrary location, at an arbitrary “zoom” level. In Listing 1 a FreeFormAddress is used to position the exact center of the tile grid at the specified address. This has tremendous advantages for simplifying the JavaScript code. Essentially, the single-line address entered by the user can be inserted directly into the PortrayMapRequest resulting in a perfectly centered grid. No adjustment is needed to center the grid, as would be the case for a tiling system relying on pre-rendering.

As with any PortrayMapRequest, the spatial positioning of the grid can be accomplished by either a BoundingBox, CenterContext, or CenterAddress. Figure 2 shows the spatial extent of a TileGrid corresponding to a 1 km CenterAddress. The CenterAddress is an innovative feature developed by deCarta as a result of feedback from our developer community. In the past, applications had to make two requests in order to retrieve a map. First, an address had to be geocoded to a lat/lon. Second, the lat/lon had to be used to populate either a BoundingBox or a CenterContext. This forced an application to make two round trips to the server, as well as necessitating handling code for the initial geocode request.

deCarta’s solution is the CenterAddress. When a CenterAddress is used, the server automatically geocodes the address, and returns a single image or grid of images that is centered on the highest-quality address candidate match. The response from the server also includes the alternate address

candidates. This allows the application to easily present alternate address choices to the user. The CenterAddress and many other features of DDS Web Services have been developed to reduce the amount of client-server communication and minimize handling code on the client.

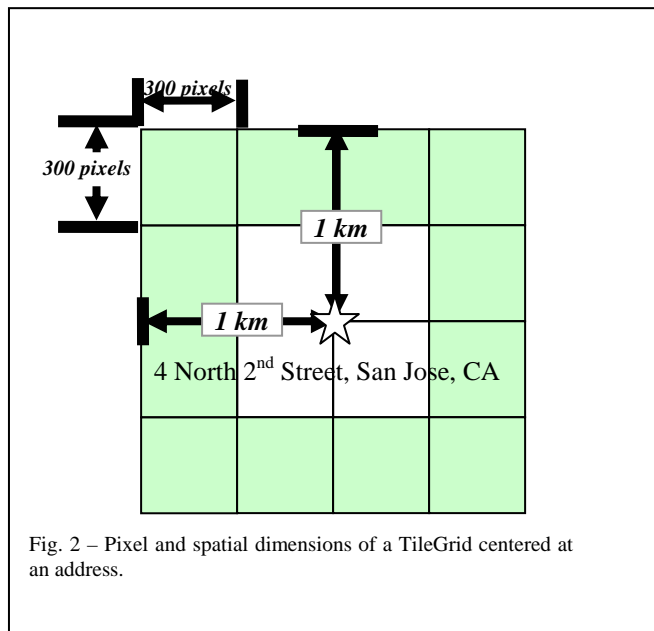


Fig. 2 – Pixel and spatial dimensions of a TileGrid centered at an address.

V. NORTHING AND EASTING PARAMETERS

In response to a PortrayMapRequest, the server returns a PortrayMapResponse containing a TileGrid element populated with individual Tile elements. Several important pieces of information are returned in the tile grid:

1. The overall spatial extent of the TileGrid
2. The row and column index of each Tile element
3. The spatial extent of each Tile
4. An image URL for each Tile

```

1 <TileGrid columns="4" rows="4">
2 <CenterContext SRS="WGS-84">
3 <CenterPoint>
4 <pos>41.002 -72.000896</pos>
5 </CenterPoint>
6 <Radius unit="KM">1.0</Radius>
7 </CenterContext>
8 <Tile col="0" row="0">
9 <Map>
10 <Content height="300" format="GIF"
11 width="300">
12 <URL>http://ws.deCarta.com:8080/opens/image/TILE?
13 LLMIN=41.00361987041037,-72.00447335745504
14 &LLMAX=41.00469978401728,-72.00304241447303
15 &WIDTH=300&HEIGHT=300&FORMAT=GIF&
16 CLIENTNAME=someclient&SESSIONID=999&
17 N=0&E=0</URL>
18 </Content>
19 <BBoxContext>
20 <pos>41.00361987041037 -
    72.00447335745504</pos>
    
```

```

21 <pos>41.00469978401728 -
22 72.00304241447303</pos>
23 </BBoxContext>
24 </ns1:Map>
25 </ns1:Tile>
    
```

Listing. 2. A TileGrid element taken from a PortrayMapResponse. Only the first Tile in the TileGrid is shown.

Listing 2 shows a single Tile element from within the TileGrid returned in the PortrayMapResponse. The URL for each Tile image is not a static file URL, but rather a set of query parameters that allow the tile to be rendered “just in time”. The client can ignore all the query parameters in the URL, except for the last two parameters. These parameters are highlighted in Listing 2 and are best described as the “Northing” and “Easting” parameters. As we shall see in the next section, the Northing and Easting parameters are essential components to a simple map dragging architecture.

VI. DRAGGING AND TILE FLIPPING

The JavaScript architecture for map dragging uses the HTML ‘DIV’ element. The DIV is used as a container for the image tiles. Dragging is accomplished by detecting JavaScript ‘onmousedown’ and ‘onmousemove’ events and altering the DIV’s CSS “top” and “left” attributes.

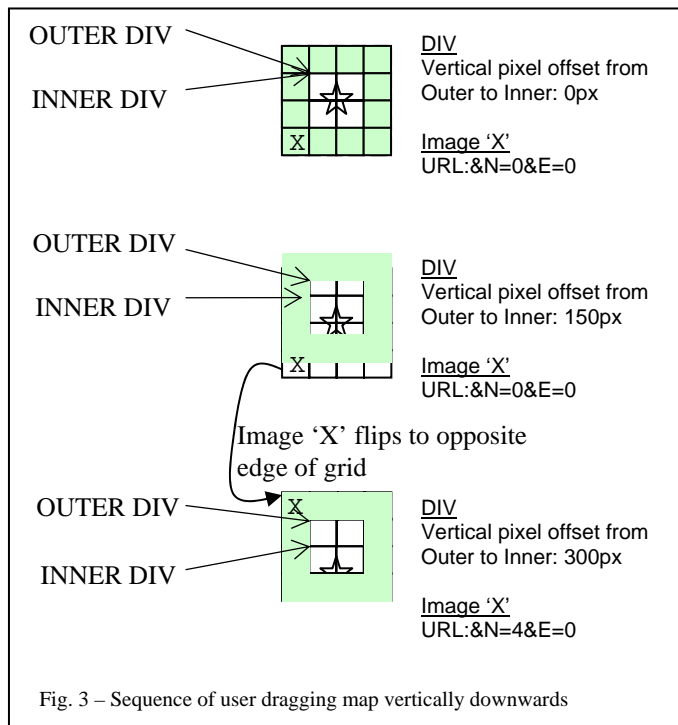


Fig. 3 – Sequence of user dragging map vertically downwards

Figure 3 shows a sequence in which the user drags the map down the page. The star represents the CenterAddress of the TileGrid. The outer DIV frames the inner DIV to create a viewable area centered on the inner 4 tiles. The CSS ‘overflow’ attribute is set to ‘hidden’ on the outer DIV so that off-screen peripheral tiles are not seen. The outer DIV has a higher CSS

Z-index than the inner DIV.

In the first snapshot, the map has not been dragged. Therefore, the inner DIV has a vertical offset of zero from its enclosing outer DIV. The 300-pixel by 300-pixel tile image labeled 'X' has a URL with zero Northing and zero Easting. Tile 'X' has an initial vertical offset within the DIV of 600 pixels.

The second snapshot in Figure 3 shows that the user has dragged the map 150 pixels downwards. Neither of the images has flipped nor changed its offset within the inner DIV. Only the inner DIV has moved relative to the outer DIV.

In the third snapshot of Figure 3, the inner DIV has been pulled down 1 complete tile (300 pixels) from the enclosing outer DIV. At this moment, the image labeled 'X' has flipped from the bottom of the grid to the top of the grid, acquiring an offset of -300 relative to the inner DIV. Similarly, the image's URL acquires a Northing value of 4. This is computed by adding the entire grid height (4 tiles) from the image's initial Northing of zero tiles.

VII. ZOOMING

The TileGrid element includes explicit support to allow zoom functionality to be applied in conjunction with panning. At first, one might think that the JavaScript client would be responsible for computing the spatial location resulting from the user's drag operations, prior to zooming. How else would it be possible to determine which tiles to load for the zoomed view?

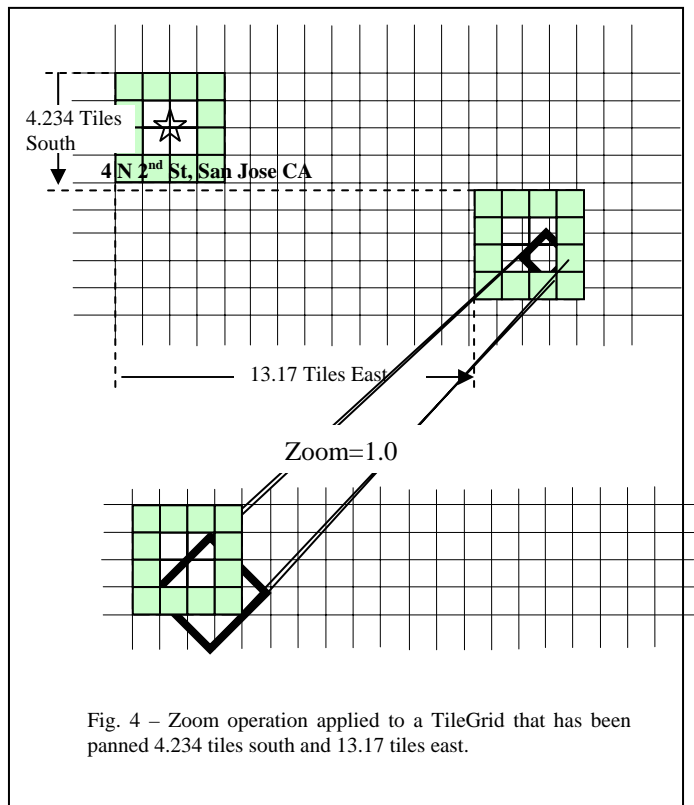


Fig. 4 – Zoom operation applied to a TileGrid that has been panned 4.234 tiles south and 13.17 tiles east.

The answer lies in allowing the client to track the user's dragging in the simplest manner possible. The client does not

need to compute the lat/lon that the map has been dragged prior to zooming. Instead, the TileGrid request allows the client to simply specify the number of tiles that the map has been dragged, in any direction. This is accomplished by the inclusion of Pan elements in the TileGrid request.

Figure 4 shows a TileGrid that the user has dragged 4.234 tiles south and 13.17 tiles west. A zoom operation is then applied to the TileGrid, resulting in a fresh TileGrid returned from the server.

```

1 <PortrayMapRequest>
2 <Output width="300" height="300" zoom="1.0">
3 <CenterContext SRS="WGS-84">
4 <CenterPoint>
5 <pos>41.002 -72.000896</pos>
6 </CenterPoint>
7 <Radius unit="KM">1.0</Radius>
8 </CenterContext>
9 <TileGrid rows="4" columns="4">
10 <Pan direction="E" numTiles="13.17"/>
11 <Pan direction="S" numTiles="4.234"/>
12 </TileGrid>
13 </Output>
14 </PortrayMapRequest>
    
```

Listing. 3. A PortrayMapRequest combining a zoom on top of a panned TileGrid

Listing 3 shows a PortrayMapRequest that requests a zoom factor of 1.0. Line 2 shows the zoom attribute. The request includes information about how the existing TileGrid has been panned. Lines 10 and 11 show the discreet Pan elements that tell the server how much the user has dragged the map. The server needs this information because it must apply the zoom factor to the final location the user dragged the map to. Note that while the client could have provided original center address, in this case the client has utilized the CenterContext provided in the response shown in Listing 2. This is shown on lines 3 to 8 of Listing 3. By utilizing the latitude/longitude-based CenterContext provided by the server along with the initial TileGrid response, geocoding is only performed on the initial positioning of the TileGrid, rather than on each subsequent zoom operation.

VIII. ROUTES AND OVERLAYS

DDS Web Services supports rendering Overlay elements on the TileGrid. Overlays can be RouteGeometry, POIs, text, etc. An advantage of dynamic tile rendering, as opposed to pre-rendered tiles, is that all routes and other overlays are part of the image tiles. In systems that use pre-rendered tiles, routes must be rendered as a PNG with a transparent background, and layered on top of the tiles in the JavaScript client, increasing the complexity of the system.

Overlays are specified as part of the initial TileGrid request and are associated with tiles via the sessionId. If a sessionId is not provided in the request, the server will return a unique sessionId. The URL for individual tile images will include this sessionId. If an application provides a sessionId, the

application should choose a sessionID that is unique to each user so as not to “spill” routes and other overlays onto a different user’s map.

```

1 <PortrayMapRequest fitOverlays="true">
2   <Output height="400" width="400">
3     <CenterAddress>
4       <Radius unit="KM">5</Radius>
5       <Address countryCode="US">
6         <freeFormAddress>300 Second Ave,
7         11111</freeFormAddress>
8       </Address>
9     </CenterAddress>
10    <TileGrid rows="3" columns="3"/>
11  </Output>
12  <Overlay>
13    <RouteGeometry>
14      <LineString ">
15        <pos>41.0039 -72.003</pos>
16        <pos>41.003 -72.003</pos>
17        <pos>41.003 -72.003</pos>
18        <pos>41.002 -72.003</pos>
19        <pos>41.002 -72.003</pos>
20        <pos>41.001 -72.003</pos>
21        <pos>41.001 -72.002</pos>
22        <pos>41.001 -72.002</pos>
23        <pos>41.001 -72.001</pos>
24        <pos>41.001 -72.001</pos>
25        <pos>41.001 -72.00011</pos>
26      </LineString>
27    </RouteGeometry>
28  </Overlay>
29 </PortrayMapRequest>

```

Listing. 4. A request for a TileGrid with a route drawn on the tiles.

Listing 4 shows a PortrayMapRequest that includes a TileGrid and RouteGeometry to overlay on the tiles. Line 1 shows the FitOverlays attribute. This attribute will cause the spatial extent of the tile grid to “stretch” to cover all the overlays. Lines 13-27 show the route geometry. In a typical AJAX application, an initial DeterminRouteRequest would be used to obtain the route geometry, and a subsequent PortrayMapRequest would be made to overlay the geometry, as shown in Listing 4. Subsequent zoom operations must also include the route geometry.

IX. SWITCHING THE LOOK AND FEEL

The look and feel of a map tile includes factors such as color scheme, label placement, feature density at different zoom levels and image download time. DDS Web Services uses a simple attribute named “Configuration” to apply image rendering styles and dataset selection to a TileGrid. Internally, DDS Web Services maps this configuration name to DDS image settings, which are highly customizable and include explicit support for tiled imagery. Unlike a pre-rendered tiling system, the configuration attribute can be changed on the fly—producing a very different look and feel of the map. This

allows an AJAX application to present the user with a choice of styles, typically through a dropdown menu. Style choices can range from functional (selecting a map style with high contrast and low feature density to facilitate printing) to humorous (making the map look like an “ancient” map of Middle Earth).

X. FREEFORM GEOCODING

The entry point into most web-based mapping applications is the “address” entry form. In the past, the entry form has typically included separate fields for “street”, “city”, and “postal code”. However, advanced geocoders are now allowing users to enter their address in a single line.

The key to effective single-line freeform geocoding is an advanced freeform parser that is tolerant of typos and robust with regard to punctuation and style of address. For example, the address “second and santa clara, san jose” contains a number of ambiguities: there are multiple streets in San Jose named “santa clara” and Santa Clara is also a neighboring city and a county in California. Furthermore, there is more than one municipality named “san jose” in the United States.

The DDS Web Services geocoder excels at handling a variety of input formats:

1. City only
2. State only
3. Postal code only
4. Cross streets (“second and/&/@ santa clara”)

The geocoder provides robust handling of these typical user errors:

1. Typos
2. Streets without a street number
3. Wrong postal code
4. Nonexistent cross street
5. No street number provided, or wrong street number provided

The geocoder also supports ‘locals’. That is, each address can include not only a country code, but also a language code. This is useful in countries like Canada where the origin address for a route may be specified in French (*e.g.* “121 RUE DE L'AVOINE, ST-AUGUSTIN-DE-DESMAURES, QC G3A 1R7”) but the destination may be in English (*e.g.* “121 YONGE ST, TORONTO, ON M5C 1W4”).

XI. OFF-THE-SHELF JAVASCRIPT SOLUTIONS

The deCarta Developer Zone includes a number of comprehensive open-source JavaScript applications that demonstrate AJAX map tiling. These applications are fully functional and can be easily re-branded and deployed immediately by customers.

As we have discussed, DDS Web Services XML schema has been carefully crafted to minimize the amount of JavaScript coding. The following features and benefits are available in our pre-built, easily customizable draggable map applications:

- Cross-browser support for Internet Explorer and Mozilla/Firefox
- Freeform, single-line address geocoding
- Click to re-center the map
- Drag to move the map

- Change map look and feel on the fly via dropdown
- XML debugging console
- Route computation and route overlay

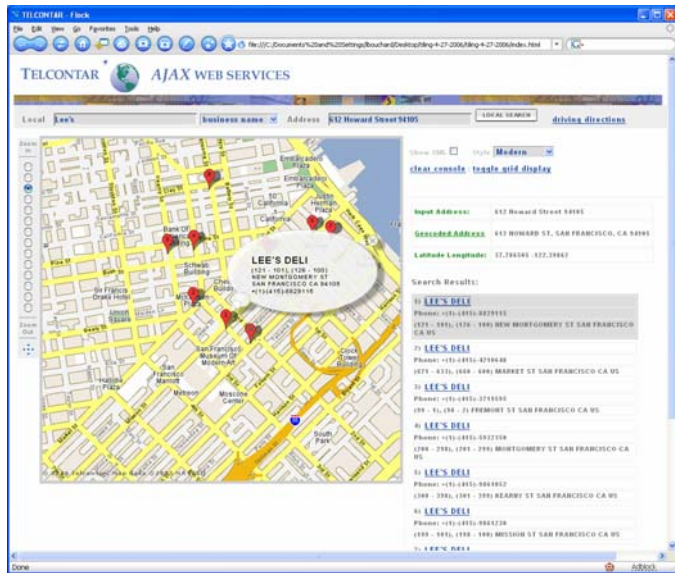


Fig. 5 – Off-the-shelf JavaScript application for draggable maps and local search

XII. DEVELOPER ZONE AND COMMERCIAL HOSTING

Whether or not a deCarta customer currently hosts an installation of DDS Web Services, the deCarta Developer Zone is the most direct way to get started with DDS Web Services. The “Dev Zone” is free of charge and includes:

- Active discussion forums
- Access to best-in-class map and POI data from leading vendors
- FAQs
- Downloadable Java 2 Standard Edition client for DDS Web Services
- XML schemas
- Web-based XML developer tools for submitting XML to DDS Web Services and viewing the response
- Sample XML documents
- A shared instance of DDS Web Services suitable for prototyping your application, without feature limitations.

Dev Zone customers can attend monthly web-based training, and receive rapid responses to forum postings, all free of charge. For developers who want to focus exclusively on developing a rich client experience and prefer not to install DDS locally, deCarta offers a commercial-grade hosted DDS Web Services cluster. This high-performance architecture is suitable for deploying production applications. For customers desiring an exclusive cluster of DDS Web Services, exclusive plans are available on request.

XIII. HIGH-PERFORMANCE SHARE-NOTHING ARCHITECTURE

DDS Web Services build on the well-known scalability of DDS. Figure 6 shows an architecture diagram of a DDS Web Services deployment. Each server in the cluster includes a clustered DDS and a servlet engine. A load balancer performs sticky load balancing across the cluster. Because each server in the cluster shares no resources with any other server, the architecture is scalable without limit. This allows customers to perform capacity planning incrementally and affordably, using commodity hardware.

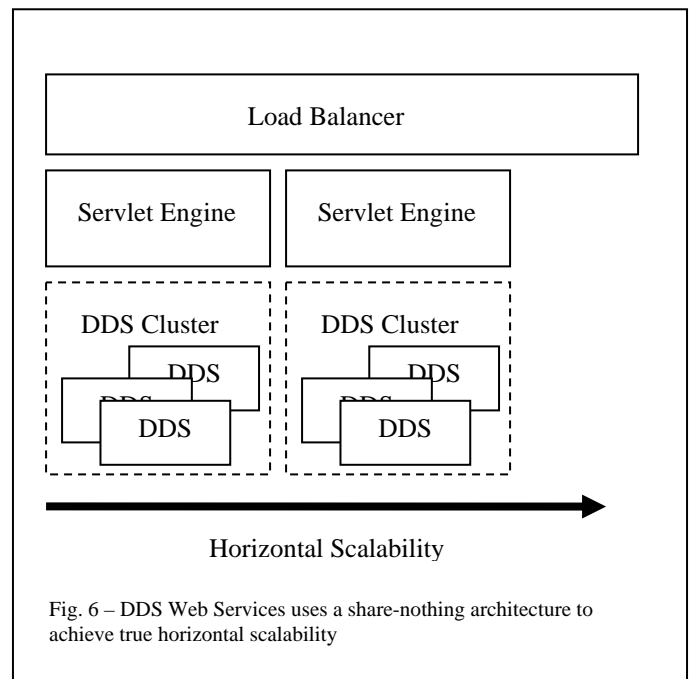


Fig. 6 – DDS Web Services uses a share-nothing architecture to achieve true horizontal scalability

XIV. CONCLUSION

deCarta’s DDS Web Services is the industry’s most advanced and easy-to-use solution for high- performance AJAX draggable maps. Complete off-the-shelf JavaScript solutions combined with a free hosted Dev Zone eliminate all development barriers. When the time is right to deploy an application for commercial use, customers can choose from commercial-grade hosting options or deploy DDS locally in their own environment. deCarta’s Drill Down Server Web Services provide the highest performance architecture, with industry leading flexibility, scalability and customizability.

About deCarta

deCarta provides the enabling geospatial software platform that powers today’s leading location-enabled applications. The company specializes in providing solutions ideal for Internet, mobile and telematics applications where scalability, speed and reliability are vital. Its DDS geospatial software platform and Rich Map Engine® software libraries powers today’s most successful and widely deployed LBS applications. deCarta is privately held and headquartered in San Jose, California with offices in Europe and Asia. Additional information about can be found at the company’s web site: <http://www.deCarta.com>.