Ka-Map: an interactive web-mapping application

L. Becchi[1]

[1] Libre profesional del grupo Ominiverdi.org, lorenzo@ominiverdi.com

ABSTRACT

ka-Map ("ka" as in ka-boom!) is an open source project that is aimed at providing a javascript API for developing highly interactive web-mapping interfaces using features available in modern web browsers.
ka-Map currently has a number of interesting features. It sports the usual array of user interface elements such as:
- *interactive, continuous panning without reloading the page*
- *keyboard navigation options (zooming, panning)*
- *zooming to pre-set scales*
- *interactive scalebar, legend and keymap support*
- *optional layer control on client side*
- *Server side tile caching*
- *many more...*

Key words: *Web Mapping, tiled caching, UMN Mapserver, Mapscript, AJAX.*

INTRODUCCIÓN

ka-Map, an interactive web-mapping application

ka-Map is ...
- a javascript API for rendering fast, tiled maps
- a set of tools for navigating and basic application needs
- a dynamically generated tile caching server

Its good for ...
- Static base map data with dynamic point overlays
- fixed set of scales
- large maps (full screen)
- dynamically resizeable maps

Its not good for ...
- dynamically changing base data
- Very large amounts of data

Ka-Map is one of the first Open Source web mapping applications making use of dinamic HTML and Ajax.

PROJECT OVERVIEW

What Ajax is?

From Jesse James Garrett article: (http://www.adaptivepath.com/publications/essays/archives/000385.php):

Web interaction designers can't help but feel a little envious of our colleagues who create desktop software. Desktop applications have a richness and responsiveness that has seemed out of reach on the Web. The same simplicity that enabled the Web's rapid proliferation also creates a gap between the experiences we can provide and the experiences users can get from a desktop application.

That gap is closing. Take a look at Google Suggest. Watch the way the suggested terms update as you type, almost instantly. Now look at Google Maps. Zoom in. Use your cursor to grab the map and scroll around a bit. Again, everything happens almost instantly, with no waiting for pages to reload.

Google Suggest and Google Maps are two examples of a new approach to web applications that we at Adaptive Path have been calling Ajax. The name is shorthand for Asynchronous JavaScript + XML, and it represents a fundamental shift in what's possible on the Web.

Defining Ajax

Ajax isn't a technology. It's really several technologies, each flourishing in its own right, coming together in powerful new ways. Ajax incorporates:

- standards-based presentation using XHTML and CSS;
- dynamic display and interaction using the Document Object Model;
- data interchange and manipulation using XML and XSLT;
- asynchronous data retrieval using XMLHttpRequest;
- and JavaScript binding everything together.

The classic web application model works like this: Most user actions in the interface trigger an HTTP request back to a web server. The server does some processing — retrieving data, crunching numbers, talking to various legacy systems — and then returns an HTML page to the client. It's a model adapted from the Web's original use as a hypertext medium, but as fans of The Elements of User Experience know, what makes the Web good for hypertext doesn't necessarily make it good for software applications.

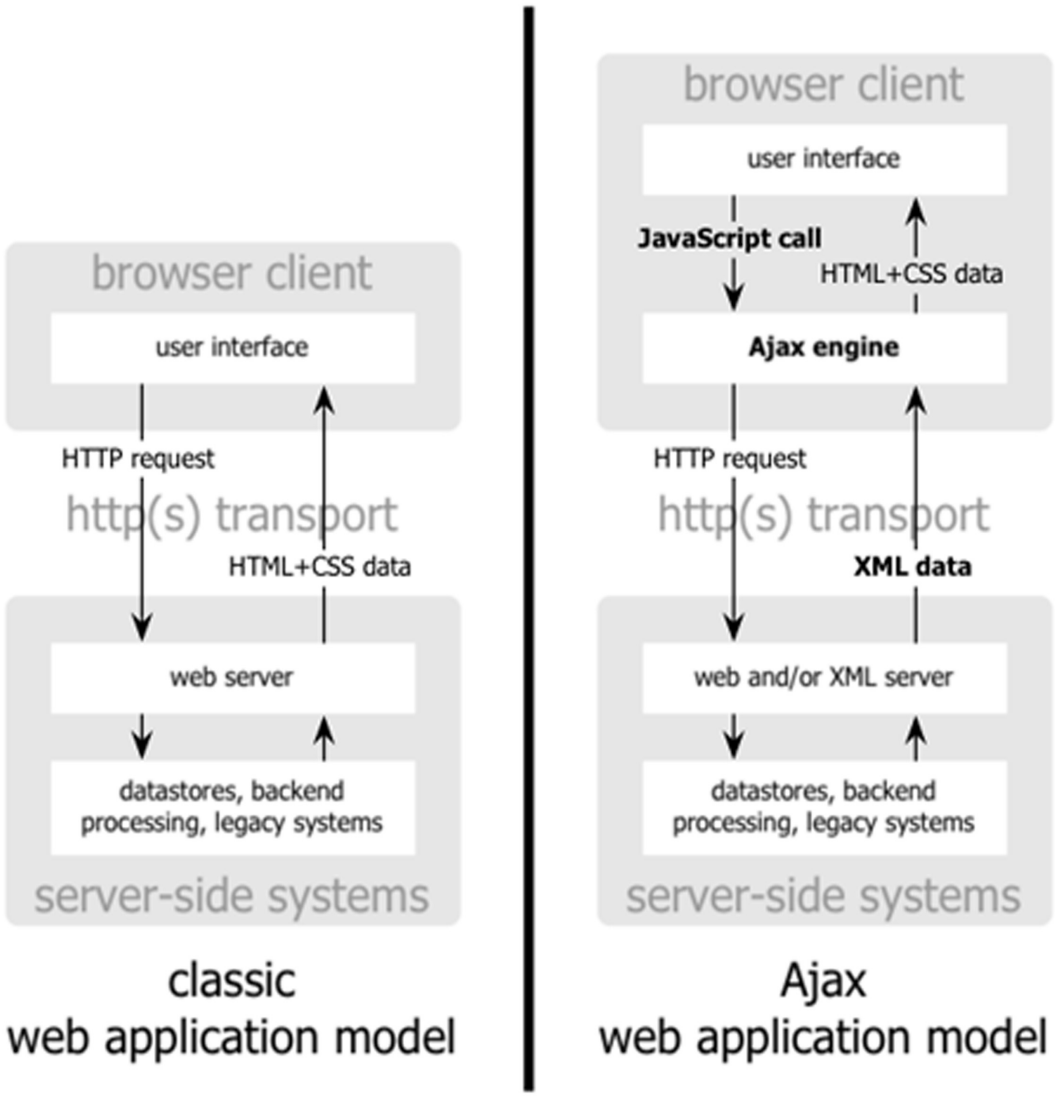

Fig 1: The traditional model for web applications (left) compared to the Ajax model (right).Source: http://www.adaptivepath.com/publications/essays/archives/000385.php

His approach makes a lot of technical sense, but it doesn't make for a great user experience. While the server is doing its thing, what's the user doing? That's right, waiting. And at every step in a task, the user waits some more.

Obviously, if we were designing the Web from scratch for applications, we wouldn't make users wait around. Once an interface is loaded, why should the user interaction come to a halt every time the application needs something from the server? In fact, why should the user see the application go to the server at all?

How Ajax is Different

An Ajax application eliminates the start-stop-start-stop nature of interaction on the Web by introducing an intermediary — an Ajax engine — between the user and the server. It seems like adding a layer to the application would make it less responsive, but the opposite is true.

Instead of loading a webpage, at the start of the session, the browser loads an Ajax engine — written in JavaScript and usually tucked away in a hidden frame. This engine is responsible for both rendering the interface the user sees and communicating with the server on the user's behalf. The Ajax engine allows the user's interaction with the application to happen asynchronously — independent of communication with the server. So the user is never staring at a blank browser window and an hourglass icon, waiting around for the server to do something.

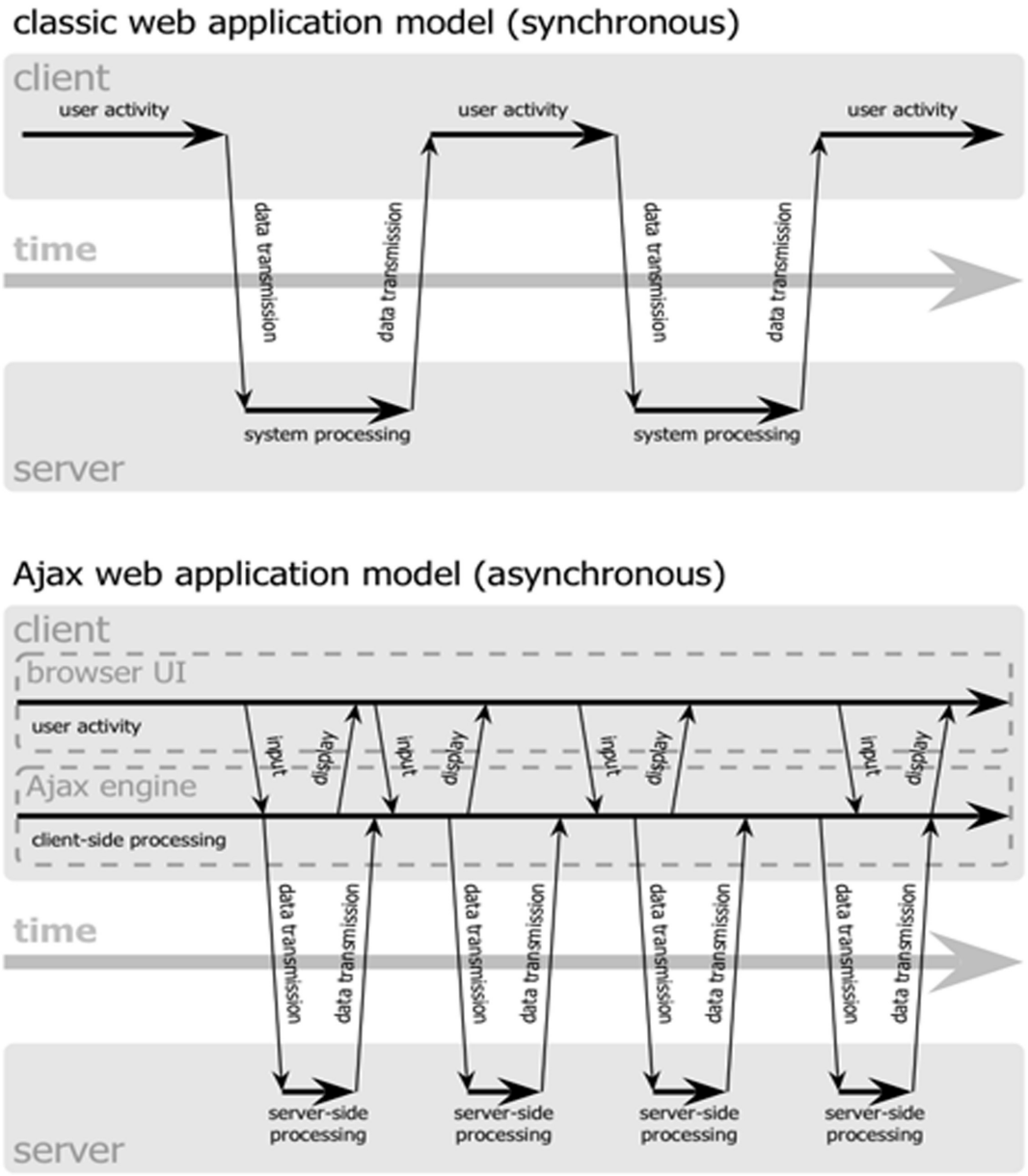

Figura 2: The synchronous interaction pattern of a traditional web application (top) compared with the asynchronous pattern of an Ajax application (bottom).Source: http://www.adaptivepath.com/publications/essays/archives/000385.php

Every user action that normally would generate an HTTP request takes the form of a JavaScript call to the Ajax engine instead. Any response to a user action that doesn't require a trip back to the server — such as simple data validation, editing data in memory, and even some navigation — the engine handles on its own. If the engine needs something from the server in order to respond — if it's submitting data for

processing, loading additional interface code, or retrieving new data — the engine makes those requests asynchronously, usually using XML, without stalling a user's interaction with the application.

CACHING SYSTEM

One important thing in common with Google Maps is the caching system.

Standard mapserver applications are not using this systems and every time a map is requested by a user the Mapserver has to access data informations before creating the new image, even if it is a pan movement from the client. This means that the web server query Mapserver, Mapserver open all data sources , prepare the response and give it back to the browser with the entire HTML code of the template page.

In Ka-Map, Mapserver can collect data access just once and create the full ammount of derired tiles of 200 px width and height. Every time a map is shown with ka-Map at a certain scale, ka-Map control the exact area and get the tiles representing it. If this tiles already exists there's no access to Mapserver and to datasets.

It is available a pre-cache system to create all needed tiles once for all and publish a site to a huge public with no need to access Mapserver calculation anymore. This means a strong optimization of the system. Limitation now are related only to available server bandwidth and filesystem access time.

TILE SYSTEM

All cached tiles are then aggregated as in a single unique mosaic by the javascript API on the user browser. That's what allow to pan continuosly without having to reload the page at each movement. Moreover on each movement only needed tiles around the viewable area are downloaded opitimizing bandwidth needings.

Flexible configuration based on UMN Mapserver

AS we've described before ka-Map is a UMN Mapserver client. Data access is then managed entirely by Mapserver and every information is related to the Mapserver mapfile.There you can define general information about the map (extent, projection, output format, ecc.) and all layers you want to be displayed (name, status, type, dataset, min scale, max scale).

Only few parameters are defined in ka-Map configuration file: tiles dimensions (in pixels), which mapfiles to show, which name to use and which predefined zoom levels to use.

Extending funtionalities

Ka-Map has a good set of tools but you can even enhance it using its **API** (Application Programming Interface).

Naturally you'll need a certain level of skill to practice with AJAX and DHTML but it is not impossible. Use the community WIKI (http://ka-map.ominiverdi.org) to find some more informations about ka-Map.

USER INTERFACE

ka-Map is only arrived at its version 1.0 but has a few interfaces to show.

Default interface

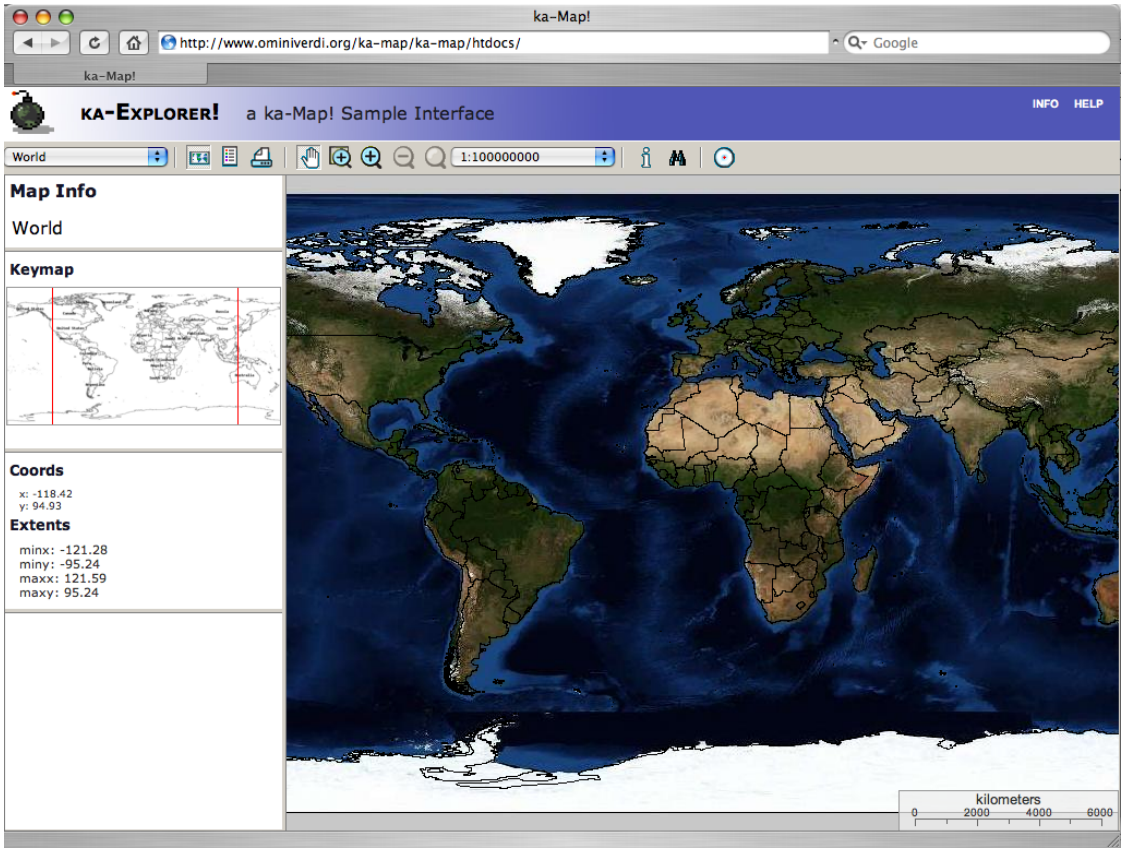

Fig 3: Default Interface. Source: http://www.ominiverdi.org/

Fig 4: Embedded Interface. Source: http://www.ominiverdi.org/

BROWSERS COMPATIBILITY

As much of all Ajax solutions, ka-Map is compatible with most diffused Web Browsers:

1. Internet Explorer 6.0 or sup.
2. Firefox 0.8 or sup.
3. Safari 1.2.4 or sup.
4. Netscape 7.1 or sup.
5. Mozilla 1.4 or sup.
6. Opera 8.02 or sup.

HOW DOES KA-MAP WORK

theInsideLayer

To maintain high performances only viewport tiles are downloaded from the server.

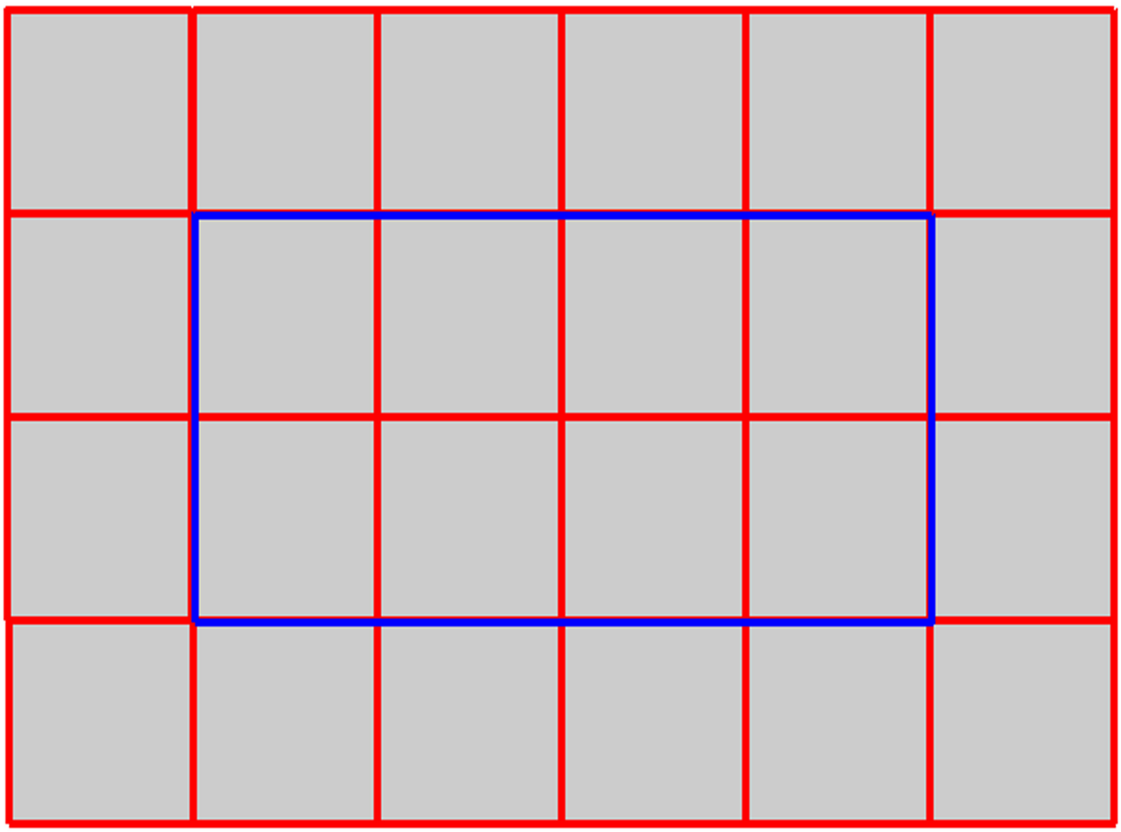

Fig 5: InsideLayer structure. Source: Paul Spencer, DM Solutions Group, 2005

viewport

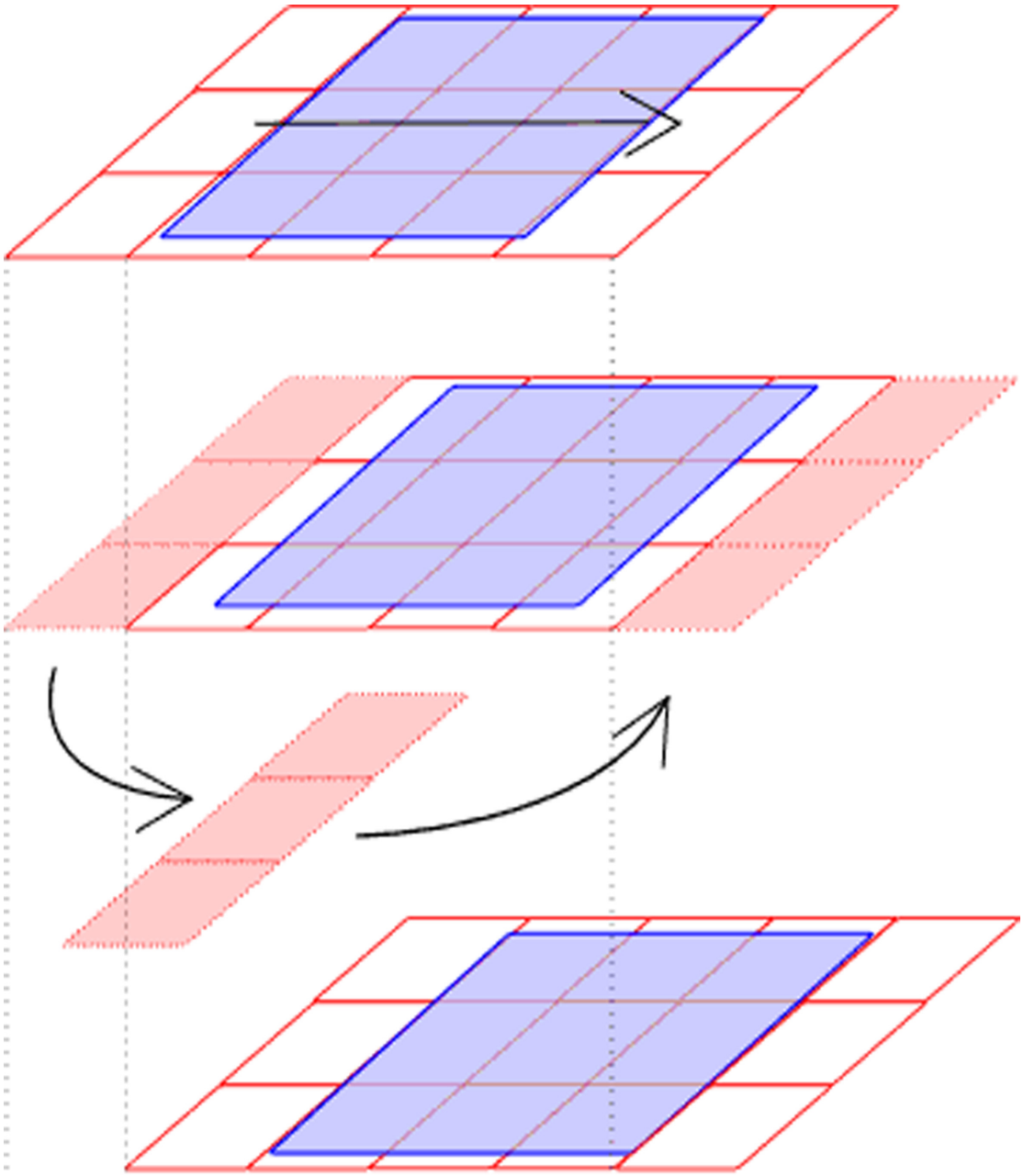

Fig 6: cashing procedure during panning. Source: Paul Spencer, DM Solutions Group, 2005

CONCLUSIONS

Ka-Map has finally arrived at version **1.0.0**. The development is still healthy and many funtions has been developed since the beginning. The community is growing continuosly and sponsorships allow the project to grow on many different directions as for other Open Source projects.

USEFULL LINKS

1. ka-Map Home Page: http://ka-map.maptools.org/
2. Documentation Wiki: http://ka-map.ominiverdi.org/

3. Ominiverdi's Home Page, co-developers group: http://www.ominiverdi.org
4. Google Maps: http://maps.google.com