

# Launching Context-Aware Visualisations

Jaakko Salonen and Jukka Huhtamäki

Tampere University of Technology, Hypermedia Laboratory,  
Korkeakoulunkatu 1, FI-33720 Tampere, Finland  
{jaakko.salonen, jukka.huhtamaki}@tut.fi

**Abstract.** Web is more pervasive than ever: it is being used with an increasing range of browsers and devices for a plethora of tasks. As such, exciting possibilities for information visualisation have become available: data is potentially highly relevant, easy to access and readily available in structured formats. For the task of visualisation, the diversity of the Web is both a challenge and an opportunity: while rich data may be available in well-defined structures, they need to be extracted from an open ecosystem of different data formats and interfaces. In this article, we will present how context-aware visualisations can be launched from target Web applications. Methods for accessing and capturing data and web usage context are presented. We will also present proof-of-concept examples for integrating launchable context-aware visualisations to target applications. We claim that by maintaining loose coupling to target Web applications and combining several data and context capturing methods, visualisations can be effectively launched from arbitrary Web applications with rich usage context information and input data.

**Keywords:** Ubiquitous computing; information visualisation; context-awareness; mobile Web; augmented browsing.

## 1 Introduction

Web has become an intangible part of our everyday lives, and is now more pervasive than ever: online services and other Web resources are used for work and play, in many environments including desktop and laptop computers, mobile phones and embedded devices. Quite recently, Internet-connected smartphones and netbooks have gained popularity, enabling Web usage on mobile devices. In living room context, Web browsing is also slowly being enabled by new TV devices, but also by some latest gaming consoles.

As the contexts in which Web is being used are becoming increasingly varied, capturing and understanding Web usage context is becoming even more important. For example, as different devices have different capabilities, a Web application may be required to adapt to the device context in order to guarantee good quality of service. Also, location and social environment may have important effects on usage: for example browsing the Web in a crowded bus, with strangers potentially looking at your screen, may have a strong influence on Web usage.

Possibly due to the advent of Web 2.0 *mashup* scene, tools of *information visualisation* are gaining popularity in everyday Web applications. In general, a mashup is a Web application created “by combining content, presentation, or application functionality from disparate Web sources” [1]. In information visualisation, the underlying objective is to serve as an amplifier of the cognition of a user through expressive views giving insight on a certain phenomena represented by the data [2]. This objective is not far away from those of mashups: in fact, many mashups can combine and represent existing data in a novel fashion and thus may be regarded as information visualisations.

It would be beneficial to enable visualisations to have mashup-like features for context and data capturing. These features would not only make easier for visualisations to access data, but also to better support users in different domains (see e.g. [3]). At their best, web visualisations would not be available as detached tools but, instead, as integral part of web browsing experience in the fashion of augmented browsing. For instance, a web browser extension could be used to pass additional information from the browser to visualisations.

For smooth web browsing experience, it is important to design the integration of visualisations as unobtrusive: they should not interfere or annoy users. Instead, they should launch, when needed, to display relevant information or provide intelligent “cues” in relevant places. For this design task, capture and use of available web usage context information is potentially very useful. By adapting visualisations to captured context, users can be provided with relevant information and tools.

In this article, we will study and identify different means of launching visualisation tools within heterogenous Web applications. We will also discuss different possible approaches for automatically capturing information about Web usage context and methods for transmitting the information when launching visualisations in the Web.

The rest of the article is organised as follows: in section 2 we will define background of our work with regard to context information and context-awareness, related work and known challenges, in section 3 we present methods for capturing Web usage context, in section 4 we will go through an example of launching visualisations from web browser with different means and discuss their differences. We conclude our work in section 5.

## 2 Background

### 2.1 Context and Context-Awareness

Key motivation for capturing context is the ability to make applications *context-aware*. Dey and Abowd [4] define that “a system is context-aware, if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task”.

In their popular, widely accepted definition for context related to context-awareness research, Dey and Abowd state the following: “Context is any information that can be used to characterise the situation of an entity. An entity is a person, place,

or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves". As such, "if a piece of information can be used to characterise the situation of a participant in an interaction, then that information is context". [4].

When building context-aware applications, the important question is that what contextual information is likely to be useful. According to Dey and Abowd, there are certain types of context that are, in practise, more important than others: activity (A), identity (I), location (L) and time (T) [4]. These types are defined as the primary context types; they not only answer important questions of who, what, when, and where, but may also be used as indices to further, secondary contextual information.

While types time, identity and location are fairly self-explanatory, we see that the activity dimension of context needs to be elaborated. According to Dey and Abowd, activity answers the fundamental question what is happening in the situation [4]. We see that the problem of obtaining knowledge about activity context is a reflection of the general problem of context-awareness: situational and contextual information are often implicit, and thus not directly available for computers.

Related to automating the collections methods of metadata on media captured with mobile devices, Davis et al. [5] present an alternative categorisation for contextual metadata. While we recognise that the three categories - spatial, temporal, and social contextual metadata - are included in Dey and Abowd's classification, we see that the social dimension is worth mentioning explicitly.

## 2.2 Web Usage Context

When capturing Web usage context, we are interested in contextual information about usage of web documents and applications.

As a ubiquitous system supporting an ecosystem of devices, applications and formats, definition of web usage context is non-trivial. However, some assumptions can be made that valid often enough (for a human user):

- Web is accessed with a user agent device that displays content on a colour screen and has at least one input device
- A web browser with rudimentary support for browser extensions, HTML<sup>1</sup>, CSS<sup>2</sup> and JavaScript is used
- In order to access web documents and applications, the browser connects to a (remote) web server
- Browser maintains state and history of the web browsing. This includes maintaining the state of currently loaded web documents and applications

All of the entities presented above – user agent device, web browser and target web documents and applications – are potential sources of usage context information. However, several challenges in accessing the context information exist.

First of all, context information is often fragmented across the different entities: relevant state information and data may be scattered across user agent

---

<sup>1</sup> HyperText Markup Language.

<sup>2</sup> Cascading Style Sheets.

device, web browser and (remote) web documents and applications. In order to obtain a clear picture of a usage context, information needs to be integrated from multiple sources.

Secondly, access to context information in the different entities is often limited to local APIs. For instance, relevant context information available on a user agent device may not be accessible from a web browser. Mechanisms for passing context data between web browser, local and remote systems are needed.

Finally, as a broad-spectrum of user agent devices and web browsers are in use, design of an all-inclusive web context capturing system is likely to be very demanding if not impossible. On the other hand, the use of single system or vendor-specific API cannot be considered sufficient. A considerably efficient approach would be to use (open) standard APIs whenever possible, and vendor-specific interfaces only when required.

The range of different user agent devices, operating systems and Web browsers make describing Web usage context especially challenging. Some typical choices, however, can be detected:

- Popular desktop operating systems for Web browsing include Windows XP, Windows Vista and Mac OS X, with Windows family systems taking easily over 90 percent market share [6]. On mobile OS markets, the most commonly used operating systems are iPhoneOS, SymbianOS (e.g. Nokia devices), and RIM (BlackBerry devices).
- It is estimated [7] that on desktop browser markets, Internet Explorer and Firefox take more than 80 percent share of the markets. Mobile markets are much more fragmented; Opera (incl. Opera Mini), iPhone, Nokia and iTouch browsers all four have a market share larger than 10 percent [6].

While specific configurations of Web browsers and devices are not easily defined, two different Web usage contexts clearly emerge: *desktop context* and *mobile context*. Statistics suggest that the configurations vary between these two settings: where desktop systems seem to be typically Windows systems with Internet Explorer or Firefox are mobile systems more typically smartphones using their vendor's own Web browser, or alternatively Opera. Only a fraction of Web usage can be detected from media devices like Playstation, suggesting Web usage from *living room context* is either very minimal or not visible in Web usage statistics.

### 2.3 Related Work

Numerous earlier and related approaches on mashup development and context information capturing influence and inform our work.

In project Mobilife, a context management framework was defined, according to which software called ContextWatcher was implemented as a mobile application for a limited set of Nokia 60 series phones (see [8]). ContextWatcher automatically collects, enriches and shares context information focused on visited places and people met. ContextWatcher is presented as a way to keep in touch with relatives and friends: use cases include managing social relationships, taking photos and receiving points of interests [9].

A fairly similar design in context management frameworks is introduced in Hydrogen. Similar to ContextWatcher, Hydrogen [10] specifies an architecture in which ContextServer is used as an intermediary component between applications that use context information and various adaptors providing context data.

Yahoo! Research's ZoneTag<sup>3</sup> presents a more practice-oriented solution for contextual data. ZoneTag is a rich mobile client that focuses on enabling context-aware photographs uploads from camera phones [11]. Additionally, the software supports media annotation via context-based tag suggestions. Also a mobile photo browser client, Zurfer<sup>4</sup>, has been implemented and can be used to access photos based on geographic location as captured by ZoneTag.

Another example of a more specific context application is Nokia's Sports Tracker<sup>5</sup>. Sports Tracker is a GPS<sup>6</sup>-based activity tracker for compatible Nokia mobile devices and records information such as speed, distance, and time from a training routine. Context information also includes optional playlist from used mobile device.

A variety of tools and frameworks supporting the development of mashups with widget-like rich user interfaces and content processing pipelines exist. Examples include Yahoo Pipes<sup>7</sup>, Intel Mash Maker<sup>8</sup> and Google Mashup Editor<sup>9</sup>. Also information visualisation tools including Adobe Flex<sup>10</sup> and Prefuse Flare<sup>11</sup> and web widget frameworks such as Dojo<sup>12</sup>, Ext JS<sup>13</sup>, Yahoo YUI library<sup>14</sup> and jQuery<sup>15</sup> contribute to the development of mashups.

## 2.4 Challenges

Several challenges in leveraging the potential of information visualisation in mashups exist. Means for accessing context information is often limited to simple attributes representing the capabilities of the device in use, user language and, recently, user geolocation. No harmonised solution to access context information is provided for either mashup or information visualisation developers.

Due to the limitations of web browser as a run-time platform, visualisations that use large data sets or are required to be highly interactive may not be feasible to be implemented as conventional mashups. Additionally, access to local and remote resources is very restricted in typical web browser security models. In effect, mashups or mashup systems based on web browsers may not be able to access and represent all information of interest.

---

<sup>3</sup> <http://zonetag.research.yahoo.com/>

<sup>4</sup> <http://zurfer.research.yahoo.com/>

<sup>5</sup> <http://sportstracker.nokia.com/>

<sup>6</sup> Global Positioning System.

<sup>7</sup> <http://pipes.yahoo.com/>

<sup>8</sup> <http://mashmaker.intel.com/>

<sup>9</sup> <http://code.google.com/gme/>

<sup>10</sup> <http://www.adobe.com/products/flex/>

<sup>11</sup> <http://flare.prefuse.org/>

<sup>12</sup> <http://www.dojotoolkit.org/>

<sup>13</sup> <http://www.extjs.com/>

<sup>14</sup> <http://developer.yahoo.com/yui/>

<sup>15</sup> <http://jquery.com/>

Other, more powerful visualisation tools such as Orange<sup>16</sup> and Vizster<sup>17</sup> or the ones implemented with Processing<sup>18</sup> often operate as separate applications: they take data from web applications as input and process it independently of the original application, often in a pipelined transformation process. In this line of thinking, web applications are seen as data sources that are only accessed in the initial step of the process. As many web applications expose similar application programming interfaces (APIs), it is in theory sufficient to create new data export component only once for each type of interface.

Based on our experiences, the creation of a general-purpose data export component has proven to be a difficult task. In a common pattern of practical visualisation development, a lot of information about application context, such as authorisation and authentication information, and location and format of the target data, has to be specified. An exporter component then needs to not only retrieve the desired data, but also perform tasks required to access the data, such as to perform user authentication. Also, data access process is often very application-specific: for instance, while two applications may share common data format, they may use incompatible authentication methods. In order to make data export components more general purpose, more parametrisation is to be added to make the use of these components more complex.

In a worst-case scenario, external export component may not be able to access data at all. For instance, it is common for web applications that require high level of security such as online banking system or institutional single-sign on services, to take additional measures to specifically stop non-human agents accessing any content at all. This kind of a “walled-garden” service provision policy found from many other domains as well may effectively block visualisation data export.

### 3 Context Capturing Methods

In this chapter, we will define and discuss various methods that are feasible for capturing web usage context.

#### 3.1 HTTP Request Interpretation

HTTP Request contains multiple headers that are typically present and can be used for context interpretation. A commonly used, standardised request header is USER-AGENT, which contains information about the user agent originating the request [12]. Another widely supported standard, User Agent Profile (UAProf), specifies X-WAP-PROFILE header for transporting references to Composite Capability/Preferences Profiles (CC/PP) [13]. Also, proprietary header extensions may be used to obtain capability information. Opera Mini, a data transfer-efficient mobile browser system based on proxy architecture, for example, specifies several custom headers that are helpful in determining device capabilities [14].

---

<sup>16</sup> <http://www.aillab.si/orange/>

<sup>17</sup> <http://jheer.org/vizster/>

<sup>18</sup> <http://processing.org/>

A widely used method for extracting capability information is to map header data to device profile and user agent databases. Commonly used mappings and databases include resources provided by `browscap.ini`<sup>19</sup> and `WURFL`<sup>20</sup>. Capability information that is potentially easy to extract with the mapping approach includes the following: operating system or platform name, browser name and version, and browser capabilities including CSS support level, support for specific HTML elements such as frames and tables, cookies and scripting support.

By interpreting clients IP<sup>3</sup> address, HTTP request data can also be used for locating user. For example, `hostip.info` provides a public Web API (see [15]) that can be used to translate any given IP (Internet Protocol) address in to a geographic position. While not typically very accurate, this method provides us with a mean for making a rough guess of user's position and enabling at least some awareness of user's location context.

### 3.2 Generic DOM and JavaScript Access

Information on different capabilities may also be also obtained with JavaScript via Document Object Model (DOM), an API provided by Web browser and its various extensions. Information about browser capabilities is available via `window.navigator`. Screen capabilities (width, height, depth) can be accessed from `window.screen` (see e.g. [16]). Current browsing location is accessible via `window.location`. Also, an object containing information about browsing history is available, but does not actually expose any history data besides the number of elements in the history list (e.g. [17]). The objects are defined in DOM Level 0: a specification of JavaScript objects that existed in browsers before actual DOM specifications [18]. Most modern browsers still provide DOM 0 elements (e.g. [19]), thus making them a viable source of capability information.

Even some input metrics can be captured with scripting. Mouse coordinates relative to the screen can be obtained from property pair `event.screenX` and `event.screenY`, both available on all major Web browsers (see e.g. [20]). By creating a listener for mouse move events in JavaScript, input metrics from mouse can be captured within individual Web pages. In a similar fashion, by listening to `onkeydown`, `onkeyup` and `onkeypress` events, a limited set of events from keyboard can be captured.

APIs for accessing user's geographic position via browser APIs are also being actively developed. Google Gears<sup>21</sup>, a browser extension for Firefox, Internet Explorer and Safari, implements a Geolocation API [21] that can be used to obtain positional data. Later on, this work has been progressed in the form of W3C's specification for Geolocation API that has been published as an Editor's Draft [12]. Also an experimental Opera build with Geolocation support exists [23]. In general, several browser vendors are supporting the API, making it a viable source for positional data.

<sup>19</sup> <http://browsers.garykeith.com/downloads.asp>

<sup>20</sup> <http://wurfl.sourceforge.net/>

<sup>21</sup> <http://gears.google.com/>

W3C's Geolocation API can be used to fetch user's position as a latitude-longitude value pair, but it also supports retrieving altitude, speed, heading and information about the accuracy of positioning (*accuracy*, *altitudeAccuracy*). A particularly convenient feature in Geolocation API, when properly implemented by a browser, is that it automatically tries to combine different sources of data to obtain accurate positioning. For example, in Firefox [24], information about nearby wireless access points and computer's IP address is gathered. This information is then sent to geolocation service provider, Google Location Services by default, to get an estimate of user's location. Data from GPS devices is similarly exploited when available. Note that without an established trust relationship between the user and a service requesting geolocation, user will be prompted for approval for sharing the location.

### 3.3 Context Scraping

Similar to *data scraping* or *screen scraping* where data is programmatically extracted from views that are primarily intended for humans to read, context information can be extracted from documents.

Web hypertext documents, however, already generally include information in formats particularly designed for machines to process. This information is often represented as *web feeds* in RSS (Really Simple Syndication) and RDF<sup>22</sup> Site Summary) or Atom formats. Also, Web documents may include explicit information in Microformats<sup>23</sup> or other form of *semantic markup* that enables the automated processing of the content.

Further, the output of a particular content management system or other known platform follow a specific markup scheme. A popular wiki engine MediaWiki, for example, lays out the editing history of a given page as an HTML list structure. Detecting that a page is put out by an instance of MediaWiki can be identified on basis of document metadata, such as follows:

```
<meta name="generator"
content="MediaWiki 1.16alpha-wmf" />
```

Once MediaWiki is recognised, a revision history of a given MediaWiki page may be fetched, for example, in order to define a social context for a wiki page (if this is something that an application developer wants to do). In practise, this is simply done by adding a request parameter (*?action=history*) to the URI of the wiki page. Whereas interpretation surely is involved in making this kind of information explicit, a rough set of contextual information on social activity within a wiki can be automatically collected.

### 3.4 Application-Specific API Access

In many cases, accessing the content of a system via external components such as crawlers is either difficult or impossible. Examples include Facebook and other

---

<sup>22</sup> Resource Description Framework.

<sup>23</sup> <http://microformats.org/>



services that actively attempt to restrict crawler access for reasons of privacy and business. Injecting an intelligent probe may help in these cases. A probe implemented as a Greasemonkey<sup>24</sup> script, for example, is even able to access the Javascript functions of a specific service via its *unsafeWindow* object. Do note, however, that the use of *unsafeWindow* claimed to be data-insecure and, more importantly, does often meet the characteristics of a cross-site scripting (XSS) attack.

### 3.5 User Input

Direct or indirect input from user is also an option to collect context information. We see, however, that to be consistent with the unobtrusive approach of information visualisation integration, user input should always be considered as the last resort.

Some of the context information is always unavailable for automatic collection. Moreover, we agree with Dey and Mankoff [25] in that automatically collected context information is often ambiguous, thus a user should always have the option to review and refine the interpreted context information. We see that, when complemented with the option to refine the context information, visualising the automatically collected and interpreted context is a valid option for providing user with a *mediator* [25], a component that supports handling ambiguous information. With inspiration from the visualisation-driven, iterative process of *knowledge crystallisation* [26], in which the most relevant information in a given case is collected, often with the help of tools of information visualisation, we might want to refer to *context crystallisation* when collecting context information in co-operation with the user.

## 4 Launching Context-Aware Visualisations

In this chapter, we will go through an example on launching visualisations from web browsing context with different means.

### 4.1 Launching Visualisations as Stand-Alone Applications

Let us first consider how visualisations are launched – as how we see they typically are – as stand-alone applications.

As an example of a candidate visualisation to be launched, we present RSSVis. RSSVis is a visualisation application implemented with *Wille 2 Visualisation System*. The application consists of three widgets for representing RSS information: a timeline, a data table, and a map for RSS data that includes geolocation information in, e.g., GeoRSS<sup>25</sup> format.

Data source to RSSVis is given by adding parameter *rss* with full feed URL either as a URL-encoded parameter (GET) or encoded in HTTP request content (POST). The actual feed content is then retrieved by RSSVis. The aim of the visualisation is to serve as a general-purpose RSS visualisation tool: different feed formats and metadata fields are automatically detected and parsed. If an RSS

---

<sup>24</sup> <http://www.greasemonkey.net/>

<sup>25</sup> <http://www.georss.org/>

aggregator component was added either as independent component or integrated to RSSVis, data from several RSS feeds could be displayed in an aggregate data view.

In casual web browsing, RSS content is often stumbled upon, as an alternative representation of data primarily available in HTML format. When an autodiscovery mechanism such as RSS [27] or Atom Auto-Discovery [28] is used, modern web browsers are able to detect and inform users about available RSS content. By opening the actual RSS URL in browser, an HTML version of the feed is often displayed.

A simple option to enable the launch of RSSVis is via a regular HTML form that the visualisation user pastes the URL of the RSS feed to be visualised into. If user uses an RSS-enabled browser, it is sufficient for example to simply copy and paste the discovered URL to the HTML form.

## 4.2 Passing More Context with Bookmarklets

Launching a visualisation, such as RSSVis, as a standalone application, requires user to manually re-enter information about the RSS feed. A more handy option can be provided with a *bookmarklet*, a functional JavaScript-based bookmark capable of accessing the DOM of the document to be visualised. By using a bookmarklet, some of the context information available in browsing window may be passed automatically to the visualisation.

An example of a simple bookmarklet for launching a more context-aware RSSVis is presented in the following listing:

### RSSVis Launcher Bookmarklet Code Example<sup>26</sup>

```

01 javascript:
02 appurl='http://localhost:8080/apps/rssvis/?uri=';
03 u=document.location.hostname+document.location.pathname;
04 t=document.title;
05 w=screen.width;
06 h=screen.height;
07 void(window.open(appurl+escape(u)+'&width='+
08 escape(w)+'&height='+escape(h),'_blank'));
```

Going through the listing, the following actions are performed by the bookmarklet:

1. URL of the target visualisation application is specified (line 2)
2. Document identifier metadata (URL and title) of the currently open document is retrieved via DOM (lines 3-4)
3. Some browser properties (screen width and height) are also retrieved (lines 5-6)
4. Finally, a new pop up window is opened with all retrieved variables encoded to target URL as URL arguments (lines 7-8)

Once tested thoroughly for browser-compatibility, bookmarklets operate in a wide range of desktop web browsers. Bookmarklets with limited functionality can be implemented also for Opera Mini and other mobile browsers.

---

<sup>26</sup> Note that line breaks have been added for clarity: in a functional bookmarklet, everything has to be encoded within a single line of code. Also, if the bookmarklet is encoded as a hyperlink (*a* element), some characters may need to be escaped and entity-encoded as well.

To make the launching process more straightforward, we can implement support for auto-discovery of RSS feeds into RSSVis. Auto-discovery is based on Web document metadata stating the existence of RSS feed(s) related to the document. Browsers and RSS aggregators, for example, implement auto-discovery to ease the subscription of Web feeds. With auto-discovery, a visualisation user can launch the visualisation directly from a Web document. The existence of RSS feeds can be detected either with JavaScript or by the actual RSSVis implementation.

### 4.3 Proactive Detection of Visualisable Contexts

For a true visualisation-augmented web browsing experience, bookmarklets may not be sufficient: a user needs to actively click a bookmarklet in order to activate a visualisation. In a more sophisticated solution, a visualisation could automatically capture the context and suggest possible visualisations for the currently active document or application.

In Mozilla Firefox, Google Chrome and other compatible browsers, Greasemonkey can be used for augmenting browsing with visualisations. Greasemonkey is a browser extension that allows implementing scripts that tailor Web documents on-the-fly as the documents are loaded into the browser. A URI space is defined for a given script to handle.

With Greasemonkey, we are able to implement a proactive mechanism for launching RSSVis where the browser automatically injects either an intelligent visualisation launcher or the actual visualisation into the document that falls into the defined URI space or, more generically, has an alternative representation in RSS.

In the following listing, an example of Greasemonkey's usage for visualisation augmented browsing is presented. The idea is to automatically discover data sources actively, while user is browsing, and inject a link to a visualisation whenever appropriate.

Greasemonkey code example

```

01 postdata = 'uri=' +
02 escape(unsafeWindow.document.location.href) +
03 '&representation='+ escape(new
04 XMLSerializer().serializeToString(unsafeWindow.document)
05 );
06
07 GM_xmlhttpRequest({
08   method: "POST",
09   url: "http://localhost/apps/rssvis/isvisualisable/",
10   data: postdata,
11   headers: {
12     "Content-Type": "application/x-www-form-urlencoded",
13     "Content-Length": postdata.length,
14   },
15   onload: function(response) {
16     if (response.status == 200) {
17       document = unsafeWindow.document;
18       link = document.createElement('a');
19       // Set up appearance and target of the link
20       // (Code removed for brevity)
21       unsafeWindow.document.body.appendChild(link);
22     }
23   }
24 });

```

Going through the listing, the following actions are performed:

1. Context data, including the whole content of the current page is captured and URL encoded (lines 1-4)
2. The captured context data is sent to an analyser application in order to determine if it can be visualised with RSSVis (lines 6-23)
3. In case visualisable data is available, a link to the RSSVis tool is injected directly to the page (lines 16-20)

A Greasemonkey script can be configured to execute on every page or to an enumerated set of URLs. Through configuration we can state, for example, that every user profile in a social bookmarking service delicious.com is equipped with a timeline representing the latest bookmarkings of the user. Further, when geolocation information is available in an RSS feed, the geolocation of a user can be resolved and used to set the original focus of the map visualising the items in an RSS feed.

#### 4.4 Summary and Discussion

The presented visualisation launching strategies let users to maintain their current web workflows, while making potentially insightful visualisations available in right contexts. When an intermediary context collector and visualisation selector such as presented Visualise app is used, the approach is more proactive than actively browsing to an alternative location to operate the visualisations.

The most pervasive approach is to use a script injected with Greasemonkey or similar system. This approach, however, suffers from the lack of support in some browsers. Security considerations may also be biased: it is questionable if one wants an external application to actively monitor the web usage.

Bookmarklets, on the other hand, while less capable, are discrete and leave the decision for activation to the user. They can be also used as a fallback or as a selectable alternative to systems like Greasemonkey: if user-scripting is not supported by browser or disallowed by the user, bookmarklets can be used instead.

As a final fallback, an HTML form can be also used. In this case, we may lack much of the desired context data. This implies that potentially more user intervention is required for manual input of context information.

The coverage of data and context capturing can be also extended to environments that e.g. require authentication or do not directly support RSS. For example if the RSS feed requires user authentication, the visualisation application may initially try to retrieve data loaded to browser in client-side or as a fallback, ask user for credentials and perform data retrieval on server-side.

As the visualisation system supports local machine installation, local resources can be potentially accessed as well. Thus, an RSS representation of, for example, a local Zotero<sup>27</sup> article database could be implemented and visualised with RSSVis. Having the possibility to utilise context information in mind, new interesting applications are easy to invent. A researcher who is currently going through an academic article database, can be provided with a timeline representation of articles in the local database. The presented approach even makes it possible to mash up and display local

---

<sup>27</sup> <http://www.zotero.org/>

and remote data in browser: in this fashion, an integrated article timeline could contain both local and remote resources.

Access to documents and application APIs is complemented with DOM manipulation and access to the local files system is, when put together, a highly expressive combination that has to be dealt with care for reasons of data security. For now, we have focused to harness the full power of the technology stack; the production use of the defined approach, however, insists the availability of solutions increasing the user control. Most likely, the expressivity is compromised accordingly.

## 5 Conclusion

In this paper, we have described a model for Web usage context. Also, based on practical experimentation and literature review, we have identified and described generic methods for capturing Web usage context. Finally, we have presented a real- life example on launching information visualisation tools in heterogenous environment and on passing the context information to the application.

According to our approach, Web usage context capturing should be based primarily on exploiting information readily available with different automated means. If a specific piece of context information is not available but required, methods that may require user intervention can be used. Further, mediation of ambiguous context information can be supported with context information visualisation. As a last resort, user will be directly asked for direct input. User identification is an exception in this strategy: manual identification is always preferred.

An advantage of the given approach is that it very often minimises the need for both user intervention and creation of custom context capturing applications. As such, we could describe it as a lightweight strategy for Web usage context capturing. A limitation in the given approach is that, when followed, it does not itself guarantee a given coverage, accuracy or level of available contextual information. In the worst- case setting, this means that no contextual information is available for the application.

We see that context capturing is always restricted by several factors. Firstly, the more platforms we wish to support, the more work it always demands. Secondly, due to legislative and non-technical restrictions, capturing some context information may always require user intervention or is not possible at all. Finally, it seems that context information will be often left incomplete: a setting in which data is not available must be tolerated.

Future work includes extending the context-collection framework and enabling a more harmonised way to access context data for visualisation tool developers. On basis of this data, a more intelligent mediator application can be developed to support the visualisation-selection. Similarly to context data access, generic means to inform visualisation tools about the existing points of accessing input data should be developed. Our long-term goal is to enable a visualisation-launching ecosystem that different visualisations can be mapped into.

**Acknowledgements.** European Commission (IST network of excellence project OPAALS of the sixth framework program, contract number: FP6-034824) supports our work.

## References

1. Yu, J., Benatallah, B., Casta, F., Daniel, F.: Understanding Mashup Development. *IEEE Internet Computing* 12(5), 44–52 (2008)
2. Ware, C.: *Information Visualization, Perception for Design* (Interactive Technologies), 2nd edn. Morgan Kaufmann, San Francisco (2004)
3. Galli, L., Guarneri, R., Huhtamäki, J.: *VERTIGO: Find, Enjoy and Share Media Trails across Physical and Social Contexts*, London, UK (2009)
4. Dey, A., Abowd, G.: Towards a Better Understanding of Context and Context-Awareness. Submitted to the 1st International Symposium on Handheld and Ubiquitous Computing (HUC '99) (June 1999)
5. Davis, M., King, S., Good, N., Sarvas, R.: From Context to Content: Leveraging Context to Infer Media Metadata. In: *Proceedings of the 12th annual ACM international conference on Multimedia*, pp. 188–195. ACM, New York (2004)
6. Statcounter.com: Top 8 Mobile OS from July 1, 2008 to 28 July 28, 2009, [http://gs.statcounter.com/#mobile\\_os-ww-daily-20080701-20090728](http://gs.statcounter.com/#mobile_os-ww-daily-20080701-20090728) (accessed January 8, 2010)
7. Wikipedia: Usage share of web browsers, [http://en.wikipedia.org/w/index.php?title=Usage\\_share\\_of\\_web\\_browsers&oldid=3364023](http://en.wikipedia.org/w/index.php?title=Usage_share_of_web_browsers&oldid=3364023) (accessed January 10, 2010)
8. Floreen, P., Przybilski, M., Nurmi, P., Koolwaaij, J., Tarlano, A., Wagner, M., Luther, M., Bataille, F., Boussard, M., Mrohs, B., Lau, S.: Towards a context management framework for Mobilife. In: *Proceedings of the 14th IST Mobile & Wireless Summit* (2005)
9. Kernchen, R., Bonnefoy, D., Battestini, A., Mrohs, B., Wagner, M., Klemettinen, M.: Context-Awareness in MobiLife. In: *Proceedings of the 15th IST Mobile & Wireless Summit* (2006)
10. Hofer, T., Schwinger, W., Pichler, M., Leonhartsberger, G., Altmann, J., Retschitzegger, W.: Context-Awareness on Mobile Devices - the Hydrogen Approach. In: *Proceedings of the 36th Hawaii International Conference on System Sciences, HICSS'03* (2002)
11. Ahern, S., Davis, M., Eckles, D., King, S., Naaman, M., Nair, R., Spasojevic, M., Hui-I Yang, J.: ZoneTag: Designing Context-Aware Mobile Media Capture to Increase Participation. In: *Mobile Systems, Applications, and Services*, pp. 36–44 (2004)
12. Fielding, R., et al.: Hypertext Transfer Protocol - HTTP/1.1. Request for Comments (RFC) 2616. The Internet Society (1999)
13. Wireless Application Forum, Ltd.: Wireless Application Protocol WAP-248-UAPROF-20011020-a, <http://www.openmobilealliance.org/tech/affiliates/wap/wap-248-uaprof-20011020-a.pdf> (accessed January 10, 2010)
14. Von Streng Hsehre, K.: Opera Mini request headers (November 7, 2007), <http://dev.opera.com/articles/view/opera-mini-request-headers/>
15. Mozilla Developer Center: window.screen (August 25, 2009), <https://developer.mozilla.org/en/DOM/window.screen>
16. Community Geotarget IP Addresses Project: Using the Database - IP Address Lookup, <http://www.hostip.info/use.html>
17. Mozilla Developer Center: window.history (August 25, 2009), <https://developer.mozilla.org/en/DOM/window.history>
18. Koch, P.: JavaScript - Level 0 DOM, <http://www.quirksmode.org/js/dom0.html>
19. Aptana, Inc.: HTML DOM Level 0 Reference Index (January 8, 2010), <http://www.aptana.com/reference/html/api/HTMLDOM0.index.html>

20. Koch, P.: W3C DOM Compatibility – CSS Object Model View (March 29, 2009), [http://www.quirksmode.org/dom/w3c\\_cssom.html](http://www.quirksmode.org/dom/w3c_cssom.html)
21. Google: Geolocation API – Gears API – Google Code (2010), [http://code.google.com/apis/gears/api\\_geolocation.html](http://code.google.com/apis/gears/api_geolocation.html)
22. Popescu, A. (ed.): W3C: Geolocation API Specification. W3C Editor’s Draft, June 30 (2009), <http://dev.w3.org/geo/api/spec-source.html>
23. Mills, C.: Find me! Geolocation-enabled Opera build (March 26, 2009), <http://labs.opera.com/news/2009/03/26/>
24. Mozilla: Geolocation in Firefox (2010), <http://www.mozilla.com/firefox/geolocation/>
25. Dey, A.K., Mankoff, J.: Designing mediation for context-aware applications. *ACM Transactions on Computer-Human Interaction* 12, 53–80 (2005)
26. Card, S.K., Mackinlay, J., Shneiderman, B.: *Readings in Information Visualization: Using Vision to Think*. Morgan Kaufmann, San Francisco (1999)
27. Cadenhead, R., Holderness, J., Charles Morin, R.: RSS Auto-discovery. RSS Advisory Board (November 27, 2006), <http://www.rssboard.org/rss-autodiscovery>
28. The Internet Society: Atom Auto-discovery (November 11, 2005), <http://philringnalda.com/rfc/draft-ietf-atompub-autodiscovery-01.html>