Bringing Web 2.0 to the Old Web: A Platform for Parasitic Applications

Florian Alt[1], Albrecht Schmidt[1], Richard Atterer[2], Paul Holleis[3]

[1] Pervasive Computing Group, University of Duisburg-Essen, Germany
{florian.alt,albrecht.schmidt}@uni-due.de
[2] Media Informatics Group, Ludwig-Maximilians-University Munich, Germany
richard.atterer@ifi.lmu.de
[3] DOCOMO Euro Labs, Munich, Germany
holleis@docomolab-euro.com

Abstract. It is possible to create interactive, responsive web applications that allow user-generated contributions. However, the relevant technologies have to be explicitly deployed by the authors of the web pages. In this work we present the concept of parasitic and symbiotic web applications which can be deployed on arbitrary web pages by means of a proxy-based application platform. Such applications are capable of inserting, editing and deleting the content of web pages. We use an HTTP proxy in order to insert JavaScript code on each web page that is delivered from the web server to the browser. Additionally we use a database server hosting user-generated scripts as well as high-level APIs allowing for implementing customized web applications. Our approach is capable of cooperating with existing web pages by using shared standards (e.g. formatting of the structure on DOM level) and common APIs but also allows for user-generated (parasitic) applications on arbitrary web pages without the need for cooperation by the page owner.

1 Introduction

A drawback of the WWW has always been that information only flows in one direction i.e. from the author of a web page to its readers. To resolve this shortcoming, several approaches have been proposed, such as guest books, bookmarks and discussion boards. However, most of these approaches are not capable of providing a true two-way flow of information, neither by modifying the information resource itself nor by extending the unidirectional flow in a way that indirectly associates separately stored additional information with the original source. A popular solution which allows interaction and shared working on documents is Wikis. However, they still require installation of the Wiki software on the server side and are hence limited to specific web sites. Especially with the advent of Web 2.0 technology, annotation systems and tools supporting automation and customization of rendered web pages have become popular. Yet, those approaches are static in a sense that the applications cannot be easily distributed and made available to other users or programmers.

Our contribution in this paper is twofold. First, we provide a modular technical platform which permits novel types of applications to be deployed on top of existing web applications. This is possible using statically added components which are implemented directly within the platform, such as a tool which allows users to leave annotations on arbitrary web pages. More advanced applications are customizable and dynamic, e.g. a voting tool allowing the user to specify questions, options and the pages to deploy the tool on. Finally, user-generated applications are also possible, they allow programmers to execute code of their choice on arbitrary pages. The platform is based on an HTTP proxy which modifies page content before sending it to the browser, together with an application/database server for storage of code and data of the deployed applications.

Existing solutions in this area rely on client-side software installation, which has the unfortunate effect that only users who install the software can use the application. In comparison, our solution merely requires the user to reconfigure their browser's HTTP proxy setting. Additionally, existing efforts cannot easily support dynamic applications requiring for example a database connection due to the same origin policy. We solve this problem by providing users a simple high-level API that offers methods for database access and XMLHttpRequests.

Our second contribution is at the conceptual level: The abovementioned applications can be deployed in a symbiotic or a parasitic way. In the first case, they can use the existing API of the web application they extend. In contrast, parasitic applications can be built on top of any web page without the need of cooperation of the site owner. This enables many interesting application concepts to be realized, and allows new user interface components to be added to many sites in a consistent way.

This paper is organized as follows: First, we introduce challenges arising from the use of parasitic code on arbitrary web pages. Based on this we present a technical approach for allowing the deployment of static, dynamic, and user-generated code. In chapter 4 and 5 we present to case studies, demonstrating how our platform can be used to augment web pages with annotations and dynamic code and which issues arise thereof. Finally we present related work in chapter 6.

2 Challenges of Parasitic Applications

In contrast to Berners-Lee's vision of the WWW, the web as we experience it today has many restrictions and follows standards only to a certain degree. This makes it difficult to build applications on top of web documents which enhance those documents.

2.1 One-way Information Exchange

When considering the flow of information of web documents on the World Wide Web, it is obvious that a one-way information exchange from the author to the reader prevails, since the user sitting in front of the browser has read-only access to web documents. Many different tools are available for the document author to visualize his information, while the interaction of the reader is limited to viewing the page, clicking on hyperlinks and creating bookmarks. Other types of communication on the web, such as forms or email, do not have these restrictions. It is apparent that the unidirectional information exchange limits the system in its communication potential.

Tools supporting the deployment of user-generated code on web pages can partly help to overcome this problem by providing the readers a form of backchannel. With the help of such a backchannel, they can not only address the author but also provide others with tools to embed information and thus modify the page.

2.2 Structural Diversity

Another problem of web documents is the lack of a structural layout standard, which leads to the use of arbitrary layouts. As an example, a margin is not any longer a mandatory element of each document. While numerous web pages have a fixed width, which results in a margin being displayed if the browser window size is large, there are also a lot of web pages that adjust automatically to the window width.

This raises severe problems when it comes to implementing applications that try to interact with the web page's static code. Our approach tries to overcome this problem by providing a way to symbiotically interact with pages, e.g. based on a common standard, but also explores ways of how to interact with pages without knowledge about their structure.

2.3 Multidimensionality of Web Documents

In contrast to traditional documents, digital documents allow to overcome the two-dimensionality of documents by adding new layers. The use of CSS offers the chance to position elements above each other, thus providing content to be added similar to post-its that are stuck to a sheet of paper. This way, an almost unlimited amount of additional space is available for new content. However, this raises several issues such as how to create a relation between this content and the original document, and how to define and display an anchor in the original text.

2.4 Reliable Modifications to Existing Page Layouts

At a more technical level, adding new code and layout elements to existing web pages is a non-trivial task if we consider that the existing page may change subtly over time. For example, if an annotation was placed on a page, it should still be attached to the sentence it was added to, even if other parts of the text content change. This makes it necessary to define different levels on which positioning of added content is possible, and to extract high-quality anchor information which allows proper repositioning even in case the source document is modified [11, 25].

3 A Platform for Parasitic Applications

Existing systems supporting client side interaction need to make a trade-off between several advantages and disadvantages. Systems requiring client side software installation tend to have a more intuitive and responsive user interface since they benefit from the tight integration with the browser and client side integration of the features. On the other hand, the main advantage of server-based approaches is that site visitors do not need to install software on their computer. Instead, they can start using the server side system immediately to deploy any changes.

It is the aim of our work to combine both aspects. Even though no software installation should be required, the user interface should be intuitive to use and responsive, for example by avoiding re-downloading and redisplaying the entire page during working. This is achieved by combining an AJAX-based architecture for responsive client side performance with an HTTP proxy approach which allows the deployment of code on arbitrary pages.

3.1 Requirements

In our experience, the following requirements were important for such a system:

- In the same way as the other technologies of the WWW, our implementation should achieve platform independence on the server side (i.e. with arbitrary web server solutions) and the client side (different browsers and operating systems).
- Furthermore, it should be minimally invasive in terms of required client and server side changes. In contrast to systems that require the installation of software, which is always a possible source of errors and implies an additional burden to the user, this approach reduces the users' effort: they only need to change the browser preferences for the HTTP proxy. (Moreover, by deploying the proxy as a transparent proxy for a whole network, even this small change can be eliminated.)
- Maintainability of code for the platform and for applications should be ensured. Because the JavaScript code for the application platform is delivered by the proxy each time a page is loaded by the user, the code can be modified at any time without requiring the user to make an explicit software update. Also, the strong conceptual isolation of the web server, proxy and browser makes it easier to replace any of these components.
- The platform should support different types of applications, such as static applications provided by the platform owner, dynamic applications that can be configured by the user and user-generated applications.
- Finally, the result should be responsive despite the fact that loading and storing of modifications requires a lot of traffic between the client and scripting/database server. An efficient implementation using AJAX technology is essential to avoid negative effects of the platform on the user experience.

3.2 Supported Types of Applications

Especially since the advent of Web 2.0 technology, a lot of research has gone on into the area of automating and customizing web pages based on user-generated code and content. Widely available examples are tools for annotation, adding links, building custom portals, and making alternative queries (see related work). All those tools have in common that they allow users to add content to web pages without any programming knowledge. On a lower level, several approaches allow users to execute their code on arbitrary web pages. Prominent examples are toolkits such as Greasemonkey and WBI [4] or high-level programming languages such as WebL [15] or Chickenfoot [6].

However those applications have major drawbacks. First, high level applications (such as annotation tools) are static in the sense that they do not allow for customizing or modifying by the user. Second, solutions allowing the deployment of user-generated code are not easily available, since first, the toolkit itself has to be installed as a plug-in which limits its use to certain browsers, and second, the scripts are not centrally available. Finally, existing approaches cannot easily be extended or modified though probably intended by the author, since placing user-generated content requires the use of, e.g., a database. This is difficult due to the same origin policy of modern browsers.

Our system tries to integrate the advantages of different approaches. First, we support pre-implemented applications deployable by users without any programming knowledge. Second, we allow providing applications that can easily be customized, and finally we provide means to implement JavaScript-based applications.

Static Applications: As mentioned before, a lot of effort has been put into the development of technologies supporting the deployment of applications on top of web pages, hence easing the automation and customization for users. The challenging part for such applications is the storage of data, since web pages are only virtually modified. Most of those applications are static because extending the functionality would require major changes on the provider side and cannot be simply achieved by writing client side code. Yet such applications are very useful, since no programming knowledge is required for their deployment. An example for a static application that can be distributed via an application platform such as the one presented in this work are annotation tools. In chapter 5 we explain how such a system can be integrated with our platform.
Dynamic Applications: A similar, yet more generic approach is the support of dynamic applications. Although those applications also have to be deployed within the application platform, they leave more space for customization. An example would be an application allowing for generating customized surveys. Connections to external storage such as a database server again have to be implemented within the application platform. Yet the code for the application itself is created dynamically based on user requirements.
User-implemented applications: Finally we also support applications implemented by users. We provide a module which allows for inserting arbitrary JavaScript code in any web page through the application platform. The JavaScript code is stored in a database and can be fetched and executed on demand. In order to enable users to create dynamic applications, we further provide a simple high-level API realizing access to a database. The API provides methods such as insert(key, value) which writes a (key, value) pair into the database and get(key) which returns the value for a given key. value can be an arbitrary string which allows for storing 2-dimensional data sets. Programmers can simply parse the value variable in order to store multiple attributes. This provides an easy way of avoiding the same origin policy and additionally supports users in creating dynamic applications without the need to care either for XMLHttpRequests or for database connections.

In order to allow the use of the database by multiple applications, we use prefixing for the key values in the form {app1}_{local|global}_key. Hence it is not only possible to use multiple applications but also to determine between local and global entries in the database within one application. Taking the voting system as an example, local entries would be the available options (e.g. votingApp_local_1 = "option 1"), global entries would be the answers of the users (e.g. votingApp_global_1 = "1").

3.3 Parasitic vs. Symbiotic Applications

We now introduce the concept of parasitic and symbiotic applications. In the WWW, the client side has read-only access to web resources. To virtually take control over a web page, pages need to be manipulated directly before or after they are rendered in a browser. Hence an illusion for the users is created pretending that they are given the power to modify a web page itself.

We call a web application *parasitic* if it is capable of editing, inserting or destroying content on a web page without the need for server side cooperation. We call a web application *symbiotic* if it uses functionality provided by the server side or provides functionality that can be used by the server side to modify web content. Table 1 gives examples for the different types of applications that become possible with our platform.

Table 1. Classification of parasitic and symbiotic applications.

	Non-cooperative (parasitic)	**Cooperative (symbiotic)**
Static applications	Annotations tool for arbitrary web pages	Annotation tool supported by web pages using common guide-lines
Dynamic applications	Voting tool	Customized search tool
User-based applications	Script for increasing contrast of web pages	Websites using user-based APIs (e.g. drag and drop)

Parasitic Applications: Parasitic applications interact with web pages without the explicit permission of the site's owner. This creates new opportunities since it allows users to adjust web sites to their needs.

While this may sound unattractive at first, parasitic code can be useful in a number of ways: an interested party can increase the accessibility and usability of the web application without having to coordinate this activity with the provider of the application. Furthermore, opposing goals of the application provider and of the users can be solved by users. At the simplest level, this can involve removing advertising, but more controversial changes are also possible, such as preventing users from accidentally signing up for a service they have to pay for. Finally, it is possible to enrich existing applications with new functionality, e.g. by interfacing it with other online services such as maps, dictionaries or even related services of competitors.

Symbiotic Applications: Web pages and applications deployed on top of them can also interact in a symbiotic way. Web page owners can support the use of applications provided by our platform in different ways:

– *Page formatting*: Repositioning of additional UI elements is not an easy task. Web pages that use identifiers for areas containing text can support applications in a way such that places where insertions or modifications happen can easily be retrieved once the page is loaded, especially if they moved to a different location.
– *Provide APIs*: Web site owners can provide APIs to be used by applications deployed via the platform. Hence, programmers can be supported and encouraged to write applications, thus increasing the value of a page. Like this, dynamic applications can be supported by providing them access to, e.g., a local database.

Further, page owners can also benefit from deploying platform-based applications:

– *Piggyback applications*: Page owners can use the APIs provided by the platform to implement applications outside their web server. This allows the use of applications among multiple page owners. An example would be a rating system supported among a company's web pages. Hence a user-generated rating could be created based on comments and ratings and stored in a third party's location (in this case the application platform) thus increasing its credibility and liability.
– *Increasing usability/functionality*: The platform can offer scripts that increase the usability and add functionality to websites by offering tools to the user for customizing and formatting of web pages based on their needs. A simple example would be a script to adjust the font-size according to the users' preferences.

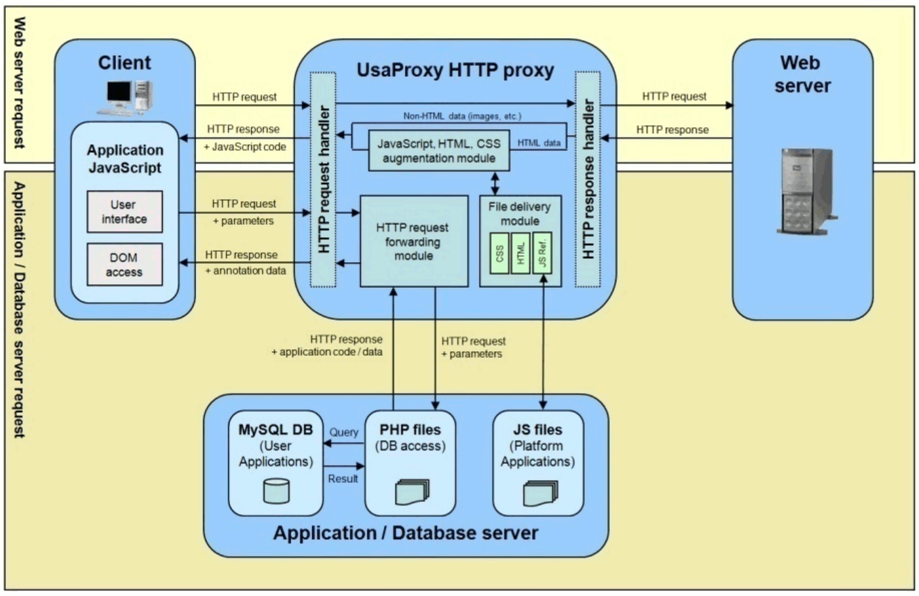


Figure 1. Components of a proxy-based application platform: application server, HTTP proxy and client side JavaScript. With this approach, no installation of software is necessary on the client browser or the web server.

3.4 Implementation

The implemented application platform consists of three components: an HTTP proxy, an application/database server and the client side JavaScript code which supplies the main functionality as well as the user interfaces. Figure 1 gives a simplified overview of the interaction of the components during operation of the application platform.

HTTP Proxy: The HTTP proxy UsaProxy ([3, 4, 5]) forms the center of the application platform since it connects the client side JavaScript code and the application server. Its first task is to embed Java- Script code on-the-fly on any page that is sent from the web server to the client in response to a standard HTTP request. This makes it possible to realize the embedding of content on the client instead of the proxy. In order to add the JavaScript code, UsaProxy monitors all HTTP requests which pass through it. In case the server delivers HTML content to the browser, the content type of the server response is text/html or text/xhtml, and the returned document is modified. Other content types such as videos and images are forwarded without any changes. The modifications to the original HTML content are small: a <script> tag is added inside the document's <head>, and its src attribute references the annotation JavaScript. The same approach is used to include a CSS style sheet which controls the layout of the user interface for the application platform as well as the layout of the modifications themselves. Finally, the elements for the user interface and the API is inserted after the opening <body> tag.

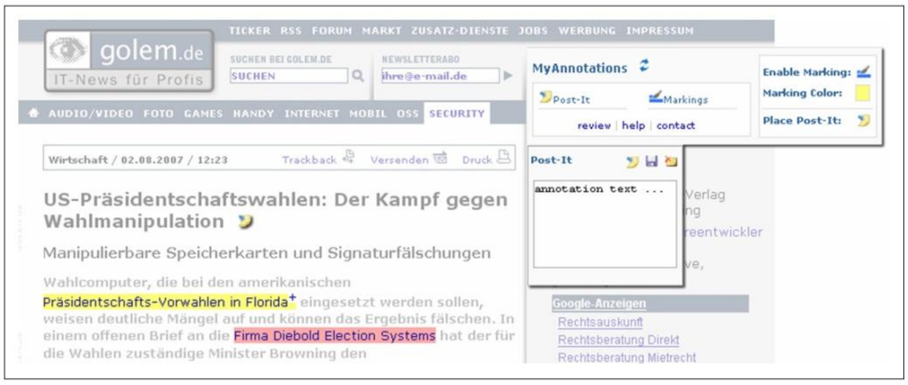
To access the application (script) data in the database on the annotation server, the JavaScript which is run inside the browser as a result of the above modifications uses further HTTP requests. XMLHttpRequest objects provide a convenient way of downloading the data. A problem when doing so is that, for security reasons, modern browsers require that the requests are made to the same server which also supplied the original web page. This "same origin" policy is circumvented in the following way: the JavaScript simply makes a request to the same server that the HTML was requested from, which is allowed by the browser. The requested URL is special in that it apparently attempts to access the directory /usaproxylolo/httprequest/ on the server. However, in reality, the request never reaches the original web server. Instead, triggered by the special directory name, it is intercepted by UsaProxy and redirected to the application server, which answers the query and returns the required data.

Application and Database Server: The purpose of the application server is to store the code for both, platform-side applications and for client side JavaScripts in a database, and to retrieve it later upon request. The database is accessible via PHP scripts, which handle storage and retrieval of the data. Furthermore, they pre-process data before returning it to the browser, which simplifies the work of the JavaScript.
Based on the type of modification required by the deployed applications (annotations, voting tools, text marking), different types of information are stored. They can be separated into three classes:

– *Content information*: data such as the code for creating the voting tool or text that is selected by a marking.
– *Positioning information*: the topmost positioning information is the URL of the page an application or modification was created for. Additionally, x/y coordinates are stored for relatively positioned content whereas for selections, a string representation of the DOM path, the surrounding context, the actually marked text, and, if available, the ID of the node is stored.
– *Additional information*: all types of information not directly related to the content or the positioning such as the date a modification was inserted or updated, the author or the title of the page.

Client-side JavaScript: The client-side JavaScript code is inserted into every page by means of the HTTP proxy. Its purpose is to provide the interface for loading and executing available applications from the database. Access to the database is realized using XMLHttpRequest to server-side PHP scripts.

In a similar fashion, the API is made available to programmers using the platform for distributing their applications. The high-level functions that allow programmers to use the platform's database are written in JavaScript and by default delivered by the proxy by embedding a script tag in the page <script type='text/javascript' src='UsaAPI.js'>. Further methods can be simply added by updating the remote JavaScript source file. The API would be available immediately for all users hence meeting the requirement of easy maintainability.



Wirtschaft / 02.08.2007 / 12:23

US-Präsidentschaftswahlen: Der Kampf gegen Wahlmanipulation

Manipulierbare Speicherkarten und Signaturfälschungen

Wahlcomputer, die bei den amerikanischen Präsidentschafts-Vorwahlen in Florida+ eingesetzt werden sollen, weisen deutliche Mängel auf und können das Ergebnis fälschen. In einem offenen Brief an die Firma Diebold Election Systems hat der für die Wahlen zuständige Minister Browning den

annotation text ...

Figure 2. Elements of the system's user interface: control panel, sticky notes, markings.

Modifications to the Existing Page Layout: Modifications of the original HTML can happen at different granularity levels: The same change (e.g. adding a layer with UI elements) can be performed on all pages of a domain, or it can be tailored for exactly one page on a website. Within pages, the modification can apply to a certain node which must be identified. At the most accurate level, it is specific to individual characters, such as text that has been highlighted by the user.

To be able to implement applications capable of making changes on these levels, different types of information are needed for reinserting the changes correctly on a page. This includes anchor text information and surrounding context [8] as well as structural document information and absolute positioning information. Thus, the platform for deployment of applications supports not only the simple case that absolute positioning is used to add new elements at fixed positions on the page, but also that the positioning depends on the properties of a certain element in the existing document's Document Object Model (DOM) tree. To allow individual characters to be addressable, e.g. to ensure that an annotation for part of the text appears next to the relevant words, the platform can identify the characters using the offset within their enclosing element. Alternatively, it can store the marked text, i.e. the words or sentences that the user selected when he created the annotation, and employ a substring search at a later time to find it again. This approach can be made more robust against changes on the page by not only storing the marked text, but also some of the text surrounding it.

4 Case Study 1: A Web Annotation Tool

One of the best-researched piggyback applications on the web are annotation tools. Hence we had a student implement such an annotation tool as a proof-of-concept for our application platform during a master thesis. An annotation tool is an example for a static application deployed on the platform side. However, we added several dynamic elements that allow for customization such as dynamically choosing the marking color.

In this section we give an overview of the design process and implementation and in a final step the evaluation in a user study and a real-world deployment.

4.1 Implementation

The annotation tool provides two basic forms of annotations: the marking of text, similar to the use of a highlighter, and placing sticky notes onto a website, similar to sticking paper post-its to a sheet of paper. Additionally, inline comments are introduced, which allow for associating comments with marked text passages. Figure 2 shows the different types of supported annotations and the expanded control panel.

The basic idea for the marking tool is to simply change the background color of the text selected by the mouse, thus creating a similar effect to using a highlighter in the real world. However this approach is technically limited to the marking of text so that it is not possible to highlight arbitrary page elements.

In order to offer the opportunity to create a text comment related to a marking, similar to notes scribbled between the lines or in the margin near the annotated text on physical paper, the marking concept is extended by so-called inline comments. Once created, they can be displayed either as tool tips or as text rendered next to the marked text. This way of displaying inline comments differs significantly from real-world annotations, so special attention was paid to it during the evaluation.

The use of sticky notes is similar to the real-world Post-It counterparts which can be placed everywhere on a page. To allow moving the sticky notes around a page, a drag-and-drop functionality is implemented so that sticky notes can easily be positioned in arbitrary locations. An interesting extension of this concept trying to deal with the limited space on a web page is to provide a minimize function which transforms sticky notes into small icons that can similarly be dragged around, but do not obscure any elements on the page.

To enable access to the previously described functions, it is necessary to provide a control panel, a layer that is automatically inserted on each page. The control panel offers access to other features such as hiding, displaying, expanding and collapsing the annotations, creating summaries or overviews, and using the notification tool.

The positioning of the control panel is a non-trivial problem. For automatic positioning, the system would have to interpret the structure and the content of the page in order to determine whether important areas of the page are obscured. To deal with this, the control panel can be dragged to any position on the page by the user. Additionally, a minimizing feature of the control panel is provided that reduces the control center to a small box showing only the most important functions.

4.2 Real World Usage Scenario

In order to assess the applicability of tools developed for the use with our application platform in WWW, we tested the annotation tool in the real world. The system was set up and adjusted for the online archive of the German weekly newspaper *Die Zeit*, in preparation for productive use with the pages which comprise the archive.

The goal of the evaluation was to discover potential issues that arise from using our system on pages in the World Wide Web. Therefore, we not only intensively tested all features of the system, but also closely examined the internal HTML structure of the page, the layout and the use of CSS styles.

It turned out that the entire functionality of the annotation tool could be used throughout the website without any restrictions. However, we discovered some potential issues that might interfere with the use not only of the annotation system but also with other tools deployed via the application platform. Most of the issues are related to the modification of content and/or structure of web sites.

– *Dynamic URLs*: since each annotation is uniquely defined by the URL of the page it belongs to as well as its ID, the insertion algorithm relies upon the URL for correctness. Hence, URLs that change according to session IDs or dynamic parameters may lead to duplicate pages so that annotations can no longer be associated with a specific web page and get orphaned.
– *Dynamic page width*: while markings are positioned directly in the DOM structure, sticky notes strongly rely on the layout of a page since their position is recorded in pixel coordinates. This way, pages that do not have a fixed width and/or are not left-aligned lead to sticky notes being displayed in different locations for different browser window sizes.
– *Dynamic page content*: for pages with dynamic content, the DOM tree can change thus causing the positioning information of annotations to become invalid. This can be circumvented by predefining areas for dynamic content. Thus the DOM path will only be changed on a level where it does not affect the positioning algorithm for the annotations.
– *Overriding global style sheet settings*: the CSS rules which are intended for formatting the annotations should be designed with care to avoid that they influence the original site layout. This is achieved by defining a special class for all parts of the annotation UI, and assigning it to the UI elements.

4.3 Summary

The implementation of the annotation tool prototype helped us considerably in understanding issues and challenges arising from the deployment of piggyback applications. We think that applications provided via our platform could especially benefit in symbiotic scenarios by following common design guidelines or even standards. Applications will be most successful once they are well integrated and their functionality tailored towards specific tasks.

However, we also showed that such tools can be deployed as parasitic applications. Yet, creating reliable parasitic applications requires a lot of effort, such as the implementation of complex positioning algorithms.

5 Case Study 2: UsaScript

In a second case study, we implemented UsaScript, a tool for testing the deployment of user-generated code on arbitrary web pages. The tool uses the infrastructure of the application platform to store JavaScript code in the database and to make it available on any web page to any user.

A simple example would be a script that overcomes the very common problem of web pages with low contrast between background and text. A user could implement a script based on three lines of code that sets the background color of an arbitrary web page to white and the standard text color to black:

```
var bodyNode = document.getElementById('body');
bodyNode.style.background = '\#FFF';
bodyNode.style.color = '\#000';
```

A user who wants to use that piece of code can simply choose it from a list of available applications provided by the platform. The script is loaded into a <script> tag embedded in the page by the platform and executed immediately.

However, the tool also allows for writing more complex applications that provide, for example, a GUI. In order to prevent the code of being immediately executed once loaded, programmers can define JavaScript methods to be called later, even by other scripts. By following this approach, UsaScript does not only support the deployment of user-generated scripts, but also gives programmers the opportunity to provide APIs for other programmers or the site owners.

An example would be an API that enables drag and drop for page elements. This API could provide a function makeDraggable(id) which assigns drag and drop functionality to the element id. Page owners could then define page elements which should be draggable (e.g. products in an online store that could be dragged to the shopping cart). To users who are not using the application platform, the web page appears normal. However, once they use the application platform, they are be able to drag and drop elements without any further required action since the API for the drag and drop functionality can be loaded based on the URL of the page.

6 Related Work

Since the advent of the WorldWideWeb, programmers try to realize Tim Berners-Lee's vision of its interactive and bidirectional use. There have been numerous attempts to build systems enabling users to customize and modify pages of the WorldWideWeb by adding content and controls. However, existing solutions require either a special server side setup or installation of software on the client machine, both of which limit the areas in which the system is useful.

First we focus on research in the area of customization and modification in general, second we look at research that has been carried out in order to achieve this through augmenting web pages by content and controls. As a third part we especially focus on annotation tools as an illustrative application.

Bolin et al [7] have proposed a categorization of tasks supported by tools that deal with the automation and customization of web pages on the client side. The categorization distinguishes between automating repetitive operations, integrating multiple websites (e.g., incorporating a map service inside a web page), and transforming a web site's appearance.

Suitable approaches for automation include scripting languages such as Perl, Python, or WebL [15] but also tools that support the recording of macros such as LiveAgent [16] or WebVCR [1]. Those tools allow for recording the actions necessary to access hard-to-reach content and replay it later.

Examples for approaches that deal with transforming a website's appearance are toolkits such as the browser extensions Greasemonkey and Platypus, as well as WBI [4], a pre Web 2.0 approach that observes user interactions by using different kinds of agents.WebL [15] and Chickenfoot [4] provide a high-level language to ease the manipulation of web pages. The advantage of Chickenfoot is that it supports the modification of web pages without requiring knowledge about HTML programming.

Concerning modifications of a page, mechanisms and strategies are required to insert content into the page. Bouvin et al [7] present an overview of web augmentation strategies. They define a tool as a hypermedia augmentation tool if "it through integration with a web browser, an HTTP proxy or a Web server adds content or controls [. . .] with the purpose to help users organize, associate or structure information found on the web." Such tools can be divided into four categories: structuring/spatial, link creation and traversal, guided tours, and annotations/discussion support.

First, spatial hypermedia describes applications where link structures are not shown explicitly any more but rather implicitly based on the spatial relationship between objects, hence providing a powerful tool for organizing and structuring the web. An example is Web Squirrel [20] that helps users to organize their URLs in information farms.

Second, example tools for link creation and traversal are Chimera and DLS. Chimera [1] is an experimental system that allows for displaying structural information of a page in a separate program (applet) or within the browser by hooking up a web server to the Chimera server hence translating the Chimera structures into HTML. The Distributed Link Service (DLS) [10] is based on the MicroCosm hypermedia system [13]. It allows for attaching a link service menu to the browser by using a wrapper. This wrapper would contact the link server once a link was clicked in this menu.

Third, tools supporting guided tours are mainly used in educational settings. Walden's path [11] uses a path authoring tool such as VIKI [19] to compose a trail that students have to pass. Trails are stored on a Path Server as well as CGI scripts to provide an interface to the path. A similar approach is followed by WebVise [12], an open hypermedia service which provides a link, annotation and guided tour authoring interface integrated with MSIE. It can be accessed in arbitrary Web browsers via a proxy server interface.

Finally, annotation tools such as implemented in the presented case study have been widely examined and numerous approaches of deploying them exist. Most common are client side browser extensions such as Yawas, Diigo, Fleck, and Stickis, or bookmarklets such as sharedcop. Yet, also entire browsers have been implemented which support annotations such as comMentor [17]. On the other side, also several serverbased approaches exist such as CritLink [21] which works based on prefixed URLs or Dashnote which is, similar to Wikis, entirely deployed on web servers. Between those two solutions also hybrid approaches exist such as Annotea [14] which allows for storing the annotations locally or on an annotation server.

7 Conclusion

In this work, we have introduced the concept of parasitic and symbiotic applications capable of bringing Web 2.0 to any page of the World Wide Web. This was realized by implementing a proxy-based application platform. Our approach offers the opportunity to deploy static and dynamic applications provided by the platform owners as well as the deployment of user-based code.

For the implementation of the concept, UsaProxy served as a basis for inserting JavaScript code on-the-fly into each web page delivered from a web server to the client. This JavaScript code provides the basic functionality for inserting or editing the content of web pages. Furthermore, a HTTP proxy was extended to allow for XMLHttpRequests to a remote application server, thus avoiding the same origin policy of modern browsers. Hence, content can be dynamically loaded from or stored to the application/ database server using AJAX technology.

As a proof-of-concept we presented two case studies. One focused on deploying a parasitic application supporting both static and dynamic elements in the form of an annotation tool. We presented challenges arising from dealing with arbitrary web pages and provided potential solutions either by complex client side mechanisms or by symbiotic deployment. Issues like scalability and extensibility still need further research.

Second, we presented UsaScript, a tool for deploying user-based scripts and APIs on web pages. We outlined how this approach can also be used among web site owners to cooperate with the application platform thus creating additional benefits for the user.

References

1. K.M. Anderson. Integrating Open Hypermedia Systems with the World Wide Web. In Hypertext'97, 1997.
2. R. Atterer, M. Wnuk, and A. Schmidt. Knowing the User's Every Move - User Activity Tracking for Website Usability Evaluation and Implicit Interaction. In WWW'06. 2006
3. R. Atterer, A. Schmidt, and M. Wnuk. A Proxy-Based Infrastructure for Web Application Sharing and Remote Collaboration on Web Pages. In INTERACT'07. 2007.
4. R. Barrett, P.P. Maglio, and D.C. Kellem. How to Personalize the Web. In CHI'97. 1997.
5. J.P. Bigham, and R.E. Ladner. Accessmonkey: a Collaborative Scripting Framework for-Web Users and Developers. In W4a'07. 2007.
6. M. Bolin, M. Webber, P. Rha, T. Wilson, and R.C. Miller. Automation and Customization of Rendered Web Pages. In UIST '05. 2005.
7. N.O. Bouvin, Unifying Strategies for Web Augmenting. In Hypertext'99. 1999.
8. A.J. Brush, D. Bargeron, A. Gupta, and J.J. Cadiz. Robust Annotation Positioning in Digital Documents. In CHI'01. 2001.
9. J.J. Cadiz, A. Gupta, and J. Grudin. Using Web Annotations for Asynchronous Collaboration around Documents. In CSCW'00. 2000.
10. L.D. Carr, W. De Roure, W. Hall, and G. Hill. The Distributed Link Service: a Tool for Publishers, Authors and Readers. In The Web Journal 1, 1. 1995.
11. R. Furuta, F.M. Shipman, C.C. Marshall, D. Brenner, and H. Hsieh. Hypertext Paths and the World-Wide Web: Experiences with Walden's Paths. In Hypertext'97. 1997.
12. K. Grønbæk, L. Sloth, and P. Ørbæk.Webvise: Browser and Proxy Support for Open Hypermedia Structuring Mechanisms on the WWW. In WWW'99. 1999.
13. W. Hall, H. Davis, and G. Hutchings, Rethinking Hypermedia: the Microcosm Approach. Kluwer Academic Publishers. 1996.
14. J. Kahan, M-R. Koivunen. Annotea: an Open RDF Infrastructure for Shared Web Annotations. In WWW'01.
15. T. Kistler and H. Marais. WebL - a Programming Language for the Web. In WWW'98. 1998.
16. B. Krulwich. Automating the Internet: Agents as User Surrogates. IEEE Internet Computing 1, 4, 1997.
17. T.A. Phelps and R. Wilensky. Robust Intra-document Locations. University of California, Berkeley, 2000. http://www9.org/w9cdrom/312/312.html
18. M. R¨oscheisen, T.Winograd, and A. Paepcke. Content Ratings and Other Third-Party Value-Added Information: Defining an Enabling Platform. Stanford University, Stanford, 1995.
19. F.M. Shipman and C.C. Marshall. Spatial Hypertext: an Alternative to Navigational and Semantic Links. ACM Comput. Surv. 31, 4. 1999.
20. R.M. Simpson. Experiences withWeb Squirrel: my Life on the Information Farm. In Hypertext'01. 2001.
21. K-P. Yee. CritLink: Advanced Hyperlinks Enable Public Annotation on the Web, University of California, Berkeley, 2002.