A Meta-reflective Wiki for Collaborative Design

Li Zhu
Università degli Studi di Milano
Via Comelico 39/41
20139 Milano, Italy
+39 02 50314007
zhu@dico.unimi.it

Ivan Vaghi
EarlyMorning
Via Santa Croce 4
20122 Milano, Italy
+39 393 9507609
ivan@earlymorning.com

Barbara Rita Barricelli
Università degli Studi di Milano
Via Comelico 39/41
20139 Milano, Italy
+39 02 50314007
barricelli@dico.unimi.it

ABSTRACT
This paper presents MikiWiki, a meta-reflective wiki developed to prototype key aspects of the Hive-Mind Space model. MikiWiki is aimed at supporting End-User Development activities and exploring the opportunities to enable software tailoring at use time. Such an open-ended collaborative design process is realized by providing basic boundary object prototypes, allowing end users to remix, modify, and create their own boundary objects. Moreover, MikiWiki minimizes essential services at the server-side, while putting the main functionalities on the client-side, opening the whole system to its users for further tailoring. In addition to traditional wikis, MikiWiki allows different Communities of Practice to collaboratively design and to continuously evolve the whole system. This approach illustrates the meta-design concept, where some software collaboration between professional developers and end users is made possible through communication channels properly associated with the environment. As such, the MikiWiki environment is presented as a 'concept demonstrator' for meta-design and end-user tailoring.

Categories and Subject Descriptors
D.2.1 [**Software Engineering**]: Requirements/Specifications – *elicitation method*. D.2.2 [**Software Engineering**]: Design Tools and Techniques – e*volutionary prototyping, user interfaces*. D.2.6 [**Software Engineering**]: Programming Environments – *graphical environments, interactive environments*.

General Terms
Design, Experimentation, Human Factors

Keywords
Hive-Mind Space model, Meta-design, End-User Development, Boundary Objects, Co-evolution, Habitable Environment, Wiki, MikiWiki, Mikinugget.

1. INTRODUCTION
Web 2.0, social media and advanced information technology are changing the role of end users and the way they share and manage knowledge, and encourage cultures of participation [1]. Complex design projects also need to actively engage all stakeholders in the design process. However, communication among stakeholders often breaks down due to differences in cultures, backgrounds and modes of communication. Moreover, the co-evolution of design communities and software systems [2] requires open software development environments to support emerging needs.

The paradigm of End-User Development (EUD) aims to explore the opportunities to enable software development at use time involving the end users as experts of specific domains [3]. In fact, EUD techniques propose various approaches that allow users of software systems, who are not professional software developers, to create, modify or extend software artifacts at use time [4]. Viewing software as a continuously evolving artifact considerably contributes to the progress of software development.

The Hive-Mind Space (HMS) model [5] [6], a meta-design [7] conceptual model, has been proposed to tackle the fore mentioned issues and to support collaborative creativity among design teams. This paper describes the architecture and design principles of MikiWiki, a prototype of the HMS model that we built to evaluate the technical feasibility and the effectiveness of the model. By implementing the HMS model, we also hope to be able to evaluate the meta-design system and discover new design opportunities. We call MikiWiki a meta-reflective system, meaning that it is reflective because it makes visible and accessible parts of its infrastructure to the users. The 'meta' refers to the meta-design features available to the users. The contribution of this work is to show the feasibility of implementing HMS by extending the concept of wiki with programmable concepts, focused within a conceptual framework.

The following section briefly introduces the HMS model. Section 3 explains the reasons why we chose a wiki model as the basic infrastructure for prototyping the HMS model and introduces MikiWiki. Section 4 describes the MikiWiki architecture. Section 5 explains how MikiWiki reflects upon open structure, boundary objects and habitable environments concepts of the HMS model. Section 6 demonstrates how MikiWiki can be used in collaborative iPhone mockup design. Section 7 outlines some possible application domains. Some reflections and a brief conclusion are given in the last two sections.

2. HIVE-MIND SPACE MODEL
The Hive-Mind Space model explores the meta-design approach and aims to bring software engineers, domain experts and end users together to collaboratively work on design projects. A fundamental objective of meta-design is to create socio-technical environments that empower users to engage actively in the continuous development of systems rather than being restricted to the use of existing systems. The meta-designers design the design process as well as design context [7]. Each stakeholder belonging to a specific design community (i.e. a Community of Practice (CoP) [8]) in the HMS model is provided with a 'habitable

WikiSym '11, October 3-5, 2011, Mountain View, CA, USA.

environment' [9]. Each habitable environment provides essential tools for a CoP and allows its members to perform design activities. Moreover, it is localized to the CoP's culture, role and digital devices in use [10] [11]. The HMS model derives from the Software Shaping Workshop (SSW) methodology [3] and supports three different levels of participation and design activities (see Figure 1): i) meta-design level, where software engineers maintain the system and design environments for domain experts; ii) design level, where domain experts design environments for end users; iii) use level, where end users tailor and use the environments and tools.

In addition, the HMS model has an open infrastructure, i.e. CoPs can tailor their habitable environments.

To enhance communication among CoPs, the HMS model introduces a central communication channel serving as a boundary zone [12], where different CoPs can create, exchange, and cooperate around boundary objects [13]. For example, an architecture model or a sketch can be used as boundary objects among architects, clients and civil engineers for reasoning about design. The HMS model explores boundary objects as a) a means to enhance the communication among different design communities, and b) artifacts that mediate communication. The role of mediators is supported by the fact that boundary objects are localized differently in terms of users' role in the context, culture, expertise and background and device in use. The HMS model has been applied before to two different cases, mechanical engineering and collaborative knowledge management for tourism [5][6]. In these two applications, the environments were designed and implemented using a specific software framework, the BANCO framework [10][11]. In this paper, we present a new architecture and a different approach of prototyping the HMS model.

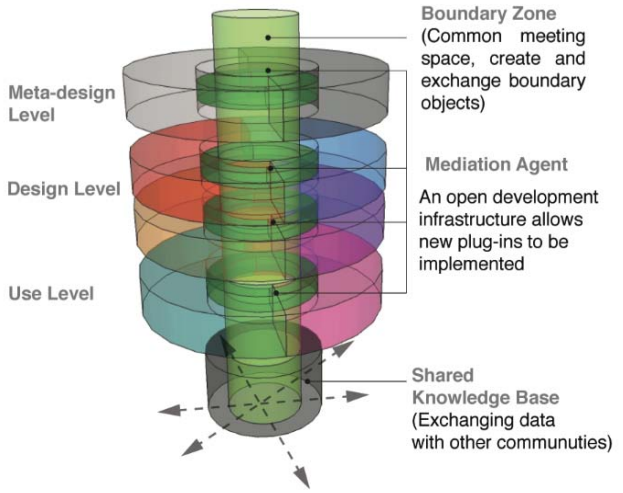

Figure 1. Hive-Mind Space model

3. MIKIWIKI

Wikis are a collection of pages that can be edited by anyone, at any time and from everywhere, and they are becoming a popular format for sharing knowledge in both academia and professional domains [14].

3.1 Starting from Wiki

Our approach and motivation to use a wiki model as the basis to prototype the HMS model are based on the following reasons:

1. Wikis encourage a culture of participation, since wikis enable users to share and develop knowledge from a wide range of domains. Cunningham suggests that wikis are useful tools for building CoPs [14].

2. Wikis have an open structure and foster social creativity.

3. The distinction between design time and use time is blurred.

4. Wikis allow incremental knowledge creation and enhancement. In order to collaboratively build up knowledge or perform design activities among diverse design teams, a wiki can be an ideal platform and format [15].

5. Wikis have also been shown to express emergent organization and a degree of meta-design via communities using wikis to discuss how the wiki itself should evolve, making the wiki both the product and the medium of communication. The open editable structure, pages as basic units of sharing, existing documented architecture models and implementation of traditional wikis, make them a good starting point for prototyping the HMS model.

3.2 Extending the Wiki

Typical wikis lack support to create new domain-oriented tools within the wiki itself. Existing wiki engines only allow users to enter passive content and do not allow users to customize wiki pages. Therefore wikis cannot be used to host or author rich dynamic and interactive content [16].

On the other hand, application wikis enhance wiki systems with lightweight programming features that aid in making data structure and processes explicit. Using these features, end-users can program a wiki to better support their collaboration [17], for instance SnipSnap [18] and XWiki [19]. This approach, however, is limited since language constructs are often domain specific [17] and users are constrained to write code in markup languages rather than to interact and inspect objects [16] [20].

In the next section we will explain how MikiWiki extends wikis with typed pages [21], which allow users to link structured text and data pages to specific templates and layouts. This mechanism allows users to incrementally evolve part of the wiki text from informal text to structured contents. The MikiWiki system is open and it goes beyond templates, allowing advanced users to process structured content by defining their own page types using JavaScript.

4. ARCHITECTURE

Client-side Web development is an emerging trend, since it provides the possibility to empower end users to create applications by merely using a Web browser [20]. MikiWiki utilizes a client side Web development approach to put end users in charge of their design problems, to enhance their tailoring power and thus to create situational applications [22] for better collaboration and problem-solving.

4.1 Architectural Philosophy

In MikiWiki, the server side supports the minimal amount of features and services that maintain basic functionality. Whenever it was possible, functionality was moved to the client-side in the form of JavaScript code. The server-side primitive services are still available to the client-side via AJAX calls.

Some functionality cannot be removed from the server side specifically, all the basic facilities that handle the sharing of information as the server acts as a repository and single aggregation point for wiki pages. We created an abstract system for sharing pages, without encumbering the server side with the specifics of what is being shared, therefore keeping it very simple.

The semantics of what is being shared are expressed on the client side, where eventual mikinuggets are executed. A mikinugget is a page embedded within another page in order to create sharable remixable components. Mikinuggets in MikiWiki are explicitly designed to reflect the HMS model's boundary objects' concept.

As a concept demonstrator, MikiWiki aims to explore some key characteristics of meta-design, i.e. design infrastructure tailorability and EUD. Therefore, the system architecture mainly addresses the tailoring flexibility of the client side, experimenting with the concept of empowering the end users to evolve the behavior of the system according to their collaboration.

Issues of security and scalability are not explicitly addressed since they are out of our research focus and we mean to keep the architecture tidy, lightweight and fully focused on collaborative tailoring and open-ended evolution.

The architecture of MikiWiki is depicted in Figure 2.

4.2 Architectural Implementation

MikiWiki is implemented as a Ruby [23] web application written on top of the Sinatra framework [24] on the server side and as HTML and JavaScript on client side, also making ample use of the jQuery framework [25] and its plug-ins.

JavaScript was chosen as the boundary-object (that is, MikiWiki's mikinuggets), creation language for several reasons: the code is interpreted on the client's side and runs on most devices, within the user's browser, and various libraries are available for building cross-platform and cross-browser JavaScript applications. There is also a growing popularity of frameworks such as node.js [26], enabling JavaScript on the server side, which we hope to leverage in the future. Although JavaScript supports dynamic scripting, it is seldom used to empower end users to influence and even change the behavior of Web programs.

In MikiWiki, the server side supports the minimal amount of features and services that maintain basic functionality. The server side handles all the tasks and rules related to the page and environment infrastructure, the basic navigation framework, authorization and notification services, while the client handles the rendering and the management of the interaction with the users.

Some functionality cannot be removed from the server side, especially all the basic facilities that handle the sharing of information, since the server acts as a repository and single aggregation point for wiki pages. Whenever a feature does not necessary have to be run on the server side, it can be expressed as JavaScript code within a wiki page. This allows the feature to be available for inspection and tinkering by the users, who might decide to customize it for their environment. Centralized 'primitive' services can be accessed by the client-side features with AJAX calls.

We mean to keep the server-side architecture very simple. The server does not have any information about the content of the pages or the meaning of the data that is being served. The associations among content page, data page and format page are authored by users, while the server only keeps track of these relationships rather than their semantics.

Once the pages are loaded on the client side and the embedded mikinuggets are expanded, then the page content is interpreted according to the format of the data page.

The server provides some additional functionality to user generated pages on the client side by exposing an AJAX interface to authentication, synchronization and awareness and versioning services.

Figure 2 shows the MikiWiki layered architecture. Embedded mikinuggets are executed within the browser as JavaScript code. When a resource is requested during the rendering of a mikinugget, static text or JSON content and related dynamic JavaScript are retrieved, passed to the rendering engine, rendered in HTML format and displayed in the user's browser.

A lifecycle for fetching and rendering a page in MikiWiki is described below.

A user accesses MikiWiki by loading the URL of a page within the Web browser. This request gets sent to the MikiWiki Server.

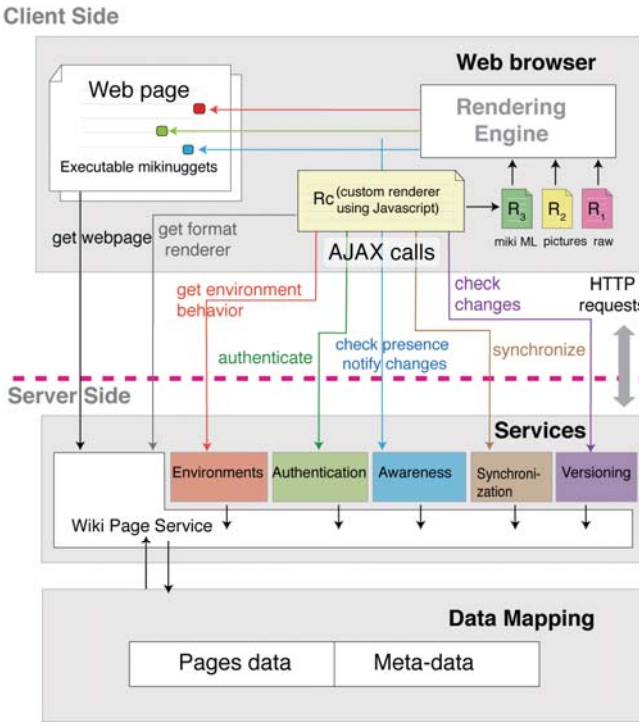

Figure 2. MikiWiki architecture

A page load request is routed to the Wiki Page Service, within the Services layer. The Wiki Page Service takes care of fetching the page content and metadata, and answers queries related to a page and its context: for instance whether this page has child pages or a parent page, whether it is an environment, etc. Additionally, the service layer provides CRUD functionality (Create, Read, Update, Delete), environment assessment information, authentication of the user, enhancing awareness, synchronizing communication and tracking versions.

The information returned to the browser might not be simple HTML, since it possibly contains mikinuggets, which are placed in hidden DIV tags. All the mikinuggets must be interpreted, expanded and rendered to become visible HTML within the page.

Once a page is loaded, the MikiWiki Rendering Engine is activated. Every mikinugget in the loaded page gets passed to the Rendering Engine to be executed and materialized according to visible HTML code.

The Rendering Engine firstly checks the format metadata of the mikinugget to see whether the predefined rendering strategies can handle it. The three basic rendering strategies support pictures, plain text (to be used when editing) and MikiWiki Markup Language.

If the format does not fall into any of these three rendering categories, the Rendering Engine exploits a custom rendering strategy. The name of the format corresponds to a MikiWiki page, containing detailed rendering instructions in the form of an HTML template or JavaScript code.

The Rendering Engine fetches the JavaScript contained by the format page via an AJAX call and executes it in the browser. The JavaScript may further make its own AJAX calls to the MikiWiki services in order to retrieve all the information needed to perform its rendering for instance, fetching further pages containing data in a JSON format, querying the environments or getting user profile information, checking file versions and so on.

Finally, the JavaScript produces the HTML code and makes it visible. If the returned HTML code contains further boundary objects, the expansion step is executed recursively until all nested mikinuggets are fully expanded and rendered in the Web browser.

5. MIKIWIKI DESIGN PRINCIPLES

The MikiWiki architecture has been designed to allow the implementation of mechanisms that support the HMS concepts. This is not a one-to-one mapping, as many theoretical concepts, such as boundary objects, cannot be reduced to a simple software system component.

5.1 MikiWiki pages as organizational structure

Information in MikiWiki is organized by pages. MikiWiki pages can act as the materialized representations of boundary objects and every page can easily embed other pages. CoPs can manipulate, create and reuse pages created by other users by referencing their URL, uniquely identifying them.

The HMS does not specify the shapes of the boundary zone, boundary objects or environments. It specifies 'bordering' and 'containment' relationships, but not the nature of the space or its navigation. The focus of the HMS model is not on the space metaphor, and thus in MikiWiki we chose the simplest interpretation of *space* that was consistent with the model. We decided to represent the collaborative space as a tree structure of content, allowing the organization of artifacts in environments and sub-environments.

A page can be either a "Folder page", used primarily to group and navigate sub-pages, or a "Content page", used to convey users' content. From within a folder it is possible to create new pages and to upload pictures.

A folder can also be promoted to an environment, a basic mechanism to manage and organize boundary objects in the HMS. The members of a CoP can tailor an environment to reflect their thinking and workflow. Moreover, within this local environment, a CoP can create pages related to the tasks at hand as well as create sub- environments.

5.2 Content pages, Data pages, Format pages

MikiWiki pages are accessing points to the deeper level of the system. They are the building blocks of a complex, multi-level medium that can be constantly tailored by users, thus allowing evolving the system's structure and behavior to evolve. MikiWiki supports three basic types of pages: text pages, data pages and format pages. All of them act as boundary objects, in the sense that they can be shared and extended by users.

Content pages can embed other content pages or data pages. Data pages are typed pages with structured text (either in JSON, XML or any text convention that the user chooses) and they are associated with a specific format page. A format page can be either an HTML template with insertion points or JavaScript code. A format page is basically a set of rules or some code that defines how to render a data page.

Figure 3 shows a rendered content page containing a chat mikinugget. Advanced users can choose to highlight the mikinuggets embedded within a page and make visible a link to the pages containing their data (wall-data in the figure) as well as their behavior (wall in the figure). These links facilitate and stimulate advanced users to access meta-level functionality and modify or clone existing functionality and tools.

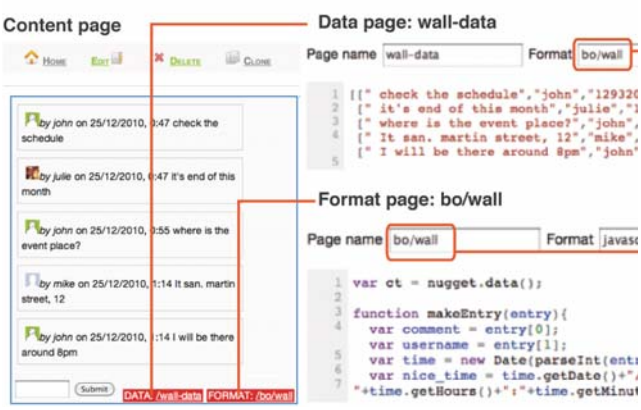

Figure 3. Content page, data and format page of a chat mikinugget

5.3 Mikinuggets

MacLean suggested that a more incremental approach is desirable to allow end users to express their customization requirements as much as possible using skills they already possess, and to equate increases in customization power with proportionate increases in the level of expertise required [27].

We provide a set of mikinuggets to be embedded for instance ranking, commenting, annotations, drawing tools, notification, online presence, change-tracking, user-tracking, chat, to-do list, video embedding, access control and profile. By utilizing these mikinuggets, non-programmers can easily start using and remixing existing objects, while advanced users can clone and modify these mikinuggets and consequently introduce new behaviors.

The separation between user interface and application (the surface and the deep [28]) restricts end users to simple manipulation of surface features, while the deeper system remains only accessible to developers. However, developers often do not know all the ways in which the system might be used by different end users over time [29]. Hence some lower-level details of system behavior should be also available for customization at the user interface.

Mikinuggets allow end users further appropriation of the system. Mikinuggets' pages act as a mechanism and interface for supporting the creation and evolution of software artifacts beyond their initial form. Moreover, mikinuggets are also a medium made of captured knowledge. CoPs can incrementally construct knowledge via mikinuggets during collaboration and communication.

Figure 4 shows a screenshot of a page containing a description of a set of role-playing game's characters and three mikinuggets, namely, stickyNotes, tag and notifychanges nuggets. The

<<nugget's name:parameters>> syntax demonstrates the way to include mikinuggets.

In this case, a player can create stickynotes, add textual annotations and place them at any location on the page. Stickynotes' color (pink, blue, green) can also indicate the importance of the annotation or annotation types according to

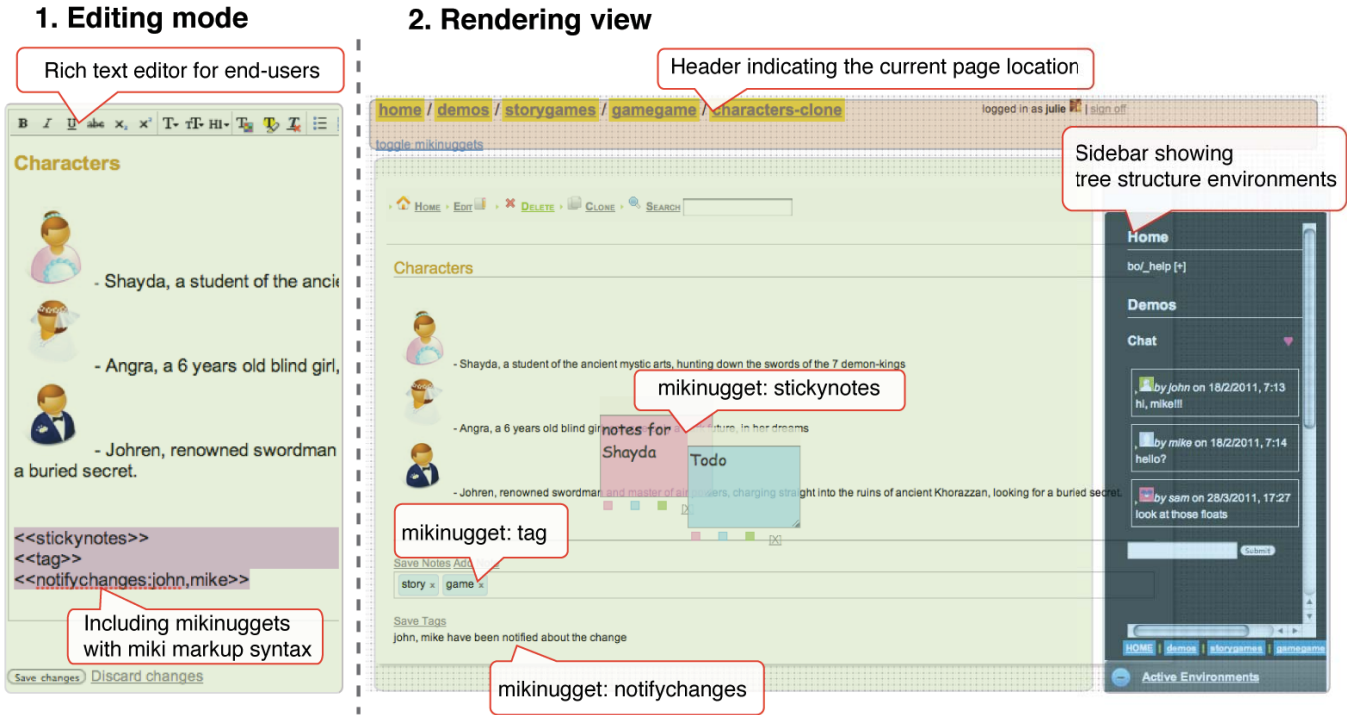

Figure 4. Embedding multiple mikinuggets

player's need.

The tag mikinugget allows players to assign keywords to pages. The auto complete tags can be predefined in a glossary file. Players can create their own tags. The notifychanges mikinugget allows players to specify who should be notified via emails whenever the page is modified. <<notifychanges:john,mike>> indicates that Mike and John will be notified by email when this page is modified. Mikinuggets also enable individuals or CoPs to take control of their communication experiences.

A MikiWiki page consists of three parts, a header Div to indicate the current page path and a user's profile information, a content area and a sidebar showing the active environments.

To better support different levels of participation, MikiWiki provides two different content editors, a rich text editor, which is a WYSIWYG editor that allows novice users to easily create text context, and a JavaScript editor aimed at expert users who are able to programming. Figure 4 illustrates the characters page in editing and rendering two different views.

5.4 Habitable environments

In accordance with the HMS model, a flexible mechanism is designed to allow CoPs to partition and locally configure communication. Environments are used as a way to associate specific behavior to a large set of pages.

5.4.1 Designing Habitable Environments

Any folder in MikiWiki can be promoted to an environment and it can be customized by embedding a set of mikinuggets.

Environments do not impose a predefined structure on all CoPs, but allow sharing specific features among selected members. For example, access control in MikiWiki is not an inherent property of all environments, but it can be achieved by including a "check point" mikinugget in the environment settings page: all the pages within this environment therefore inherit the access control property.

If an environment loads an "online-presence" mikinugget, all users that are accessing that environment become reciprocally aware of one another's presence as MikiWiki starts tracking user activities within the environment and displays which users are browsing that environment.

The sidebar shows all the active environments: the current environment and all the upper level environments enclosing it (Figure 4). Since a sub-environment inherits all the characteristics of its upper level environments, the mikinuggets of the current environment and its parent environments are activated.

5.4.2 Habitable Environments for Mediation

The HMS model addresses the complexity of design projects by bringing together different CoPs into a Community of Interest (CoI) [30] to work together. Design activities require collaboration among different CoPs and are characterized by symmetry of ignorance [7]. This means that no CoP possesses all the knowledge, or more important knowledge than others, but rather that all the knowledge is symmetrically important in the problem solving process and tacitly distributed among the community [31]. Exploiting the power of the "symmetry of ignorance" leads to creative results [32].

In a collaborative system, there is always a trade-off between convergent and divergent points of view. Within the same knowledge system, a CoP can take advantage of a shared background and communication is comparatively easier internally than with outsiders. Members of the CoP tend to be biased by

communicating with people sharing the same practices. Divergent viewpoints in a CoI are therefore needed for solving complex design problems and coming up with innovative solutions. Nevertheless, CoPs with different cultural backgrounds use different systems of signs, languages and representations [33] and may have different perceptions as well as interpretations. Communication is needed to reach a common understanding about the messages they exchange.

Environments are also a mechanism to negotiate the awareness of convergent and divergent viewpoints regarding an object of interest by presenting it in a meaningful way to different users. For example, when designing an apartment, the same information might be presented differently in different environments, i.e. floor plans in an environment for architects, construction details in an environment for civil engineers, a 3D rendered model in the buyers' environment, and a price table in the contractor's environment. Moreover, being aware of those differences eventually enhances the mutual understanding and supports CoP collaboration.

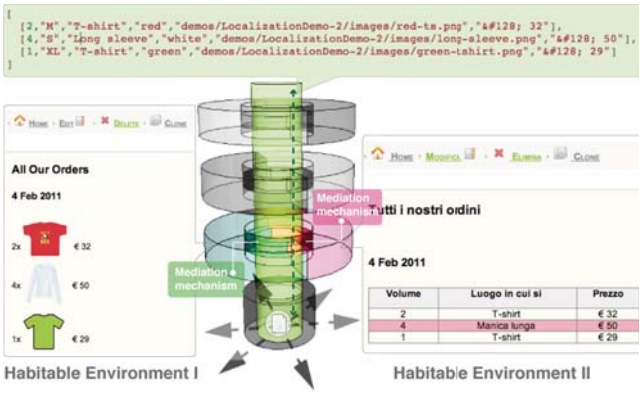

Figure 5. Two different habitable environments

MikiWiki supports the mediation mechanism of the HMS model by allowing mikinuggets to be represented differently within individual environments.

Figure 5 shows an example where the same data is visualized differently in two different habitable environments: shopping items represented as photos for Americans and as a price list table for Italians. The data page is stored in the knowledge base and can be accessed by all the design communities. On the other hand, each environment has its own mediation mechanism to materialize the shared data in a way that is meaningful to its inhabitants.

The mediation mechanism, as envisioned in the HMS model, was initially based on user's role, culture and platform in use. However since the mediation mechanism is also handled as a wiki page, it is open to modifications. Instead of gathering data about the end users' culture, role and device, and automatically filtering information for them, the open mediation mechanism goes beyond the "perfect personalization" dilemma [34] and enables the environment designer to decide directly what is meaningful information and how to represent it.

Making the mediation mechanism open and editable empowers CoPs to design their own environment and optimize their workflow at different design stages.

5.4.3 Mediation Mechanism for Tweaking and Tinkering

A habitable environment is also a space for end users to further refine their means of communication, tinkering with new tools and tweaking mikinuggets according to their aims.

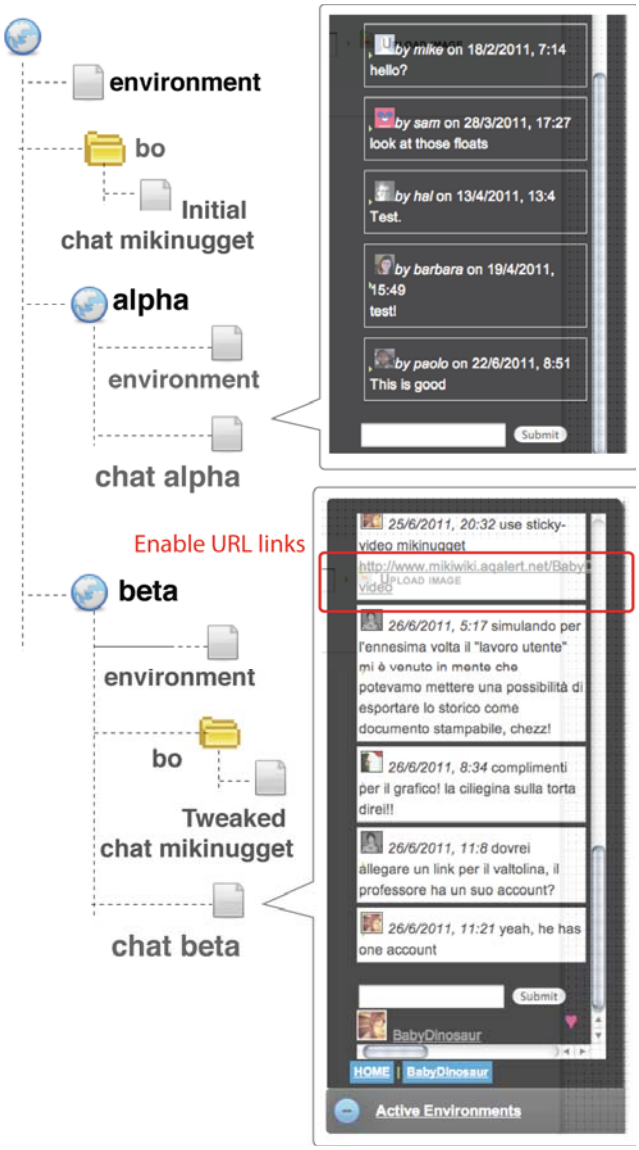

Figure 6. Initial chat mikinugget tweaked for better collaboration

Figure 6 shows how students tweaked the existing chat mikinugget to support their collaboration on a project. The students started using the chat mikinugget to exchange ideas that were then copied to wiki pages. This presented a few unexpected issues: links copied to the chat were not clickable and the chat text was written with a white font, making it unreadable when pasted on the white wiki page.

When a chat page is created, it loads the standard chat behavior, as in the case of 'chat alpha' in Figure 6. The students cloned the chat code page within their own environment ('beta' in the figure) and started modifying it. Consequently, every chat page visualized within the beta environment picked up the new local chat code.

In section 8 we give more details on the page lookup mechanism and how it is used to support the mediation mechanism.

In the *tweaked chat mikinugget,* three changes have been made to the initial chat mikinugget: i) each chat entry's font color is black, allowing users to copy text to wikipages with a white background; ii) the chat entries' background color is changed to white to offer some contrast to the black font; iii) each entry is parsed and if a URL link is found, it is automatically made clickable in order improve contextual navigation.

Some weeks later, another group of users involved in a different project appropriated the tweaked chat created by the first group and started using it in their own environment.

Notably, mikiwiki supports extending new functionalities without compromising the working of an existing mikinugget for other users; the students cloned the *initial chat mikinugget* code page in a local environment and worked on their own version. If the tinkering did not work out, the students could have deleted the local version and simply used the closest available mikinugget with the same name, in this case the *initial chat mikinugget*. The tweaked mikinuggets can then be shared with other CoPs.

6. MIKIWIKI IN USE

The first MikiWiki use case was developed to support diverse CoPs (mainly software engineers, designers and clients) collaboratively designing iPhone applications.

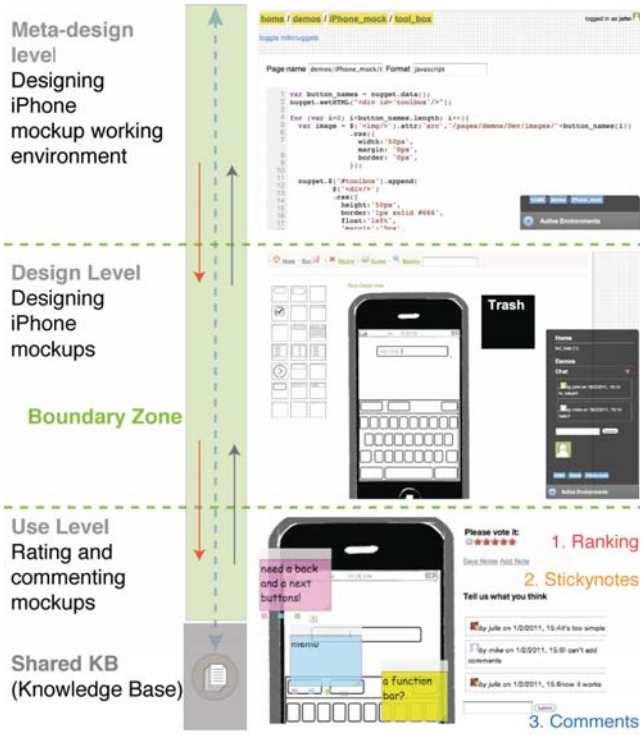

Figure 7. Three different levels of participation

1) At the meta-design level as shown in Figure 7, software engineers design the workflow and interactions, which generate a mockup design environment for designers to use. In this case, the mockup design environment is composed of three mikinuggets written in JavaScript, namely toolbox, canvas, and trash.

2) At the design level, designers can use a mockup environment to create an iPhone mockup by simply dragging and dropping components and sharing their results with their team. Alternatively, designers can create their own toolbox mikinugget with customized design components, while the iPhone canvas mikinugget can be easily replaced by a new mobile device or platform for another design case. Figure 7 shows a wireframe iPhone mockup environment.

In this case, the designed mockups are also wiki pages, acting as boundary objects containing other boundary objects, exchanged and shared in the community.

3) At the use level, users can vote on annotate and discuss the mockups. The final end user environment consists of a mockup sample by designers and three mikinuggets, ranking, stickynotes and comment respectively.

Using the MikiWiki markup language, a user can easily include these mikinuggets by using <<ranking>>, <<stickynotes>>, <<comments>>. Stickynotes can be used by clients to ask for new features and modifications or to annotate the mockup with suggestions and comments.

Nevertheless, all the design activities can happen in the same environment. The levels of participation are not predefined but emergent in terms of users' skills as well as roles, which are highly dynamic.

7. Application Domains

MikiWiki could provide a fundamentally different design methodology for a broad spectrum of application domains, including the following:

- **Mockup Environments**. Many applications for mockup design have been developed, such as MockingBird [35], Balsamiq [36] and MockFlow [37]. In the design of a UI mockup system, MikiWiki would not only support rapid prototyping, but also it would bring together different design teams and support their communication. Mockups could be easily stored and shared by the communities within MikiWiki.
- **Productivity Applications.** Applications such as Google Docs, GoogleTalk and Gmail already offer a high level of sharing and collaboration within an organization, between friends and with the larger public. These applications however lack support for customization, which is often achieved only by using specialized browser plug-ins. The use of some of the techniques employed by MikiWiki would allow building of a custom features and leverage the existing communication mechanisms to share them with other users.
- **Online Spreadsheets.** Google Spreadsheet and other online applications allow users to work on the same spreadsheets. Google Spreadsheet went so far as allow the scripting of new formulas in JavaScript. We can see how these formulas and macros could be shared and maintained collaboratively across an organization using mechanisms similar to the ones employed by MikiWiki.
- **Collaborative Creative Writing.** MikiWiki can be used as a brainstorming tool and platform to build dedicated tools for defining basic story principles and structure, and leave participants generating ideas and developing plots.
- **Online Games.** Games have been open to user customizations (known as 'mod') for a long time. SecondLife is a great example of how to provide extensive customization and scripting from within the environment itself [38]. As in MikiWiki, online gaming platforms could benefit from an environment for social customization and sharing of programmable artifacts and characters.
- **Collaborative Software Design [39].** Software design and design process documentation can be easily integrated in

MikiWiki. Beginner, advanced and expert users are encouraged to participate in software design, discuss it, share code and test application prototypes. Github (social project sharing) [40], Cloud9 (team based online development environment) [41] and jsFiddle (online JavaScript prototyping) [42] are all projects working in this direction.

- **Web Widgets.** Projects such as Netvibes [43] and iGoogle [44] focus on creating cross-platform reusable widgets. These widgets are generally hosted by third parties, although Google has been working on integration with its own Google App Engine [45]. MikiWiki could have the same potential as well as highlighting code sharing and the social aspects of development.
- **Social Networking Platforms.** Currently, the most popular social networking platforms, such as Facebook and LinkedIn, support the development of third party integrated applications, yet are lacking an integrated development environment and custom services live on third party servers called via callback APIs. MikiWiki-inspired techniques would work on client-side extension mechanisms and integrated shareable code environments, relying on the existing social features.
- **E-learning and Knowledge Sharing.** Knowledge can be made more informative and more vivid by combining multimedia and mikinuggets. For instance, users can attach interactive examples to any articles. Articles then are not just a static form but become interactive essays by combining more complex interactive behaviors. Sharing course notes, articles, discussions can also be made explicit within the environment.

  Based on GoogleWave technology [46], Google Shared Spaces [47] is a wiki-like environment that allows end users to create sharable online 'spaces'. Spaces can also embed widgets, which however are developed and hosted outside the Google Shared Space. Google Shared Spaces is very close to the idea of a programmable wiki, and it could be enhanced by integrated client-side programming and social sharing features.

In short, MikiWiki can be applied to systems that require creative work, allow a high degree of customization and benefit essentially from social sharing. Comparatively, the advantage of MikiWiki is that it does not rely on external servers; it provides integrated environments and allows immediate feedback, through which code and features become social artifacts. However, it could also suffer from potential trust issues related to the code. The scripting language is limited to JavaScript and thus relies heavily on a backend with a set of given functionalities.

8. REFLECTIONS

Our understanding of the HMS model also evolved as a result of implementing it, as technical affordances were exploited to evolve the initial model and open up new opportunities. In this section we present some observations on this process.

Tinkering, mashups and intercommunicating components

We found HTML and JavaScript to be extremely effective for quickly building new features and mashups with Web applications, mostly due to the recent adoption and growth of the jQuery framework within the open-source community. Adding language translation to an environment can be as simple as accessing the Google translation APIs via a ready-made JavaScript library.

HTML and JavaScript also facilitate building loosely coupled systems. We did not design MikiWiki with the idea of creating interoperable artifacts, yet the event-based model of JavaScript allowed us to create independent components that could interact with one another at runtime, as in the case of the iPhone application. The toolbox mikinugget allows selected UI elements to be draggable by mouse, the canvas mikinugget accepts selected UI elements being dropped on, and the trash mikinugget not only accepts draggable UI elements but also erases them. However, these three mikinuggets can be used independently.

Openness vs. blank page syndrome

As users of the system, it was not always clear what the next obvious step could be. The system is very open, and thus at times this very openness encourages ad-hoc activities, yet this could leave a user at a loss what to do next. We still need users to design the initial environment before co-evolution can take place.

We found MikiWiki to be very effective for tinkering. It is easy to access something that works, clone it and tweak it into something new that still works. Conversely, it is much harder to come up with something new from scratch. To encourage cultures of participation within MikiWiki, we should carefully plan the initial environment, design rewards or incentives, foster public commitment, establish clear contribution norms, and provide clear mechanism for conflicts solving [48].

Levels of participation

Before designing MikiWiki, we perceived the three levels of participation (meta-design, design and use) as being different 'places' within the HMS and with different users. During the implementation we realized that they are not as much places as modes of work. Users can now decide to create different environments to carry out meta-design or design activities, but this is not necessary. The meta-design, design and use modes can coexist within the same environment and users can seamlessly move between them, achieving higher levels of system tailorability. Therefore, MikiWiki is both a development environment and a collaborative environment. It also provides users with a gentle learning curve and a high ceiling.

Page lookup mechanism

The practical difficulty of having to specify a full path for every wikipage every time we referenced it inspired us to come up with a relative lookup system for wiki pages. Whenever a resource is referenced, MikiWiki looks it up starting from the immediate environment, then proceeding to the containing environments. The implication is that locally defined resources override globally defined resources. This mechanism became the basis for the mediation mechanism and creation of local configurability of resources within a larger interconnected environment.

Mediation mechanism

The way we imagined the mediation mechanism to work has also evolved. We started out imagining how an abstract shared object could be materialized according to sets of rules depending on a user role, culture and device in use. As we developed the system we realized that these rules were cumbersome to express in practice and that it was easier to allow mediation by exposing the rendering mechanism to the users and then allowing different environments to override the rendering mechanisms of the same data page.

This greatly simplifies the system by making the mediation mechanism socially based and removing the need for an explicit representation of device role and culture within MikiWiki.

9. CONCLUSIONS

MikiWiki is a work-in-progress prototype to support and evaluate the HMS model. MikiWiki brings diverse CoPs together to participate in the design process, support their communication and evolve all the system components as well as the communities themselves.

We want to underline the importance of prototyping within the model-building process. Developing and interacting with MikiWiki changed and evolved the initial HMS concepts, grounding them in real-world constraints and at the same time opening up new design opportunities.

MikiWiki is not designed to be an environment for software development, but rather as a shared environment where design teams can communicate or write basic wiki style markup language, using HTML as well as JavaScript to tailor the communication and collaboration tools from within the shared space.

The contribution of this work is to demonstrate the feasibility of implementing the HMS model. MikiWiki combines the functionalities of traditional wikis with EUD activities and meta-design concepts, focused within the HMS conceptual framework.

In this context the boundary between system developers and users is blurred, and there is no need to identify a final delivery or the end users. Considering situated innovation emerging in local contexts, MikiWiki aims to provide a just-enough infrastructure based on under-design principles, which allows users to further build, extend and develop their own environment.

The next step will be to apply MikiWiki to various use scenarios and run usability tests to improve its functionality and to support design teams with an iterative implementation process.

10. ACKNOWLEDGMENTS

The work of Li Zhu and Barbara Rita Barricelli is supported by the Initial Training Network "Marie Curie Actions", funded by the FP 7 - People Programme with reference PITN-GA-2008-215446 entitled "DESIRE: Creative Design for Innovation in Science and Technology."

11. REFERENCES

[1] Kolbitsch, J. and Maurer, H. 2006. The Transformation of the Web: How Emerging Communities Shape the Information we Consume. J. Universal Computer Science 12(2), 187—213.

[2] Bourguin, G., Derycke, A. and Tarby, J.C. 2001. Beyond the Interface: Co-evolution inside Interactive Systems - A Proposal Founded on Activity Theory. In: Blandford, A., Vanderdonckt, J., Gray P. (eds.) Proc. of IHM-HCI 2001, pp. 297--310. Cépaduès-Éditions, Toulouse.

[3] Costabile, M.F., Fogli, D., Mussio, P. and Piccinno, A. 2007. Visual Interactive Systems for End-User Development: A Model-based Design Methodology. IEEE TSMCA 37(6), 1029—1046.

[4] Lieberman, H., Paternò, F., Klann, M. and Wulf, V. 2006. End-User Development: An Emerging Paradigm. In: Lieberman, H., Paternò, F., Wulf,V. (eds.) End User Development, pp. 1--8. Springer, Dordrecht.

[5] Zhu, L., Mussio, P. and Barricelli, B.R. 2010. Hive-mind space model for creative, collaborative design. In: Proc. of DESIRE '10, Lancaster, UK, 121-130.

[6] Zhu, L., Barricelli, B.R. and Iacob, C. 2011. A Meta-design Model for Creative Distributed Collaborative Design. IJDST, 2(4).

[7] Fischer, G. 2000. Social Creativity, Symmetry of Ignorance and Meta-design. Knowledge-Based Systems Journal 13 (7-8), 527--37

[8] Wenger, E. 1998 Communities of Practice. Learning, Meaning, and Identity. Cambridge University Press, Cambridge.

[9] Reenskaug, T.M.H, 2003. The Model-View-Controller (MVC) - Its Past and Present. JavaZONE, Oslo.

[10] Barricelli, B.R., Marcante, A., Mussio, P., Parasiliti Provenza, L., Valtolina, S. and Fresta. G. 2009. BANCO: a Web architecture supporting unwitting end-user development. IxD&A - Interaction Design & Architecture(s): Design for the Future Experience, 5-6, 23--30

[11] Barricelli, B.R., Iacob, C. and Zhu, L. 2010. BANCO Web Architecture to Support Global Collaborative Interaction Design. In: Proc. of IWIPS 2010, pp. 159--162. P&SI.

[12] Andersen, R. and Mørch, A.I. 2009. Mutual Development: A Case Study in Customer- Initiated Software Product Development. In: Pipek, V., Rosson, M.B., de Ruyter, B., Wulf V. (eds.) Proc. of IS-EUD2009, pp. 31--49. Springer-Verlag, Berlin.

[13] Star, S.L. and Griesemer, J.R. 1989. Translations and Boundary Objects: Amateurs and Professionals in Berkley‟s Museum of Vertebrate Zoology, 1907-1939. Social Studies of Science 19(3), 387—420.

[14] Leuf, B. and Cunningham, W. 2001. The Wiki Way: Collaboration and Sharing on the Internet. Addison-Wesley

[15] Schadewitz, N. and Zakaria, N. 2009. Cross-cultural collaboration Wiki: evolving knowledge about international teamwork. In: 2009 International Workshop on Intercultural Collaboration (IWIC '09), Palo Alto, CA, USA, Feb 2009, 301-304.

[16] Krahn, R, Ingalls, D., Hirschfeld, R., Lincke J. and Palacz, K. 2009. Lively Wiki: A development environment for creating and sharing active web content, In: Proc. of WikiSym '09, ACM Press, New York.

[17] Anslow, C. and Riehle, D. 2008. Towards End-User Programming with Wikis. In WEUSE ’08: Proceedings of the 4th international workshop on End-user software engineering, pp. 61–65, New York, NY, USA.

[18] SnipSnap, http://www.snipsnap.org/

[19] XWiki, http://www.xwiki.org/xwiki/bin/view/Main/

[20] Taivalsaari, A., Mikkonen, T., Ingalls, D. and Palacz, K. 2008. Web Browser as an Application Platform: The Lively Kernel Experience. Technical Report SMLI TR-2008-175, Sun Microsystems.

[21] Correia, F., Ferreira, H., Flores, N. and Aguiar, A. 2009. Incremental knowledge acquisition in software development using a weakly-typed Wiki. In Proceedings of the 5th International Symposium on Wikis and Open Collaboration (WikiSym '09). ACM, New York, NY, USA, Article 31.

[22] Riehle, D. 2008. End-User Programming with Application Wikis: A Panel with Ludovic Dubost, Stewart Nickolas, and Peter Thoeny. In: Proc. of the 2008 International Symposium on Wikis (WikiSym ‘08). ACM Press, New York.

[23] Ruby, http://ruby-lang.org/

[24] Sinatra, http://www.sinatrarb.com/

[25] jQuery, http://jquery.com/

[26] node.js, http://nodejs.org/

[27] MacLean, A., Carter, K., Lövstrand, L. and Moran, T. 1990. User-tailorable Systems: Pressing the Issues with Buttons. In: Proc. of CHI 90, pp. 175--182. ACM Press, New York.

[28] Dourish, P. 1995. Developing a Reflective Model of Collaborative Systems. ACM Transactions on Computer-Human Interaction 2(1), 40--63

[29] Greenberg, S. and Marwood, D. 1994. Real Time Groupware as a Distributed System: Concurrency Control and its Effect on the Interface. In: Proc. of CSCW 94, 207—217.

[30] Fischer, G. 2001. Communities of Interest: Learning through the Interaction of Multiple Knowledge Systems. In: Bjornestad, S., Moe, R., Morch, A., Opdahl A. (eds.) Proc. of IRIS 2001, 1—14.

[31] Kirsh, D. 1995. The Intelligent Use of Space. Artif. Intell., 73(1-2), 31—68.

[32] Engelbart, D.C. 1995. Toward augmenting the human intellect and boosting our collective IQ. Commun. CACM 38(8), 30—32.

[33] Snow, C.P. 1993. The Two Cultures. Cambridge University Press, Cambridge.

[34] Pariser, E. 2011. The Filter Bubble: What the Internet Is Hiding from You. New York: Penguin Press HC.

[35] MockingBird, https://gomockingbird.com/

[36] Balsamiq, http://balsamiq.com/

[37] MockFlow, http://app.mockflow.com/

[38] LSL, http://wiki.secondlife.com/wiki/LSL_Portal

[39] Henderson, A. and Kyng, M. 1991. There's no place like home: Continuing Design in Use. In: Design at work: Cooperative Design of Computer Systems, pp. 219--240. Lawrence Erlbaum Ass.

[40] Github, https://github.com/

[41] Could9, http://cloud9ide.com/

[42] jsFiddle, http://jsfiddle.net/

[43] Netvibes, http://tour.netvibes.com/overview.php

[44] Google App Engine, http://www.google.com/enterprise/cloud/appengine/

[45] iGoogle, http://www.google.com/ig

[46] GoogleWave, https://wave.google.com

[47] Google Shared Spaces, http://sharedspaces.googlelabs.com/

[48] Grudin, J. and Erika Shehan Poole. 2010. Wikis at work: success factors and challenges for sustainability of enterprise Wikis. In Proceedings of the 6th International Symposium on Wikis and Open Collaboration (WikiSym '10). ACM, New York, NY, USA, Article 5, 8 pages.