RWTH AACHEN UNIVERSITY

WIDE Workstation Independent Desktop Environment

Diploma Thesis at the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Noriyasu Vontin

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Prof. Dr. Klaus Wehrle

Registration date: Feb 28th, 2008
Submission date: Aug 14th, 2008

I hereby declare that I have created this work completely on my own and used no other sources or tools than the ones listed, and that I have marked any citations accordingly.

Hiermit versichere ich, dass ich die vorliegende Arbeit selbsta"ndig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Aachen, August 14th, 2008

Contents

List of Figures

List of Tables

Abstract

The convergence of the web and the desktop led to web applications that can be considered almost on par with desktop applications. While many systems have emerged in the field of web applications and remote computing, none of them succeeds in creating a single information space for web applications.

WIDE, a Workstation Independent Desktop Environment, is a framework for web applications that connects the user, his data, and his applications. It poses as a single information space that allows sharing of data across applications.

To gain further insight of the user's behaviour and concerns when interacting with online information and applications, we conducted a survey. The results of the survey and literature in the field of online trust highlight that trust in web applications is likely to be lower than in desktop applications. Therefore, to strengthen trust and ensure privacy of the user's data, WIDE uses a request-based access control mechanism. To minimize bothersome interruptions in the work flow it also provides a single sign-on mechanism that allows users to seamlessly move between web applications, similar to changing from one application to another on desktop systems.

Through different client prototypes of potential web applications for WIDE, we reconsidered the design resulting in adding support for Flash and JavaScript-based applications. Adding support for these application effectively enhanced their functionality and improved the range of applications.

Another aspect of refinement considers the integration of existing applications. A review of alternative approaches highlighted benefits and deficits of each a method. A method for external integration of existing web application was developed and is demonstrated successfully by working prototypes.

Further, we critique our design decision's in a design rationale and criticize our explore context, expected problem, and unsolved obstacles that should be addressed in future designs.

Überblick

Das Verschmelzen von Web und Desktop führte zu Webanwendungen, die man als nahezu gleichwertig mit Desktopanwendungen betrachen kann. Während viele Systeme im Bereich der Webanwendungen und des Remote Computings entstanden sind, hat kein System erfolgreich einen einzelnen einheitlichen Informationsraum umsetzen können.

WIDE, ein Arbeitsplatz unabhängige Desktop-Umgebung, ist ein Framework fü Webanwendungen, die den Nutzer, seine Daten und seine Anwendungen verbindet zu einem einzigen Informationsraum, der die Mitbenutzung der Daten von verschiedenen Anwendugen erlaubt. Um weitere Kenntnisse über das Nutzerverhalten und die Bedenken der Nutzern beim interagieren mit Informationen und Anwendungen online zu erhalten, haben wir eine Befragung durchgeführt. Die Ergebnisse der Befragung und die Literatur, die sich mit Vertrauen in Online-Umgebungen befasst, heben hervor, dass das Vertrauen in Webanwendungen dazu tendiert, niedriger zu sein, als das in Desktopanwendungen. Zur Vertrauensstärung und zum Schutz der Benutzerdaten verwendet WIDE deshalb ein Anfragen basierte Zugriffskontrolle. Zum Minimieren von lästigen Unterbrechungen des Arbeitsflusses durch Passwortabfragen hat WIDE einen Single Sign-On Mechanismus implementiert, der es Nutern erlaubt nahtlos zwischen Webanwendungen zu wechseln, ähnlich wie bei einer Desktopumgebung.

Durch das Ausarbeiten von verschiedenen Client-Prototypen von potenziellen Webanwendungen für WIDE haben wir das Design überdacht. Diese Überlegungen führten zur Unterstützung von Flash und JavaScript basierten Anwendungen. Diese Unterstützung erweiterte ihre Funktionalität und erweiterte die Anwendungen um mögliche Einsatzgebiete. Ein andere Verbesserung behandelt die Integration von bereits existierenden Webanwendungen. Nach einer Untersuchung von verschiedenen Durchführungsmöglichkeiten, die ihre jeweiligen Vor- und Nachteile bei der Integration von Anwendungen aufzeigte, wurde eine Methode gewählt und erfolgreich durch funktionierende Prototypen demonstriert.

Desweiteren erörtern wir krtisch unsere Designentscheidungen in einer Design-Begründung und untersuchen den Kontext der Arbeit, die zu erwartenden Probleme und ungelöste Probleme, die im Fokus einer Weiterentwicklungen liegen sollten.

Acknowledgements

First, I would like to thank Prof. Dr. Jan Borchers for making my studies and this research possible. It was him who seeded my interest in human computer interaction. I also especially want to say thank you to my advisor Jonathan Diehl for his open-minded support and ideas he shared with me.

This thesis would not be possible without all the people who supported me during my studies. I appreciated the exchange of ideas with a lot of people of the chair. I especially would like to thank Leonhard Lichtschlag with whom I spent lots of late hours in the lab, Moritz Wittenhagen and Mei Fang Liau for the weekend cooking that kept motivation at a high level even on sundays. Also, I appreciate the time I spent with my "sports" buddies Dieter Drobny and Christopher Gretzki.

Of course, I would like to thank those who helped me to realize the survey by helping me to translate it: Ai, Akiko, Aoi, and Reina, and all the participants of the survey. The whole implementation would have been a lot more difficult without the advise of Ralph who helped whenever I was in trouble.

I especially would like to thank those who accompanied me from the very first day at RWTH Aachen and since then I had a lot of fun with and shared many valuable experience with.

Thank you, Sarah, Kai, Jens, David, and Christian!

A very special thanks goes to Gyan who had to bear with me during my stressful time and of course, my family who made my studies possible and supported me all the time.

Conventions

Throughout this thesis we use the following conventions.

Source code and implementation symbols are written in typewriter-style text.

`myClass`

The plural “we” will be used throughout this thesis instead of the singular “I”, even when referring to work that was primarily or solely done by the author.

The whole thesis is written in British English.

Unidentified third persons are always described in male form. This is only done for purposes of readability.

Chapter 1

Introduction

"The Internet? Is that thing still around?"

—Homer Simpson

The world wide web was initially a pure information-gathering service with limited interactions, in which users consumed web pages one after another. Through typing in a new address or following a hyperlink, the user could request a page from a web server.

This page-based consumption, rather than interaction with online contents, has changed to a more interactive consumption, such as guest books and billboards where users generate additional content by leaving comments and posts. However, interaction was still page-based. Along with web 2.0, a term coined by O'Reilly [2005], web sites have become more responsive and interactive. Thus, they have developed to resemble applications rather than pages. Some of these web applications, such as the online word processing application Zoho Writer[1], can be considered on a par with simple desktop applications.

In addition, they provide benefits that stems from the fact that they are online, such as workstation independence due to web browsers being installed on almost every internet

[1]`http://writer.zoho.com`

enabled computer. Instead of being only able to use an application on a single computer, this property has now allowed users to access their application from any connected computer.

1.1 Mobility

The idea of working online is not novel

Although these new web applications have enabled the user to work online , Weiss [2005] asserts that the idea of working online is not novel but points out that the reasons for the failing of earlier approaches was that not only the hardware performance was weak but the lack of mobility. However, with computers becoming more mobile in recent years, web applications for working online are likely to establish as an alternative to desktop applications.

More mobile computers sold than desktop computers

Notebook computer sales surpassing the number of desktop computers sold in the U.S retail segment for the first time in 2005[2] are indicating such trend to more mobility in computing. Furthermore, many new classes of mobile internet-enabled devices have emerged, such as mobile phones, Personal Digital Assistants (PDA), ultra-mobile personal computers (UMPC)[3], and mobile internet devices (MID)[4].

A new mobile computer class: the Netbook

The *Eee PC*[5], for example, is a notebook that, despite its low hardware specifications, such as its reduced screen size, is very popular. Between January 1st and February 21st 2008 the *4G Surf* model of the *Eee PC* was the most popular laptop among the laptop brands that consumer bought, according to the Consumer Behavior Report[6] from PriceGrabber.com. The *Eee PC* introduced a new class of low-cost mobile devices, the so-called Netbooks, for which Bergevin [2008] despite their low hardware specifications suspects a general high demand around the world. It indicates a new

[2]http://news.zdnet.com/2100-9584_22-146603.html
[3]http://www.microsoft.com/windows/products/winfamily/umpc/default.mspx
[4]http://www.intel.com/products/mid/
[5]http://eeepc.asus.com
[6]https://mr.pricegrabber.com/March_CBR_Portable_Laptop_Trends_v9_FINAL.pdf

direction to less hardware oriented computing which due to hardware limitations relies on services of distant remote servers, such as Wikipedia[7] that can be directly accessed from the *Eee PC*'s desktop environment like a regular program.

Cheap mobile hardware favours ubiquitous computing

Furthermore, small affordable online connected devices approaches to what has been envisioned by Weiser [1991] at PARC. He imagined devices called tabs, "inch-scale machines that approximate active Post-It notes", which together with other devices, pads and boards, will count "hundreds of computers per room".

For such ubiquitousness, hardware must be affordable and to provide a small form factor, power is limited. Since high performance results in high power consumption, such devices are likely to have low hardware performance. Moreover, already today falling hardware costs lead to people have multiple computers, in form of mobile phones, digital music players, and personal computers. As purposes of these overlap, same data is shared across devices, such as contact information stored on the user's mobile phone, his mobile computer, and his personal digital assistant. Changing contact information in one device requires synchronization for consistency of the data. To overcome both problems, people, instead of working on the actual physical machine, can simply log into their virtual computer that is distributed over the internet. Data does not exist in multiple instances but can be accessed online avoiding the problem of synchronization.

1.2 Challenges of web applications

Web applications do not provide a unified user experience

Web applications, such as web mail, are already used by many people on a daily basis as they are convenient to use and can be accessed from any computer with internet connection and yet, web applications still do not give the user a unified user experience. Quite to the contrary, working with a multitude of separated web applications is troublesome to the user as explored as follows.

Working with two web applications on one document is not possible

[7]`http://www.wikipedia.org`

While in a desktop system documents can be easily shared across applications, benefiting the user with the use of a wide range of applications, web applications are isolated and sharing across applications is in general not possible. This is because documents on the desktop computer are stored on a common information space, the shared hard disk. Such a common information space, however, does not exists for web applications. Currently, documents are transferred from one web application to another by downloading and uploading. This is cumbersome and intervenes with the user's work flow. Moreover, web applications are protected by log-in mechanisms, generally asking for a user name and a password when using the application, which furthers interruptions to the user.

The web is untrustworthy

The web also harbours risk, such as viruses infecting the computer, phishing sites[8] and identity theft. For these reasons, the web is considered rather untrustworthy which has has prevented people from further embracing web applications as an alternative to desktop applications.

In this regard, we will present the design process of a framework for web applications, we call WIDE, Workstation Independent Desktop Environment, that addresses these concerns.

WIDE is a framework for web applications

WIDE seeks to provide a single shared information space with single sign-on mechanism to allow users to work with various web applications in a more desktop-like manner. Our goal is to demonstrate a working system with prototype applications that aims to offer a workstation independent solution to the problems mentioned above:

- Single information space helps the user to organize his documents by placing them in one location rather than being distributed. Furthermore, it supports document sharing across applications which allows efficient and natural working with multiple applications.
- Single sign-on minimizes unwelcomed breaks in the work flow and frees the user from password concerns for the increasing number of online accounts

[8]sites that mimic other sites to obtain customers passwords

- Direct access supports web applications best as many of them are, too, directly accessible. Moreover, it helps to engage users to work with it.

1.3 Chapter overview

The thesis is organized as follows:

- In *Related Work*, we describe works in the field of web applications and remote computing and identify lacking support for the features of WIDE.
- The *Trust* chapter introduces the theory regarding general and online trust as well as a survey we conducted to get further insight in user's online behaviour.
- In *Architecture* we present the WIDE server and how it interacts with the clients to realize authentication and authorization.
- *Implementation* describes how we implemented the proposed architecture and how through implementing prototypes we developed support for JavaScript and ActionScript-base applications.
- *Integration* explores ways to add support for WIDE to existing web application and presents implemented prototypes successfully demonstrating the integration.
- In *Validation*, we discuss the our choice of an appropriate evaluation methodology resulting in presenting the project's context, the reasoning behind the design decisions made, and expected problems.
- In *Summary and Future Work*, we summarize our conclusions and explore how future work can further improve our work.

Chapter 2

Related work

In this chapter, we first try to find a categorization for the field we investigate to provide a further understanding of the benefits and deficits in context of the respective approach under consideration. For a better comparison with WIDE, we state features that we deem optimal for an environment to support best working online:

- **Single information space** allow users to share data across applications
- **Single sign-on** mechanism avoids work flow interruptions
- **Access control** mechanism protects the user's data from unwanted access
- **Open system** is extensible and flexible regarding to the needs of the user
- **Support any type of data** to not restrict user's in their usage of such systems
- **Workstation independence** to not exclude existing workstation independent online applications
- **Instant access** requires no installation and encourage flexible usage

After the overview we give a comparison to show that the presented systems do not fully support these features.

2.1 Classifications

In section tries to explain the different approaches and how to they can be classified in the field of web and desktop applications, and remote computing.

2.1.1 Web and Desktop Applications

It is difficult to grasp an exact definitions of web and desktop applications. Furthermore, emerging convergence of the web and the desktop has led to various kinds of applications between these. As Moritz [2008] points out

> "The terminology of Desktop and Web applications and everything between is difficult and confusing. Finding unique and unambiguous names seems to be difficult."

Categorization of web applications is difficult

The table 2.1 shows how the borders between the technologies of these terms are blurred and make a categorization difficult.

User Perception-based Definition

Borders between categories are blurry

A clear technical differentiation seems difficult, as terms like *rich internet application* seems in some cases to apply for desktop applications that need to be installed and in some other cases to apply for specific web sites. Therefore, we propose a user perception-base definition of what a web application is.

An application that runs in the web browser and is perceived by the user to relate to the web, such as for example, having a server back-end, being originated from a web

Term	Properties	Examples
Desktop application	Need installation on the computer	Microsoft Word, Adobe Photoshop
Internet-enabled desktop application	Uses network support, can also run offline, perhaps with limited functionality	Outlook, Thunderbird
Rich internet application (RIA)	Located in between, combination of desktop application and web application	
Web browser-based RIA	Web sites with more richness of user experience, more response and personalization facilities	social network sites, calendars, online email services
Desktop-related RIA	Look and feel of desktop application but strong focus on web	Apple iTunes
Smart client	Very related to RIA, technology between thin and rich client	
Web 2.0	Applications providing classical desktop and web features	
Web site, web application, thin client	Rarely need to be installed, started and loaded via network, personalized log-in, running connection, often run in a web browser	

Table 2.1: Overview of some Terminologies Describing Desktop and Web Related Applications, According to Moritz [2008]

page, or resembling another web site, we consider a web application.

Although this definition still leaves some space for interpretation, it will more likely reflect the user's view on web applications.

2.1.2 Non Web-based Remote Applications

There are also other ways to realise working remotely with applications as follows:

- Distributed Window System

- Remote Computing
- Application Virtualization

Distributed window system: separation of interface and application

Distributed window systems allow the separation of interface and applications, some particular kind of remote access to application. In section 2.4.1 we explore it exemplified by the X window system.

Remote computing: maintenance of distant computers

Remote computing refers to the concept of remotely controlling a computer through software. This is, for example, used to maintain distant computers. Section 2.4.2 will give an overview of such a system by examining Virtual Network Computing.

Virtualization encapsulates applications and hides the actual enviromment

Application virtualization refers to encapsulating the application in an virtual environment and separate it from the operating system, according to Amir [2008]. The application however, is acting as if it is running directly on the operating system. The encapsulating layer is the so-called virtualization layer which intercepts all operating system related operations and redirects them into a virtualised location, such as a single file. An example of such virtualization is given in section 2.4.3. In contrast to application virtualization, desktop virtualization encapsulates the entire desktop, an example can be found in chapter 2.2.3

2.2 Information Spaces

According to Newby [2002] information space is defined as the set of concepts and relations among these held by an information system. Guarino [1998] states resources, user interfaces, and application programs as the main components of such an information system.

On workstations for example, the desktop environment is such an information space that usually consists of concepts - files, file systems, applications, and the user - and their relationship to each other - e.g. folder hierarchies, access rights, and installation of applications.

On the web various kinds of online accessible information spaces exist. Therefore we define the following types for a better categorization:

- *Online Storage Services* focuses on storage and access of information in form of files in a file system and as databases.
- *Web Applications* form their own information spaces containing their data, such as documents for office applications or personal information for a social network site. Application spaces differ in their presentation as they depend on the domain and type of the application. The focus of application spaces is how to best support the service provided by the application.
- *Web Desktops* are a specific type of web applications similar to the traditional desktop systems on local workstations.

Working in an environment, such as the world wide web, in which services are widely distributed poses the question of to whom or what the information belongs. The statement that a piece of information belongs to an application because the user created it with the help of the application is untenable as this does also not hold true for the real world. Furthermore, applications are tools that, without the user, do not access information. Information should be, therefore, owned by the user and belong to a space that is controlled by the user and not by the applications.

Information created by a user should belong to that user

2.2.1 Online Storage Services

Online storage spaces are information spaces that allow a user to store and organize information so that it is accessible online. In the beginning, these types of services were rather simple, giving the user online accessible web storage with a file system where files were either public, that is accessible to all, or not visible, being only accessible when logged in. Recent storage systems offer additional value

Storage services add value to their service like different interfaces for applications the files can be accessed through. The approach of storage spaces is document centric as storing and accessing to data remains the main focus. However, to increase its usefulness some web storage have provided additional services, such as adding meta-data, being information about information, through the tagging of files or built-in applications that can directly use the stored data.

Amazon S3

Amazon Simple Storage Service[1] (S3) is an online storage service offered by Amazon. The customers pay for the amount of stored data and for data transfer.

Data and their meta-data together form so-called *objects*. Objects are stored in buckets and each has one unique key with which to address it within the bucket. Thus, the bucket and the key of an object identify the object which can be accessed by other applications and users. The access control policy is defined by each object's access control list (ACL). When an object is accessed it is checked to determined whether the object is allowed to be read or written depending on the rights stated in the ACL. With sufficient rights even the ACL can be accessed and modified.

Access control lists define access control policy

With the ACL a dynamically adjustable fine-grained access control is offered. However, for applications that have only rights to read certain files due to limited trust, it is not possible to easily set an exception granting one time object access. Furthermore, although other applications can access S3 and S3 has knowledge about the users through the ACL, on the other hand S3 cannot call the applications. Thus, unless the user only wants to organize his data into buckets, S3 provides an infrastructural service to other applications rather than a framework for working online.

In contrast WIDE offers an entry point for both data and applications. With WIDE, documents can also be opened directly from the information space. Access to files for un-

[1]http://aws.amazon.com/s3

trusted application can be easily granted through the request based access control without concern for unwanted access (see 4.3).

AOL Xdrive

AOL provides an online storage service called Xdrive[2]. Beside a web front end for uploading data, unlike S3, Xdrive does not offer an interface for other applications to access its storage. However, AOL provides stand alone software such as Xdrive Desktop Lite and built-in functionalities to add value to the service.

Xdrive Desktop Lite is a Flash based desktop application that runs in Adobe AIR (see 2.3.1) that allows drag and drop interaction to upload data to and download from the storage. Furthermore, its web interface includes basic functionalities, such as displaying pictures and playing music with its integrated music player. This added value actually shifts Xdrive from an purely storage focussed approach to a more general information space that provides, to a certain degree, integration into the desktop and offers further interaction over and above simple organization.

Added value through integrated applications

However, in contrast to the prior example Xdrive is a closed system that cannot be extended.

Instead of adding built-in functionalities, WIDE proposes an information space that delegates document interaction beyond organizing them to external applications that in contrast focuses on processing the information rather than managing them.

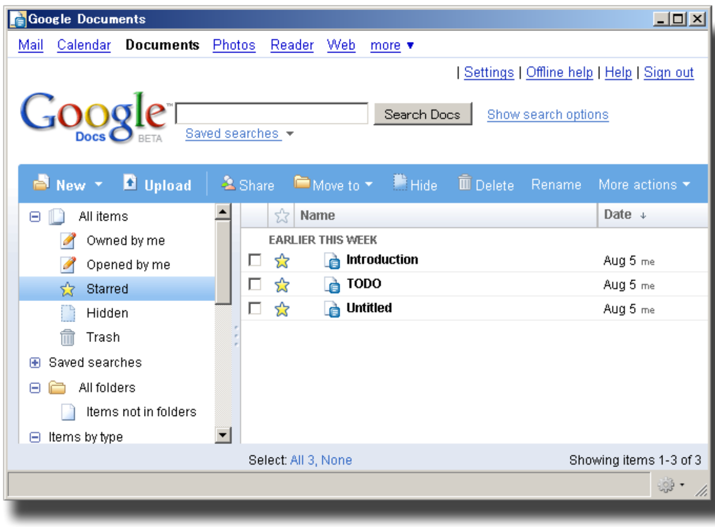
2.2.2 Web Applications

An example of how applications are also an information space similar to the aforementioned storage services is web mail. Web mail applications allow users to access their

[2]http://www.xdrive.com

mails and therefore, hosts their mails to present them over the web for reading purposes. Mails can usually be filed in folders, searched or can be filtered automatically as junk mail.

Google Docs



Name	Date ↓
EARLIER THIS WEEK	
Introduction	Aug 5 me
TODO	Aug 5 me
Untitled	Aug 5 me

Figure 2.1: Google Docs, a Unified Information Space for its applications

Google Docs is a web-based office suite providing word processing, spreadsheet, and presentation application services. Documents are saved on the shared web storage system which can then be tagged, placed in folders, and shared with other users.

Interfaces provide access to documents

Although Google Docs was initially developed as an application providing attached storage space, stored documents can now be directly accessed through several interfaces Google offers. Through the interfaces, applications can retrieve documents, upload documents to Google Docs. Spreadsheets can also be used as a simple database for applications. Furthermore, it can be used offline with Gears(see 2.3.1) that also provides additional desktop interactions.

The provided interfaces focus on information retrieval and

Webtop	Open source	Engine	Own server	Free
EyeOS	yes	PHP+AJAX	yes	yes
G.ho.st	no	Flash	no	yes
Stoneware webOS	no	AJAX	no	no

Table 2.2: Comparison of webtop systems

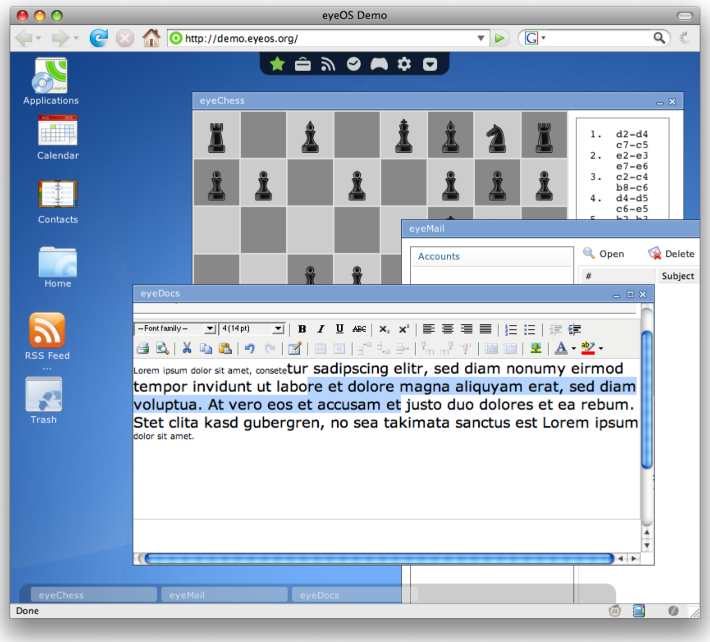
meet the needs of application that solely consumes the contents of stored information. It is however, not powerful enough to satisfy those applications that require bidirectional transactions which allows the application to store arbitrary data on their storage space.

WIDE, in comparison, does provide transactions for both receiving and sending of arbitrary data and thus, supports more versatile applications that require more complex information exchange mechanisms.

2.2.3 Web Desktops

Currently available Webtop systems are usually online desktop environments that mimic traditional desktop system's user interfaces, such as MacOS or Windows, on remote servers. Also file systems and application management systems resemble those of commonly known desktop systems. Webtops are offered as a internet service for which the user sets up an user account. This service itself is a web application and as such, it runs workstation and operating system independently in the web browser's environment. The different realisations of webtop systems exemplified by EyeOS, G.ho.st and Stoneware webOS are presented in table 2.2. In the following the differences of these will be further explored.

Mimic traditional desktop systems



```
1.  d2-d4
    c7-c5
2.  e2-e3
    e7-e6
3.  c2-c4
    b8-c6
4.  d4-d5
    c6-e5
```

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Figure 2.2: EyeOS, a Web Desktop System.

EyeOS

EyeOS[3], one of many Webtop systems, will present itself to the user just like an ordinary desktop system. After the user logs in, a desktop environment is presented, with icons, applications running in windows etc.. For visualization of the user interface HTML and JavaScript are used, while some other Webtop Systems use Flash instead. In that desktop environment, the user can start programs that are stored on the server on which the system is running. It is possible to extend the range of application by developing new applications with the provided eyeOS API. External applications (written by a third party developer) can be submitted to a repository. Alternatively applications can be downloaded in so-called eyePackages that can be uploaded to the eyeOS. Similar to the traditional desktop applications need to be installed. In EyeOs this is only possible for the root user. That means that regular user cannot extend the set of application. However, EyeOS can be downloaded and run

EyeOs can be extended by third party applications

[3]http://www.eyeos.com

on a separate server and user can create their own EyeOS server which then can be modified.

G.ho.st

In contrast to EyeOS, G.ho.st is a Flash-based webtop system and partly does rely on hosted web applications, that are web applications that have been integrated into the system. For example, Zoho Writer, an online word processing application similar to GoogleDocs (see 2.2.2), is integrated into G.ho.st as a hosted web application. Instead of accessing the service via web browser, the user in G.ho.st starts it like any other application directly from the desktop. The window in which Zoho Writer is loaded, provides additional options, as it is shown in figure 2.3. As a G.ho.st user the edited document can be directly saved to the desktop. On the other hand, options, such as for sharing and publishing, that usually are included are omitted. In ad-

Flash-based webtop with hosted web applications

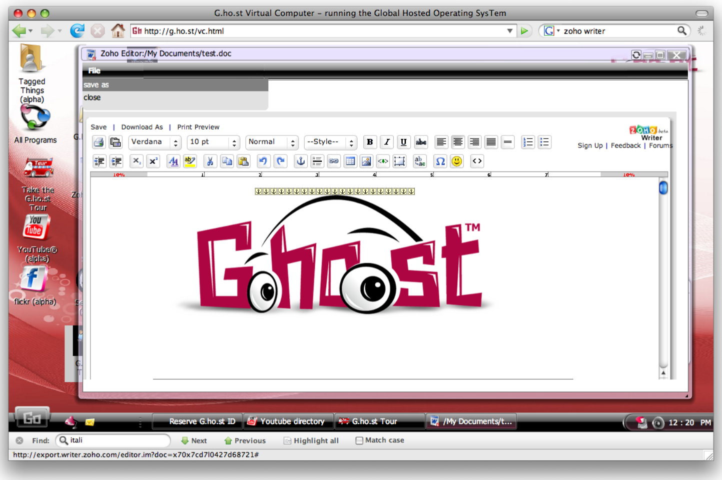


Figure 2.3: G.ho.st, a web desktop system providing hosted web applications.

dition, it also provides some specific applications to access other web services, such as YouTube or Flickr. These applications are consuming the data of the services and do not provide full functionality of the services the data is consumed from. Such an application is depicted in figure

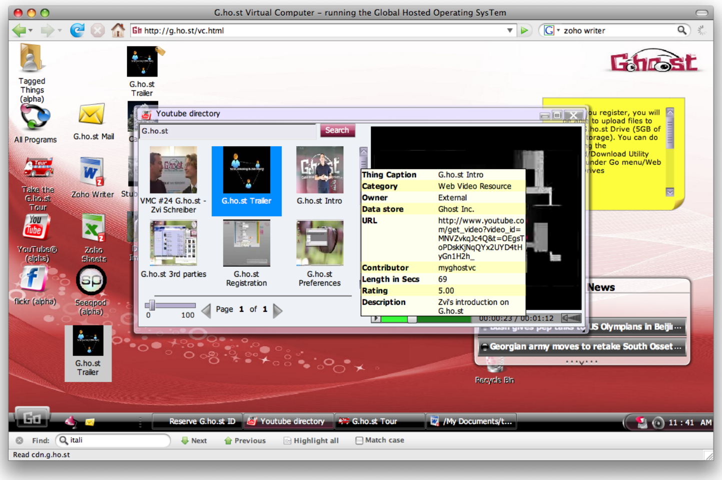


Figure 2.4: G.ho.st: Specific applications to access online content, such as YouTube

2.4. Another difference to EyeOS is that G.ho.st is still not open-source but, according to the website[4], the developers are currently working on opening their webtop system and developing their API. Open-source increases transparency and thus, lessens uncertainty resulting in more trust in the application (see 3.1).

Stoneware webOS

Stoneware's webOS is different to the prior examples since its focus lies on desktop virtualization. It provides desktop virtualization over the web and as such is independent of the underlying operating system and hardware. It provides virtualized desktop applications, such as windows programs through utilization of Microsoft terminal server, and also access to web applications as hosted applications. It is designed as a single access point to all corporate computing resources. With its access to corporate web applications, they can be located securely in the corporate network, reducing the vulnerability of the corporate computing resources.

Focus on corporate computing

[4]`http://g.ho.st`

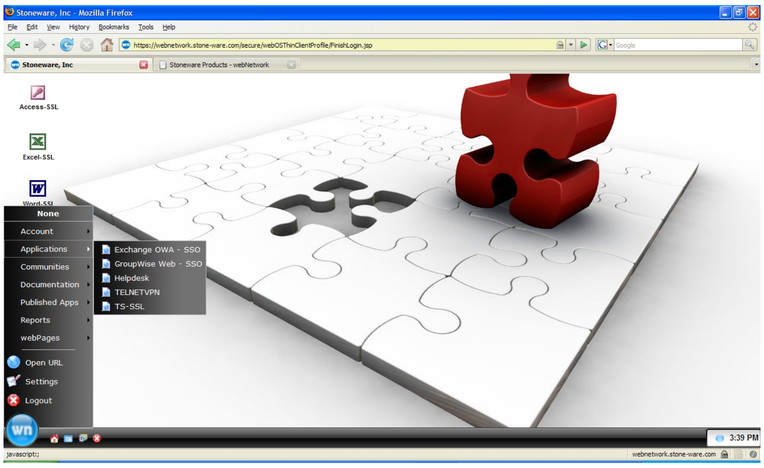


Figure 2.5: Stoneware's webOS, a web desktop virtualization system.

Conclusion

In summary, webtop systems succeed in moving the desktop environment to the web. This is adding value to the desktop computing experience as it can be accessed independently from the workstation. Extending of open webtop systems is difficult as APIs include new widget toolkits the potential developers have to familiarize with and existing source code, e.g. based on standard HTML widgets, cannot be re-used easily. Contrary, WIDE enhances web applications with new functionality and focusses on transaction instead of dictate a user interface toolkit.

Existing web applications cannot easily adjusted for webtop systems

2.3 Hybrid applications

Despite the convergence of web and desktop applications (see also 2.1.1), there are still gaps between these two worlds, such as sharing of documents between applications and trust in online applications (see 3.1).

2.3.1 Desktop Integration

In this section three approaches are shown to integrate web applications into the desktop environment that offers more sophisticated interactions, such as drag and drop, and to provide support for working offline and direct access to applications through the desktop. This is similar to the goals of WIDE to support working with multiple applications in one single information space.

Adobe AIR

Adobe Integrated Runtime (AIR)[5] is a runtime environment for internet applications similar to the web browser. It provides a virtual machine for JavaScript and ActionScript. In AIR the applications run similarly to traditional desktop applications as AIR, unlike a web browser, does not have any visible controls and thus, is invisible to the user.

A cross-platform runtime for web applications

The cross-platform environment of AIR adds an abstraction layer to the operating system. It hides operating system specific behaviour from the applications and provides the same runtime on different operating systems, such as Mac OS X or Windows Vista. Thus, developers only need to write code once without modifying it for each operating system. To run in AIR, applications need to be installed on the workstation.

AIR is also adding desktop functionality to the applications. While web browsers are restrictive due to security reasons, AIR provides more interaction with the desktop as they run locally. This also provides applications with benefits, such as access to the local file system and in the case of Flash based applications, freeing the application from very limited assigned local storage. This allows the application to integrate further into the desktop system because of a more direct interaction, in which functions such as dragging a file and dropping it on a running application, are now possible. Furthermore, with the support of popular

[5]http://www.adobe.com/products/air/

web programming languages it is possible to re-use source code when an online version already exists.

Installing AIR applications locally take away some advantages of web applications, such as applications not being instantly available to all computers but must be installed locally on each computer. In other words making a previously directly accessible workstation independent application inaccessible on other workstations if it is not installed.

Prism

Prism[6] is an add-on for the Mozilla Firefox web browser that allows the user to “save” a web site as an desktop application. As such the user can start the application from the desktop directly in the same manner as other desktop applications.

Disguises web applications as desktop applications

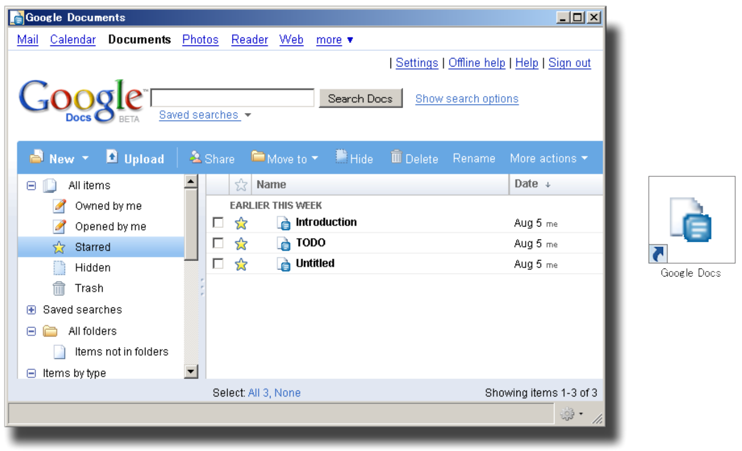


Figure 2.6: Prism hides the browser window’s navigation to mimic an usual application window (left) and can be started over a desktop link (right)

The transformation to a “desktop application” however is achieved by removing the browser user interface, such as the page navigations or the location bar, from the window and wrapping it into an executable link. Although with Prism, web sites appear as desktop application, unlike AIR,

[6]http://developer.mozilla.org/en/docs/Prism

desktop interactions other than starting the applications directly from the desktop are not possible. Furthermore, the application is still a web application and is actually executed from the web despite the direct start through the link.

In contrast to Prism WIDE does not focus on merging offline and online applications but rather on shifting the desktop experience online. WIDE enhances the online experience by integrating desktop-like interaction processes.

Gears

Adds offline support and desktop interaction

Gears[7] is an open source project by Google that enhances web applications by adding features to the web browser. With Gears the data layer of an application can be separated to add offline support as illustrated in figure 2.7 and also to provide additional desktop interactions. It is realized as an web browser add-on that provides a *Database module* to store data offline, a *LocalServer* module for caching and serving application resources, and a *Desktop module* that provides the bridge between the browser and the desktop for desktop interactions.

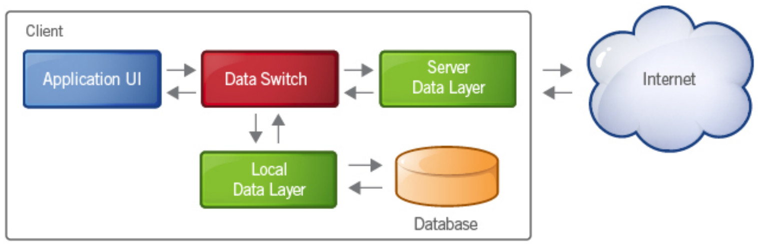


Figure 2.7: Gears separates the data layer from the user interface. A Data switch communicates with the offline local data layer and the online server data layer.

Gears stores data in the Database module and communicates offline to the LocalServer module. The offline stored data is synchronized with the online storage when online connectivity is provided. If there is no internet access at the time, synchronization is deferred during the period offline. Gears is not only used by web applications to add

[7]http://gears.google.com

offline synchronization but also to speed up web application by reducing network load on the servers because the potentially slow, high latency internet connection is only accessed for synchronization instead of each time the user interacts with the web application. By doing so, Wordpress, a weblog service, could reduce the number of request from about 50 to two to three.[8] Unlike AIR applications, Gears supported applications are web applications and it is possible to use the applications without Gears. However, to gain the benefits offered, Gears must be installed to the local workstation.

Gears enhances the interaction on workstations. WIDE focuses solely on the online interaction rather than taking into consideration also offline access and desktop integration. This makes the two concepts complementary rather than contrary.

2.4 Desktop Applications over the Network

The opposite approach is to take offline desktop applications over the network and operated them from remote computers.

2.4.1 X Window System

The X window system, proposed by Scheifler and Gettys [1986], is a windowing system that was specifically designed to be used over network connections. As such, it does not imply the application, the X client, and the user's local computer to be at the same location. The X client sends requests for graphical output over the network to the X server. On the other hand the X server sends sends user input events back to the X client.

However, X server software is not directly available on all platforms although X terminal, a client software that

[8]http://trac.wordpress.org/ticket/6965

Not natively supported by all platforms

Graphical output requires high bandwidth

runs the X server, can be downloaded. In addition, the graphical output that is sent over the network requires high bandwidth that for internet connections cannot be ensured. Therefore, WIDE is based on web technology that sends rather high level output, such as buttons and tables, to the web browser rather than pixels. Furthermore, a web browser is a software that many people use on a regular day-to-day basis and thus, much more familiar than an X terminal.

2.4.2 Virtual Network Computing

Virtual Network Computing (VNC) is a platform-independent client/server-based protocol developed by Olivetti & Oracle Research Laboratory (ORL) for remotely controlling a computer(see Richardson et al. [1998]). The implemented display of the protocol allows the user to connect a VNC client managed computer, the *viewer*, to a computer managed by the VNC server, the so-called *desktop*. Once a connection is established, the *desktop* sends its screen content over the connection to the *viewer* and in return the *viewer* sends its keyboard and mouse input to the *desktop*. Hence, to the user it appears as if he is interacting with the remote *desktop*. The server on the *desktop* side also provides a web server offering a Java-applet, a small Java program that can run in a web browser, posing as a VNC client. This allows the user to access the VNC server with any Java capable web browser and therefore, can be, to a certain degree, considered workstation independent.

Server offers Java applet for workstation independent access

Similar to X, VNC transports the screen contents over the network connection requiring more bandwidth compared to those of web applications. When the user logs into a remote computer, VNC then will display it to the user as if sitting in front of it. In contrast, WIDE can be used by many people at a time and coordinate their work and interaction with external web applications. As WIDE is based on web technology, file transfer, being download and upload, are supported. VNC, however, does not implement file transfer, i.e. files of the remote computer cannot be directly downloaded to the local computer.

2.4.3 Citrix XenApp

XenApp[9] provides two types of application virtualization technologies: client-side application virtualization and server-side application virtualization. The two types refer to the location where the application is actually executed. Therefore, with server-side application virtualization application can be used on hardware that does not satisfy the application hardware requirements because the application runs entirely on the server that indeed must meet the requirements. The client handling the virtualization layer, exist for several platforms, such as Microsoft Windows, Mac OS, and Linux.

Although the concept of virtualization relates to the concept of thin clients in case of the web applications, virtualization requires the client, similar to a runtime environment, that must be installed on the user's local computer preventing direct access. Moreover, Citrix focus lies on corporate networks, relatively closed network environments to which only particular users have access to in contrast to web applications that can be basically accessed by everyone through a web browser.

2.5 Development of Web Applications

The focus of the here presented works is to ease the development of web applications by providing methods that allow development that is similar to the of desktop applications. This can make source code of the existing offline application re-usable when developing a corresponding web application.

[9] `http://www.citrix.com/English/ps2/products/subfeature.asp?contentID=163987`

GWT

Google Web Toolkit[10] is an open source Java development framework for web applications that focuses on browser independent development. With GWT, web applications are developed and debugged in Java. For deployment, the compiler translates the Java application into a browser-compliant JavaScript and HTML application that runs in a web browser.

Web application development in Java on client and server side

GWT eases development as Java's object orientation helps to improve comprehension. Furthermore, many development tools exists for Java that helps to accelerate development. Through the compiler generated code, the resulting application is performance optimized, while debugging is more efficient as web browser incompatibility has been attended to.

Despite using Java for development, widget sets of Java, such as the Abstract Window Toolkit or Swing, are not included. Instead the user interface is created with components of the GWT Web UI class library which holds widgets like buttons, text fields etc.. Thus, as an example, user interface code of Java applications have to be rewritten although other parts can be re-used.

XML11

XML11 proposed by Puder [2006] is an XML based abstract windowing toolkit inspired by the X11 protocol of the X window system (see 2.4.1). It migrates native Java applications to client applications that run in a web browser.

Migration of Java application through bytecode translation

In order to run a Java application over the network in a browser it is translated into a JavaScript and HTML application. The translating process begins on byte code level where Java commands are dissembled to atomic instructions. First, Java classes are translated into an XML based programming language, so-called XLMVM. This XMLVM program is then translated into JavaScript by mapping via

[10]`http://code.google.com/webtoolkit/`

style sheets. The resulting JavaScript program can generally run in a web browser. However, some classes, such as a database, require fixed resources that cannot migrate. Therefore, a configuration file determines which of the XMLVM classes can be migrated to the client and which need to remain on the server side. To communicated between such classes, a proxy marshals parameters and sends them to the server. The proxy appears to the application as a remote object.

Although XML11 uses remote communications like X11, differences are considered. While X11 is used with high bandwidth networks rather than a medium bandwidth network with a high latency like the internet. Thus, to adapt to the different network environment, XML11 transfers widgets, instead of pixels, to the web browser and events back to the server back end.

In comparison with GWT, XML11 is also different as it focuses on the migration of desktop application and GWT on web application development. The translation of Java into JavaScript differs, also. Unlike GWT, XML11 does not require adjustment to the original Java code. One reason for it is that XML11 uses the native widget set AWT. Another difference happens in the translation process. GWT translates the Java source code while XML11's cross-compiler translates the byte code. Hence, features of newer versions of Java can be supported as byte code instructions stay the same.

However, similar to web desktop systems XML11 applications are remotely controlled through the web browser. The resulting application does not interact with the web or follow its principles. The desktop or back end in this case remains isolated on one server. With WIDE, the aim is to keep the distribution of web applications and to provide a single information space interacting with the distributed network of various services.

2.6 Comparison Table

System	Single information space	Single sign-on	Access control mechanism	Extensible, external interfaces	Type of data	Workstation independent	Instant access
Amazon S3	yes	no	ACL	yes	any	yes	yes
XDrive	yes	no	n/a	no	any	yes	yes
Google Docs	yes	yes	sharing	limited	documents	yes	yes
AIR	local desktop	n/a	no	yes	desktop	?	no
Prism	n/a	no	n/a	any website	n/a	yes	n/a
Webtop	yes	n/a	no	no	any	yes	yes
Gears	n/a	?	n/a	yes	n/a	yes	no
X	yes	n/a	no	yes	n/a	yes	no
VNC	remote OS	n/a	no	yes	any	yes	yes
XenApp	local desktop	?	no	yes	any	?	no
XML11	n/a	n/a	n/a	n/a	n/a	n/a	yes

Table 2.3: Comparison of the various systems

Chapter 3

Initial study

"The city's central computer told you? R2D2,
you know better than to trust a strange computer!"

—C3PO

3.1 Theory

When interacting with the web, and more precisely with service running on the web, the user enters a trust relationship. This trust relationship is important for smooth and successful interaction. Thus, it is of particular interest to know what trust exactly is, what effect does trust have on the working process, and how trustworthiness is communicated to the trustor.

First, some definitions are explored for a better understanding of trust. *Trust* is property of a relationship of two or more parties. The trusting party is called the *trustor* and the trusted party is called *trustee*.

The concept of trust appears in many different fields, such as sociology, psychology, and computer science among others. In Sociology the degree of trust is defined as a measure of belief of how honest, benevolent, and competent the trustee is. In the field of computer science, different kind of trust relationships, such as computer mediated trust and online trust, are of interest. Computer mediated trust describes the trust in others in situations where interaction is mediated by computers as it is the case when chatting, mailing, or online gaming with others. Online trust relates to human-computer trust in an online environment, such as website interaction.

3.1.1 General Trust Model

A detailed general model of trust is proposed by Riegelsberger et al. [2005]. In their model they argue that the trustee must meet contextual conditions to be trusted. These conditions are described as temporal embeddedness, social embeddedness, and institutional embeddedness. Temporal embeddedness relates to the trustee's dependence on the trust of the trustor at a later point of time. When interacting multiple times, the trustor will react to the prior behaviour of the trustee and if the trustee has previously acted untrustworthily, the trustor might not trust the trustee again. Social embeddedness refers to the trustor's social environment, particularly other potential trustors of the trustee. Untrustworthy behaviour results in decrease of reputation and thus, other parties might not place trust in the trustee in future interactions. The institutional embeddedness relates to the trustee being in an institution. They describe trust as institutions motivating trustworthy behaviour and sanctioning non-fulfilment. Therefore, however, the trustee's actions must be traceable and investigation and actions must be relatively easy in comparison to the non-fulfilment. These three aspects motivate the trustee to act in a trustworthy manner and communicate - as signals - the trustworthiness of the trustee to the trustor. In addition to these external incentives the trustee's motivation and abilities will affect his actions.

Contextual conditions

Separation in time and space were further discussed with respect to their effect on the trust relationship. Spatial separation leads to loss of information regarding the trustee and thus, increases uncertainty. Separation in time, such as delayed response, increases the effect as it prolongs this period of uncertainty. Uncertainty increases the feeling of risk

Separation increases uncertainty

and thus, increases the need of trust.

Although Riegelsberger et al.'s framework illustrated trust in general and not specifically in relation to online trust, many aspects holds true for online trust.

3.1.2 Online Trust

However, as WIDE is an online system, this thesis' primary focus lay in online trust. Corritore et al. [2003] define online trust as an attitude of confident expectation in an online situation of risk that one's vulnerabilities will not be exploited. Such vulnerabilities can be, for example, loss of privacy.

In their work, they present a model of trust for informational and transactional websites, excluding computer mediated trust scenarios. Their model consists of external factors, physical and psychological, and perceptional factors, being credibility, ease of use, and risk.

Credibility is an important characteristic of trust, both online and offline, and is closely related to the predictability of an user interface. Ease of use deals with how easily an user can achieve his goal when using a computer. The consistency of the user interface can also improve the ease of use. The third perceived factor of risk, is known as a key factor in offline trust literature and it has also been discussed extensively in the area of online trust.

As ease of use increases the feeling of control and credibility is known to lower the perception of risk, both impacts risk, as depicted in figure 3.1. Furthermore, Corritore et al. argue that increased ease of use reduces the cognitive workload and thus, the user will have more cognitive resources to notice credibility cues. In the model the external factors affect all perceptional factors. The perception of risk and the credibility of a website is also known to have direct effects on trust.

Similarly, Hampton-Sosa and Koufaris [2005] define trust as the willingness of the trusting party to rely on the trustee.

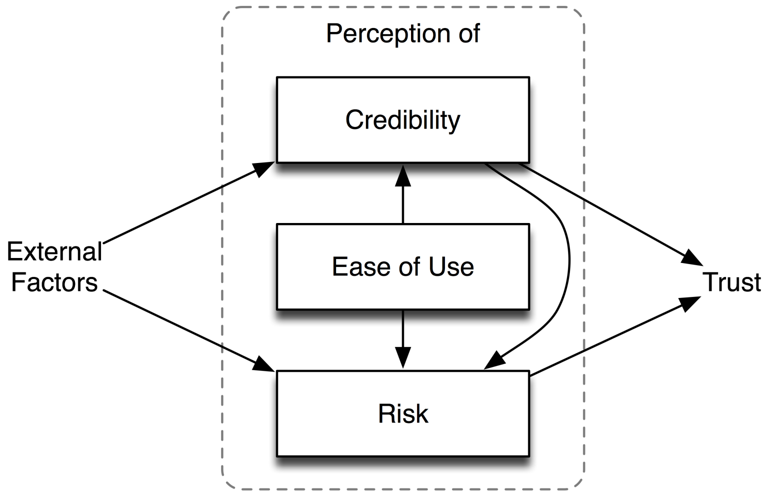


Figure 3.1: Trust model by Corritore et al. [2003]

In their work they focused on the factors that leads to initial trust which then encourages the user to use a website for the first time. The results showed that a web site's appeal, namely the perceived usefulness and enjoyment, had a significantly positive effect on the initial trust. Despite the test's results being unable to support their hypothesis that a web site's usability, control and ease of use, is positively related to initial trust in the web site's company, they pointed out that this could be due to the short time the users interacted with the website and that longer interaction could indeed increase the impact of usability on initial trust.

Another perspective on trust and credibility was presented by Cyr et al. [2005] who led studies in design preferences of different cultural groups and their effect on trust or e-loyalty. Culture is assumed to be important as cultural background in the effect on one's perception and it has been shown that trust related to perception, such as perceived feeling of risk. A study with groups of Americans, Canadians, Germans, and Japanese on their preferences of local and foreign web sites gave mixed results with the Japanese group unexpectedly displaying higher levels of satisfaction and loyalty to the foreign website than the local. Consequently, they have proposed to further research whether the internet itself will become or has become an internationalized common culture. A model incorporating

Internet: internationalized common culture?

these aspects among others may result in a better picture of preferences across cultures.

3.1.3 Implication

From the aforementioned studies of trust certain implications for a platform for web application, such as WIDE, can be derived. In the following we describe such two implications: control and transparency.

Control

Although control is not explicitly a separate factor of these models, the perception of being in control is affected by the ease of use, or usability. The relationship of control to risk is of inverse correlation. Corritore et al. [2003] stated that

> "The body of work on risk has also shown that control reduces risk and that risk is higher in the absence of control."

Consequently, total control eliminates any uncertainties leading to a situation without risk in which trust is not necessary.

Transparency

According to Riegelsberger et al. [2005] institutions motivate trustworthy behaviour. In an online environment like that the web such an institution can be a network of various web site. They can act as an institution for each of its members. In such a relationship, transparency is likely to increase trustworthy behaviour as it increases the traceability of actions and thus, lowers the cost of investigation, supporting sanctions. Transparency to the user reduces the degree of uncertainty presented which itself is closely related to risk.

3.2 Survey

To gain further insight in online behaviour, trust towards online web sites, and the general understanding of online web applications we conducted an online survey.

The survey asked about general online experience, such as how often people work online or what kind of services they use.

3.2.1 Participants

For a more representative result that points out potential regional differences the survey was was conducted in English and in Japanese. In total 183, 58 female participants and 125 male participants, participated in the survey. The average age was 26.63. More than half, 103, were students. About 56% said that their degree was rather techical. Furthermore, 55% of the participants estimated their general computer skills to be very good or excellent.

3.2.2 Online Behaviour

While 17.7% of the participants do online shopping at least once a month, 7.2% stated that they never do online shopping. Of those who do online shopping, 76% stating that they are mostly or completely satisfied. The fact that more people do buy things online than people who do not indicates that despite online shopping harbouring not only privacy related risks but also financial risks for the user, is accepted and regularly used.

In contrast to 25% storing their password for online accounts in the web browser, 30% store some passwords depending on various aspects. Most often answers were related to the importance or the amount of private information. The three most stated reasons for storing passwords in the web browser for those participants who neither generally store nor not store their passwords were the importance and level of privacy of data, the computer they work with at that time, and the frequency they need to enter a password. While importance of information and the computer the user is using are indeed privacy related factors, the frequency of use indicates that at least certain number of people find passwords generally cumbersome to remember and type in. This findings corresponds well to the number of distinct passwords people use for their online accounts, being in average 4.64 and 10.14 respectively.

3.2.3 Trust

The average rating for trust in online application was 5.06 opposed to 6.86 for trust in offline applications (10 highest), with less than 5% rating their degree of trust in online applications higher than trust in offline applications and over 70% rated it lower. Despite over two thirds using web mail to access their mails, a web application, 60% gave 5 or less points for general trust in online applications opposed to 21% for local applications. Regarding the privacy concerns, over 80% of the participants agreed that they worry about privacy issues concerning personal information.

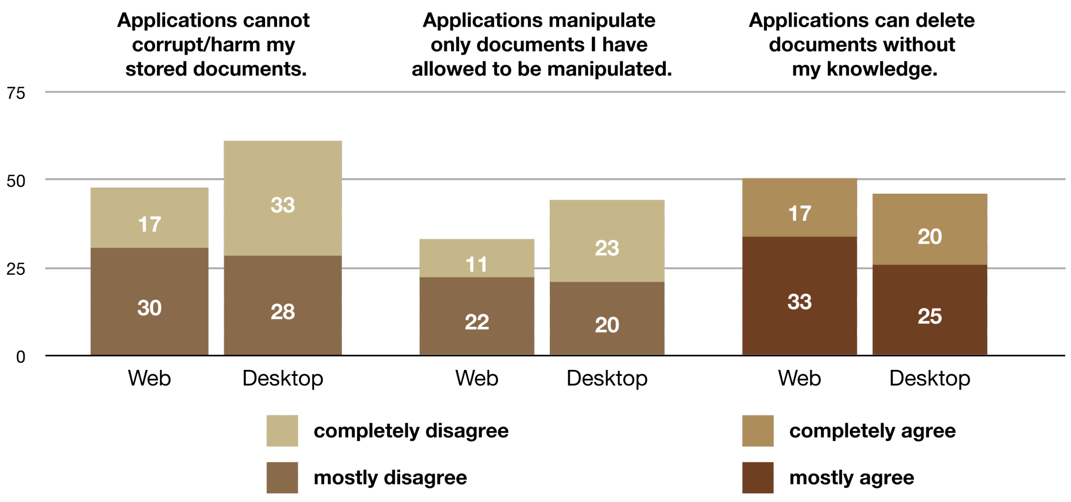


Figure 3.2: Results regarding the awareness of application's abilities

Moreover, about 75% of the participants either completely agree or mostly agree that they worry about privacy concerning documents of theirs being online. Also, more than half of the participants worried about data loss of their online documents. The awareness of online application's abilities to delete files, access files, and corrupt files was similar to those of offline applications, as it is depicted in figure 3.2.

3.2.4 Data online

Furthermore, the participants felt more convenient accessing files offline than online with an average rating of 8.15 in contrast to 6.13 for the level of convenience when accessing files online, with 8.2% finding it more convenient to access online data.

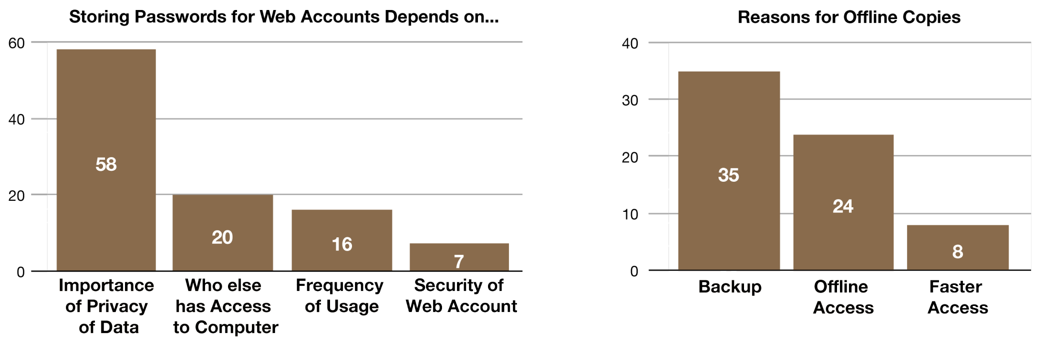


Figure 3.3: Results regarding password storage behaviour (left) and reasons for offline copies of online data.

Also, about 33% of the participants do not think that having an online copy of local data is a good idea. Of those who gave reason, all stated they worry about unwanted access of their data. Furthermore, reliability and accessibiliy concerned the participants; 35% preferred to have a local backup of online data and 24% wanted offline copies to have acccess when internet connectivity is not available.

3.3 Conclusion

In general the survey confirmed lower trust in online application. However, with about two thirds of the participants using web mail, a web application to access one's emails, and 17.7% doing online shopping once a month, it illustrates how web applications are already used by people.

Furthermore, the following conclusions about the users' relationship to web applications can be derived:

- Users generally trust web applications less than traditional offline applications
- Users worry about their privacy being kept when data is stored online
- Users would like a local copy to access data when internet is not available
- Users use only a small number of distinct passwords

Chapter 4

Architecture

WIDE focusses on two aspects that are closely related to each other: *Authentication* and *Authorization*.

Authentication of a subject is the action of validating or verifying the subject as authentic, that is, it is real true. With respect to the login mechanisms of web pages, authentication of the user seeks to verify that the user logging in is really the user he claims to be. Often the user authenticates himself with a secret that should only be known to him and thus, the web site can verify him.

Authorization is the concept of granting access to resources only to those with permission to do do and in doing so, protects resources from others. As such, information access is essentially a problem of authorization, especially for private information, as the owner has a stronger desire to ensure that only those he has allowed, access to such sensitive information.

Furthermore, the conducted survey indicated that users tended to worry about storing data online (see 3.2.4). Despite general trust in online applications appearing to be lower than in offline applications, some online applications are in fact very successful and are used on a day to day basis. Web e-mail systems are a classic example of an information oriented online service that many people use daily. The survey's result also highlighted the fact that users are aware of the potential risks of information being stored online and thus, do not place trust in some web applications.

The above leads to two basic requirements for application interaction:

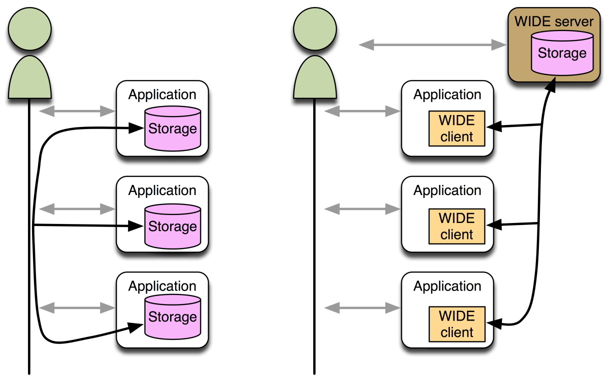
1. The user, in general, will prefer a smooth interaction with as few interruptions as possible. Since trusted applications do not pose a threat or risk to the user, interacting with a trusted application should minimize interruptions caused by, for example, increased security requirements.
2. Applications that users are unfamiliar with and are perceived to harbour potential risks to the user are not trusted. Such a web application could, however, be still considered by the user to be useful or interesting, despite the lack of trust. Thus, when using such an untrusted application, it must not gain any other information than what is explicitly permitted by the user, for example, information that the user does not consider to be private or sensitive.

In the following we describe the overall architecture of WIDE to support both authentication and authorization satisfying the above stated requirements to provide an user experience closer to that in desktop computing where privacy concerns and information access are perceived to be less problematic.

4.1 General Architecture

WIDE realizes a client/server architecture on the distributed systems of the different applications. It consists of two parties being the WIDE server and the clients - the web applications. To ease management of information, WIDE separates data from the applications and the information is accessed through the WIDE server that holds a storage space and handles authentication and authorization, like depicted in figure 4.1. This also adopts a more document

WIDE separates data from applications and centralizes it centric working approach as an alternative to current application centric working. The requests for accessing data are realized through so-called transactions. However, the applications themselves are not affected and interact directly with the user. As WIDE is designed as an information space users can sign in and manage the information, application, and data.



(a) Storage space implemented in current web applications

(b) WIDE provides a single shared storage space

Figure 4.1: Storage Space Solutions for Applications

The following sections describe the main components of the WIDE framework, the WIDE server and the application clients. An overview of the WIDE server structure and its interaction with other entities is provided in figure 4.2.

4.1.1 WIDE Server

The central component of WIDE is the WIDE server. It connects the user, web applications, and data to one information space. As such, its main purpose is in the handling of authentication and authorization so as to protect the information stored by the user.

For each connection to the involved parties a particular interface will handle specific tasks. The application interface is responsible for providing authentication to web applications and serves information access requests, so-called

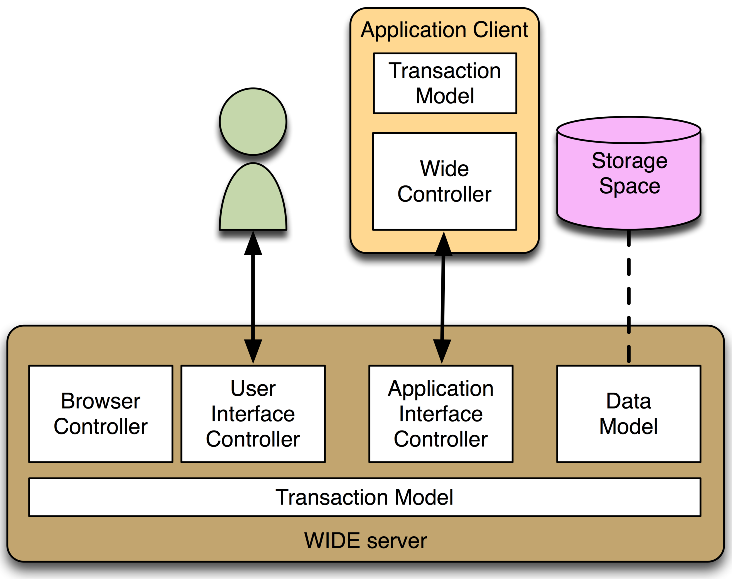


Figure 4.2: General Structure

transactions, from the web application to the WIDE server. The user interface controller prompts the user for authorization of such transactions. The execution of transactions, such as that to gain access to information, is managed by the data model.

Combining tags and hierarchical filesystem

The storage space, realized as a part of the server, supports various applications and their individual needs by way of a database-based hierarchical file system with tag support. It was chosen as it poses a superset of a purely tag-based file system and the traditional folder-based file systems. The hierarchy helps users to find files and discover files by browsing as shown by Jones et al. [2005]. However, Quan et al. [2003] and Marsden and Cairns [2003] have further found out that tags can improve a hierarchical file system and result in faster searching.

Transactions are further discussed in section 4.1.3 as they are equally involved both the server and the clients. The authorization process utilizing the transactions is explored separately in section 4.3.

4.1.2 Application Client

The application client's primary component is the WIDE controller which is responsible for communication with the server. The main concern when implementing the client side framework was to ensure that is would integrate well into applications without interfering with other toolkits. Therefore, aspects other than server communication were not explored as they are already a multitude of other toolkits that exist from which developers can choose.

The general native client framework is for the development of web applications. Description of how existing web applications can be extended can be found in 6.1.

The client framework consists of one controller component, the WIDE controller. It serves as an interface to the server for authentication and transaction requests. It communicates with the application via a controller of the developer's choice.

4.1.3 Transactions

For accessing data on the storage of the WIDE server, the web application requests so-called transactions. To support applications that require multiple transactions at a time, transactions are bundled to transaction bundles. Transactions must be bundled into transaction bundles as a single transaction is not sent separately to the server. Hence, the bundle holds information about the web application, user, and status.

Transactions are bundled

Each transaction defines an *action* on *content*. An *action* is access or manipulation of data that is described in *content*.

The *action* defines the kind of manipulation the application requests. A list of supported *actions* can be found in the appendix B. Some of these actions require user interaction and therefore, will render a user interface on screen requesting user input for execution. For example, when the user triggers an *open file* command, the application might want to let the user choose the file to open. These prompts will not be rendered by the application but instead by the server. This is to protect the stored information from the application and satisfies the requirement in section 1. Furthermore, it will provide a more consistent user interface. The transaction's *content* may be the files and folders depending on the defined *action*. The value of *content* is XML encoded to provide flexibility for further modifications.

4.2 Authentication

The general authentication is achieved by a cookie-based authentication, such as found in other web applications. In addition, the web application connected to WIDE benefits from WIDE's single sign-on mechanism. Once logged into WIDE, the user does not need additional passwords to authenticate as the user's identity is already known to the WIDE server. To log in to client applications the WIDE server uses the user information to authenticate the user to the application without requiring additional authentication by the user.

This approach was taken in order to further unify the information space. In addition, log-in procedures often introduce unwelcomed breaks when working with various applications by interrupting work flow which violates the requirement in section 4.

Single sign-on approach supported by survey

Further support for the single sign-on approach is highlighted by the survey. One of the motivations for users to web passwords was the frequency of usage (see 3.2.2). Moreover, in a traditional desktop environment, although the user desktop is often protected by a log-in themselves form part of the desktop and thus, do not request from the user additional login. As WIDE is a distributed system, such an behaviour can be achieved by a single-sign on mechanism with a central login instance, being the WIDE server in this case.

When logging out on such an distributed system, the user must be logged out on every application. Therefore, WIDE performs its own log-out procedure after all running applications of the user have been logged out. In addition, each application can use WIDE's authentication information as a base for its own authentication method to protect the application.

Assuming someone was able to gain access to an application it would still not be possible for that person to access the user's information on the WIDE server itself. WIDE applications may store some settings to personalisation but they do not host the data. As each application holds an individual authentication cookie that is only used for the application's own authentication the WIDE server and the user's hosted data are not affected and exposed to risk.

Each application holds its individual cookie

When the web application is started, WIDE's authentication mechanism connects to the server's *ai/logged_in_user* to receive information about the status of authentication as depicted in figure 4.3. If the user is not yet logged into WIDE, he is prompted to do so. Only with the user logged in, that means owning the cookie that the authentication set, a random ID is generated and linked to the user's identity and passed to the user for redirection to the application. The application receives the ID and request the user's identification by passing the ID to *ai/user_hash*. Once successfully logged in at the application, the client framework sends a notification to the WIDE server for a status update and later, log-out.

Similar to log-in procedures, logging out is also communicated to the server after a successful log-out to update the application's status on the WIDE server. When the user logs out of the WIDE server, all applications are requested to perform their log out. When an application does not log out, the WIDE server cancels the log out. This behaviour, however, can be overridden in the case of a non-responsive application server. Once the user is logged out of all his applications, the WIDE will log out the user.

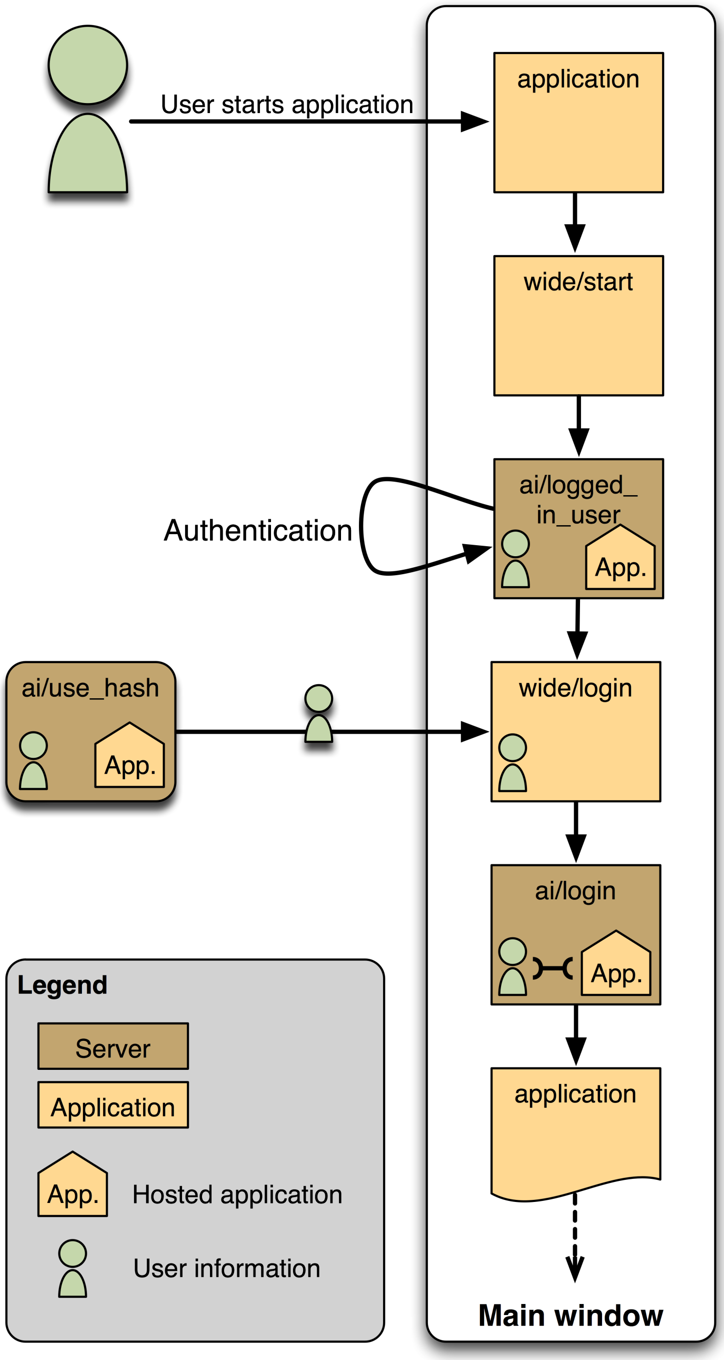


Figure 4.3: Authentication process at start of client application

4.3 Authorization

The access control mechanism of WIDE is request-based, that is, authorizations for a request's execution is granted request-by-request basis. Each transaction request from an application must be authorized individually on the WIDE server's side. In the case of transactions from an untrusted application, the user is prompted by the WIDE server to authorize the request or discard the transaction. This mechanism, is thus able to provide general protection of all the user's information. Were it to be realized on an exceptions of the general access policy, a continually increasing number of exceptions could make the user lose track of the exception and ultimately lead to unwanted access, e.g. through misplaced private information. This is not the case with WIDE as authorizations given by accepting the prompts are only for one particular transaction and do not affect any future transactions.

With request-based authorization no exceptions needed

However, such prompts cause interruptions and as such, can harm the user's work flow. For that reason, when trusted application requests a transaction, the server grants access directly without asking the user for authorization. This does not violate the requirements because trusted applications are not considered to be a risk and thus, information access can be granted.

Therefore, the request-based access control of WIDE is able to satisfy both stated requirements of application interaction. The technical realization of the described transaction granting request-based access control is illustrated in figure 4.4.

When the user interacts with an application and triggers an action that requires, for example, stored information, the application sends an request to the WIDE server. Once the user has permitted the transaction the server executes the transaction and returns the result to the application.

The authorization process consists of multiple steps to satisfy the needs of protecting information stored on the server and minimize any lack of responsiveness due to bad network performance.

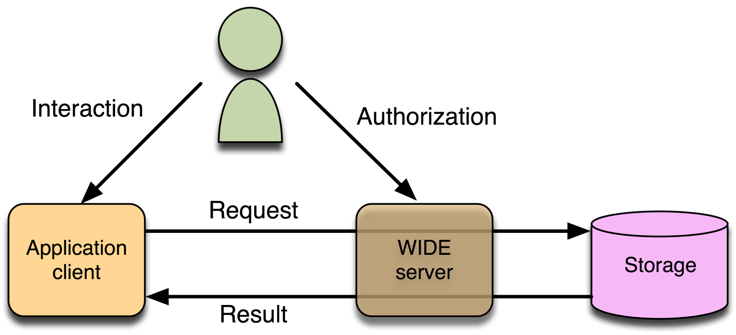


Figure 4.4: Request Based Authorization: The User Stays in Control.

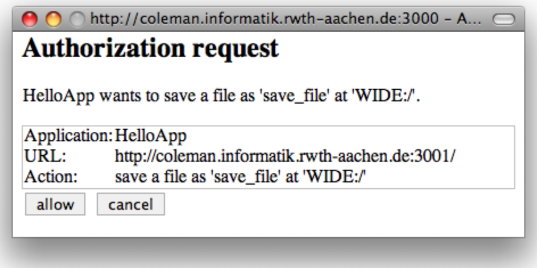
Figure 4.6 provides an illustrative overview of the whole authorization process. When the user interacts with the application and triggers a command that requires access to the data storage, a corresponding transaction is formed and bundled in a transaction bundle. The application then requests the server to create a new transaction bundle.

Pop-up window for rendering authorization requests

This and the following steps are performed in a pop-up window for multiple reasons. An additional pop-up window does not interrupt or leave the current page of the application. This pop-up is also used for rendering the authorization dialog by the WIDE server. It is similar to what users are accustomed to in desktop user interactions. Furthermore, it provides a direct connection between the user and the server through which the server can authenticate the user with the authentication cookie that is set by the authentication mechanism. The user is crucial in the authorization and execution process. Without the user neither will run as only the user has the cookie holding the authentication information that is required for the authorization to progress.

The server then in return requests the transactions of the bundle from the application that then returns the transactions' actions, however, the content of the transactions is left out. Content of transactions can be substantial and require a long period of time when being transferred depending on the network's transfer rate. A long file for example, can delay the following steps even when the transaction is rejected by the user. Thus, the content of transactions is transferred after the authorization by the user has taken place.

When the application is not trusted, the server requests authorization for each transaction. In the case of a dialog being displayed, such as a file open dialog, the completion of the interactions - in the given example to select a file and trigger the open command - is taken as the authorization. Another authorization prompt is not necessary because when completing the dialog interaction it can be assumed that the user is aware of the action and by submitting his choice, authorization is given.

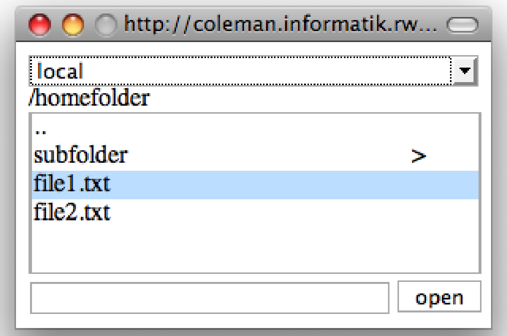


Authorization request

HelloApp wants to save a file as 'save_file' at 'WIDE:/'.

Application:	HelloApp
URL:	http://coleman.informatik.rwth-aachen.de:3001/
Action:	save a file as 'save_file' at 'WIDE:/'

(a) Simple Authorization Request Prompt



(b) Transaction Requiring Further Actions by the User

Figure 4.5: Authorization Prompts

Figure 4.5a portrays a prompt for a simple transaction, such as to delete a file, that requires an authorization from the user. For more complex transactions , such as that to open a file the user selects, the required dialog for further user input is presented as shown in figure 4.5b.

These authorization prompts are rendered by the server and thus, information about the user's information cannot be reached by untrusted application as it does not reach the application at any point during the authorization. Moreover, it removes the need for users to have to familiarize themselves with each application's dialog styles. It, therefore, improves consistency because all dialogs are rendered by the one party - the server.

Server rendered authorization prompts increase consistency

After successful authorization of all transactions, the content is transferred to the server where the transactions are executed. The results of the transactions are returned to the application and the URL for returning to the application is requested from the application. Once the URL has been returned to the server, the server guides the user back to the application by redirecting to the received resume URL that lets the application further process the results.

4.3.1 Example

The following provides an example scenario to illustrate the authorization process. In this scenario the user works with an online word processor that supports WIDE. The user is logged in and the user wishes to open a file to edit.

1. The user clicks on the "file open" button in the application menu.
2. The button click triggers the application to form a new transaction bundle that contains a transaction with a "file open dialog" action. Once created, the application opens a pop-up window to the WIDE server's application interface.
3. The application interface creates an empty transaction bundle and links it to the application's bundle. In the next step of the creation, the bundle fetches the transactions, excluding the content, from the application. Once all of the transactions has been received, the user is redirected to the user interface controller.

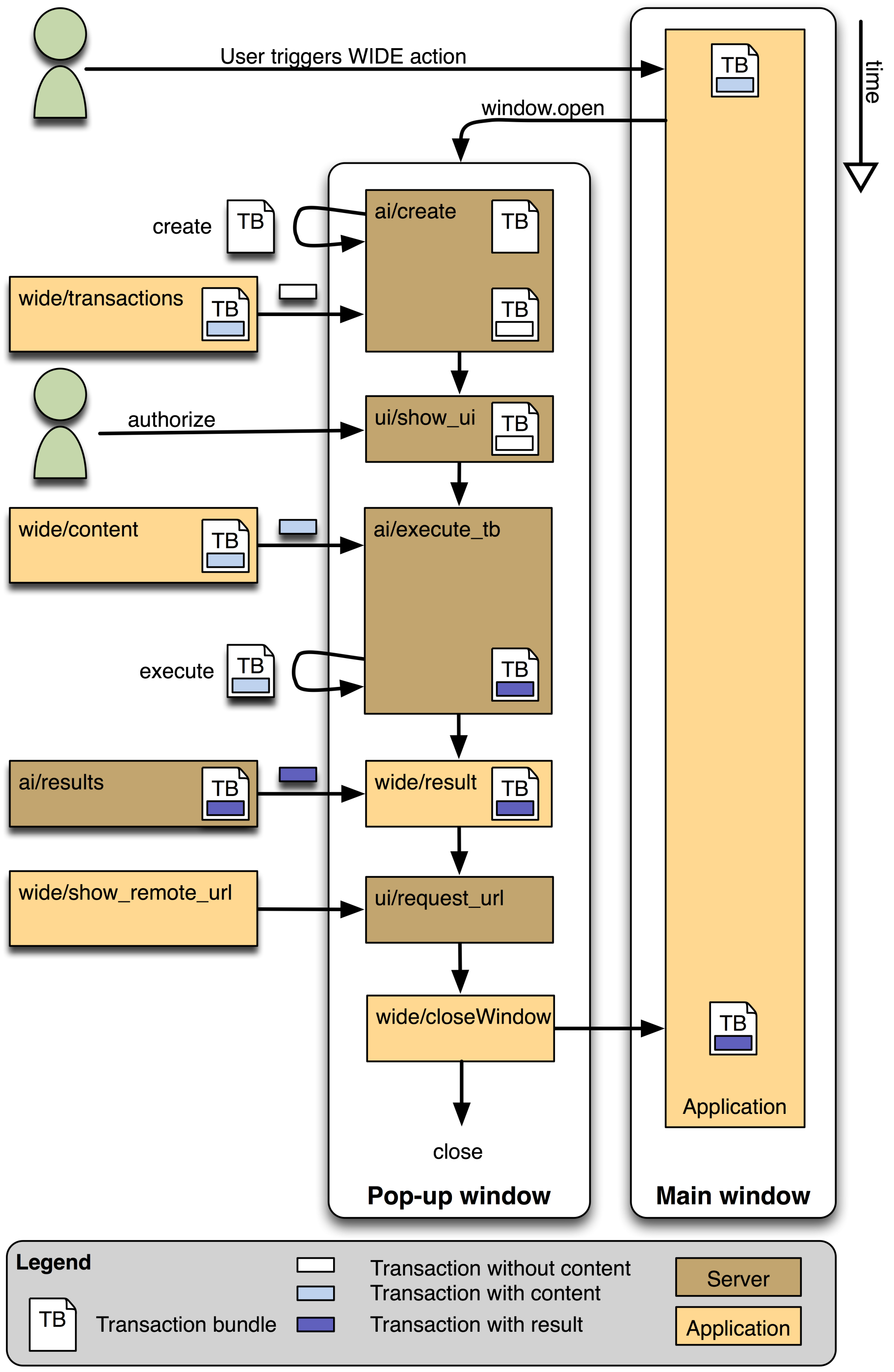


Figure 4.6: Message exchange for transaction request and authorization by the user.

4. The user interface controller renders a dialog that prompts the user to select a file to open.

5. When the user chooses his file and completes the request, he is completing the authorization. The user interface calls the transaction bundle to execute its transactions which requests the content of each transaction before the actual execution. The result, the content of the file the user has chosen to be opened, is stored in the transaction.

6. After the execution the server redirects back to the application that requests the results and stores it in the transaction's content.

7. In the final step the application's *closeWindow* calls the result receiving callback action, that resumes the application, and closes the pop-up window.

8. The callback function then displays the result of the transaction as text in the text area to the user.

Chapter 5

Implementation

We implemented the WIDE server as a Ruby on Rails application and several clients in various languages with different characteristics, a native application that corresponding to the server is also implemented as Ruby on Rails. However, we identified certain clients that natively are not supported by WIDE. By developing adjusted frameworks, as explored in section 5.3 and 5.4, we increased the range of application classes WIDE supports and provide useful enhancement to the applications.

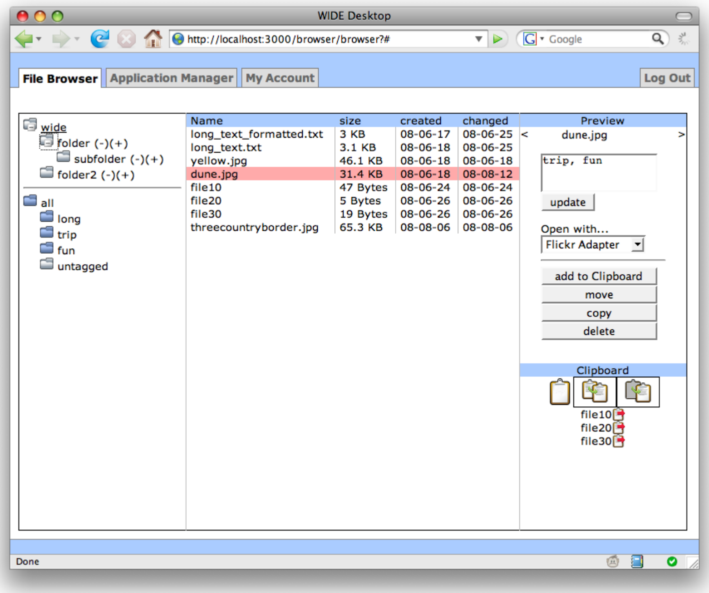
5.1 Server Implementation

The architecture of the WIDE server is implemented as Ruby on Rails utilizing a MySQL database. For communication with the clients we chose XML as it poses a standard for data representation and parser for XML are available in many programming languages.

The user access the server to interact with both the data and the application working on the data. Therefore, we decided to implement a data browser and an application manager.

5.1.1 Data Browser

The *Data Browser* is a file browser that provides basic file system interactions, such as copy, move, and delete. Furthermore, it allows to add and remove tags. Tagged items appear in automatically created tag folders that display all items with a particular tag. As an alternative to the application centric way to work, it allows the user to first select a file and then choose the application of choice to open the file with. In summary, it is to maintain the information space on the data side.



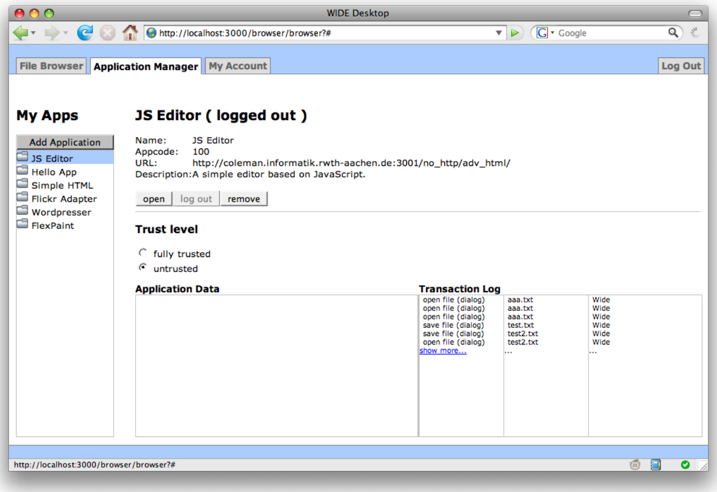
Name	size	created	changed
long_text_formatted.txt	3 KB	08-06-17	08-06-25
long_text.txt	3.1 KB	08-06-18	08-06-25
yellow.jpg	46.1 KB	08-06-18	08-06-18
dune.jpg	31.4 KB	08-06-18	08-08-12
file10	47 Bytes	08-06-24	08-06-24
file20	5 Bytes	08-06-26	08-06-26
file30	19 Bytes	08-06-26	08-06-26
threecountryborder.jpg	65.3 KB	08-08-06	08-08-06

trip, fun

Figure 5.1: Data Browser

5.1.2 Application Manager

The *Application Manager* maintains the user's web applications. There user can add and remove web application, analogous to installing and de-installing on traditional desktop systems. The list of applications also indicates the login status of each application. A panel presents additional actions and information and allows the user to start an application or log out. The user can also set the trust level in this panel. Furthermore, the details to each application are presented to the user, such as the URL of the application, its description, and the last transactions.



JS Editor (logged out)

```
Name:       JS Editor
Appcode:    100
URL:        http://coleman.informatik.rwth-aachen.de:3001/no_http/adv_html/
Description:A simple editor based on JavaScript.
```

Trust level

( ) fully trusted
[✓ untrusted]

Application Data

Transaction Log

```
open file (dialog)   aaa.txt     Wide
open file (dialog)   aaa.txt     Wide
open file (dialog)   aaa.txt     Wide
save file (dialog)   test.txt    Wide
save file (dialog)   test2.txt   Wide
open file (dialog)   test2.txt   Wide
show more...         ...         ...
```

Figure 5.2: Application Manager

5.2 Native Client: Wordpresser

For prototyping we decided to implement a small publishing application for Wordpress[1], a weblog system that is used by many people. The application, we called it Wordpresser, publishes a text post to a weblog that can be read by others. Text editors supporting WIDE that do not provide publishing functionality can thus still be used to write a post and then post the saved text file with Wordpresser. Therefore, this small prototype well illustrates how a workflow in WIDE could look like.

[1]`http://www.wordpress.org`

5.2.1 Design

As a publishing tool, Wordpresser allows the user to open files, add a title, and publish the post to the weblog. The weblog is determined by settings, such as the weblog's address and other information required for remotely publishing a post, that the user must enter before publishing. The settings are stored separately by the application.

5.2.2 Implementation

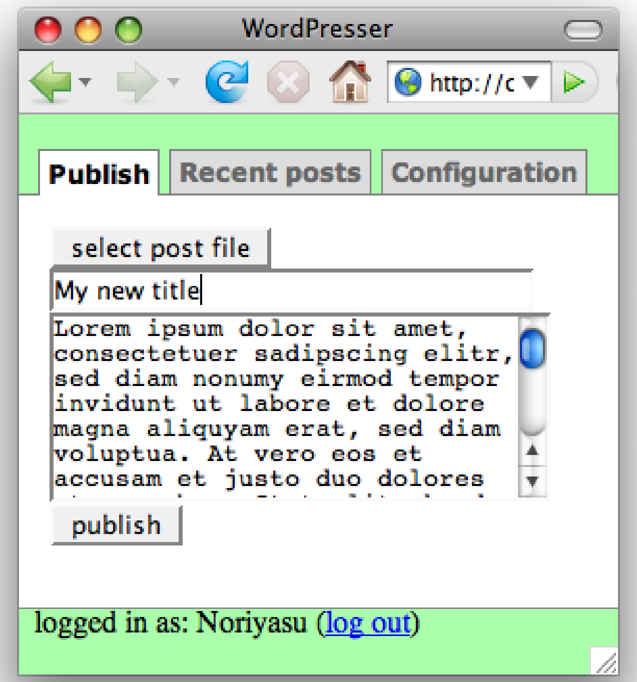
To implement the prototype we used the native client framework in Ruby on Rails and a MySQL database. The database also stored the settings of the user's Wordpress weblog. For the publishing the application accessed the weblog via XML remote procedure calls[2] (XMLRPC).

5.2.3 Interaction

For publishing a post the user starts Wordpresser and then opens a text file from WIDE or types in a new text. Due to the alternative more document centric approach, another way of opening a text file with Wordpresser is to select the file in the Data Browser and open it with Wordpresser. Into the input field at the top the user can further type in a title for the post, as depicted in figure 5.3. To send the post to the weblog, the user clicks the *publish* button. Once published, the most recent posts are displayed to the user to confirm the successful post to the user. In the recent posts panel, posts can be directly accessed through the provided hyperlinks.

The user does not have to enter the setting required for publishing because WIDE's authentication identified the user to Wordpresser that then loads the corresponding settings.

[2]http://www.xmlrpc.com/spec



My new title

Lorem ipsum dolor sit amet, consectetuer sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores

logged in as: Noriyasu (log out)

Figure 5.3: Wordpresser is a Simple Publishing Tool That Posts Text Files.

5.2.4 Conclusion

The Wordpresser prototype implemented successfully the WIDE request based authorization. For other kind of applications however, a different framework is needed. The following section describes such a framework for JavaScript based applications.

5.3 JavaScript Client Framework

When analysing the framework, a lack of support for JavaScript stand-alone application was discovered. Stand-alone JavaScript applications are small applications that work without a server back-end. These kinds of applications do not work with the general WIDE framework as is relies on direct server communication.

They are limited as they can process data but cannot access and manipulate stored data. For example, with a JavaScript based drawing program, one could draw simple images, but not save them for future editing. These kinds of applications can be considered stateless as states of the application cannot be stored in a database. Instead, the state exists in the current instance of the browser environment of the application. A reload resets the instance and the state is no longer available. This is also true for another window of the same application that creates a new instance of the application that is independent of the other instance. A problem presents itself here as instances are independent from each other. Callbacks addressing web pages of the application creates new instances that cannot retrieve the data of the other instance. Furthermore, instead of having a database on the server for such applications a local database solution is required that runs in the application's instance in the browser environment. Another difference is that in contrast to a server, HTML applications cannot send HTTP requests to domains other than of the originating server. To overcome this restriction of the JavaScript environment, another method has to be chosen for both inbound and outbound communication.

5.3.1 Design

The WIDE client framework for JavaScript addresses these key problems and provide a local WIDE client implemented in JavaScript that runs in the browser's environment. Thus includes a simple database to store transaction data, communication across window instances, POST method based transaction requests, and alternative data retrieval.

When the web page of the application is loaded, the transaction bundle database is initialized. New transaction bundles are stored in that database and referenced through its index in the database.

To bridge instances of the application, the frame makes use of the *opener* attribute of the *window* instance. This is achieved, for example, when opening a new window, such as a pop-up window, this window can back-reference to the originating window via `window.opener`. This bridges the two instances and data can be accessed across instances if there is such a parent-child relationship. This relationship holds even when containing documents changes, as long the currently referencing document is of the same domain.

To send transaction requests to the WIDE server the framework uses a HTML form that is submitted via JavaScript to post data. For receiving messages, such as the result of transactions, the framework dynamically pulls the results from the WIDE server by loading a ⟨*script*⟩tag onto the application's page when the transaction has been executed on the WIDE server. These mechanism however require a different behaviour of the WIDE server.

5.3.2 Implementation

The framework consist of the following two Javascript files and three HTML files.

- *wide_settings.js*
- *wide_no_http.js*
- *post_tas.html*
- *post_tas_content.html*
- *result_receiver.html*

wide_settings.js holds constants for communication with the WIDE server, such as the application's code and URL. These settings must be defined correctly to ensure that the framework will run.

wide_no_http.js handles transaction creation and the communication with the pop-up window instances. In a Javascript based client, the instantiation of a transaction bundle is similar to a server based client. First, `newTB()` creates a new bundle and stores it in the local database. Through the attribute *resume_function* , the function which is called after the results are received, is specified. New transactions are appended to the array `transactions` of the bundle by `newTA()` , a function of the bundle. To specify the transaction, attributes, such as `ta_action`, `ta_name`, `ta_content` , can be set. With the submit method `submit_tb()` of the transaction bundle the authorization starts.

An example of a simple initialization for saving a document is listed here.

```
var tb = newTB(); //new bundle
var ta = tb.newTA(); //new transaction
ta.ta_action = SAVE;
ta.ta_name = 'text';
ta.ta_content = 'hello world';
tb.resume_function = 'saved';
tb.submit_tb(); //submit to server
```

The three HTML files are for the communication of the authorization process that is undertaken entirely in a pop-up window created when the transaction bundle is submitted as depicted in 5.4. With the three HTML files, the transactions are sent to the WIDE server and the result is requested from the server in the form of a JavaScript file that executes the resume function of the transaction bundle together with the result.

In more detail, the submitted transaction bundle's ID is sent to the WIDE server's application interface to create a corresponding transaction bundle. Due to the specific *no_http* mode sent alongside the request, the server redirects back to the application. *post_tas.html* instantaneously retrieves the transaction data from the application host instance over the `window.opener` context and posts the transaction details without the content to the *ta_receiver* action of the WIDE server's application interface where the authorization of the transaction occurs.

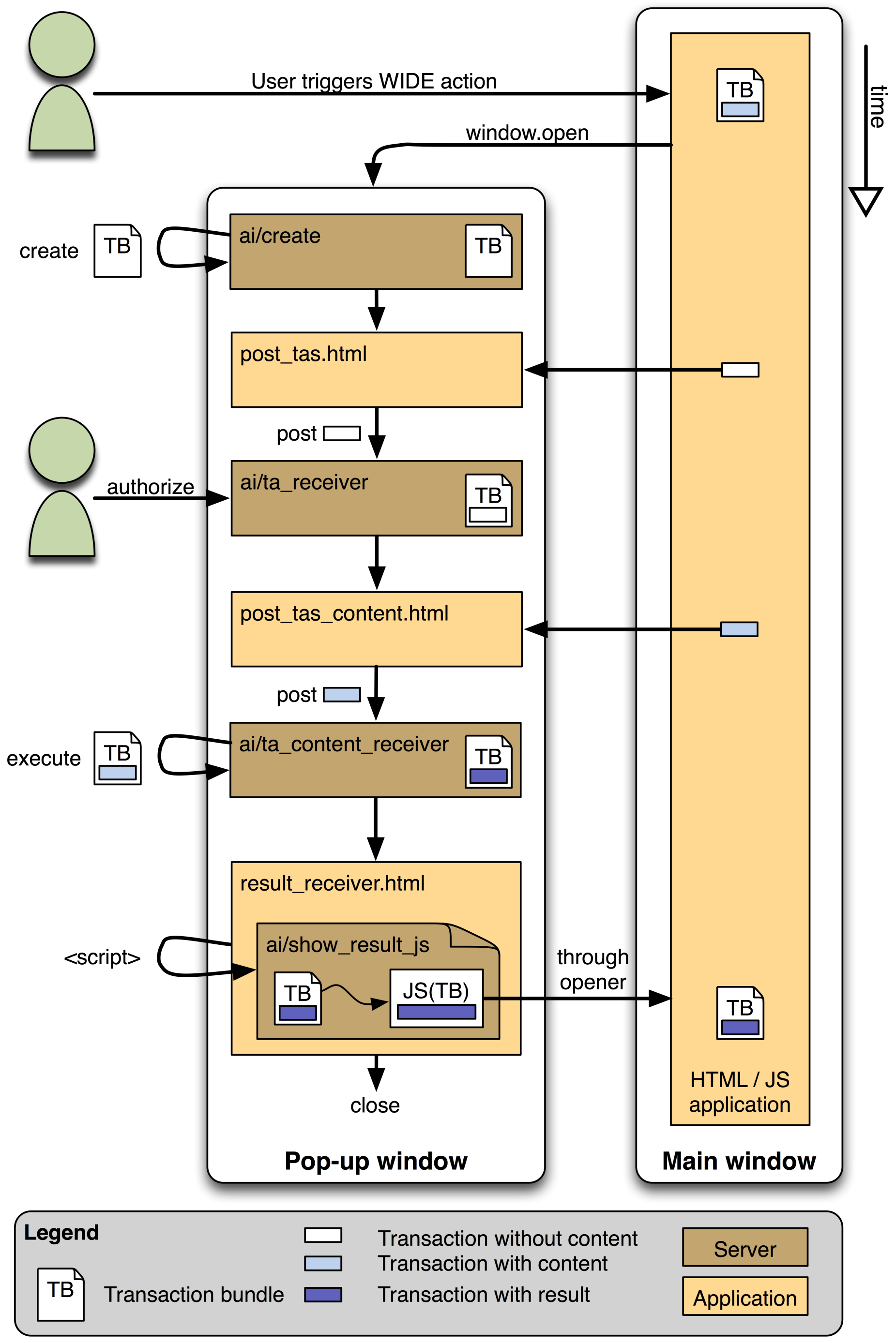


Figure 5.4: Overview of WIDE's JavaScript Framework

After successful authorization, the server redirects back to *post_tas_content.html* that posts the transactions' content to the *ta_content_receiver* action that executes the transactions and redirects to *result_receiver.html*. With the received index, the result is pulled from the WIDE server by inserting a $\langle script \rangle$ tag into the DOM that loads a JavaScript file at *show_result_js*. This JavaScript file is dynamically created from the content of the associated index. It contains code that transfers the content of the transactions to the application's host instance's database and calls the corresponding resume function. The resume function has to accept one parameter which will be the resulting transaction bundle, like that shown in the following example.

```
function saved(tb){
  var ta = tb.getTransactionByName('text');
  myText = ta.ta_content;
}
```

Through the adjustments explained above, the framework supports the same mechanism of transaction handling for JavaScript-based applications. This was prototyped with the implementation of the JS Editor prototype application.

Prototype: JS Editor

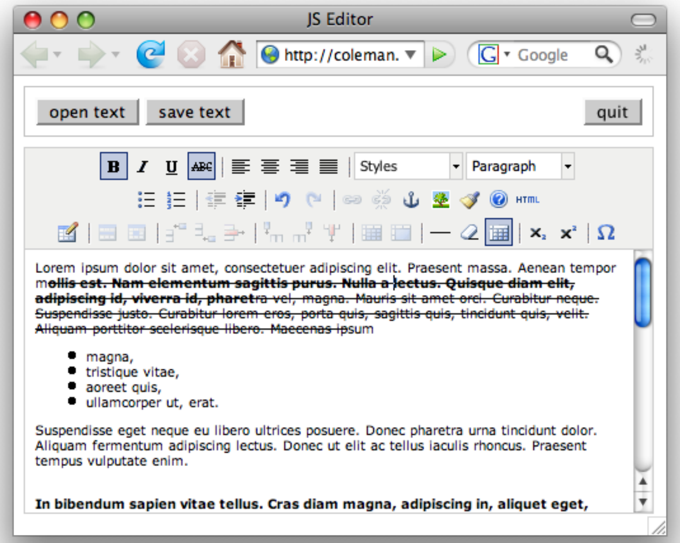
Since JavaScript-processed data generally cannot be stored online due to the restrictions of JavaScript, sophisticated applications are difficult to realise. To complement the sketched potential workflow for Wordpresser (see 5.2 we decided to implement a text editor based on JavaScript named *JS Editor*. JS Editor is a text editor web application to manipulate text which the user can save and load through the WIDE framework.

Implementation

TinyMCE JavaScript Content Editor[3] (TinyMCE) provided a powerful text editor framework for the JavaScript based

[3]`http://tinymce.moxiecode.com/`

editor. However, it does not define from where the content to be edited comes or how the data is saved. The getter and setter methods for the currently edited content provided also by TinyMCE. These were used to implement the open and save functionalities with the described JavaScript Framework.



Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Praesent massa. Aenean tempor m~~**ollis est. Nam elementum sagittis purus. Nulla a lectus. Quisque diam elit, adipiscing id, viverra id, pharet**ra vel, magna. Mauris sit amet orci. Curabitur neque. Suspendisse justo. Curabitur lorem eros, porta quis, sagittis quis, tincidunt quis, velit. Aliquam porttitor scelerisque libero. Maecenas ipsum~~

- magna,
- tristique vitae,
- aoreet quis,
- ullamcorper ut, erat.

Suspendisse eget neque eu libero ultrices posuere. Donec pharetra urna tincidunt dolor. Aliquam fermentum adipiscing lectus. Donec ut elit ac tellus iaculis rhoncus. Praesent tempus vulputate enim.

In bibendum sapien vitae tellus. Cras diam magna, adipiscing in, aliquet eget,

Figure 5.5: JS Editor, a JavaScript-based editor that supports WIDE

When JS Editor is started, the user can use the top buttons to open a text file and to save the current text, for example, to continue editing at a later point in time. Furthermore, the top right button quits the application.

In summary, we successfully demonstrated that WIDE's JavaScript client framework provided additional value to JavaScript stand-alone applications by way of data manipulating and storing functionality that is not possible solely with JavaScript and HTML.

5.4 ActionScript Client Framework

The JavaScript framework provides a solution for JavaScript applications that is similar to Flash-based applications. They, like JavaScript applications, run in the browser and do not necessarily rely on a back end server. However, Flash does provide methods for HTTP requests without restrictions, that is, a server outside the domain of the originating server can be accessed. This functionality can replace the problematic reception of the results in the JavaScript client framework. Therefore, we designed a client framework by re-using parts of the JavaScript client framework and using the external interface of ActionScript to communicate with it.

5.4.1 Design

The ActionScript client framework consists of JavaScript, HTML files, and ActionScript classes. The communication during the authorization process is achieved analogous to the JavaScript client framework. The external interface of ActionScript provides callbacks from JavaScript functions to call ActionScript functions in the Flash application, and external calls to run JavaScript functions in the embedding HTML file from the Flash application. These methods are used to communicate the transactions natively formed in ActionScript to the external JavaScript and to return the callbacks of the WIDE server to the Flash application as the user instance of it cannot be called directly from outside the browser environment. Finally, the results are received via a HTTP request similar to server based applications.

5.4.2 Implementation

In general the implementation of the ActionScript client framework consists of a modification of the JavaScript client described before and the following ActionScript classes and JavaScript file.

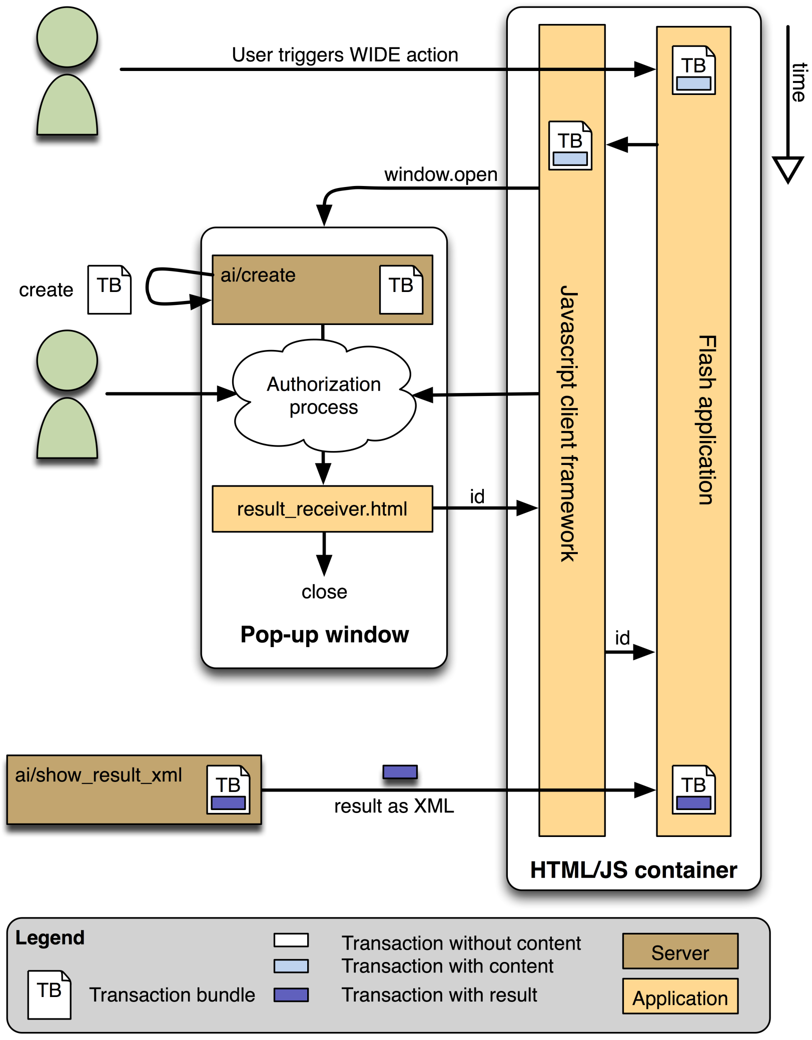


Figure 5.6: Overview of WIDE's ActionScript Framework

- *WideClientSystem*
- *Transaction*
- *TransactionBundle*
- *no_http_flex.js*

The *WideClientSystem* class holds settings, the initialization, and the transaction database. It also communicates the initialization to the external JavaScript environment.

The setup of transactions is similar to other client frameworks. An example is shown here.

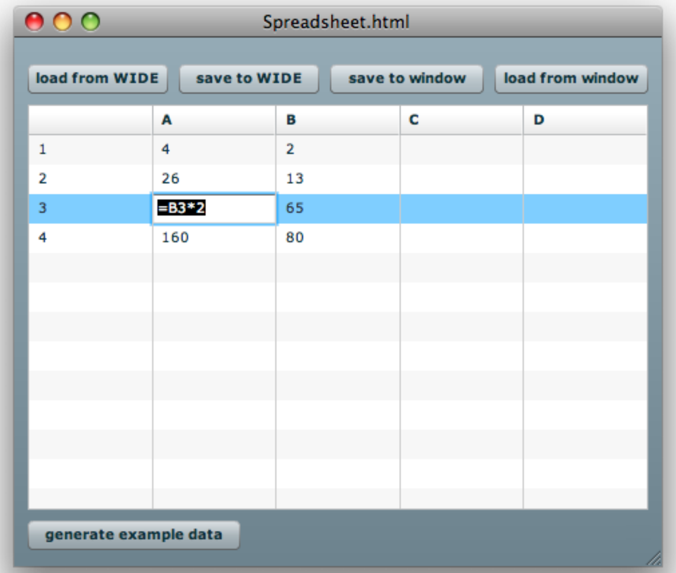
```
var tb:TransactionBundle = wideClient.newTB();
var ta:Transaction = tb.newTA();
ta.ta_name = 'save_transaction';
ta.ta_content = 'hello flex';
ta.ta_action = wideClient.SAVE_AS;
tb.resumeFunction = 'savedToWide';
tb.submit();
```

When a transaction bundle is submitted, the transaction is sent to the external JavaScript framework that initiates the authorization process. For a detailed description of the authorization process communication see 5.3.2. However, when *result_receiver.html* is called by the WIDE server, it calls the Flash framework with the received index. The framework then requests the result via a ActionScript HTTP request. An overview of the whole process is provided in figure 5.6

5.4.3 Prototype: Flex Spreadsheet

After having implemented two applications dealing with text, the prototype of a Flash-based web application we decided to be a spreadsheet application. Spreadsheets being also text-based are good for prototyping as textual data is small and easier to read. Furthermore, spreadsheet applications seems to pose the next most important office applications after word processing applications as they can be commonly found in office suites, such as Google Docs and Zoho.

We developed *Spreadsheet* with Flex Builder 2 and integrated the ActionScript framework. As data format for the spreadsheets we chose comma separated values in favour of quick prototyping as it can easily edited manually. The spreadsheet application provides basic functionalities including labelling cells and entering simple formulas for calculation. In addition, similar to JS Editor it provides a WIDE supported loading and saving functionality.



	A	B	C	D
1	4	2		
2	26	13		
3	=B3*2	65		
4	160	80		

Figure 5.7: Spreadsheet, a Flex-based Spreadsheet Application Supporting WIDE

The *Load from WIDE* button loads a file from WIDE after which, depending on the contents, the application displays a table with the corresponding cells or a notification that the contents is not in the right format. The user can edit cells individually and enter formulas as shown on figure 5.7. For later editing the spreadsheet then can be saved to WIDE.

5.5 Conclusion

In this chapter we described two frameworks that enabled new web application classes to utilize WIDE and thus, expanded the range of supported web applications. Both addressed application classes can be offered by a simple web storage service as they do not rely on a server and if doing so cannot provide functionality as it is provided by the frameworks.

Through the prototyping of the new frameworks we adjusted the WIDE server to the new frameworks, in particular the returning of results and handling of requests.

While WIDE supports ways to develop web applications, it is unlikely that existing well established web applications will instantaneously support WIDE. Thus, chapter 6 explores ways to integrate existing web application externally.

Chapter 6

Integration

6.1 WIDE Support for Existing Applications

In contrast to applications that are being developed, existing applications cannot be adjusted easily. Although some applications provide APIs to access the application's service or plug-in support, others do not offer any support for external adjustment.

Integration of applications is crucial for several reasons. A User who is content with his current web application is unlikely to change his favourite application. Furthermore, data of that application might have taken a long period of time to be gathered and if migration is possible, at least cumbersome to the user. For some applications the accumulation of data itself is a valuable information that is lost when the data is migrated. In such a case, integrating the existing application solves the problem without losing any data of the user. In addition, with integration, applications do not have to be developed from scratch and development effort is reduced to the development of an appropriate integration.

Therefore, this chapter explores methods to incorporate WIDE support into existing applications.

6.1.1 API

APIs can be used to interact with the application through other applications, but they do not offer methods to integrate new functionality to the existing web application. Integration of new functionalities requires the application to act and react instead of merely returning replies to incoming request from the API. To overcome this a new view can be created that interacts with the API. This, however, results rather in a new application since APIs often do not provide all the functionality the application provides. Instead, APIs can be used together with other methods to integrate new functionality to the existing application. The following sections will discuss these methods.

6.1.2 Plug-Ins

As adjustment of the original existing application is important in keeping the user experience when working with the application, it is necessary to seamlessly integrate without any additional user interactions. Plug-ins offer such seamless integration. Plug-ins are computer programs that can extend (be plugged into) the application, the so-called host application. To interact with the host application, plug-ins use the application's API. The plug-ins are usually called automatically from the host application itself and thus, do not require additional user interaction to be run. Furthermore, plug-ins are installed to the host applications which keeps the web application's workstation independence.

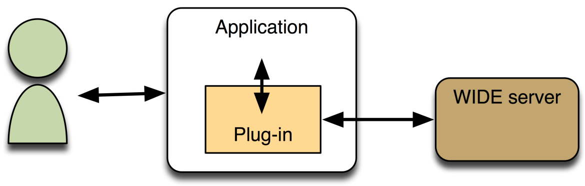


Figure 6.1: Possible Architecture for Integration: Plug-ins

Therefore, plug-ins in combination with APIs offer a good solution for WIDE to integrate into existing applications.

6.1.3 Adapters

For web applications that do not support plug-ins, a more general approach to insert new functionality to an existing web application is needed. Flash-based applications are difficult to adjust as they run in the web browser's sandbox that cannot be accessed from the outside without changing the original code inside the application. In comparison, HTML-based application's views are in text and thus, are easier to to interact with. Two possible approaches are tunnelling and code injection.

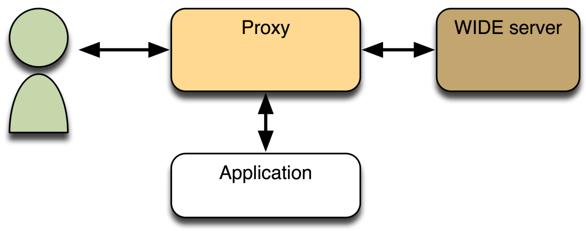


Figure 6.2: Possible Architecture for Integration: Tunnelling via a Proxy Server.

Tunnelling

Tunnelling requires all the interaction to be tunnelled through another web server that inserts some extra functionality or adds some WIDE specific interface. The tunnelling option gives the adapter application full control of the adapted application. However, the additional web server between the adapted server, being the application server, and the user also means that interaction is delayed by the factor of two, because every request and response is routed through the adapter server. Furthermore, it also increases the risks of identity theft as all interactions are passing through the adapter server. Thus, login information, such as passwords, and other information, such as document data, can be monitored easily by the adapter. In fact, this kind of misuse is not possible to detect because it is undertaken on the server with server sided processing and as such, can occur without the user's knowledge.

Code injection

The other approach is to inject some JavaScript code into the web application's site to be adapted. JavaScript provides means to insert, remove, and manipulate elements of a web page. In contrast to the tunnelling approach, JavaScript is executed on the client's machine which leads to better scaling as opposed to a server sided solution. Even with an optional backend server that the injected JavaScript code communicates with, only those interactions requiring the backend server will need this additional communication while other parts of the web application's interaction remain the same.

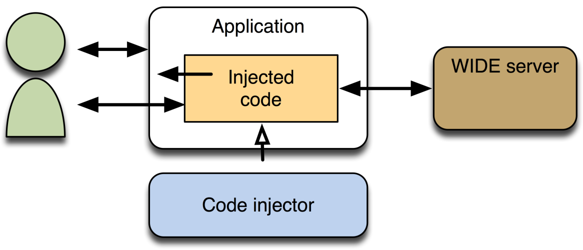


Figure 6.3: Possible architecture for integration: Code injection via web browser plug-in.

However, as code injection requires the injecting software and the code to be injected to be situated on the web browser's workstation, this approach is not workstation independent. Despite the lack of independence, locally executed JavaScript has one major advantage over the tunnelling approach. One of the major drawbacks of tunnelling was the potential identity theft due to invisible server sided code. In contrast, since JavaScript runs in the web browser it is possible to inspect the code at any time. This openness cannot prevent identity theft but facilitates detection of it and thus, lowers the probability of malicious code. Furthermore, JavaScript injection code does not require a running web server giving more developers access to the development of adapters.

In light of the above factors, we have chosen the code injection approach. The code to be injected varies depending on the web application because the user interface, in which the adapter needs to be inserted, differs. As the injection code needs to be downloaded and stored locally on the workstation the problem of different versions, like those of traditional software, can still arise. This effect can be reduced however, through dynamic loading of external JavaScript code. The scripts are dynamically requested at runtime from a single source, a web storage, and thus, unless the loader for dynamic loading requires change different versions of code should not appear.

6.2 Prototype Implementation

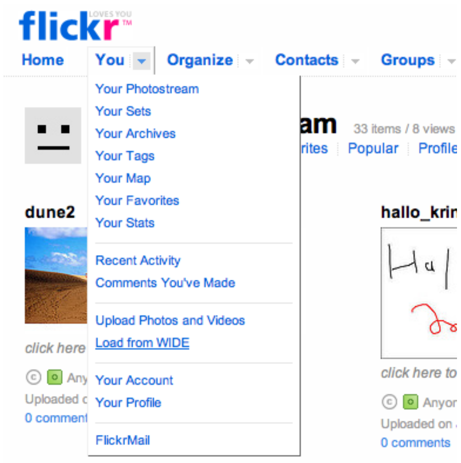
6.2.1 Flickr Adapter

The Flickr Adapter incorporates WIDE support to Flickr[1], an online community platform for hosting videos and images, by integrating methods to interact with the storage of WIDE. It provides two basic functions: loading pictures from WIDE to the user's Flickr account and saving Flickr pictures to WIDE. Technically, the adapter is realized by a server supported web application and Greasemonkey[2], a Mozilla Firefox extension that allows users to run JavaScript scripts on web pages.

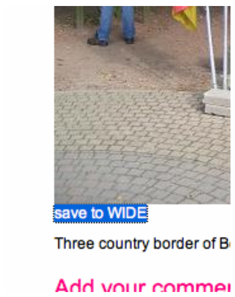
When the user interacts with Flickr, Greasemonkey runs the Flickr Adapter script that inserts enhancements to the user interface and catches user events. It shows a 'Load from WIDE' option in the 'You' menu as depicted in figure 6.4a and it also inserts 'Save to WIDE' at the bottom of pictures as shown in figure 6.4b. For pictures to be loaded from WIDE, the script calles the Flickr Adapter server that communicates with Flickr via its API. Furthermore, the script assists the Flickr Adapter server in notifying the WIDE server when the login status changes.

[1] `http://www.flickr.com`

[2] `https://addons.mozilla.org/de/firefox/addon/748`



(a) Additional Menu Item to Load Pictures from WIDE Inserted into the Flickr Menu.



(b) Inserted Hyperlink to save Pictures to WIDE.

Figure 6.4: Injected user Interface Manipulation in Flickr by the WIDE Adapter.

Load from WIDE

When the user chooses to load a picture from WIDE, the file open dialog is presented from which the user can select a picture. After the upload has been completed, he is redirected to the picture page of Flickr and can edit details, such as description, and add tags to the picture.

Save to WIDE

The download of pictures to WIDE is also done through the server. After the adapter server has downloaded the picture, it forms a transaction and request a save dialog.

6.2.2 gDoc Picture Insertion Adapter

The gDoc Picture Insertion Adapter introduces an option to insert images loaded from WIDE. It uses a Greasemonkey script and the JavaScript client framework discussed in 5.3. It integrates “From WIDE” as a third option, besides

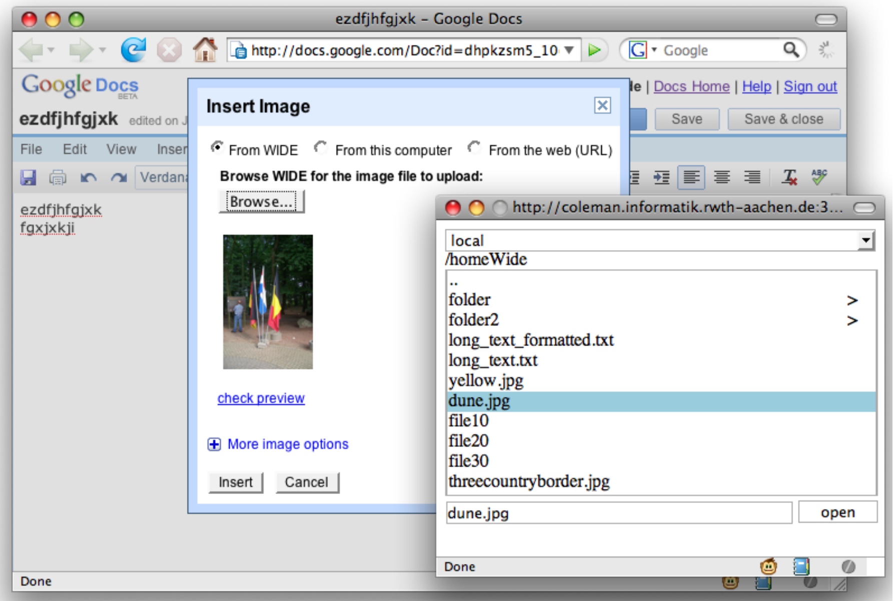


Figure 6.5: The gDoc Picture Inserter Adjust the Google Docs Insert Image menu to Support Images Loaded from WIDE

'URL' and 'upload', into the image insertion dialog. When the users clicks the "Browse..." button the "File open" dialog pops up and the user is prompted to select a file. Once the transaction is completed successfully a small preview thumbnail appears. By clicking the "Insert" button the image is inserted at the position of the cursor.

6.3 Conclusion

Although the latter adapter is rather an add-on than a full adapter that completely integrates Google Docs, it provided access to the shared information space of WIDE for the particular functionality of inserting picture. With the adapter prototypes successfully implemented the integration of existing web application's functionalities in WIDE, it demonstrates how third party developers can extend application to utilise WIDE.

Chapter 7

Validation

"How can we create what could become culturally significant systems if we demand that the system be validated before a culture is formed around it?"

—Bill Buxton et al.

In considering WIDE as a proof of concept of a distributed desktop environment, we argue that a usability evaluation is not an appropriate approach to validate the system. WIDE as an architecture and framework instead, focusses on usefulness, rather than that usability which, however, should form the focal point of the applications itself. Also, WIDE communicates mainly with the applications while user interaction is limited to authentication and authorization. WIDE's evoked working environment is in some aspects similar to or resembles offline systems. When evaluating, users might mistakenly compare it to offline systems in which performance is, in many respects, better due to the innate nature of the web and the local workstation. Hence, such evaluations are more likely to show limitations rather than evaluate the novel approach and concept itself introduced in this thesis. As, according to Greenberg and Bux-

WIDE is not very visible to the user

ton [2008],

> "the choice of evaluation methodology - if any - must arise and be appropriate for the actual problem or research question under consideration."

WIDE purposes a system for working online although such a culture of working online has yet to come despite the popularity of the web in general as it is yet not seen as place to place information and people prefer to have an offline copy (see 3.2.4).

Design rationale to validate WIDE

Taking the aforementioned arguments into account, we believe that a usability evaluation is not beneficial and therefore, we are going to validate our work by a design rationale. Hereby, we follow what has been recommended by Greenberg and Buxton and critique our design by reason our design decisions, what alternative were considered, the context of the work, the problems that can be expected in such context and what to do next.

7.1 Design Rationale

In this design rational we reason about the goal WIDE seeks for, the decisions made to approach this goal, but also what have been learned and what problem we are expecting concerning WIDE.

7.1.1 Challenge

Increased complexity through distributed information spaces

As pointed out in chapter 2 current distributed applications lack centralized online information access which results in decreased efficiency when working with them as the distribution of information across several information spaces adds complexity to information retrieval as the information spaces are not linked together. That means, e.g. that users have to search for a file in one information space after another, instead of one search over all information spaces.

This happens when users work with various applications and files can be opened by more than one application. When the user wishes to find that file, there are no cues in which application to find that file. Such reduced visibility lowers the information in the world and increases the knowledge that is required to be in the head which is harder to obtain as stated by Norman [2002]. This is particularly problematic as a file's position can change over time and learned knowledge becomes useless and furthermore, is likely to mislead the user to an outdated location.

Synchronization increases effort

Having a file at all the applications that can open it, does ease the problem but on the other hand introduces the problem of different versions when edited by an application. A solution to this could be to synchronise information which increases the effort and thus, reduces the efficiency of work.

Accessible Unified information space

Furthermore, to gain the most benefit from online systems, the framework must be able to work platform independently and with no additional application requiring setup or installation. Also, users want to keep their applications of choice due to their personal preferences and thus, keep their diversity and distribution. However, efficient working requires an unified information space, as highlighted above, and a single entry point for applications. In addition, applications must in general be able to access all data necessary even although some of the information can be considered private.

The idea of a single central external entry point would introduce a new concept to web applications but it must also provide a way to keep existing applications and provide ease of implementation for the development of new applications.

7.1.2 Solution

To resolve these requirements, a single point of information is needed that ensures trustworthy behaviour by means such as 'human-in-the-loop'. An information browser and its induced document-centric work flow will offer an alternative, more natural way of working. Furthermore, adapter applications help to integrate already developed applications and the use of standards eases the integration into new applications and makes it in general run in web browsers without any modifications. Thus, WIDE provides a unified workstation independent information space for distributed applications that, due to its usefulness, helps to increase the efficiency of work flow in an distributed online environment.

7.1.3 Decisions

WIDE separates the data from the applications. While with separated data the user could decide by placing the file which application to give access the file, in a single information space this needs to be addressed. WIDE, therefore, focusses on keeping the user in control of his documents and protect them from untrusted applications and unwanted access. Furthermore, we decided to utilize the external window management that the underlying web browser and its operating system provide. Both decision and the reasoning behind it are as follows.

Server Rendered Requests

One major design decision to support control of the user, was to render the authorization requests by the server. The advantage of such an approach is that the user interface of requests are the same, independent of the requesting application. This satisfies one of the best known usability heuristics in computer interaction, being to keep the interface consistent.

Be consistent

Shneiderman and Plaisant [2004] proposes consistent style, layout, and terminology to help the user to find necessary information more efficiently.

The style of user interface that is rendered by the WIDE server resembles dialogs of the well-known desktop. The familiarity improves, beside consistency and predictability among others, the user interface's learnability, as stated by Dix et al. [2003]. Improved learnability also increases the communicated trustworthiness, as the user has a higher perception of control (see 3.1).

higher trustworthiness and usability through server rendered requests

External Window Manager

When interacting with WIDE another aspect is how applications are displayed to the user. While webtop systems, such as EyeOS and G.ho.st, use an internal window manager to display applications inside their environment, WIDE utilizes the web browser's and its underlying operating system's window managing capabilities by opening applications in new tabs or windows.

This decision was made for several reasons. External rendering of the application maximizes the application's screen space available as it uses all the window's screen space for itself instead of sharing it with internally rendered window decorations and other applications. Furthermore, windows are handled either by the web browser or the underlying operating system which is likely to be more powerful and provides more functionality to handle them. It is also more responsive to manage the windows on a local system opposed to an online system that runs in a low-performance environment or lags due to bad connectivity.

Independent window management increases responsiveness

An external window managing also prevents WIDE from observing the user when working with an application. Also interaction with WIDE supporting application and regular web application do not differ and thus, increases further familiarity and consistency.

7.1.4 Lessons Learned

Currently, many different forms of web application populate the web and are provided to the user. This is partly because the internet makes it so easy to distribute and promote new ideas. The variety of web toolkits for web development is similarly diverse. And still, the web is changing at a high pace. However, it makes it difficult to find the best possible solution for development that benefits the users. Especially, with cross-web browser incompatibility not yet resolved and different paradigms for development (see 2.5 and 2.5). Complexity is particularly high when the development of a web application includes several different frameworks that need to work together.

7.1.5 Expected Problems

According to O'Reilly [2005], "data is the new intel inside" and control over data has become key advantage of some companies over their competitors. That means, not only the service provided by an application is of value but also the data created within the service. As WIDE proposes to separate data from the applications and to move it to the WIDE server, this is unlikely to be tolerated by web application providers.

Application/data separation is unlikely to be tolerated.

However, O'Reilly also points out that a free data movement, similar to the free software movement, will emerge.

7.1.6 Context and Vision

The motivation of this project was briefly explored in the introduction (see 1). While the potential and benefits of web applications are unambiguous, current systems to not support these. With WIDE the user can organize his data and coordinate it with the applications. Further development will allow users to change their information space provider, that are web sites providing services like WIDE offers, without loosing data and need to re-install all the applications. Information spaces will be as easily to switch as themes in current desktop systems. Furthermore, information system like WIDE will not only incorporate applications but also storage providers. They will offer a single entry point to a distributed information space in which elements can be added, changed, and removed, very similarly to how people can change hard disks of their computer and change the programs installed on the computer.

Therefore, the next steps to further improve WIDE and how to approach the envisioned idea by addressing unresolved issues are described in the future work section 8.

Chapter 8

Summary and Future Work

"The future is here. It's just not widely distributed yet."

—William Gibson

While working on this project we encountered many ideas for further improvement for which there was not sufficient time to implement them. In this chapter these ideas are further explored.

8.1 Summary

WIDE is a framework for more desktop-like working with web application. It provides a workstation independent shared single information space and further ensures trustworthy behaviour through giving the user enhanced controls over his files.

First, we analysed different approaches that aim for more convergence of the web and desktop on various levels, such as user experience and development, and explored alternative remote computing approaches, such as virtualization.

We could, in a comparison, identify a lack of support for the requirements that a system like WIDE seeks to satisfy.

To gain further insight of the users' perception and concerns when working with online information and storing information online, we conducted an online survey confirming the requirements. We also reviewed current models of online trust on which we based parts of the design for the architecture.

Then, we proposed the architecture of our WIDE framework for supporting working online and protection of information. Through several client prototypes of potential web applications interacting with the WIDE system, we refined the design, added support for Flash-based and JavaScript-based applications. As part of the implementation, we further reviewed different ways to integrate existing applications and presented our choice for integration which we also were able to successfully implement as prototypes.

In the validation we argued our evaluation methodology and stated our rationale in which based on HCI literature, common user heuristics, and the gained knowledge about trust we reasoned our design decisions that successfully enhances usefulness of web application and therefore widens their application area.

We also gave insight in other considerations and finally embedded our design in the vision WIDE seeks to approach.

8.2 Future Work

Although WIDE can be already utilised by web applications, many aspects could not be implemented due to time limitations.

- notifications for reactive transactions
- integration of external storage spaces

- exchanging of WIDE systems

Also, we expect near future changes of the restrictions on XMLHttpRequest, as the W3C workgroup web application is currently working on access control for cross-site requests[1]. Once such an access control is available cross-site JavaScript calls would be possible and make some aspects of the proposed JavaScript client framework redundant requiring it to be adjusted.

Evolution of web standards helps to simplify the WIDE framework

8.2.1 Notifications for Reactive Transactions

Current transactions are created by the application, most likely due to a user's action. However, transactions that may be required after an event that occurred without any direct interaction of the user, are not supported yet. Thus, adding notification, that communicate such events to a particular web application, would enrich interaction between information space and application. For example, a weblog application could update a post when the user adds pictures or modifies the text file of the post's folder on the WIDE storage space. Such reversed transactions tighten the relationship of server and application, similar to agents in desktop environment that observe a certain aspect.

Notification increase the level involvement of applications

8.2.2 External Storage Space Integration

Existing web application have their own storage space to host the data of the application. For example, Flickr host pictures uploaded by the user. These currently need to be migrated to WIDE to share them across applications. Such a collections of pictures however can grow over time to thousands of pictures. Moreover, meta-data added to the pictures, such as comments by other users, cannot be migrated with the data resulting in complete loss of such information.

[1] `http://www.w3.org/2008/webapps/charter/webapps-deliverables.html`

To avoid such a scenario, WIDE should in future also support storages in a similar manner it currently supports applications. A storage can be plugged to WIDE, analogous to a external hard disk drive that is plugged into the computer, and provide the data to the WIDE system. However, passing such external data through the WIDE server does not seem appropriate as transfer time is likely to double with the WIDE server in between the data and the application. Instead, for data transfer, such as when opening a file, the server should arrange a direct connection between the application and the data source for the data transfer.

External data sources should be able to plugged into WIDE

This approach promises to be a way to integrate data, that is already made available online, to WIDE.

8.2.3 Exchanging WIDE

Currently web application know of one WIDE system they belong to. However, this is a restriction and should be avoided. In future, application should instead by able to connect to multiple WIDE systems and allow the user to choose a WIDE system he wants to use. When changing from one WIDE system to another application should allow to exchange the WIDE server they are connected to. This provides smooth migration without any lost of information, similarly to exchanging a window manager in the X window system.

Appendix A

Preliminiary Study: Online Survey

A.1 General Information

1. What is your age?

   Age in years ______________________

2. Are you female or male?

   ○ female

   ○ male

3. What country are you from?

   ____

4. In which of the following fields is your current occupation?

   ○ Administration
   ○ Advertising, marketing and PR
   ○ Arts, design and crafts
   ○ Construction and property management
   ○ Counselling, social and guidance services
   ○ Education, teaching and lecturing
   ○ Engineering
   ○ Finance and management consultancy
   ○ Health care
   ○ Human resources and employment

◯ Insurance and pensions and actuarial work
◯ IT, economics, statistics and management services
◯ Law enforcement and public protection
◯ Legal services
◯ Leisure, sport and tourism
◯ Logistics and transport
◯ Manufacturing and processing
◯ Publishing, media and performing arts
◯ Sales, retail and buying
◯ Scientific services
◯ Student
◯ Other (please specify) ____________________

5. What is your highest degree of education you have obtained or, if you are a student, you are studying for now?
◯ High School
◯ Bachelor
◯ Master/Diplom
◯ Ph.D.
◯ Other (please specify) ____________________

6. Of which field is that degree?
◯ rather technical
◯ rather non-technical

7. How do you consider your skills?

	excellent	very good	good	fair	poor
General computer skills	◯	◯	◯	◯	◯
Web skills (familiarity with the web)	◯	◯	◯	◯	◯

A.2 Computer Usage

8. How many days in an average week do you use..

	never	less than once a week	1 or 2 days	3 days	4 days	5 days	6 or more days
...your computer at home?	○	○	○	○	○	○	○
...your computer at work?	○	○	○	○	○	○	○
...a different computer that was not initially intended to be used by you?	○	○	○	○	○	○	○

9. How do you rate the level of convenience when using a computer with an environment different than yours, such as different OS, different browser, mediaplayer etc.?

	completely convenient	mostly convenient	neither convenient nor inconvenient	mostly inconvenient	completely inconvenient	N/A
different OS	○	○	○	○	○	○
Different programs	○	○	○	○	○	○

A.3 Internet Usage

10. How many days in an average week do you access the internet with a computer?

	never	less than once a week	1 or 2 days	3 days	4 days	5 days	6 or more days
At home	○	○	○	○	○	○	○
At work/school	○	○	○	○	○	○	○
At other places	○	○	○	○	○	○	○

11. At what other places than at home and at your work do you access the internet?
☐ Internet cafe
☐ Library
☐ Hotel

☐ Airport/station
☐ Other (please specify) ______________________

12. How frequently do you purchase online?
◯ Never
◯ Rarely
◯ Several times a year
◯ About once a month
◯ About once a week
◯ Several times a week

13. How frequently do you purchase songs online?
◯ Never
◯ Rarely
◯ Several times a year
◯ About once a month
◯ About once a week
◯ Several times a week

14. How satisfied are you with online shopping?
◯ Completely satisfied
◯ Mostly satisfied
◯ Neither satisfied nor not satisfied
◯ Mostly not satisfied
◯ Completely not satisfied
◯ N/A

15. In the past 2 months, how many different web accounts have you used that require login?
Answer with number ______________________

16. How many different passwords do you use for these accounts?
Answer with number ______________________

17. Do you store passwords in your browser, i.e. your browser fills out the login form for you?
◯ Always
◯ Never
◯ Depends (please specify on what it depends) ____

18. How many times in the past two weeks have you forgotten a password?
Answer with number ______________________

19. How do you retrieve your password when you cannot remember your password?
    ☐ Look up the password in the registration mail
    ☐ Look it up somewhere else
    ☐ Answer the security question
    ☐ Request the application send you the/a new password by mail
    ☐ Other (please specify) ______________________

20. How many times in the past two weeks have you forgotten your username?
    Answer with number ______________________

21. How do you retrieve your user name when you cannot remember your user name?
    ☐ Look up the user name in the registration mail
    ☐ Look it up somewhere else
    ☐ Request the application send you the user name by mail
    ☐ Other (please specify) ______________________

A.4 Data on the web

22. Which of the following statements are true?
    ☐ I have a web space account to store documents on it.
    ☐ I am aware of some of my documents being online available.
    ☐ I access my email account over a web site.
    ☐ I access my instant messenger client online, i.e. via web messenger.
    ☐ I have contact information stored online.

23. How do you make documents available to others?
    ☐ I do not make documents available to others
    ☐ Email
    ☐ Upload to a web space service
    ☐ Instant messenger
    ☐ To make a document available online I do something else (please specify) ______________________

24. Why do you make the documents available online?
    ☐ I do not make documents available online
    ☐ Share documents

☐ Collaborate
☐ Backup
☐ Availability / Access
☐ Other (please specify) ____________

25. How often do you back up your documents on your computer?
◯ never
◯ rarely
◯ about once or twice a month
◯ about once or twice a week
◯ about every or every second day

26. What kind of documents do you back up?
☐ I do not back up documents
☐ Office documents
☐ Contact data
☐ Mails
☐ Schedule
☐ Media
☐ Other (please specify) ____________

27. How do you back up your documents on your computer?
☐ I do not back up documents
☐ Backup on a media (CD, DVD etc.)
☐ Backup on another hard drive
☐ Backup to a web space
☐ Other (please specify) ____________

28. Do you think it is generally a good idea to have a copy of your online documents on your own computer?
◯ Yes
◯ Depends
◯ No

29. Why do you think so? ____________

30. Do you think it is generally a good idea to have an online copy of your local documents?
◯ Yes
◯ Depends
◯ No

31. Why do you think so? ____________

A.5 Web applications

32. I am interested in what you consider a web application. Please try to give some examples of what you think web applications are.
Please enter the name and/or the web address.

    (a) ______________________________

    (b) ______________________________

    (c) ______________________________

    Some examples for web applications are

    - web-based mail services like hotmail (http://www.hotmail.com)
    - map services like google maps (http://maps.google.com)
    - e-commerce sites like amazon (http://www.amazon.com)
    - web-based office application like google docs (http://docs.google.com)
    - web-based instant messenger like meebo (http://www.meebo.com)
    - auction sites like ebay (http://www.ebay.com)
    - social networks like facebook (http://www.facebook.com)
    - blogging sites like wordpress (http://www.wordpress.com)

33. With these examples, do you know some web applications?

	I do not know any	I know some	I use one	I use more than one
Web mail service	○	○	○	○
Web instant messenger	○	○	○	○
Office application	○	○	○	○
Map service	○	○	○	○
Auction site	○	○	○	○
Online Shop	○	○	○	○

A.6 Trust

34. Do you agree with the following statements?

	Completely agree	Mostly agree	Neither disagree nor agree	Mostly disagree	Completely disagree	N/A
Web applications cannot corrupt/harm my online stored web application documents.	○	○	○	○	○	○
Web applications manipulate only online stored web application documents I have allowed to be manipulated.	○	○	○	○	○	○
Web applications can delete online stored web application documents without my knowledge.	○	○	○	○	○	○

35. Do you agree with the following statements?

	Completely agree	Mostly agree	Neither disagree nor agree	Mostly disagree	Completely disagree	N/A
Software on my computer cannot corrupt/harm my locally stored documents.	○	○	○	○	○	○
Software on my computer manipulate only local documents I have allowed to be manipulated.	○	○	○	○	○	○
Software on my computer can delete local documents without my knowledge.	○	○	○	○	○	○

36. Do you agree with the following statements?

	Completely agree	Mostly agree	Neither disagree nor agree	Mostly disagree	Completely disagree	N/A
I am worried about privacy issues concerning personal information about me.	○	○	○	○	○	○
I am worried about privacy issues concerning my documents being online.	○	○	○	○	○	○
I am worried about data loss on my hard disk	○	○	○	○	○	○
I am worried about data loss of my online documents	○	○	○	○	○	○

37. Do you agree with the following statements?

	Completely agree	Mostly agree	Neither disagree nor agree	Mostly disagree	Completely disagree	N/A
My Software is updated regularly	○	○	○	○	○	○
I do not update software if not needed	○	○	○	○	○	○
Updating software is good	○	○	○	○	○	○
Updating software distracts from my flow of work	○	○	○	○	○	○
I trust updated software more than not updated software	○	○	○	○	○	○

38. Rate your general trust in the following.
(answer with numbers, whereas 0 stands for 'no trust at all' and 10 stands for 'trust without doubts')
General trust in software on your computer ________
General trust in web applications ________________

39. Rate the level of convenience in accessing information/documents in the following situations
(answer with numbers, whereas 0 stands for 'unacceptable' and 10 stands for 'comfortable')
documents/information online (assuming you are connected to the internet) ________________
documents/information on your hard disk ________

Thank you very much for your participation!

Appendix B

TITLE OF THE SECOND APPENDIX

The different *actions* of a transaction are introduced in the following list.

- *OpenFile* opens the file that is defined in content and returns the file.
- *SaveFile* saves the file that is defined in content.
- *DeleteFile* deletes the file that is defined in content.
- *CopyFile* copies the first file defined in content to the second file defined in content and returns the destination file.
- *MoveFile* moves the first file to the second file defined in content.
- *SelectFileDialog* opens a dialog that prompts the user to select a file.
- *OpenFileDialog* opens a dialog that prompts the user to select a file to open.
- *SaveAsFileDialog* opens a dialog that prompts the user to select a file or enter new filename to save a file under a particular filename.

- *OpenDir* opens the folder defined in content and returns a listing of the contained files and folders.
- *CreateDir* creates the folder defined in content and returns the created folder.
- *DeleteDir* deletes the folder that is identified in content.
- *CopyDir* copies the first folder defined in content to the second folder defined in content and returns the destination folder.
- *MoveDir* moves the first folder defined in content to the second folder defined in content and returns the destination folder.
- *SelectDirDialog* opens a dialog that prompts the user to select a folder and returns the listing of the contained folders and files.
- *CreateDirDialog* opens a dialog that prompts the user to enter a name and select a location to create a new folder and returns the empty folder.
- *DeleteDirDialog* opens a dialog that prompts the user to enter a name or select a folder to delete that folder.

Bibliography

Amir. weblog, 2008. URL `http://vdiworks.com/wp/?p=15`.

Paul Bergevin. Thoughts on netbooks, 2008. URL `http://blogs.intel.com/technology/2008/03/thoughts_on_netbooks.php`.

Cynthia L. Corritore, Beverly Kracher, and Susan Wiedenbeck. On-line trust: concepts, evolving themes, a model. *International Journal of Human-Computer Studies*, 58(1): 737–758, 2003. URL `http://portal.acm.org/citation.cfm?id=941183&dl=ACM&coll=&CFID=15151515&CFTOKEN=6184618`.

Dianne Cyr, Carole Bonanni, John Bowes, and Joe Ilsever. Beyond trust: Website design preferences across cultures. *Journal of Global Information Management*, 13(4):24–52, 2005.

Alan Dix, Janet E. Finlay, Gregory D. Abowd, and Russell Beale. *Human-Computer Interaction (3rd Edition)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2003. ISBN 0130461091.

Saul Greenberg and Bill Buxton. Usability evaluation considered harmful (some of the time). In *CHI '08: Proceeding of the twenty-sixth annual SIGCHI conference on Human factors in computing systems*, pages 111–120, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-011-1. doi: http://doi.acm.org/10.1145/1357054.1357074.

Nicola Guarino. *Formal Ontology in Information Systems: Proceedings of the 1st International Conference June 6-8, 1998, Trento, Italy*. IOS Press, Amsterdam, The Netherlands, The Netherlands, 1998. ISBN 9051993994.

William Hampton-Sosa and Marios Koufaris. The effect of web site perceptions on initial trust in the owner company. *Int. J. Electron. Commerce*, 10(1):55–81, 2005. ISSN 1086-4415.

William Jones, Ammy J. Phuwanartnurak, Rajdeep Gill, and Harry Bruce. Don't take my folders away!: organizing personal information to get ghings done. In *CHI '05: CHI '05 extended abstracts on Human factors in computing systems*, pages 1505–1508, New York, NY, USA, 2005. ACM Press. ISBN 1595930027. doi: 10.1145/1056808.1056952. URL `http://portal.acm.org/citation.cfm?id=1056952`.

Gary Marsden and David E. Cairns. Improving the usability of the hierarchical file system. In *SAICSIT '03: Proceedings of the 2003 annual research conference of the South African institute of computer scientists and information technologists on Enablement through technology*, pages 122–129, , Republic of South Africa, 2003. South African Institute for Computer Scientists and Information Technologists. ISBN 1-58113-774-5.

Florian Moritz. Rich internet applications (ria): A convergence of user interface paradigms of web and desktop exemplified by javafx. Master's thesis, University of Applied Science Kaiserslautern, Germany, 2008. URL `http://www.flomedia.de/diploma/documents/DiplomaThesisFlorianMoritz.pdf`.

Gregory B. Newby. The necessity for information space mapping for information retrieval on the semantic web. *Information Research*, 7(4), 2002. URL `http://InformationR.net/ir/7-4/paper137.html`.

Donald A. Norman. *The design of everyday things*. Basic Books, [New York], 1. basic paperback ed., [nachdr.] edition, 2002. ISBN 0-465-06710-7.

T. O'Reilly. What is web 2.0 - design patterns and business models for the nect generation of software. 2005. URL `http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html`.

Arno Puder. A code migration framework for AJAX applications. In *DAIS '06: 6th International Conference on Dis-*

tributed Applications and Interoperable Systems, pages 138–151. LNCS, Springer, 2006.

D. Quan, K. Bakshi, D. Huynh, and D. Karger. User Interfaces for Supporting Multiple Categorization. *Proceedings of INTERACT*, pages 228–235, 2003.

Tristan Richardson, Quentin Stafford-Fraser, Kenneth R. Wood, and Andy Hopper. Virtual network computing. *IEEE Internet Computing*, 2(1):33–38, 1998. ISSN 1089-7801. doi: http://dx.doi.org/10.1109/4236.656066.

Jens Riegelsberger, M. Angela Sasse, and John D. McCarthy. The mechanics of trust: a framework for research and design. *Int. J. Hum.-Comput. Stud.*, 62(3):381–422, 2005. ISSN 1071-5819. doi: http://dx.doi.org/10.1016/j.ijhcs.2005.01.001.

Robert W. Scheifler and Jim Gettys. The x window system. *ACM Trans. Graph.*, 5(2):79–109, 1986. ISSN 0730-0301. doi: http://doi.acm.org/10.1145/22949.24053.

Ben Shneiderman and Catherine Plaisant. *Designing the User Interface: Strategies for Effective Human-Computer Interaction (4th Edition)*. Pearson Addison Wesley, 2004. ISBN 0321197860.

Mark Weiser. The computer for the twenty-first century. *Scientific American*, pages 94–110, 1991. URL `http://www.ubiq.com/hypertext/weiser/SciAmDraft3.html`.

Aaron Weiss. Webos: say goodbye to desktop applications. *netWorker*, 9(4):18–26, 2005. ISSN 1091-3556. doi: http://doi.acm.org/10.1145/1103940.1103941.

Typeset October 8, 2008