
Production Systems and Information Engineering
Volume 7 (2015), pp. 55–65

WEB BASED DESKTOP ENVIRONMENT

Imre Piller
University of Miskolc, Hungary
Department of Applied Mathematics
piller@iit.uni-miskolc.hu

Sándor Fegyverneki
University of Miskolc, Hungary
Department of Applied Mathematics
matfs@uni-miskolc.hu

[Received August 2015 and accepted November 2015]

Abstract. The web browser has become the natural platform of the graphical applications. This paper introduces a desktop environment concept which makes the development and deployment of the applications easier. The approach combines the thin client technology and locally installed servers. This hybrid solution is proper for the common use cases. The paper primarily analyzes two aspects of the desktop environment; the rendering of the desktop and the synchronization of the user data. The proposed solutions are compared with the available similar alternatives.

Keywords: web desktop, thin client, display manager, version control

1. Introduction

There are usually some misconception of the user about the importance of underlying software stacks. We can found many debates on the Internet about various operating systems and kernels. Most of them only compare the thin top layers namely the desktop environment. This recognition suggests that from the viewpoint of the user the underlying layers are irrelevant because they provide non-functional features.

The graphical user interface toolkit (near the top of the software stack) has huge impact to the user experience. The commonly used components have already standardized but their behaviour and appearance are not defined exactly. It results inconsistent look and feel when we use applications from a different kind. It is the source of some further problems. The inconsistent

terminology and graphic elements make the usage of the software harder. The various dependencies of the widget toolkit libraries cause unnecessary redundancy and sometimes make the communication between software problematic. (For example special clipboard and IPC implementations.)

The personal computing means nowadays that the user typically works with at least two computers. We can assume that the computers are different in some aspects. (The user works with more computers because all of them have a benefit for example the location, the performance, the storage size or the weight is more appropriate in a given situation.) The data are usually shared between the machines. It makes necessary to synchronize the data.

The thin client technology is able to solve the previously mentioned problems. The client is only responsible for displaying the contents and for sending the commands to the servers. The applications are provided by the server therefore the set of applications can be collected carefully by the system administrator. The synchronization is not a problem because all data is stored on the server. However there are some disadvantages also. For instance the thin client conceptually has only a limited offline functionality.

The paper concentrates on the consistent visual representation of the user interface and the problematic of the synchronization. It shows compromised solutions between the thin client and offline workstation.

2. Related Works

The term "web based" here means that the desktop application is a web application where the server can be local or remote. We can found some similar architectures in the contemporary operating systems.

The Bell Laboratories has developed an operating system for distributed servers [1]. Its main design goal is to became graphical, portable and networked. The system's programming language is the Limbo. For portability it uses machine independent bytecode and the Dis virtual machine. (The concept of the project is close to the Java Environment. Basically this project was the concurrent technology when the Java project started.)

The FirefoxOS is an operating system designed for the web [2]. It uses open standards and provides a low cost alternative for smart devices. It's layered architecture consists the following parts.

- *Gonk*: A common layer for the Linux kernel and the Hardware Abstraction Layer.
- *Gecko*: Web browser engine. Its main task is to render HTML pages.

- *Gaia*: User interface implementation written in HTML, CSS and JavaScript. It communicates over the Open Web API therefore the applications can be works in other web browser.
- *HTML5 applications*: The FirefoxOS is able to run web applications. It provides access to device drivers.

We can use applications in online or offline mode. The online mode is the same as in the case of desktop web browsers. For offline usage we need to install the application. In this case, instead of nativ application we can run the application server on the given device.

The Chrome OS (and its open source version called Chromium OS) is also a browser oriented operating system [3]. Technically this is a Linux distribution with some unique features and hardware support. Primarily it is a thin client which provides access to Google services.

There are other operating systems which are designed directly to the web [4]. Their interfaces resembles to the frequently used desktop environment. The most notable representatives are the *eyeOS*, *Glide OS* and the *Joli OS*.

As we can see the browser has become a platform in the Web 2.0 era. The application of thin client is not a trivial decision. The main differences of the architectures originated from the measure of network dependency.

3. Interaction with the User

3.1. Desktop as Primary Interface

The desktop environment is an essential part of the interaction with the user. We can see the main steps of interaction cycle in Figure 1. The input and output devices are physically available for the user. In the case of web browser the typical input devices are the keyboard and mouse, the output devices are the screen and the speaker.

We can notice the *Model-View-Controller* pattern in the cycle. The controller here responsible for registering and preprocessing the input. The application logic is stateful in most cases. The layout arrangement calculates the positions and sizes of the visual elements. In the rendering step from the tree-like component structure an image will be created.

3.2. Online and Offline Works

The online services have many advantages. For instance, they require only a minimal standard installation because most of the applications are installed on the remote server.

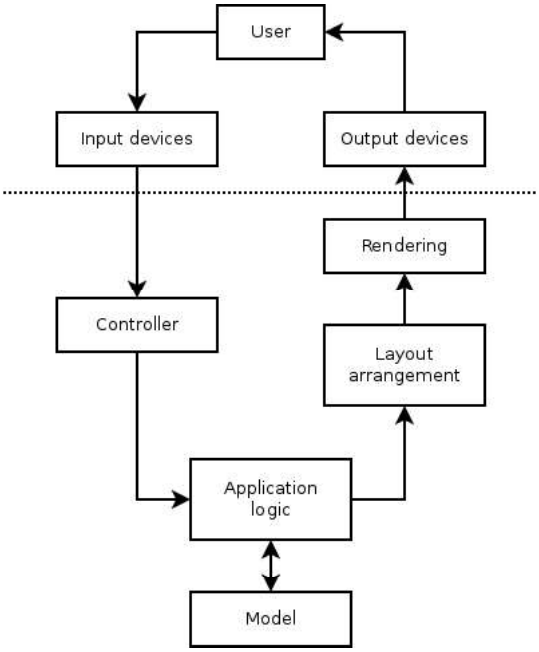

Figure 1. The main steps of the interaction.

The platform of this type of software is the web browser. It results that the applications are portable. We can use the software on any device which can run web browser.

The reliable and fast network connection is a natural requirement in information technology. In the case of online services these requirements are essential. When the network connection is slower or we have no connection at all these online services are unable to work.

The thin client technology often assumes that the client device has only a limited resources and cannot run the application locally. The computational power of smart devices is enough for managing file systems and databases. The thin clients are still preferred because intensive calculations requires more energy and the battery becomes the bottleneck.

4. Layered Architectures

4.1. Client types

We can distinguish different client types as we can see on Table 1. The clients are classified by their components. The literature use the *light*, *rich*

and *heavy* names as synonyms in general [5]. Here these names reflect to its complexity. In the following paragraphs their advantages and disadvantages will be summarized.

type	Model	App. L.	Controller	Layout Arr.	Rendering
thin					
light					×
rich			×	×	×
heavy		×	×	×	×
desktop	×	×	×	×	×

Table 1. The types of the clients.

thin client: The client only display the previously rendered image. It requires only a minimal amount of resources on the client side but assumes that the network is able to stream the view of the desktop real-time. This solution is very flexible on the client side but the rendering requires graphical capabilities on the server. For example the VNC client and remote desktops belong to this category.

light client: In this case the server send document object, vector graphics or only the difference of the desktop image from the previous one. Most of the web pages can be considered as light client because the browser waits the response from the server and render graphical elements from the HTML source code.

rich client: The web browser usually able to manage the user input events and calculate the layout arrangement. In web environment the rendering provided by the HTML rendering engine. As *rich* suggests the *Rich Internet Application*s are in this type.

heavy client: The application runs in the client. The persistent storage still on the server side. This construction is beneficial for example the case of calculation intensive applications. The drawback is that the client must have enough resource for running the application but unable to work offline. The clients of online games are heavy clients in this sense.

desktop client: The desktop client means that all components and data are available for offline working. The *client* in the name is intentional. The model is available on the machine but the data are shared or the application partially depends on online services. It can be for instance the file sharing or mail client.

4.2. Function Locations

The client-server model is a simple distributed architecture. The reliable network connections give a choice to locate some functionality either the client or the server side. We have possibility to separate some elements of the server side to remote machines. On Figure 2 we can see the two preferred construction of the layered architecture. At this step we have already assume that the user works with web browser on the client side.

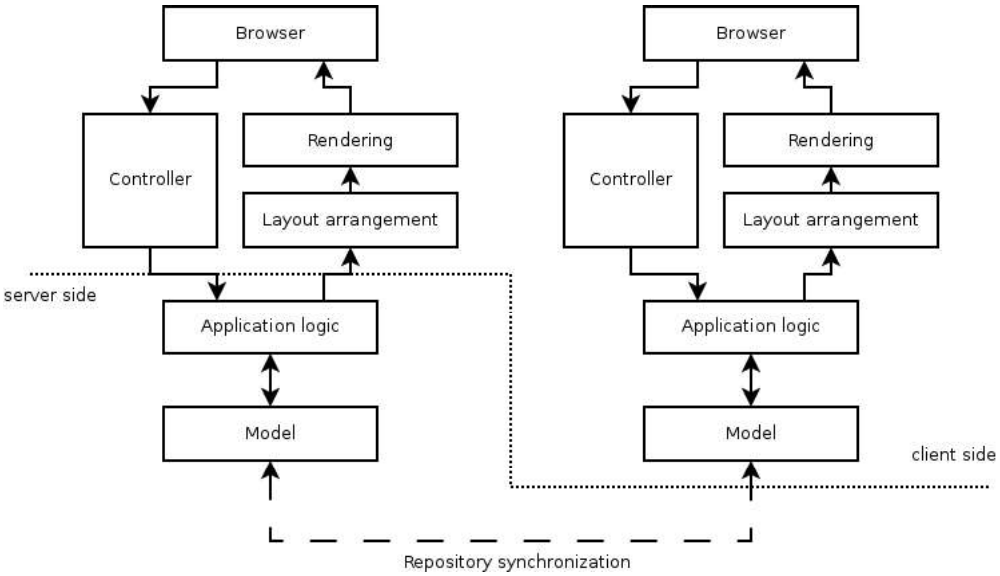

Figure 2. The preferred layered architectures.

The two configuration nearly the same because they use the same interfaces. The difference between them is the location of the application and the model. The left is a rich and the right is a desktop client. We prefer these types because in the case of desktop applications the hardware requirement of the application engine and the model are similar. (In many applications the logic and the model are not separated.)

At the bottom of the figure we can see that the models are connected and able to synchronize the data. The model basically has two parts: *file storage* and *metadata*.

The file is a unit of data. We use files from many reasons, for example,

- the data is unstructured,
- we cannot edit with online editor,

- we are trying to be compatible with other systems,
- the file is large and we cannot store it in the database,
- the filesystem is common, available and supported.

The metadata stores structured data. It provides consistency checking and flexible queries. The type of the database is not important at this point because the choice is highly depends on the given application.

From the user's viewpoint the application is a web application. Many rich clients can use the same web application while in the case of desktop client it presumably dedicated to the user. Therefore the web application is not necessarily installed by the user but enables to install locally for offline work.

The controller, layout arrangement and rendering can be implemented in the browser. It is usually developed in JavaScript as one of the most dominant programming language of the Internet. (In the web browser we have only some other alternatives, because the support of other programming languages is rare.)

5. Optimal selection of configuration

The proposed architecture makes possible to choose between configurations. The decision is based on the following factors.

- *Network availability, and reliability*: We can configure the application for online or offline work. Most of the applications are sensitive to network problems. The periodically updated local repository helps to smooth the differences between the real time online work and offline workstation-like usage.
- *Local resources*: We need to examine the impact of locally installed applications and stored shared files. In general, the synchronization of all files from the repository is not an appropriate solution.

The length of the updating process in the case of distributed configuration based on the speed of network and from the amount of data. It does not affect the behaviour of the application after the synchronization is ready.

We can consider the model as the part of the application from the aspect of performance. Therefore we need to compare the cost of local configuration $(t_l + c_l)$ and the cost of remote configuration $(t_r + c_r)$, where

- t_l, t_r: average response time for locally and remotely installed application,
- c_l, c_r: average time cost of local and remote data exchange.

The $(t_r + c_r) - (t_l + c_l)$ value shows the average response time advantage of the locally installed application.

6. Desktop Interface

The proposed concept help us to create application to this distributed environment. In the following sections the part of the toolkit will be presented.

6.1. User input and the controller

The considered desktop environment uses the keyboard and the mouse as the input devices. It defines the following events.

- *key down/up*: Uses keycode for identify the key.
- *mouse move*: It contains the position (x, y) of the mouse cursor.
- *mouse down/up*: It identifies the mouse buttons similarly to the keys.

The controller is responsible for creating or abtract the event. For instance instead of sending the raw input to the application it can send request for starting or stopping a process, clear the input field or other context dependent commands. It has an internal state but it is limited and not overlapped with the application logic.

6.2. Layout arrangement

The application only sends responses to the requests. In the layout arrangement step the following informations are combined:

- the data which should be presented for the user,
- the rules of the layout arrangement,
- the previous arrangement.

It is hard to divide the functionality of layout arrangement and rendering. As a guideline we should consider only with the structure of the user interface and leave tasks related to the appearance for the rendering.

6.3. Display renderer

The mentioned simple interface can be implemented on any hardware or software platform. The trends suggest that the most appropriate tool nowadays is the HTML5 [6]. It is able to listen the keyboard and mouse events and to display the rendered image.

We can trust the design decesions of HTML5 canvas. It inherits the primitives of widely used graphical toolkits. When we build our display manager

on the top of this primitives we can assume that it will have hardware support in the future.

The main advantage of HTML5 is the web browser support. The HTML5 canvas is available in all major browser. It has no standardized widget toolkit yet. For a higher level functionality (for example windowing) it is necessary to use a third party Javascript library or creating a new one. The portability of the applications remains but the consistent look and feel is not guaranteed.

7. Repositories and synchronization

The web application is responsible for the synchronization. It must make available the current data automatically [7]. For synchronizing the data the web applications require the change sets from other web applications instances. From this aspect we can consider the application as a version control system. The database behind the desktop environment is practically a repository [8].

We must distinguish two cases in synchronization. At the simple case the user works with only one browser at the same time. It is a plausible assumption, because the environment is designed for multitasking and for supporting the all required feature. As an advanced scenario, the user is able to work with many desktops parallel. In this case it is hard to avoid collisions and inconsistent states. The login interface helps us to restrict the user to work with a single instance of the desktop.

7.1. Log based synchronization

The structure distinguish the data and the metadata. Ideally one database should enough but the management of binary files make necessary to store the files without knowing its internal format. It require different approach for synchronizing the metadata and the file storage part.

The log based synchronization is beneficial, because

- the calculation of deltas is straightforward since all changes are discrete events with timestamp,
- makes merging easier; the problematic steps can be detected and corrected,
- helps to follow the growth of the repository.

We can group the changes into named sets of changes, but this is optional. The commit messages help to make the modifications easier to follow but the conflicts are not necessary separated by revisions.

7.2. Repository types

The local repository often contains only the subset of the managed data. We can create repository for working and for backup. We can define further types also by considering the following properties of the files.

- *file size*: We expect that the synchronization of the repository will be fast and the size of local copy is small as possible. We can avoid large local storage size via on-demand synchronization.
- *last access or modification date*: Presumably we will work on a set of files on a given period. From this reason we can remove the old files from the local repository.
- *importance*: This is a subjective factor. Some cases it does not turned out from the access or modification dates.

As we can see, the repository types are the results of adaptation to the user behaviour. It is a complex optimization problem which try to achieve small repository size and prompt access at the same time.

8. Conclusion

The paper considers two difficulties in providing convenient desktop environment, namely the graphical representation and the repository synchronization. The preferred solution is based on contemporary web technology (standards, software, tools and methodology). It has showed a possible categorization of client types according to the functions of the client. We prefer the rich and desktop clients because the comparisons show that these configurations are able to adapt to the available hardware and software environment properly. These represent a compromise in the aspect of low resource requirement and the possibility of offline work.

Acknowledgements

The presented research work was partially supported by the grant TÁMOP-4.2.2.B-15/1/KONV-2015-0003.

REFERENCES

[1] S. Dorward, R. Pike, D. L. Presotto, D. M. Ritchie, H. Trickey, P. Winterbottom: *The Inferno Operating System*, Bell Labs Technical Journal, Vol. 2, No. 1, Winter 1997, pp. 5-18.

[2] Network, Mozilla Developer, and Individual Contributors. "*Firefox OS architecture.*" (2014).

[3] Azad, S. "*Chrome OS and System Architecture.* Retrieved November 3, 2014." (2012).

[4] Lawton, George. "*Moving the OS to the Web.*" Computer 3 (2008): 16-19.

[5] Satyanarayanan, Mahadev. "*Pervasive computing: Vision and challenges.*" Personal Communications, IEEE 8.4 (2001): 10-17.

[6] W3C, *HTML5 Recommendation*, `http://www.w3.org/TR/html5/`, 2014.

[7] Karlson, Amy K., Greg Smith, and Bongshin Lee. "*Which version is this?: improving the desktop experience within a copy-aware computing ecosystem.*" Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. ACM, 2011.

[8] Constantin, Camelia, et al. "*A desktop interface over distributed document repositories.*" Proceedings of the 15th International Conference on Extending Database Technology. ACM, 2012.