On- and Off-Line User Interfaces for Collaborative Cloud Services

Wolfgang Stuerzlinger
York University, Dept of Computer Science & Engineering
4700 Keele Street
Toronto, Canada
http://www.cse.yorku.ca/~wolfgang

Abstract
We describe a vision for user interfaces of cloud-based systems that permit seamless collaboration and provide also on- and off-line access to data. All individual components of this vision are currently available in various systems, but the sum of the components will satisfy user needs much more comprehensively.

Keywords
Cloud services, synchronization, collaboration

ACM Classification Keywords
H.5.2. User Interfaces, C.2.4 Distributed Systems.

General Terms
Cloud services, synchronization, collaboration

Introduction
Cloud-based systems offer a wide variety of services to the user, such as publishing pictures, checking email, collaborative scheduling, editing an encyclopedia, storing information, tracking issues in a large open source project, and sharing files. One notable advantage of such systems is that users can access them at (almost) any time, on (almost) all platforms, and at (almost) all locations. Many of these services also permit collaboration with other users.

This document discusses a vision of better user interfaces for cloud-based systems[1]. It is based on user needs that are not satisfied in many current services. In particular, we focus on collaboration facilities combined with support for off-line interaction, as both of these issues are highly interlinked, both from the perspective of the user as well as in terms of technical issues. We discuss our vision for collaborative facilities and off-line interaction in the following sections.

A small-scale example that realizes this vision reasonably well is the calendaring system in iPhones. A highly relevant, large-scale example is the collaborative editing support in Google Documents. This feature was received very positively. With the eagerly awaited re-introduction of off-line editing this will fully realize the vision presented here in a cloud-based system.

The technical issues associated with implementing the methods discussed here are considered outside the scope of this document. It suffices to point out that practically all mentioned techniques are already available in current systems, except that very few implement all of them. Also, we do not distinguish between traditional applications and those based on browser platforms. For further technical details, we point the reader to the documentation of HTML5, Google Documents blog, smart-phone application guidelines, and the research literature.

Improved Collaboration in Cloud Services

Collaborative systems often provide facilities for easy information sharing, but also need to support awareness and coordination, both of the past, present, and the future. The World Wide Web (WWW) has made it easy to share information with others, just by publishing information as a Web page. A lot of systems allow users to edit the information on a Web page, too.

However, information sharing is only a small part of supporting collaboration. Tracking and coordination of changes is critical for any collaborative system. This is well known from research into Computer Supported Collaborative Work (CSCW) and we refer the reader to this body of literature for more information.

Consider the example of Wikipedia, which features two simple mechanisms for coordinating the past and the future. Via the "History" tab everybody can see what was edited on a page and who did it. For coordinating the future, there are discussion facilities. However, there is no direct support for coordinating concurrent activities. One desirable improvement to the History tab in Wikipedia would be to provide a view similar to the "Track Changes" feature in Word, which shows the changes directly in the context of a page. The value to the user is that changes can be seen directly in the text, which makes it much easier to understand a particular edit. Yet and with an increasing amount of changes, this feature does not scale up. One solution is to permit the explicit creation of versions after a set of changes has been accepted. This then necessitates user interface facilities that permit users to access old versions. Various on-line file sharing services, such as Dropbox, and editing services, such as Google Docs, already support versions in a variety of ways.

However, versioning is not just a mechanism to detect who did what, it also aids in comparisons between past

[1] This document presents an updated version of content presented at a workshop at IBM's CASCON 2007.

versions, permits reviewing, and helps undo accidental changes. Given that users rely on the cloud to keep their data safe (and in fact cannot be expected to keep a local copy as it is often unnecessarily hard to export data out of current cloud-based systems!), frequent and reliable backups are a must for any cloud-based system. Overlaying versions on top of that is not difficult. Another alternative to handling versions is to design new mechanism to visualize large amounts of changes in documents containing text, such as [1], or diagrams, such as [2,3]. However, these methods often require very high bandwidth, and are hence likely not yet directly applicable to cloud-based systems.

Coordination of the future is often best served by providing some form of discussion forum, usually with granularity proportional to some reasonable "chunk" of information, such as a Wikipedia page.

The coordination of the present, i.e. support for awareness of (potentially conflicting) changes by others is currently only rarely available in cloud-based systems. Yet, awareness of the actions of others helps to avoid conflicts. Google Documents, which permits simultaneous collaborative editing is a notable example. Another well-known user interface that enhances awareness of the actions of others is messaging systems, where "<user> is typing …" appears, whenever another user starts entering a message. This last idea can be expanded significantly. For example, if a collaborative system displays messages such as "Jill is currently adding a calendar entry on Oct 24 at 2pm", "John is currently adding content to his Facebook page", or "Joan started editing the Future plans section on the current page at 3pm", it becomes much easier to coordinate actions with other users.

A Note on Privacy

All data in the local cache and during transmission should be adequately encrypted to preserve privacy. However, also motivated by privacy reasons it is reasonable to permit the permanent deletion of content in the presence of versions. However, and to guard against accidental deletion of content, this should involve a multi-modal two-part process, such as a "Captcha" type of verification followed by an e-mail with a verification link.

Off-line Cloud-based Services

One of the main deficiencies of most current cloud-based services is that they do not provide for off-line operation. There are many situations where no network connectivity exists and this will not change substantially in the foreseeable future. Consider inside locations such elevators, subterranean tunnels, and rooms deep inside buildings. Outside this can happen in network blind spots, e.g. between buildings, in the mountains, in large nature parks, in airplanes and trains, and generally in areas where few people live. One may also not be able to access data services in foreign locations. Another issue is that the data fee structure of some providers is too expensive for casual use. Finally, one may not want to use available networks in insecure environments, such as a competitor's building.

Another perspective is that off-line operation is still one of the major distinguishing features between traditional applications and cloud-based services. While users like cloud-based services in general, they want to be able to access their data even when there is no network connection is available. Adding this feature will accelerate the gradual migration towards clouds.

Yet another issue is that more and more people are using mobile computing devices such as smartphones or multi-touch tablets. Some users are using their mobile devices now more frequently than their desktops. On such platforms the underlying file system is often hidden from users, which notably simplifies the user interface. This makes it infeasible to apply conventional file-based solutions. Consequently, people come up with awkward workarounds, such as using e-mail attachments as a form of temporary data storage on these platforms.

The main problem in this context is that the typical client software for cloud-based services does not offer any form of persistency. However, and especially on mobile devices, it is currently not feasible to keep a complete local copy of all data and databases that a user can access. Hence, off-line cloud-based systems should employ a form of cache, which keeps a local copy of recently used data, sometimes complemented with additional "pushed" important content.

The general idea with such caches is that the application first checks if the network is available. If it is, everything proceeds as usual, but data transferred is cached. If the network is not available, viewing of cached data is possible. Then, once the user connects again to the network, the cache is updated with the most current version of the data from the cloud.

However, this limits the user to pure viewing, which is seen as unnecessarily restrictive. Also, users often need to enter important information while they are away from their usual network. Hence, there needs to be support for off-line additions, changes and deletions.

A few important applications already provide for off-line addition or changes to data. Email on smartphones is one good example, as practically every smartphone platform caches newly created e-mails locally until network connectivity is restored, when those e-mails are sent. Another example is a calendar client, which caches a partial copy of the schedule (usually only a limited amount of the past and all future events). Once network connectivity is re-established, the changes to the schedule are uploaded to the main database and recent additions by others are downloaded. However, there is always the potential for conflicts in these sets of changes, which is why some form of data synchronization is necessary for such systems.

Finally, note that many users cannot predict reliably in advance when they are going to be off-line. Hence, it is appropriate to always cache information. Naturally, the amount that can be cached is limited by the storage available locally. However, it is generally not considered appropriate if a single application "takes over" a mobile device completely by consuming all free space.

Data Synchronization & Conflict Resolution

Whenever data is synchronized between devices where changes can occur concurrently, there is a potential for conflicts, which have to be resolved. If only one side updated the information since the last synchronization, change propagation is sufficient. However, with the potential for multiple updates on different sides, conflicts are possible. Thus, conflict resolution is a necessary feature for cloud-based services. Such conflicts can happen at different time scales in clouds. Consider e.g. a system that allows real-time collaboration, where multiple users can add entries or change a particular piece of information. Then add users that work off-line and upload their changes in batches. While different technical mechanisms may be necessary to handle both types of changes, it is best if the user interface does not distinguish between them. However, due to the asynchronous nature of cloud-based services, it is anyways essential that the system can handle updates that occur on different time scales.

Yet, the type of synchronization required depends also on the granularity of the database and the type of data. E.g. calendar or address book entries may be atomic, which means that time stamping of changes is sufficient to identify who changed a particular entry last. This kind of synchronization can even be applied at levels finer than a single database entry. Consider e.g. two changes to an address book entry, one of which updates only the title of the person, whereas the other changes the address for that person. In this situation there is no conflict between those changes.

Other types of data require different synchronization strategies. E.g. a text document typically requires an identification of who changed what part when, typically at the character or word level. This then requires each text document having revision information directly embedded into the document. Similarly, collaborative spreadsheets require tracking of changes to each cell. If there are strong dependencies between changes, e.g. linked updates to multiple accounts in money transfers, transactions are the appropriate mechanism.

User Interfaces for Synchronization

Many users see a user interface for data synchronization service as a hindrance. After all, they are often not aware that other people may have changed data independently. One solution is to provide some form of awareness for changes by others. However, this should not take the form of an email for each single change at the word level.

The ideal user interface for synchronization is that there is some form of indicator whenever synchronization is occurring, while the user can still use the system to view the information and even add to and update it, as necessary. This synchronization should always happen whenever a network connection is available at start up of the client and then at appropriate times, such as whenever the user has made a change to a record, or entered a non-trivial amount of text. Periodic synchronization, similar to the auto-save functionality in desktop applications, is often also desirable.

Note that network problems may occur during any synchronization attempt. Some feedback for the corresponding aborted synchronization attempt needs to be given to the user. It is also noteworthy that additional changes can occur (on multiple sides) while synchronization is running. However, the best user interface is still a simple status indicator, while the system maintains the integrity of the local cache so that future synchronization attempts can succeed.

User Interfaces for Conflict Resolution

Whenever conflicts occur, the best strategy is to resolve such conflicts as automatically as possible. The main reason for this is that change merging interfaces are often very attention demanding. Hence, a dialog asking the user which of two versions is the right one makes only sense whenever the user is in a state where they have the time to handle such requests. Also, the small screen sizes of mobile platforms generally make such interfaces (too) challenging.

Hence, the general guideline is that user interfaces for change resolution should be avoided as far as possible by making data synchronization as automatic as feasible. In cases where this is not possible, such as a long text document, an appropriate solution is to add versioning information to the document that effectively stores all changes and permits the user to see the history of changes and act on them directly. This again makes the user interface for change resolution as seamless as possible. Interestingly enough, the capability to access old versions is already an important requirement for collaborative systems. Given that many cloud-based services are targeted at collaboration, these history mechanisms can also be used to handle data synchronization.

Handling the Evolution of Services

One of the main features of cloud-based services is that the service can be updated at any given time. While programmers have realized the downsides of this due to the potential of breaking things globally and thus leaving all users without access to their data, it is nevertheless a great feature as it permits for rapid deployment of new or updated features. When clients may go off-line for substantial amounts of time, it becomes essential that the service can handle "old" clients and their "old" data caches correctly.

External Integration

Import of data is a mechanism that helps users migrate to the new system and often considered essential. However, history has shown that users strongly prefer not to have their data be locked in completely into a technical system. One sign of this is that the plethora of "hacks" and "workarounds" for systems that do not feature reasonably facilities for data export, yet have outlived their usefulness, do not support current user devices well, or are not competitive anymore.

Hence, bi-directional data exchange of data can be considered a strongly desirable feature for any cloud-based system. Provided that the cloud-based system fulfills user needs well, users will value the option to be able to transition, but will not leave due to the effort required in any migration of data. Hence, support for data export serves as a reassurance factor for users, which often leads to greater adoption rates. Interestingly enough, data export is also a prerequisite for users being able to keep a backup of all data stored in the cloud.

Conclusion

Many parts of the vision about collaboration via cloud-based systems and the support for both on-line and off-line access described here already exist in various implementations systems or have been demonstrated in the past. Here, we have described how such systems should work from the view of end users. The challenge for the technical community is to realize this vision in a easy-to-use manner.

References

[1] F. Chevalier, P. Dragicevic, A. Bezerianos, J.D. Fekete, Using Text Animated Transitions to Support Navigation in Document Histories, *CHI 2010*, 683-692.

[2] D. Dadgari, W. Stuerzlinger, Novel User Interfaces for Diagram Versioning and Differencing, *British HCI 2010*.

[3] L. Zaman, W. Stuerzlinger, The Effect of Animation, Dual View, Difference Layers and Relative Re-Layout in Hierarchical Diagram Differencing, in submission.