Symp.ly Real-Time Collaboration System

David W. Jia
Stanford University
djia@stanford.edu

ABSTRACT
In this paper, we describe a novel real-time collaboration system, Symp.ly that can be utilized both as a group collaboration tool, and as an individual task management tool. Existing productivity software has two main problems. First, they are either only effective for high level milestone and goal setting, or low level individual tasks; no collaboration tool solves both problems. Second, they generally do not provide real-time collaborative syncing with negligible time delay. Symp.ly solves both of these problems by providing an infinitely nestable hierarchical organizational structure, and fully integrated real-time syncing for both text and data structure. We also conducted an empirical study that demonstrated the viability and effectiveness of Symp.ly. We highlight the important features that make a collaboration tool effective in group-based work environments.

Author Keywords
Real-time collaboration; task management; hierarchical abstraction; operation transform.

INTRODUCTION AND MOTIVATION

As collaboration tools become more crucial for the growing amount of collaborative work, task management tools have also become increasingly important for workers to stay focused and efficient. However, few tools exist that allow for both individual task management and project collaboration to take place simultaneously.

There are two main obstacles that project collaboration tools face: a need for hierarchical abstraction, and a need for real-time synchronization.

Hierarchical Abstraction

Collaboration applications have an inherent need for hierarchical abstraction. They must be able to satisfy task management at the individual level as well as collaboration needs at the group level. This is because any collaboration must inherently start at the individual level. Higher level project managers or group leaders break down tasks into smaller subtasks, which are subsequently broken down into even smaller sub-subtasks, until finally they are able to be completed by an individual and check off from the list of all the tasks for the project. This means that for a collaboration tool to be effective and efficient, it must also function as a task management tool. However, there is a "gap" between collaboration and task management applications because of their inability to resolve the user interface to allow both organization of high level information and structure of low level tasks.

CHI'12, May 5–10, 2012, Austin, Texas, USA.

Real-Time Synchronization

Another important aspect of collaboration missing in the majority of existing tools is the ability to access and modify data in real-time. In other words, these tools lack data concurrency. In many cases, collaboration requires a sense of immediacy. Because of this, group members need to be able to access the same data and see the changes made by other team members as they happen [13] [23]. Efficient collaboration will inherently require immediate and real-time syncing of shared resources [19]. The necessity of real-time technologies in collaboration tools is very much analogous to the emergence of real-time communication tools like text and video chat and their synergistic relationship with over-time tools like email. As such, modern collaboration tools (and by extension, task management tools) must support real-time collaboration.

Anatomy of a Project Collaboration Application

With this in mind, we propose that an effective collaboration tool should satisfy at minimum the following criteria:

1. It should work well on the individual level as a task management tool.
2. It should have collaborative features that allow concurrent access to and sharing of data by multiple users.
3. Its interface should be both simple and not obscure the day-to-day usage, but also support advanced features "under the hood" for power users. fcall
4. It should be able to handle tasks of any size from small individual tasks to large team-based milestones with ease and through an interface that is intuitive and non-obtrusive to the user.
5. It should be in real-time to facilitate collaboration between remote teams.

While existing collaboration and task management software may satisfy one or more of the above features, none satisfies all of them. This is a primary motivation for the creation of Symp.ly. Of course there are other criteria that can be considered, such as security, serialization, interoperability between multiple devices and platforms such as iOS, Android, other

cell phones, etc, and interoperability with other supporting applications including spreadsheets and video editing. But the main purpose of this paper is to demonstrate and provide a conceptual understanding of the most crucial and pertinent aspects of an effective collaboration tool. Whereas other criteria may also be important for specific use cases, they are either secondary to the topic of this paper or can be easily included by extending Symp.ly.

For example, we can allow support for other application like spreadsheets and videos by extending Symp.ly to allow file attachments at any node. We can then implement real-time features for these applications. As such, security and other features will not be discussed in detail in this paper and can be included in future work.

BACKGROUND

In reviewing previous work, we focus on applications that satisfy a certain baseline requirement, that highlight one or more differentiating characteristics. We do not discuss other task management and collaboration tools that are similar to, or near copies of the applications discussed in this section.

Commercial Efforts

Remember The Milk (RTM)

Remembering The Milk is a classic web-based task and time management tool, and one of the earliest of its kind. Overall, many users use it as a low level task management tool for the individual. It has consistently being rated as one of the best to-do list applications [21]. RTM includes many single-user features, such as adding various fields to tasks, including location, due date, and tags.

Much of RTM's drawbacks as a collaboration and task management application lie in its lack of support for sharing and structural organization by not having subtasks. Also, its interface is poorly made for efficiently managing tasks. The user must click through multiple menus for certain actions such as reordering tasks. It has minimal sharing features, which are also not real-time. RTM is very difficult to use with groups, and most people use RTM only for simple daily todo-lists.

Basecamp

Like RTM, Basecamp is one of the earliest commercially available web-based collaboration tools [10]. Basecamp's approach to modulating complexity is to separate functions into different applications. For example, within a Basecamp project, groups can post messages, create todo lists, set milestones, and upload files. Each of these functionalities is separated out into its own application that exists on different tabs of a project's user interface [10].

At the time of its inception, Basecamp was the best alternative to using email or paying large licensing fees for Microsoft Sharepoint or other enterprise level collaboration applications. One of Basecamp's main contributions to the collaboration space is its availability to small to medium size businesses. From working on Basecamp, the original developers were able to create the Ruby on Rails framework, which contributed significantly to the rapid development of agile applications [7].

However, Basecamp's approach of separating different functionalities into different applications is ineffective in project collaboration. By compartmentalizing data into silos, the inherent hierarchical nature of data and information is lost. There is no easy way to link informations from one application to another. For example, often times a todo list will be related to a milestone, a message, or a file, but since the todo lists exist in a different application from the milestones, messages, or files application, linking a todo list to another piece of data or resource is difficult at best. Also, asides from its chatroom application, Basecamp has no real-time collaborative functionality.

Asana

Asana is a group collaboration tool for small to medium size companies. Unlike Basecamp, Asana tries to take a more integrated approach to collaboration. At its core, Asana is a task management tool with sharing and collaboration capabilities.

Asana's main contribution to the field of collaboration software is its smooth interface and its promise to create a framework called LunaScript for easily creating web-based real-time applications [3]. LunaScript was publicized by Asana near the company's conception but was never released. The purported framework would have allowed developers to bind server-side objects to front-end views without bootstrapping middlelayer logic [3].

Unlike the tools discussed so far, Asana is in semi-real-time, which means that users do not need to refresh the application to see changes. However, the time it takes for operation to sync between clients is not negligible. For Asana, the client-to-client round trip time is between 5 to 8 seconds, whereas negligible sync times are in the 200 to 500 milliseconds range.

Another major setback is that Asana only supports two levels of subtasks. Although Asana has a very clean and intuitive user interface, the lack of multi-level subtasks makes it difficult for users to organize hierarchical information. Because of its lack of hierarchy and true real-time syncing, Asana is not an ideal candidate for effective collaboration.

Google Docs

Perhaps the most commercially successful real-time collaboration tool, Google Docs provides a suite of applications that allows users to share and work on documents, spreadsheets, presentations, and other files together on a browser.

In its recent update, Google Docs now exhibits a very powerful real-time functionality that allows users to see changes made by other users within hundreds of milliseconds, making the round-trip time negligible to the user experience. Google was able to do this by leveraging the technology from acquisition of technology company Etherpad [2]. Etherpad used a Scala based backend engine and created a non-blocking webserver to allow multiple clients to concurrently access a single resource. Etherpad was one of the first to leverage this technology. With the acquisition of Etherpad, Google was able to integrate this real-time technology to Google Docs, which significantly increased its syncing speed from several seconds

to less than 200 milliseconds, which is negligible for most use cases [2].

Because of this, one of the main contributions of Google Docs to the field of collaboration software is its push to make collaboration real-time. Google Docs' main drawback is its lack of task management capabilities.

Research Efforts

Since the 1980s, much research has been done on computer-supported cooperative work (CSCW) and groupware, which are software which help groups work together, which became a precursor to today's collaboration software. Many of these software applications focus on creating a shared workspace where users can work together concurrently in a non-conflicting way.

In this section we discuss the previous research efforts from earlier CSCW research to more modern efforts such as Google Wave.

TeamRoom

One example of an early research effort in groupware is TeamRoom, a persistent and shared workspace environment that allow multiple users to work together in real-time [20].

Users can create "rooms" to which other users can login. When logged in to a room, a user can add documents to the room, such as images, post-it notes, outliners, graphs, or a custom applet. When these share resources are added, other users are also able to see them. TeamRoom also has a chat features at the bottom of the screen in which users can communicate.

This early effort in real-time collaboration does not allow concurrent editing of text or structure. TeamRoom uses a lock-based concurrency architecture where users must finish editing a note before the changes are displayed on the other client's end, and only one user can edit a document at one time. Nevertheless, early efforts like TeamRoom was able to demonstrate the feasibility of real-time technologies in collaboration software [20].

DOLPHIN

Like TeamRoom, DOLPHIN [22] allows concurrent usage of a shared workspace through non-conflicting operations such as free pen-based drawings and adding text snippets to a shared interface. DOLPHIN and early collaborative applications were made possible because they did not have to support conflict-resolution. In other words, the operations they support were never causally dependent. Even though they may be considered concurrent in time, the operations that each client supported, by construction, never conflicted with one another. So regardless of what one client does, another client can perform its own operations concurrently without having to worry about the context of these operations in the scope of any other clients. This means that the server did not need to handle conflict resolution.

Google Wave

Google Wave is a real-time collaboration application originally developed to replace the email protocol [1]. For various reasons, it has since been discontinued by Google and reorganized into an open-source project managed by Apache.

Because email was one of the earliest collaboration tools which remains prevalent so today, Google Wave sought to replace email by providing a very similar list-message interface that integrated Google Docs-like real-time collaborative features. Users can create and send messages, called "Waves," to each other, each of which mimics a free-form Google Docs document, allowing users to edit text in real-time and add gadgets such as user polls or mini-games.

Like Google Docs, Google Wave has a robust real-time interface. Its internal architecture is based on a causal, convergence, and an intention preserving operation transform architecture much like a generalized n-client version of the Jupiter Collaboration System [1]. Google Wave also supports markup in its text editing.

However, like Google Docs Google Wave lacks the nestable and hierarchical structure necessary for effective task management and collaboration. It uses essentially the same linear list and free-form document structure that Google Docs utilizes.

SYMP.LY

Symp.ly take on collaboration and task management from a new perspective. Symp.ly is a real-time collaboration application that utilizes nestable, hierarchical structured data that gives users a seamless interface of collaboration on all hierarchal levels, from high-level milestone creation to low level task management.

Functionally, Symp.ly considers collaboration and task management holistically. This is in contrast to other application, which consider collaboration on a feature-to-feature basis. Considering collaboration from a feature-to-feature perspective is flawed and often times leads to complex interfaces and obscured usability. While these application may satisfy each feature when taken separately, they make little practical sense as a whole. This is because different features lead to contradictory interfaces, designs, and functionalities. Considering features separately also does not effectively capture relationships between data. This is the case with many enterprise software such as Microsoft Sharepoint.

Symp.ly takes a different approach to design by considering collaboration and task management together and holistically. Symp.ly first considers how users think, which is inherently hierarchically. From the way we acquire and retain information, to our logic and reasoning process, to our way of reducing and managing complexity, hierarchy and abstraction are consistently a defining characteristic. Thus, a good collaboration software must take on a hierarchical structure. We will soon see that hierarchical and nestable structure is an essence of the Symp.ly collaboration software.

Symp.ly Application Architecture

At the very highest level, Symp.ly is organized into projects. Each user is able to create multiple projects. Each project can be shared with multiple users or kept as a private project.

Projects contain items. Each item has a parent element and a list of children, each of which is also an item. In this way, a project can be modeled as a hierarchical tree with unlimited depth and no leaf nodes. In addition to keeping track of parent and children elements, items in the same hierarchical level also have a context of order.

From the user interface, a project looks like a nestable bullet list. Each item is its own entity and can be moved, deleted, or changed. As such, each item can contain meta-data. This may include tags and notes. If Symp.ly is to be used as a task list, meta-data may include due dates and assignees. As mentioned earlier, each node can also represent a larger data resource such as a document, spreadsheet, presentation, or drawing, each of which exhibits its own real-time collaborative features.

A hierarchical tree structured data model was chosen because it is the most general and fundamental hierarchical structure. Symp.ly is extensible and can meet specific collaboration and task management needs.

Nestable Data Structure and The User Interface

Because of of the high frequency of usage, a user's ability to interact with the software interface is one of the key determinants of success in collaboration and task management software. With this in mind, Symp.ly employs three key user interface features: infinitely hierarchical and nestable structures; collapsable items; and zoomable items.

Hierarchical Data Structure

Symp.ly has an infinitely recursive hierarchical data structure in which items can be nested in each other with unlimited depth. This is functionally relevant in several ways. When Symp.ly is used as a task management application, the infinitely nestable items can act as sub-items, sub-sub-items, and so on. Items in Symp.ly can also be used as simple listed notes and outliners. In this case, the purpose of the infinitely nestable items are even more clear: they can act as a hierarchical information structure. An even simpler use case is a brain map, which highly utilizes hierarchical data to map out acquired information and knowledge.

Nevertheless, two important questions still arise about an "infinitely" recursive structure: first, why should the structure be allowed to recurse infinitely, instead of just allowing a constant number of hierarchical depths? Second, do users really desire a hierarchical data structure that allows for infinite nesting? The first question was indeed taken into consideration when building Symp.ly. One possible concern with an infinitely recursively structure is that it will cause clutter and become obtrusive to the interface when viewing and editing project items. This "hierarchical breakdown" is often considered to be one of the most significant problems with hierarchical user interfaces, and is difficult to solve [6].

We will see that our answer to the latter question will naturally provide an answer to the former. In short, we do believe that users care about as many layers of nesting as the application can provide. We believe that the number of levels of nesting should solely be the user's decision. The application should not limit a user's need for nesting in any way, and

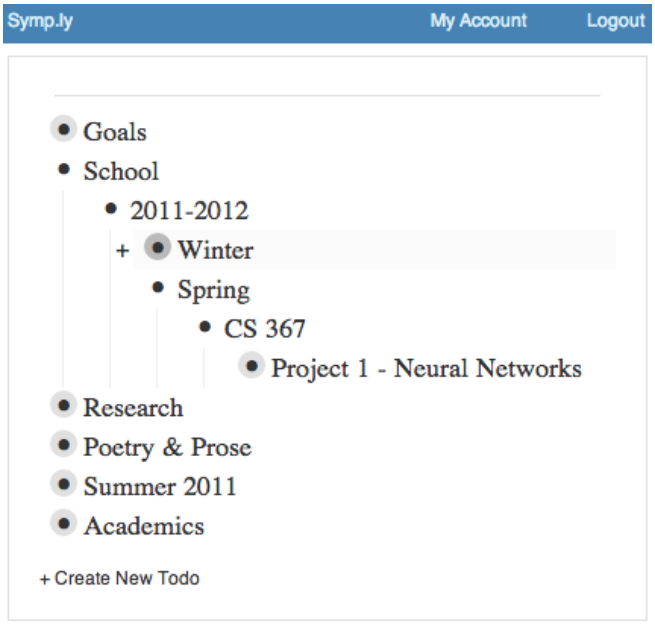

Figure 1. An illustration of the collapsable items of Symp.ly. The items that whose bullets are shaded in gray represent items that are collapsed and have more children nested under them. This allows users to hide items that are not currently relevant to the hierarchical context.

a user should not have to worry about her ability to initiate another layer of hierarchy [17] [16] [4]. For example, a daily todo list may only have two or three levels of sub-items, but in a large enterprise level use case, tens of levels of nesting may be required to organize the high level goals and milestones of the project down to the individual items. In the case of a brain map, hundreds of levels of hierarchy may be needed.

Thus, hierarchy is not only deeply rooted in the way we think and process information, but also presents itself practically. Yet, many users are unable to communicate this need. This is because they seldom have the opportunity to use applications that offer both unlimited nestable items elegantly and does not interfere with the normal use and functionality of the application [6]. Hence, users have been trained to not ask this much out of their applications. Therefore, questions about having an infinitely recursive and nestable structure are solved if we can provide an elegant interface for the user that is not only non-obtrusive, but also intuitive.

However, hierarchy is difficult to represent in a computer interface, especially one that is infinitely recursive. Screen real-estate is limited and users can only view and take in so much at once. Symp.ly solves these problem by utilizing several user interface techniques.

Collapsable Items

Collapsible items help solve problems that arise from hierarchical structures. As shown in Figure 1, each item that contains children items in the hierarchical tree has a plus or minus button to its left, which appears when the mouse cursor hovers over the item. When the button is a minus sign, the item is uncollapsed, or expanded, indicating that all of its children elements are viewable. In this expanded state, the user can click

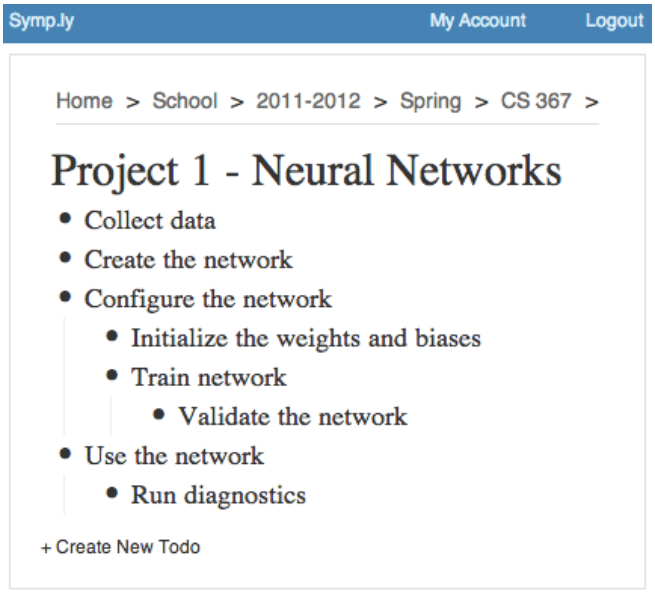

Figure 2. An illustration of the zoomable items of Symp.ly. Currently, the "Project 1" item is being zoomed. The zoomed item becomes of the root node and is displayed on the top of the contents area. Also, a breadcrumb interface appears on top of the content area to show the exact location of the current root node. This allows users to focus on a particular task while keeping other tasks in their respective hierarchies.

on the minus button to collapse the item, which will make all of its children elements non-visible, and the space they take up collapsed. When this happens, the button to the left of the item will become a plus sign, and the bullet point will be highlighted with a grey annulus. By highlighting the bullet point when the children are collapsed, users will be able to easily distinguish between collapsed and expanded items.

Having collapsable and expandable items in a hierarchical data structure has been shown in several studies to significantly reduce a user's time to seek data [6]. By displaying high level information in a cohesive and contiguous way while giving users the choice of displaying or hiding certain items on demand, collapsable items effectively solves the hierarchical breakdown problem we described above [6].

This collapsible tree interface helps solves the problem of clutter and makes viewing the infinitely recursively tree possible [14] [6]. However, collapsable items do not fully solve the problem of manipulating, editing, and navigating the hierarchical tree [24]. Thus, we introduce zoomable items to solve this problem [11].

Zoomable Items

Each item in the hierarchical tree of a Symp.ly project is "zoomable". In the user interface, the bullet point associated with each item in the hierarchical tree is clickable. By clicking on the bullet of an item, the user interface will "zoom" to that particular item. This means that the item that has been clicked on becomes the root node of the tree in the interface. On the top of the page, there is also a breadcrumbs interface that displays the full "path" to the current item, which is an arrow delimited, ordered list of all of the root item's ancestors, starting from the root node of the entire tree. In this way, each item can become the root node of the tree interface displayed as shown in Figure 2.

An analog for this zoomable interface is the foldering interface of a file system that can be navigated in any modern day windowing operating system. Folders can be seen as an infinitely recursive hierarchical tree structure [17] [14]. Any folder in the tree can be opened in its own window which will display all of its children in that window. The only difference is that Symp.ly has no terminal nodes whereas a file system generally does. In other words, file systems have files, which cannot act as folders themselves, whereas every node in Symp.ly can contain children nodes.

One of the most practical reasons to have a zoomable items interface is to allow visualization and manipulation of data at multiple scopes [11]. In the case of collaboration, while project managers and leaders may want to see the project from a high level perspective, members of the project assigned to do a specific task may want to zoom in to that task and create a sub-hierarchy of subtasks that are necessary for completing that task. This gives users both the ability to visualize at a high level, and create new information at a low level as needed.

Another important reason for the inclusion of a zoomable interface is to allow users to easily navigate the tree by pruning out nodes that are not relevant to a particular context or task [12]. Pruning, a technique for selectively hiding nodes that don't pertain to a particular context, has been shown to significantly improve a user's performance speed and overall satisfaction [12]. Studies have also shown that interfaces that contain only a collapsable interface without zooming produce significantly worse user performance results than those that contain zooming interfaces [24].

This zoomable items interface allows users to edit and manipulate any node in the hierarchical tree as if it were the root node. This prevents clutter and obtrusiveness in the infinitely recursive structure of the interface while allowing users to take advantage of all of its benefits [11]. Thus, zoomable items is one of the core features that allows Symp.ly to act seamlessly as both a collaboration and task management application.

Research Efforts in Real-Time Collaborative Technology

A key feature of Symp.ly is its real-time nature. In this section, we describe the real-time implementation in Symp.ly. First, we give background on research in the field of real-time collaborative technologies and operation transforms (OT).

From the onset of the Jupiter Collaboration System in the 1990s [15] to the mass market adoption of Google Docs in the mid 2000s, the idea of real-time collaborative systems has existed for some time. However, real-time systems are still very difficult to architect and build. Part of the reason is because of the application specific systems infrastructure that is required to support real-time systems. Different OT algorithms have to be implemented for different functionalities across different underlying data structures. There is no unifying data real-time architecture for an arbitrary data structure.

To create real-time technologies, a system must implement both an OT control algorithm and a system specific OT function [13]. The OT control algorithm regulates the flow of operations from server to client and makes sure that operations are transformed properly with the right OT algorithms. Although OT control algorithms are more or less generalizable, there is no easy way to generalize OT functions for applications because operations are specific to application data structure.

For a long time, real-time collaborative interfaces on the browser were at worst impossible and at best a hack, because most browsers did not support server-push technology. With the introduction of web-sockets and networking frameworks such as Node.js, building real-time systems from a purely networking perspective have become much simpler. So the main difficulty now lies in OT. There has been much research done in operation transform of text [13] [23] [8], and some work has also been done on operation transform of hierarchical structured data [13] [19].

Concurrency Control

Concurrency control became increasing important with emergence of database management systems in the 1970s. Concurrency control is essential for the correctness of any system where two different clients can access the same data perform one or more transactions with time overlap. This occurs in virtually in any general-purpose database system.

Before the use of operation transform was introduce, building concurrency control systems involved transactions that adhered to a set of strict rules, called the ACID rules [5], which stands for atomicity, consistency, isolation, and durability. There were several early methods developed to handle concurrency control, many of which are still used today in systems that might not require immediate reflection of changes [5]. These include locking, serialization graph checking, and timestamp ordering.

These methods worked well in early database management systems, but they are insufficient in handling modern groupware and real-time needs [9]. Modern day groupware require extremely fine-grain sharing of data with negligible response times.

Consider modern text editing as an example. Synchronization requires a level of granularity to the keystroke while response times more than several hundred milliseconds are not considered negligible. If a collaborative text editor implemented using locking, then the user would have to acquire a lock for the document on every keystroke, making real-time collaboration impossible. Similarly, consider the use of atomic transaction commits. This would mean that each user's edit would override the entire document, including the edits of all other users. Clearly, this is inefficient and ineffective for a real-time collaborative environment.

Operation Transform

To solve the problem of earlier work in concurrency control and adapt it to a real-time collaborative environment, Ellis and Gibbs introduced the concept of operation transform in 1989 [9]. In their paper, Ellis and Gibbs points out many flaws associated with using locking, especially with groupware that require immediate, real-time synchronization of changes across clients [9]. They presented an algorithm called the Distributed Operational Transformation (dOPT) Algorithm for handling concurrent operations on a shared resource without locking.

Jupiter Collaboration System

In 1995, the Jupiter Collaboration System [15] significantly improved Ellis and Gibbs' OT control algorithm by centralizing the OT logic to a server. It led a series of early papers published in the late 1980s to mid 1990s that improved and simplified the OT control and integration algorithm, which allowed for the concurrent access of data from multiple clients.

These systems all described different OT control and integration algorithms for maintaining consistency and concurrency between different clients [15], [18], [1]. Systems similar to the Jupiter adhere to the CCI (causality, convergence, and intention preservation) model, satisfying the three properties that must hold:

- Precedence (Causality) property: this ensures that the execution order of causally dependent operations is retained and kept the same as their natural cause-effect order during operation processing.
- Convergence property: this ensures that when all generated operations have been executed at all clients, the shared document at all these clients converges to the same state.
- Intention Preservation property: this ensures that when a client executes an operation on a shared document from the state that the client current resides, the effect of the operation should be the same as the intention of the operation given the current state, regardless of any other client's state.

These early research efforts paved the way for later commercial efforts in real-time technology and allowed for real-time technologies to be adopted by groupware and collaboration applications.

Share.js

Share.js is one of the most recent efforts made in the open source community to make real-time technology more available to web developers. Share.js is a javascript library for Node.js that allows web application developers to make their applications real-time.

Share.js utilizes an existing OT control algorithms and generalizes it so that developers may be able to plug-and-play into their applications. The OT algorithm still has to be written by the developer, which is tailored specifically to each different application. Share.js has received moderate success and has several example application that developers have created with Share.js, making their applications have a real-time interface. Share.js provides a simple and easy-to-use API, but its robustness and ability to create complex applications has yet to be shown.

Real-Time Interface in Symp.ly

A key feature of Symp.ly is its real-time nature. In this section, we describe the implementation of OT in Symp.ly. Then

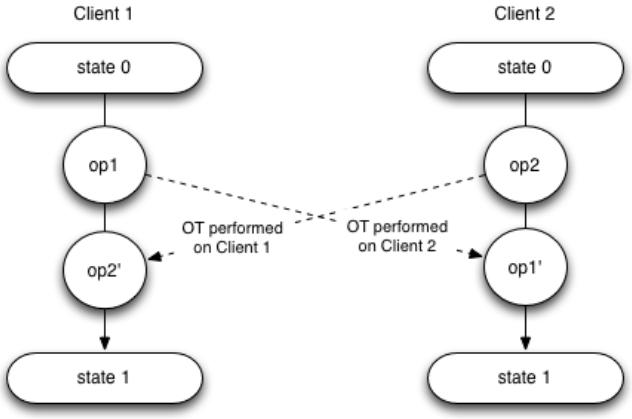

Figure 3. A high level illustration of a classical operation transform problem. Client 1 and Client 2 start with the same state 0, but create two concurrent operations, $op1$ and $op2$, respectively. The OT control algorithm uses the $xform$ function to transform each operation in the scope of the other. The resulting operations, $op1'$ and $op2'$ are executed on each client. Because of the OT control algorithm, the two clients can terminate at the same state and convergence is satisfied.

we describe two major operation transform (OT) algorithms. The first is text OT, which has been utilized by Google Docs and other online real-time text editors. The second is OT on the recursive and hierarchical tree structure. We describe these OT algorithms first from a theoretical perspective and then from an implementation perspective.

OT Control Algorithm

Real-time applications need to resolve concurrent and possibly conflicting operations on a shared resource generated from multiple clients. This requires an OT control algorithm, which regulates concurrent operations between server and clients, and an OT transform function, which transforms one concurrent operation in the scope of another as shown in Figure 3. The Jupiter Collaboration System paper [15] describes an OT control algorithm that relies on an OT transform function, $xform$. Symp.ly utilizes this algorithm generalized for n-clients. We do not discuss the OT control algorithm in detail in this paper, as it is already described in [15]. Instead, we focus on the $xform$ function, which was not described in detail in the Jupiter paper. The paper mentioned that such an $xform$ function must satisfy the following condition:

$$o_1 \cdot xform(o_2, o_1) \equiv o_2 \cdot xform(o_1, o_2) \qquad (1)$$

Here, o_i is the i-th operation, the dot product $o_i \cdot o_j$ represents performing operations o_i and then o_j, and $xform(o_i, o_j)$ gives an operation that transforms the operation o_i with a concurrent operation o_j. Given that operation o_1 and o_2 are performed concurrently on two different clients, $xform(o_2, o_1)$ will give the transformed operation o_2' that is the operation o_2 if it were to be performed after knowing o_1 was performed. Similarly, $xform(o_1, o_2)$ gives o_1', the corresponding scope of o_1.

These transformations must be made on both the client and server in order to guarantee convergence. On both client and server, the $xform$ function must be performed on incoming operations with all sent and unacknowledged operations. This is because any operations that are received have no context of all sent and unacknowledged operations. As a result, any received message is concurrent with any sent but unacknowledged operation.

Textual OT Operations

The Symp.ly Collaboration System utilizes a customized version of the $xform$ function above [15]. This $xform$ function is able to resolve conflicts from a number of operations. Symp.ly handles not only concurrent operations on text, but also operations on structural changes that affect the nested structure of a project. In this section, we describe the operation that Symp.ly uses to model textual and structural changes.

Given a document D, let D_i represent the i-th version of the document. To describe textual changes that transform D_i to D_{i+1}, we use an ordered list data structure that contains three separate types of textual operations on an entire document. This ordered list describes the change operations that need to be performed on D_i in order to obtain D_{i+1}. Each of these operation is one of three types:

- $del(i)$ - deletes i characters from the last index.
- $insert(s)$ - inserts the string s at the current index.
- $retain(i)$ - retains (keeps) i characters from the last index.

To obtain this ordered list of changes on text, Symp.ly compares D_{i+1} with D_i and records changes starting from the character in the 0-th index. Each difference in the two versions of the document is recorded by one of the three operations above and saved in the ordered list of changes. For example, if $D_1 =$ "Hello World" and $D_2 =$ "hello world!" then the ordered list of changes that will bring D_1 to D_2 can be given by the array of operations: $[del(1), insert(\text{"h"}), retain(5), del(1), insert(\text{"w"}), retain(4), insert(\text{"!"})]$.

$xform$ Algorithm for Text

Symp.ly's $xform$ function for text discovers conflicts between two sets of concurrent operations and resolves them in a way that preserves causality and client intention while guaranteeing convergence. To do this, the function loops through the first set of operations and compares each operation with each operation in the second set, while keeping track of the document index. Two sets of operations conflict when they attempt to perform different operations at the same index.

For example, consider two sets of operations O_1 and O_2 that occur concurrently. If at some index i of the document, O_1 is issuing a del operation and O_2 performed a $retain$ operation, then these two sets of operations, O_1 and O_2 will conflict on index i. Symp.ly's $xform$ operation finds and resolves all cases of conflict with a redefined set of rules and intention priorities.

We have ranked the intention priority of the operations in the following order, from lowest to highest: $retain$, $insert$, del. There are other ways to rank these intention priorities, but the current rank is a very natural one. In the case of our example, the delete operation will win since its intention is of a higher

priority because del is an explicit operation whereas $retain$ is an implicit operation.

Structural OT Operations

In this section, we discuss the OT operations that capture structural changes. Symp.ly utilizes a hierarchical tree structure so all the operations related to this structure will deal with manipulations of the tree. Structural OT can be classified into one of four types:

- $create(n, p, o)$ - creates a new node n that has parent p. The last parameter o represents an offset node which will give the relative position of the new node in the parent node and will determine its ordering. In Symp.ly, the new node n is placed right after the offset node o in the parent node p. This means that o can be $null$, which will mean that n should become the first node under p.
- $remove(n)$ - removes the node n and its descendants from the tree.
- $changeParent(n, p, o)$ - changes the parent of the node n from it's current parent to the new parent p. The offset node o functions in the same way as it does in the $create$ operation, to determine the node's ordering within its new parent.
- $reorder(n, o)$ - reorders the node n to come after the new offset node o. Note that n and o should both be under the same parent node.

We next discuss the $xform$ algorithm for these structural operations.

$xform$ Algorithm for Structure

Whereas in textual operation transform, capturing a single change of a document required an ordered list of basic operations that describe the entire document, structural OT operations are in themselves atomic. This means that each of the operations above represents a complete change in the tree, and we do not have to depend on a list of operations to describe a change. Thus, the $xform$ function needs to compare each operation type with every other operation type, possibly including itself.

Because of this, the $xform$ function for structures can be represented as a 4×4 matrix M, where each element $m_{ij} \in M$ is a function that gives the transform of operation i with operation j, where each i, j is in the set of structural operations as we defined above.

Whenever two operations types, i and j are compared with $xform(o_1, o_2)$, we simply find the element $m_{ij} \in M$ and call $m_{ij}(o_1, o_2)$ where $i = type(o_1)$ and $j = type(o_2)$. This will give us the necessary transforms for concurrent operations under the $xform$ OT control.

Sharing and Collaboration

Because Symp.ly is meant to be used for collaboration as well as individual task management, sharing is one of Symp.ly's most crucial features.

The top level structure of Symp.ly is organized into projects, each of which can be shared with multiple collaborators. When a project is shared, the project will appear on the list of projects for each collaborator.

Because of Symp.ly's hierarchical yet flexible structure, sharing a project allows different users to focus at different levels of the tree while maintaining a holistic understand of the entire project. This can be done through Symp.ly's zoomable and collapsable items.

Symp.ly's hierarchical structure naturally allows its projects to be shared and used for collaboration. A project can be seen as a collection of information: ideas, notes, tasks, and deliverables. In this way, collaboration is to make sense of these collections of information by associating them with each other in a structured way. Therefore, for collaboration to work efficiently, the ability to organize information is crucial. For example, while it is important to be able to jot down ideas, take notes, and create tasks, it is equally important in a collaborative environment to be able to create tasks related to notes, create notes related to tasks, and attach ideas to tasks and notes.

Indeed, this need to associate information is hierarchical. This desire of hierarchical information is one reason collaboration applications with a linear and siloed organization structure are not effective. Because Symp.ly's structure consists of nested, hierarchical items, each of which are non-terminal (meaning that each node can be nested further), organization of collaborative information is seamless and nonintrusive with Symp.ly Finally, Symp.ly's real-time interface also enhances collaboration by allowing geographically separate collaborators to work on the same document together synchronously.

EMPIRICAL RESULTS

In evaluating Symp.ly's effectiveness empirically, we performed user testing on 12 university students who have all previously used some sort of task management tool. The subjects were given two tasks: the first task gives the user a passage about the cell cycle and asks the user to use Symp.ly to outline the information; the second task describes a task force in charge of launching a new website for a product, and the subject is asked to used Symp.ly to map out the milestones, tasks, and subtasks, etc for accomplishing the goals of the task force. Prior knowledge of both the cell cycle and website launch are controlled by sampling a diverse pool of students mostly without prior domain expertise.

After each of the two tasks, the subject is asked the following questions:

1. On a scale of 1-5, how effective is Symp.ly as a (1) outlining or (2) task management tool compared to tools you've used in the past.
2. On a scale of 1-5, how enjoyable is Symp.ly as a (1) outlining or (2) task management tool compared to tools you've used in the past.
3. What made Symp.ly more or less enjoyable or effective than other productivity tools.

Questions	Task 1	Task 2
1. Effectiveness	3.7	4.7
2. Enjoyment	4.1	4.8

Table 1. Average ratings of effectiveness and enjoyment.

In the above questions, a 5 point Likert scale is with 5 being the best response indicating that Symp.ly was more effective or enjoyable than existing applications. The results of the first two questions are presented in Table 1, which summarizes the average scores given by the subjects to each of the two questions for each of the two tasks. In answering the third question, 9 subjects commented on the simple and responsive nature of Symp.ly, 7 on its hierarchical and nestable data structure (subtasks), and 6 on its zoomable interface as the contributing factors to its effectiveness and enjoyability (users were able to state more than one characteristic).

Results from our pilot study seem to support our hypothesis that Symp.ly serves as a more effective as well as enjoyable collaboration and task management tool. In particular, users found that, on average Symp.ly was more effective and more enjoyable not only for task management, but also as an outlining tool. Future work includes conducting a larger scale empirical study.

DISCUSSION AND CONCLUSION

One of the key oversights of existing collaboration applications is the ability to manage information at different hierarchical contexts. Because people inherently think hierarchically, we want not only to be able to understand things at a high level, such as high level projects goals and milestones, but also be able to manage low level information such as tasks. We also want to be able to access all hierarchical levels in between, zooming in and out at will. Providing such a flexible hierarchy is difficult, and existing applications fail to capture these abilities in a simple and useful way.

Some applications attempt to solve this problem by providing a finite (and small) set of hierarchical points at which users can store information. However, associating relationships between different information in this system is difficult. More importantly, users can only work within the hierarchical limitations of the application. It is impossible to add information at any other hierarchical level. Other applications ignore the problem altogether by only supporting information organization at one particular level, such as the low level of task management. This linear system of organization may work on a day to day task management scenario, but it does not work for collaborative environments.

The second crucial necessity of collaboration is having a real-time environment. Because more and more work today is done remotely, synchronous access to and work with data is becoming more important. Real-time technologies have allowed users to easy edit a document or spreadsheet together remotely. However, real-time technologies have been difficult in other forms of collaboration mainly because of the need for a real-time synced structure as well as text. Symp.ly solves these collaboration issues by providing a sensible and flexible way to manage information hierarchically.

Symp.ly utilizes an unlimited nestable hierarchical structure, which allows users to represent data and relate them flexibly. Because each node in Symp.ly is generic yet highly customizable, creating any collaborative information on Symp.ly is seamless. A node can be used as an outliner to take notes. Another node can be used as a task list that is related to those nodes. The ability to nest nodes without limit also allows different nodes to be easily associated with one another.

To solve the user interface challenges of having an infinitely nestable hierarchical structure, Symp.ly employs collapsable and zoomable items. Collapsable items allow users to hide the children of nodes that may not be relevant to the particular hierarchical context at the time. Zoomable items allow users to focus on a particular part of the hierarchical tree by making any node the root node of the tree, like a windowing file system interface. These two features give Symp.ly a simple yet flexible user interface for creating and modifying new items.

Symp.ly also employees a real-time interface, which allows remote users to collaborate synchronously. Without a real-time interface, users often times override data, and information synchronization across the group members becomes slow. For the same reason, using locks on data resources also does not solve the problem of information synchronization.

The results of Symp.ly provides progress to collaboration tools. While the real-time technology Symp.ly employees allows remote group members to collaboration in a highly synchronous way, the hierarchical structure of Symp.ly naturally mimics the way we organization information in our brain and provides for a simple yet powerful way to collaborate and manage tasks.

REFERENCES

1. Google wave operation transform. `www.wave-protocol.org`.
2. Almaer, D. Google acquires etherpad online collaboration tool, 2009.
3. Almaer, D. Lunascript: A new language and platform to take your web 2.0 apps to the moon?, 2010.
4. Bederson, B. B., and Hollan, J. D. Pad++: a zooming graphical interface for exploring alternate interface physics. In *Proceedings of the 7th annual ACM symposium on User interface software and technology*, ACM (1994), 17–26.
5. Bernstein, P. A., Hadzilacos, V., and Goodman, N. *Concurrency control and recovery in database systems*, vol. 370. Addison-wesley New York, 1987.
6. Chimera, R., and Shneiderman, B. An exploratory evaluation of three interfaces for browsing large hierarchical tables of contents. *ACM Transactions on Information Systems (TOIS) 12*, 4 (1994), 383–406.
7. Cyras, J. Ruby on rails impact on web application development, 2011.
8. Davis, A. H., Sun, C., and Lu, J. Generalizing operational transformation to the standard general

markup language. In *Proceedings of the 2002 ACM conference on Computer supported cooperative work*, ACM (2002), 58–67.

9. Ellis, C. A., and Gibbs, S. J. Concurrency control in groupware systems. In *ACM SIGMOD Record*, vol. 18, ACM (1989), 399–407.
10. Featherstone, R. Basecamp. *Journal of the Medical Library Association: JMLA 97*, 1 (2009), 67.
11. Gandhi, R., Kumar, G., Bederson, B., and Shneiderman, B. Domain name based visualization of web histories in a zoomable user interface. In *Database and Expert Systems Applications, 2000. Proceedings. 11th International Workshop on*, IEEE (2000), 591–598.
12. Kumar, H. P., Plaisant, C., and Shneiderman, B. Browsing hierarchical data with multi-level dynamic queries and pruning.
13. Kumawat, S., and Khunteta, A. A survey on operational transformation algorithms: Challenges, issues and achievements. *International Journal of Computer Applications 3*, 12 (2010).
14. Modjeska, D. K. *Hierarchical data visualization in desktop virtual reality*. PhD thesis, University of Toronto, 2000.
15. Nichols, D. A., Curtis, P., Dixon, M., and Lamping, J. High-latency, low-bandwidth windowing in the jupiter collaboration system. In *Proceedings of the 8th annual ACM symposium on User interface and software technology*, ACM (1995), 111–120.
16. Plaisant, C., Grosjean, J., and Bederson, B. B. Spacetree: Supporting exploration in large node link tree, design evolution and empirical evaluation. In *Information Visualization, 2002. INFOVIS 2002. IEEE Symposium on*, IEEE (2002), 57–64.
17. Pook, S., Lecolinet, E., Vaysseix, G., and Barillot, E. Context and interaction in zoomable user interfaces. In *Proceedings of the working conference on Advanced visual interfaces*, ACM (2000), 227–231.
18. Prakash, A., and Knister, M. J. A framework for undoing actions in collaborative systems. *ACM Transactions on Computer-Human Interaction (TOCHI) 1*, 4 (1994), 295–330.
19. Ribó, J. M., and Franch, X. A multi-version algorithm for cooperative edition of hierarchically-structured documents. In *Groupware, 2001. Proceedings. Seventh International Workshop on*, IEEE (2001), 154–163.
20. Roseman, M., and Greenberg, S. Teamrooms: network places for collaboration. In *Proceedings of the 1996 ACM conference on Computer supported cooperative work*, ACM (1996), 325–333.
21. Stern, J. The best to-do list-apps: Remember the milk, astrid, and wunderlist compared, 2012.
22. Streitz, N. A., Geißler, J., Haake, J. M., and Hol, J. Dolphin: integrated meeting support across local and remote desktop environments and liveboards. In *Proceedings of the 1994 ACM conference on Computer supported cooperative work*, ACM (1994), 345–358.
23. Sun, D., and Sun, C. Context-based operational transformation in distributed collaborative editing systems. *Parallel and Distributed Systems, IEEE Transactions on 20*, 10 (2009), 1454–1470.
24. Zaphiris, P., Shneiderman, B., and Norman, K. L. Expandable indexes vs. sequential menus for searching hierarchies on the world wide web. *Behaviour & Information Technology 21*, 3 (2002), 201–207.