Yjs: A Framework for Near Real-Time P2P Shared Editing on Arbitrary Data Types

Petru Nicolaescu, Kevin Jahns, Michael Derntl, and Ralf Klamma

Advanced Community Information Systems (ACIS) Group
RWTH Aachen University
Ahornstr. 55, 52056 Aachen, Germany
{lastname}@dbis.rwth-aachen.de

Abstract. Near real-time shared editing of documents in the Web browser has become popular for many applications like text writing, drawing, sketching and others. These applications require protocols for exchanging messages among user agents and for resolving editing conflicts. The available frameworks mostly rely on operational transformation approaches and often expose drawbacks like failing to scale, restriction to linear data structures and client-server architectures. In this paper we present Yjs, a lightweight open-source JavaScript framework that can be used for collaborative editing of arbitrary data types in peer-to-peer settings. The framework is based on a new operational transformation-like approach and supports communication protocols like XMPP and WebRTC. From an engineering perspective Yjs is easy to integrate into Web applications. Evaluations show that it has a favorable runtime complexity.

1 Introduction

Shared editing software enables multiple users to collaborate on shared data [3]. In a near real-time (NRT) setting, collaborators apply changes to their local copy, while concurrently sending and receiving notifications of those changes via some communication protocol. Google Docs is a good example of such an application. It uses Operational Transformation (OT) [1] and a client-server infrastructure for resolving conflicts occurring during NRT collaboration sessions. However, both existing literature [9] and practical experience show that current shared editing approaches do not scale well with the number of users in pure peer-to-peer settings. Moreover, they mostly enable collaboration on linear data structures but are not designed for non-linear ones (e.g., graphs, custom abstract data types). To overcome this gap we have developed Yjs[1], a framework that allows developers to conveniently add collaborative features to Web applications. It offers a customizable approach for any required shared abstract data type. Based on a modular design Yjs contains a collaboration engine that resolves editing conflicts and ensures the shared data integrity across peers, and various connectors for propagating the collaboration-related updates across the network.

[1] Developer documentation and a demo can be found at http://y-js.org.

Our demo showcases the usage of Yjs on text, JSON and DOM elements (XML), using both WebRTC and XMPP connectors. As such, collaborative editing using text, HTML5 elements and HTML5 Web pages is part of the demo scenario. The focus is on the reliability and usability of the framework in ad hoc peer-to-peer settings. Presented recently in a lightning talk at the European open source conference in Brussels (FOSDEM'15), Yjs is already garnering interest from the open-source community. We target the adoption of NRT collaboration for existing and new Web applications by leveraging lightweight applications, where collaboration logic can be easily engineered on the client side.

2 Features of the Distributed Shared Editing Algorithm

Most OT frameworks support collaboration on text, JSON and/or XML [5,2,4]. However, due to the difficulty of implementation and dealing with known OT collaboration puzzles, they fail to scale with the number of users in peer-to-peer environments [9]. Namely, they do not always converge or they need a centralized mechanism for creating a total order for local operations. Moreover, in terms of applying operations on a local copy of a shared data, in a client-server approach there are n execution orders, where n is the number of users, whereas in a peer-to-peer approach, the clients have one connection to every collaborator and there are $n(n-1)!$ execution orders that need to be considered. Yjs proposes a new type of algorithm and a modular implementation that scales to peer-to-peer communication between the clients. The algorithm, similar to efficient existing OT approaches [8,9] has a favorable time complexity. Instead of defining an operation as a tuple of position and content, we define it as an insertion between two other operations (similar to the WOOT approach[2]). This has positive implications on preserving the user intention for the order of applied operations and enables a worst-case time complexity of $O(C^2)$, where C is the number of concurrent operations at the same position.

Important advantages of Yjs' algorithm are therefore that it reduces time to synchronize shared data among clients (e.g., after a late join), can handle more clients at the same time and minimizes the space needed for propagated messages. Compared to classic OT algorithms it does not require propagation of complex internal structures (i.e., state vectors). Due to space limitations, details about the algorithm and its formal proof will be published in the near future.

3 Yjs Framework

In contrast to similar implementations that support only a very limited number of document structures, the Yjs framework encourages developers to build custom data types. A custom type can use existing types (e.g., from third parties) in order to give meaning on the actions on the data and can fire custom events. Among existing implemented types are *String*, *Array* and *Object*. Internally, a

[2] `https://hal.inria.fr/inria-00071240/document`

linear data type is represented as a doubly linked list. Each composing element (e.g., a character of a word) represents an item of the linked list having a predecessor and a successor. When an element is deleted by an operation, it will not be removed from the linked list, but its content will be set to *empty*. Only a built-in garbage collector can remove elements completely from the linked list, when it can assure that no conflicts will occur. Yjs is currently available as a collection of open-source JavaScript libraries on GitHub[3].

In order to keep the modularity and to be able to employ Yjs in various Web engineering settings, the communication protocols and the shared data type formats support are implemented as dedicated interchangeable components. This greatly simplifies the process of integrating the framework into an existing project, since existing projects typically use diverse communication protocols (e.g., Web Sockets, IWC, XMPP), and collaborate on various data types (e.g., XML, JSON, graphs). Support of different communication protocols is implemented in connector components, and the support for data types in type components. While the text document component only supports simple insert, delete and replace operations, the interface for XML documents inherits all the features (events, and DOM traversal / manipulation) of the browser DOM element. This means that the XML document can also be easily queried and manipulated via jQuery using its XML manipulation features.

4 Evaluation

To evaluate the scalability and correctness of Yjs we performed multiple automatic tests simulating many users working on a single shared document. Every user was represented by an instance of the Yjs framework. For this, we used a test connector that simulated a peer-to-peer environment, configurable with respect to number of users and actions that can be created. We connected each test user directly with all other users by means of the test connector (resulting in a total connected network). Furthermore, we enabled the option to restrict the collaboration to a specific data type (e.g., text or JSON with primitive data types and XML attributes). Among the test cases we considered various network delays in the peer-to-peer communication, generation of random operations created in different contexts and varying order of received operations at every peer [8].

Using this setting we created and executed 10000 actions on text and JSON with various test agents, ranging from 1 to 10. The tests ran on one single CPU only (Intel i7, 3.4 GHz). We ran the test 15 times to obtain solid average timings. The results showed that in average 100 actions per millisecond can be performed, which we consider to meet well the NRT collaboration expectations.

In terms of real-world applications, Yjs was used for enabling free-hand collaborative video annotation at frame level in near real-time [6]. The annotation tool evaluation also showed that the framework outputs reliable results and that it is easy to use from a developer's perspective.

[3] `https://github.com/y-js`

5 Conclusion and Future Work

Our proposed example scenarios showcase NRT collaboration on text, JSON elements and XML in terms of DOM (collaboratively building a Web page). The scenarios use both XMPP and WebRTC in order to demonstrate the trade-off between federation and responsiveness. Our results show that XMPP provides a better scaling in number of users than WebRTC, which in turn provides more efficient response times. We are currently implementing the undo functionality. Furthermore, we are investigating ways to persist the internal Yjs data representations for improved performance.

Yjs is being integrated in several open-source projects. We plan to use Yjs with our DireWolf framework [7] for widget distribution in order to synchronize the state between the Web components. Further usage of Yjs—also in the domain of mashup applications—is to integrate it as NRT collaboration and state preservation solution in modeling and widgetizing Web applications.

6 Acknowledgments

This research was funded in part by the European Commission in the projects "Layers" (FP7-318209) and "METIS" (531262-LLP-2012-ES-KA3-KA3MP).

References

1. C. A. Ellis and S. J. Gibbs. Concurrency Control in Groupware Systems. *SIGMOD Record*, 18(2):399–407, 1989.
2. Gerlicher, Ansgar R. S. A Framework for Real-time Collaborative Engineering in the Automotive Industries. In *Proceedings of the Third International Conference on Cooperative Design, Visualization, and Engineering*, pages 164–173, 2006.
3. J. Grudin. Computer-Supported Cooperative Work: History and Focus. *Computer*, 27(5):19–26, 1994.
4. M. Heinrich, F. J. Gruneberger, T. Springer, and M. Gaedke. Enriching Web Applications with Collaboration Support Using Dependency Injection. In *Proceedings of the 12th International Conference on Web Engineering (ICWE'12)*, pages 473–476, 2012.
5. C.-L. Ignat and M. C. Norrie. Multi-level Editing of Hierarchical Documents. *Computer Supported Cooperative Work (CSCW)*, 17(5-6):423–468, 2008.
6. I. Koren, P. Nicolaescu, and R. Klamma. Collaborative Drawing Annotations on Web Videos. In *Proceedings of the 15th International Conference on Web Engineering (ICWE'15)*, 2015.
7. D. Kovachev, D. Renzel, P. Nicolaescu, I. Koren, and R. Klamma. DireWolf: A Framework for Widget-based Distributed User Interfaces. *Journal of Web Engineering*, 13(3&4):203–222, 2014.
8. D. Sun and C. Sun. Operation Context and Context-based Operational Transformation. In *Proceedings of the 20th Anniversary Conference on Computer Supported Cooperative Work (CSCW'06)*, pages 279–288, 2006.
9. S. Weiss, P. Urso, and P. Molli. Logoot: A Scalable Optimistic Replication Algorithm for Collaborative Editing on P2P Networks. In *Proceedings of the 29th IEEE Conference on Distributed Computing Systems (ICDCS '09)*, pages 404–412, 2009.