Awakening Decentralised Real-Time Collaboration: Re-engineering Apache Wave into a General-Purpose Federated and Collaborative Platform

Pablo Ojanguren-Menendez, Antonio Tenorio-Fornés, and Samer Hassan

GRASIA: Grupo de Agentes Software, Ingeniería y Aplicaciones,
Departamento de Ingeniería del Software e Inteligencia Artificial,
Universidad Complutense de Madrid, Madrid, 28040, Spain
{pablojan,antonio.tenorio,samer}@ucm.es

Abstract. Real-time collaboration is being offered by plenty of libraries and APIs (Google Drive Real-time API, Microsoft Real-Time Communications API, TogetherJS, ShareJS), rapidly becoming a mainstream option for web-services developers. However, they are offered as centralised services running in a single server, regardless if they are free/open source or proprietary software. After re-engineering Apache Wave (former Google Wave), we can now provide the first decentralised and federated free/open source alternative. The new API allows to develop new real-time collaborative web applications in both JavaScript and Java environments.

Keywords: Apache Wave, API, Collaborative Edition, Federation, Operational Transformation, Real-time.

1 Introduction

Since the early 2000s, with the release and growth of Wikipedia, collaborative text editing increasingly gained relevance in the Web [1]. Writing texts in a collaborative manner implies multiple issues, especially those concerning the management and resolution of conflicting changes: those performed by different participants over the same part of the document. These are usually handled with asynchronous techniques as in version control systems for software development [2] (e.g. SVN, GIT), resembled by the popular wikis.

However, some synchronous services for collaborative text editing have arisen during the past decade. These allow users to write the same document in real-time, as in Google Docs and Etherpad. They sort out the conflict resolution issue through the Operational Transformation technology [3].

These services are typically centralised: users editing the same content must belong to the same service provider. However, if these services were federated, users from different providers would be able to edit contents simultaneously. Federated architectures provide multiple advantages concerning privacy and power

S. Omatu et al. (eds.), *Distributed Computing & Artificial Intelligence, 12th Int. Conference,*
Advances in Intelligent Systems and Computing 373, DOI: 10.1007/978-3-319-19638-1_31

distribution between users and owners, and avoid the isolation of both users and information in silos [4].

The rest of this paper is organised as follows: first, Operational Transformation frameworks' state of the art is outlined in Section 2. Section 3 depicts the reengineering approach and used technologies and tools. Concepts of the Wave Platform and changes made are explained in Section 4. Afterwards, the results are discussed in Section 5. Finally, conclusions and next steps are presented in Section 6.

2 State of the Art of Real-Time Collaboration

The development of Operational Transformation algorithms started in 1989 with the GROVE System [5]. During the next decade many improvements were added to the original work and a *International Special Interest Group on Collaborative Editing* (SIGCE) was set up in 1998. During the 2000s, OT algorithms were improved as long as mainstream applications started using them [6].

In 2009, Google announced the launch of Wave [7] as a new service for live collaboration where people could participate in conversation threads with collaborative edition based on the Jupiter OT system [8]. The Wave platform also included a federation protocol [9] and extension capabilities with robots and gadgets. In 2010 Google shutted down the Wave service and released the main portions of the source code to the Free/Open Source community. Since then, the project belongs to the Apache Incubator program and it is referred as Apache Wave. Eventually, Google has included Wave's technology on some other products, such as Google Docs. Despite its huge technological potential, the final product had a very constrained purpose and hardly reusable implementation.

Other applications became relevant during that time, such as the Free Libre Open Source Software (FLOSS) Etherpad. However, it was mostly after the Google Wave period when several FLOSS OT client libraries appeared, allowing integration of real-time collaborative edition of text and data in applications. The most relevant examples are outlined as follows.

TogetherJS [10] is a Mozilla project that uses the WebRTC protocol for peer-to-peer communication between Web browsers in addition to OTs for concurrency control of text fields. It does not provide storage and it needs a server in order to establish communications. It is a JavaScript library and uses JSON notation for messages.

ShareJS [11], is a server-client platform for collaborative edition of JSON objects as well as plain text fields. It provides a client API through a JavaScript library.

Goodow [12], is a recent FLOSS framework copying the Google Drive Real-Time API with additional clients for Android and iOS, while providing its own server implementation.

On the other hand, Google provides a *Real-Time API* as part of its Google Drive SDK [13]. It is a centralised service handling simple data structures and plain text.

In general, these solutions are centralized. Despite their claim of focusing in collaboration, users from different servers cannot work or share content. They just provide concurrency control features without added value services like storage and content management. They mostly allow collaborative editing of simple plain text format.

3 Reengineering: Technologies and Tools

This section summarises the procedure followed to re-engineer and build a generic Wave-based collaborative platform, together with the technologies used.

Wave in a Box [14] is the FLOSS reference implementation of the Apache Wave platform, which supports all former Google Wave protocols and specifications [15] and includes both implementations of the Server and the Client user interface. Most of its source code is original from Google Wave and was provided by Google, although it was complemented with parts developed by community contributors.

In particular, the Client part has been used as ground to develop the new API, with same technologies: Java and the Google Web Toolkit (GWT) FLOSS framework [16]. The Client is written in Java but is compiled and translated into JavaScript by GWT in order to be executed in a web-browser.

The lack of technical documentation forced to perform a preliminar extensive source code analysis outcoming documentation and UML diagrams. Then, initial developments within the Wave client were performed to assess whether the Apache Wave implementation could be used to develop new applications within fair parameters of quality and cost.

New general functionality was added in separated components, on top of underneath layers such as the federation protocol and server storage system. This has proved the feasibility of reusing the original code and Wave core features. The new source code is GWT-agnostic in order to be reusable in Java platforms. GWT is used to generate just the top JavaScript layer.

Concerning software testing, the JavaScript framework Jasmine [17] was used in addition to existing unit tests. The developed test suite for the API attacks the public API functions in a web-browser environment testing new layers together with the rest of the architecture stack.

The development has been tracked and released in a public source code repository [18]. It includes documentation and examples on how to use the API. Besides, during the development process, several contributions have been made to the Apache Wave Open Source community, in the form of source code patches, documentation and diagrams.

4 Generalising the Wave Federated Collaborative Platform

This section shows the fundamentals of the Wave platform and how they have been used to turn Wave into a general-purpose platform unlike the former conversation-based one.

4.1 Conversations: Wave Data Models and Architecture

This subsection exposes the conversation approach of Apache Wave, its data models and general architecture. From a logical point of view, the Wave platform handles two data models: the *Wave Data Model* [19] and the *Wave Conversational Model* [20]. First, the Wave Data Model defines general data entities used within the platform:

- **Participant**: user of the platform. It may be a human or a robot [7].
- **Document**: recipient of collaborative real-time data.
- **Wavelet**: set of Documents shared by a set of Participants.
- **Wave**: set of Wavelets sharing the same unique identifier.

Documents are the smallest entity that can store data which can be edited in a collaborative way. Documents are logically grouped in Wavelets. In addition, a Wavelet has a set of participants, which are able to access –read and edit– those Documents. Finally, the Wave Data Model defines the Wave concept as just a group of Wavelets sharing the same Wave identifier.

Data is represented in XML and the Wave Operational Transformation (OT) system [21] provides the concurrency control and consistency maintenance for editing this XML in a Document by multiple users at the same time. It also generates events to notify changes to other parts of the system, locally or remotely. XML is used to represent two types of data: rich *Structured Text* in a HTML-like format and *Abstract Data Types* (ADTs) like maps, lists, sets, etc.

On the other hand, the Wave Conversational Model was defined to manage *Conversations*, the major concept of the Wave product. A Conversation is a Wavelet having a set of participants, and a set of Documents supporting the Conversation Thread. A Conversation Thread is compound of Documents storing paragraphs as Structured Text and a Document storing the tree-structure of those paragraphs using ADTs. Conversation Metadata is also stored as a Document using ADTs. This schema is summarised below in Figure 2.

Those data models are implemented in separated layers of the Wave Client architecture as it is shown in Figure 1. All the components of this architecture are developed, packaged and deployed as an unique Java/GWT application.

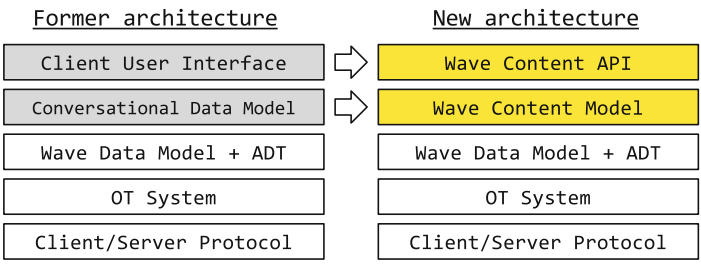

Fig. 1. Wave Client architecture

4.2 General-Purpose Collaboration: Generalising the Wave Data Model and Architecture

Last section outlined the Wave's general data model that could be used in alternative ways. This section introduces a general approach to use it (the Wave Content Model) and a mechanism to consume it (the Wave Content API).

The Wave Content Model. This is a new general-purpose and dynamic data model replacing the former Wave Conversational Model (see Figure 2). It allows to edit Abstract Data Types collaboratively on real-time by different users.

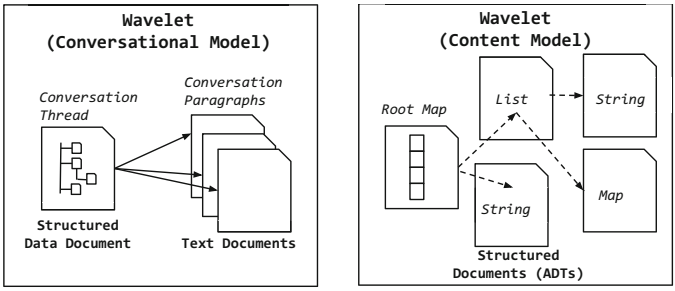

Fig. 2. Wave Data models

The main task was to develop a suitable layer that allows to dinamically create and handle ADTs within Documents of a Wavelet. ADTs are Java classes managing part of the Document content in a particular way. They can be combined declaring new compound types. The Conversation Thread implementation is an example of inmutable compound type as long as inner data structure can't be change on execution.

However, to provide a dynamic composition of ADTs, a Composite pattern [22] is applied. Such pattern defines a hierarchy of data types that can be combined and nested: map, list and string values. Each type is backed by the matching ADT; these new data types control where and how to create and handle ADTs instances within Documents:

This dynamic model is named the *Wave Content Model*. For the shake of clarity, a Wave is now called *Content Instance*, and it provides a main Wavelet where arbitrary data types can be stored dynamically starting from a provided root map. Applications can add new instances of lists and maps to this root or their nested lists and maps, and eventually store string values.

From the architecture perspective, all existing components related to Conversations have been discarded. In particular, the two top layers of the architecture have been replaced (see Figure 1). First, the Wave Conversational Model by the new Wave Content Model. Second, in order to consume the new model in a general way -not just by one single application- the old client is replaced by an API as it is depicted below.

The Wave Content API. With the new Wave Content Model any application could use collaborative data structures. However, according to the technology used in the Apache Wave implementation, just new Java or GWT Web Applications could use them directly. With the aim of delivering these new capabilities to any Web Application developed in any technology, a JavaScript API has been built.

Although GWT eventually translates Java code into JavaScript, this is not suitable to be consumed directly by non-GWT JavaScript code in a web-browser environment, for several reasons: the exception handling is not understood by outer code, and GWT-generated JavaScript syntax is obfuscated.

JSNI and Overlay Types [16] are features of GWT allowing to write arbitrary native JavaScript code and objects integrated transparently with Java code. These features have been used to develop a native JavaScript layer, following the Proxy pattern, which exposes the Wave Content Model functionality as an API. A summary of the features provided by the API follows:

Session management: controls user authentication and life cycle of content instances and

Content Instance management: Maps, lists and strings are created through a provided factory and a root map is provided as a hook.

Data types management: exposes type-specific operations such as the addition of an element to a list or getting map keys.

5 Discussion

This paper introduces the only federated platform for real-time collaboration available nowadays. However, using Wave as its starting point involves some issues.

There are several critiques concerning the complexity of the Wave OT system [11]. Its highly complex implementation –together with the lack of good documentation– causes the maintenance of the source code to be a hard task. However, OT systems are inherently complex and to design OT-based languages and control algorithms require knowledgeable people.

Some existing OT implementations are simpler, using the JSON language and a smaller set of OT operations [11] [12]. In contrast, Wave uses XML dialects that supports both, *rich text* edition straight away and structured data, instead of just plain text and JSON. Wave is the only open OT system providing full rich text and text annotations.

Regarding the API design, it works with data structures (map, list) –as the Google Drive Real-Time API–, in contrast with direct JSON objects. It is hard to conclude which approach is more appropriate for third-party developers since the lack of information about the adoption level and critics in both cases.

Java/GWT as implementation language and Jetty as the HTTP server [23], could be seen as a pitfall as long as nowadays trends are to develop using JavaScript directly and to use high-performance servers. However, GWT is still

a highly adopted and mature project which a strong community. And from the server perspective, it would be easy to adapt the code to run in non-blocking IO servers [24], extending the life of the original source code.

6 Concluding Remarks

A federated platform to develop web applications with real-time collaborative editing capabilities has been presented in the previous sections. It has been developed as a generalisation of the Apache Wave platform, the FLOSS project formerly known as Google Wave.

Nowadays there is no other federated (or distributed) platform for real-time collaboration. Moreover, this work takes the Wave Federation Protocol further, making it a general protocol. Thus, now on top of the Wave Content Model anyone can define new inter-operable collaborative data formats for text documents, spreadsheets, drawings, games, social media, social activity, etc. New applications could adopt them using an existing provider or becoming a new one. Providers can scale on interoperability since OT storage system is agnostic from underlaying content. Clients just need to be aware of data formats.

The provided API is a functional alternative to existing collaborative platforms. It provides a full-stack of software ready to be deployed, with functionalities only comparable with the proprietary Google Drive Real-Time API. Features such as the participation model, content storage and capabilities to search and manage contents, are already included in the Apache Wave platform but not implemented in any alternative.

The API is offered in JavaScript, to be integrated in web applications. Besides, a Java version will be soon released, in order to allow also Android and Java applications to have collaborative capabilities.

This work shows the unexplored high potentials of Google's original development, in spite of its complexity and lack of documentation. Thus, this work steps out engineering challenges for reusing Apache Wave and we hope it paves the way for other researchers and developers.

Acknowledgments. This work was partially supported by the Framework programme FP7-ICT-2013-10 of the European Commission through project P2Pvalue (grant no.: 610961).

References

1. West, J.A., West, M.L.: Using Wikis for Online Collaboration: The Power of the Read-Write Web. John Wiley & Sons (2008)
2. Berliner, B.: CVS II: Parallelizing software development. In: USENIX Winter 1990 Technical Conference, pp. 341–352. USENIX, Berkeley (1990)
3. Sun, C., Ellis, C.: Operational transformation in real-time group editors: Issues, algorithms, and achievements. In: Proceedings of the 1998 ACM Conference on Computer Supported Cooperative Work, pp. 59–68. ACM, New York (1998)

4. Yeung, C., Liccardi, I., Lu, K., Seneviratne, O., Berners-Lee, T.: Decentralization: The future of online social networking. In: W3C Workshop on the Future of Social Networking Position Papers, W3C (2009)
5. Ellis, C.A., Gibbs, S.J.: Concurrency control in groupware systems. In: Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, SIGMOD 1989, pp. 399–407. ACM, New York (1989)
6. Bigler, M., Raess, S., Zbinden, L.: ACE - a collaborative editor, http://sourceforge.net/projects/ace/
7. Ferrate, A.: Google Wave: Up and Running. O'Reilly Media, Inc. (2010)
8. Nichols, D.A., Curtis, P., Dixon, M., Lamping, J.: High-latency, low-bandwidth windowing in the jupiter collaboration system. In: Proceedings of 8th ACM Symposium on User Interface and Software Technology, pp. 111–120. ACM, New York (1995)
9. Baxter, A., Bekmann, J., Berlin, D., Gregorio, J., Lassen, S., Thorogood, S.: Google Wave Federation Protocol Over XMPP (2009), http://wave-protocol.googlecode.com/hg/spec/federation/wavespec.html
10. Mozilla Labs: Togetherjs, https://togetherjs.com/
11. Joseph, G.: ShareJS, http://sharejs.org/
12. Chuanwu, T.: Google docs–style collaboration via the use of operational transforms, https://github.com/goodow
13. Google Inc.: Google Drive SDK: Realtime API, https://developers.google.com/drive/realtime/
14. North, A.: Google Wave Developer Blog: Wave open source next steps: Wave in a Box, http://googlewavedev.blogspot.com.es/2010/09/wave-open-source-next-steps-wave-in-box.html (2010)
15. Google Inc.: Google Wave Protocol, http://www.waveprotocol.org/
16. Cooper, R., Collins, C.: GWT in Practice. Manning Publications (2008)
17. Pivotal Labs: Jasmine, Behavior-Driven JavaScript, http://jasmine.github.io/
18. Ojanguren-Menendez, P.: Real-time collaboration API for Wave, https://github.com/P2Pvalue/incubator-wave
19. North, A.: Wave model deep dive (2010), https://cwiki.apache.org/confluence/display/WAVE/Wave+Summit+Talks
20. Gregorio, J., North, A.: Google Wave Conversation Model (2009), http://wave-protocol.googlecode.com/hg/spec/conversation/convspec.html
21. Lassen, S., Mah, A., Wang, D.: Google Wave Operational Transformation (2010), http://wave-protocol.googlecode.com/hg/whitepapers/operational-transform/operational-transform.html
22. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Pearson Education (1994)
23. The Jetty Project: Jetty, http://www.eclipse.org/jetty/
24. Roth, G.: Architecture of a highly scalable nio-based server (2007), https://today.java.net/pub/a/today/2007/02/13/architecture-of-highly-scalable-nio-server.html