Building Real-Time Collaborative Applications with a Federated Architecture

Pablo Ojanguren-Menendez[1], Antonio Tenorio-Fornés[1], and Samer Hassan[2]

[1] GRASIA research group of Complutense University of Madrid, Madrid, Spain,
[2] Berkman Center for Internet & Society (Harvard University, US), Cambridge, US

***Abstract* —Real-time collaboration is being offered by multiple 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**

I. Introduction

Since the early 2000s, with the release and growth of Wikipedia, collaborative text editing increasingly gained relevance in the Web . The wiki software [1] (such as MediaWiki, TikiWiki and others), which enabled scalable collaborative edition of documents, rapidly became popular. Nowadays, we can see thousands of wikis used by researchers, institutions, enterprises, and a wide diversity of communities to crowdsource the knowledge of the participants. Just Wikia [2], a wiki service provider, accounts for 300K wiki communities with 135M monthly visitors.

Writing texts in a collaborative manner implies multiple challenges, especially those concerning the management and resolution of conflicting changes: those performed by different participants over the same part of the document. That is, if Alice and Bob edit the same sentences at the same time, we should make sure none of their contributions is lost. In fact, in a scenario where we have hundreds or thousands or contributors over the same pages, such conflict is not rare. These conflicts are usually handled with asynchronous techniques as in version control systems for software development [3] (e.g. SVN, GIT), resembled by the popular wikis. In these environments, the software automatically merges contributions over different sections, but users are forced to “take turns” to edit the same sentences (or otherwise manually merge the others’ contributions to theirs).

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 collaboration (simultaneously), as in Google Docs [4] and Etherpad [5]. They tend to sort out the conflict resolution issue through the Operational Transformation [6] technology which has grown to become the de-facto standard in real-time collaborative systems. 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 distribution between users and owners, and avoid the isolation of both users and information in silos [7].

The rest of this paper is organised as follows: first, the state of the art of Operational Transformation frameworks is outlined in Section 2. Section 3 depicts the re-engineering approach and the technologies and tools that were used. Section 4 covers the main concepts of the original Wave Platform, and the changes that were performed are explained in detail. Afterwards, the results are discussed in Section 5. Finally, conclusions and next steps are presented in Section 6.

II. State of the Art of Real-time Collaboration

The development of Operational Transformation (OT) algorithms started in 1989 with the GROVE System [8]. During the next decade many improvements were added to the original work and an 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 [9].

In 2009, Google announced the launch of Wave [10] as a new service for live collaboration where people could participate in conversation threads with collaborative edition based on the Jupiter OT system [11]. The Wave platform also included a federation protocol [12] and extension capabilities with robots and gadgets [13]. Allegedly because of lack of fast user adoption, in 2010 Google shut down the Wave service. However, as initially promised, Google released the main portions of the source code to the Free/Open Source community, and handed its ownership to the Apache Foundation. Since then, the project belongs to the Apache Incubator program and it is referred as Apache Wave [14]. Eventually, Google has included Wave’s technology on several products, such as Google Docs and Google Plus. Despite its high technological potential, the original final product had a constrained purpose and a hardly reusable implementation.

Other web 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 FLOSS OT-based frameworks appeared, allowing the integration of real-time collaborative edition of text and data within third-party applications. The most relevant examples are outlined as follows.

TogetherJS [15] is a Mozilla FLOSS project that uses the WebRTC protocol for peer-to-peer communication among web browsers, together with 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 [16] is a server-client FLOSS platform for collaborative edition of JSON objects as well as plain text fields. It provides a client API through a JavaScript library.

Goodow [17], is a recent FLOSS framework replicating the Google Drive Real-Time API with additional clients for Android and iOS, while providing its own server implementation.

DOI: 10.9781/ijimai.2015.356

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

In general, these solutions are highly centralised. Despite they claim collaboration, users from different servers cannot work or share content. Besides, they mostly provide concurrency control features without added value services like storage and content management. And all of them just allow collaborative edition of simple plain text format.

III. Re-engineering: 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. First, it introduces the software and technologies that have been generalised, Apache Wave and Wave in a Box, and afterwards the technologies used to develop and test the performed extensions. The description of how and where the results are shared and published conclude this section.

A. Assessment of Apache Wave & Wave in a Box

Wave in a Box is the FLOSS reference implementation of the Apache Wave platform, which supports all former Google Wave protocols and specifications [18] 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. It enables real-time collaboration over rich-text conversations in a federated infrastructure. It was designed to be an extensible platform through the use of gadgets and robots.

The existing source code is written in Java and the Google Web Toolkit (GWT) [19]. GWT is a FLOSS framework which allows to write Java code and translate it to JavaScript in order to be used in a Web browser. This approach is used to write all Wave components shared between server and client. User interface components are developed in GWT and they are strongly coupled to the Wave's business logic.

The lack of technical documentation forced to perform a preliminary extensive source code inspection, identifying main packages and interfaces and developing text documentation and diagrams. It was concluded that from a logical point of view, Wave concepts could be reused for general purposes, and that technically the source code was organised in layers properly decoupled.

B. Development & Testing frameworks

Both, server and client components of the Wave in a Box software have been extended. In particular, extensions to the server's storage system have been added to support the NoSQL database MongoDB [20] and some HTTP RESTful services have been also created. Part of new source code in client components has been written avoiding GWT dependencies in order to be reused in any Java runtime environment without adaptations. On top of this code, the JavaScript client API has been developed with some GWT specific code.

Concerning software testing, the JavaScript framework Jasmine [21] was used in addition to existing Java unit tests. The test suite attacks all JavaScript API functions in a web browser environment. These are end-to-end tests where all components of the Wave architecture are verified, from client API methods, to server's storage routines.

C. Contributions

The development has been tracked and released in an open and public source code repository [22]. It includes documentation and different examples about how to use the API.

Besides, during the development process, several contributions have been made to the Apache Wave FLOSS community, in the form of source code patches, documentation and diagrams.

IV. 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.

A. Original Wave Data Models & Architecture

This subsection describes how original Wave data models work from a logical point of view. This allows further understanding of the presented work.

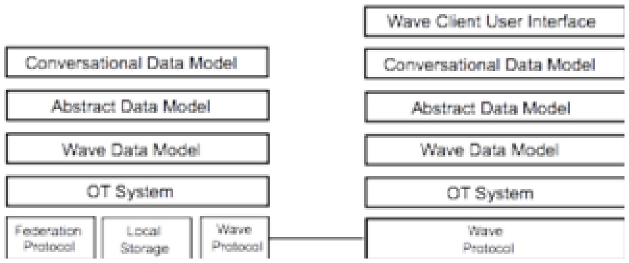


Fig. 1. Apache Wave Architecture, including data model layers.

1) The Wave Content Model

There are three different logical data models in the original Wave systems (Fig. 1). The Wave data model [23] is the basic level of data abstraction in the system providing a basic storage entity, Documents, and two aggregated entities: Wavelets and Waves.

Documents are XML documents where arbitrary data can be stored. They are logically grouped in a *Wavelet* which provides access control for the contained documents. Finally, Wavelets are grouped logically in *Waves*. A Wave is basically a unique identifier -for a particular domain- referencing a set of Wavelets which controls the access to a group of XML Documents.

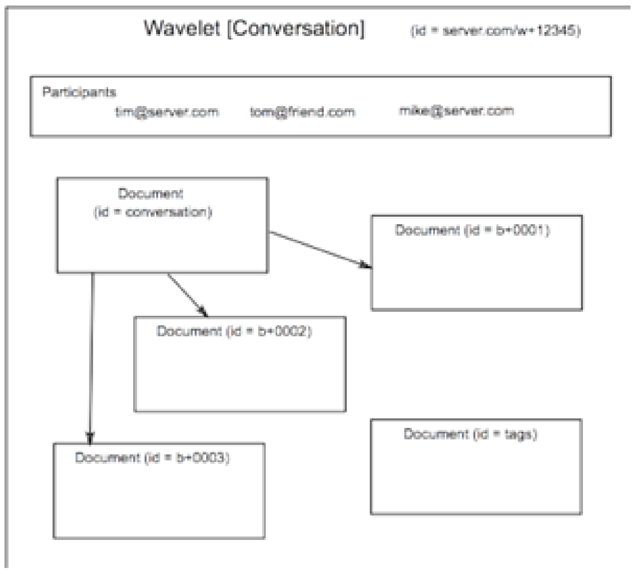


Fig. 2 Example of a Wavelet structure (Wave Data Model) representing a wave conversation (Wave Conversational Data Model)

The actual way to store these entities, and the Document's XML in particular, is through the historical set of changes performed to them. These changes are represented with a special set of character-based operations over a document: the Operational Transformations (OT) .

In the cases of having different users changing an entity at the same time, the OT's applied to the data entity through a special concurrency control algorithm ensures a consistent state of the entity, among all users, after all OT's have been applied. The OT system is responsible to implement such functionality. The implementation of the Wave Data Model allows to react when changes are performed over these entities thanks to this operation-based design.

2) *The Abstract Data Model*

In summary, the Wave Data Model enables only real-time collaborative editing of structured text (XML). However, it was convenient for the Wave system to handle non textual data as well. The Abstract Data Model provides a set of basic data structures –maps, lists and strings or Abstract Data Types (ADT)– which are represented as XML within Documents. This way, these data structures can be used by different users concurrently whereas they inherit the consistency properties of the underlaying OT system. Besides, the data model translates incoming OT's from the underlying data model in meaningful mutation events for data structures like "element is added", "element is removed", etc.

3) *The Conversational Data Model*

On top of these two layers, the Conversational Data Model [24] is placed. It provides the data entities and business logic of the original Google Wave product, focused on conversations.

A conversation is handled by a Wavelet, and each message is stored as a Document. The structure of messages is also stored in a Document but using the Abstract Data model instead: the logical structure of the thread can be seen as maps and lists of Documents' identifiers. The Conversational Data Model codifies the content's type of each Document within its identifier (Fig. 2).

These layers are deployed in a client-server architecture. The server side or "Service Provider" provides mainly OT history storage, OT system and federation control with other servers using the XMPP protocol [25]. Additional services like indexing and robots rely on the rest of already introduced data model layers. On the other hand, client side is responsible of the application logic and the user interface, therefore it handles all data layers as well.

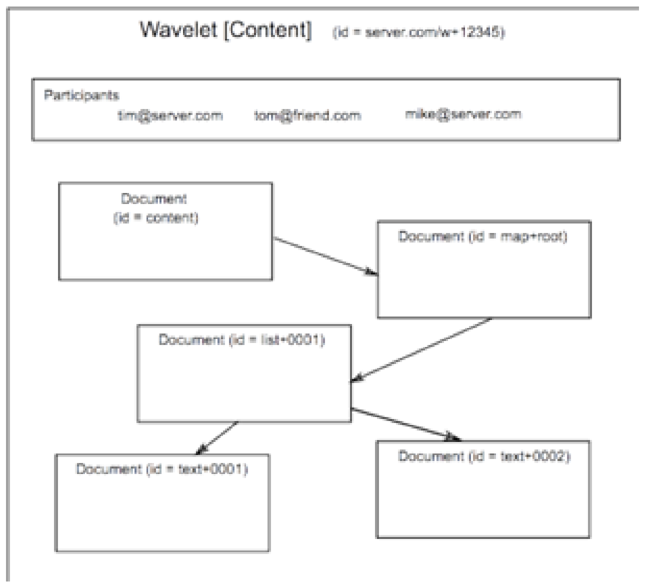
The implementation of this architecture is a Java/GWT software originally developed by Google. This technology allows to use almost completely the same source code for all layers in both, server and client modules. Java source code is translated to optimised JavaScript by the GWT compiler. Just a few and specific parts tied to the execution environment are different between server and client, such as networking and random number generation. The server-client communication between follows the Wave Client-Server Protocol. It defines a set of operations and JSON data entities to exchange Operational Transformations for Waves, Wavelets and Documents.

B. *General-Purpose Collaboration: Generalising the Wave Data Model & Architecture*

Previous section outlined the original Wave's data models and architecture. This section introduces how they can be used in a generic way thanks to the new Wave Content Model, and the Wave Content API.

1) *The Wave Content Model*

The Wave Content Model is a new general-purpose data model built on top of both existing Wave and Abstract Data Models. It provides a more convenient set of data abstractions and relationships to work with Abstract Data Types. This new data model allows to see a Wavelet as a dynamic tree of nested data objects: maps, lists, text strings and rich text documents. These objects are stored in different Documents of the Wavelet whereas the new data model manages the organization of them and their relationships among the Documents properly (Fig. 3).



Wavelet [Content] (id = server.com/w+12345)
Participants
tim@server.com
tom@friend.com
mike@server.com
Document (id = content)
Document (id = map+root)
Document (id = list+0001)
Document (id = text+0001)
Document (id = text+0002)

Fig. 3 Example of a Wavelet structure (Wave Data Model) representing a collaborative data object (Wave Content Model)

The Wave Content Model is implemented as a class hierarchy (Fig.4) controlling each possible data type –map, list, string and text– plus a controller class for the whole Wavelet, following the Composition Pattern [26].

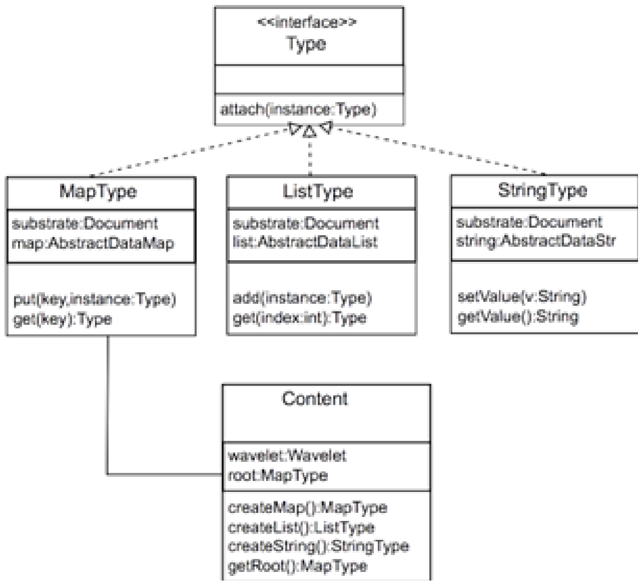


Fig. 4 Class hierarchy implementing the Wave Content Model.

A data class instance, or data objects, handles one single underlaying abstract data type instance over a single Document. New instances are initially unhooked from any Wavelet, so they must be attached to an existing parent instance. Attach process creates the underlying substrate Document, the right Abstract Data Type handler and stores the new Document identifier as reference in the parent instance. This classes allow to register callback methods to be notified on model mutations.

With this approach, Wavelets -and Waves- became generic and dynamic data containers where multiple users can create and modify a nested data structure at the same time ensuring its consistency over the time.

In comparison with the former architecture stack, in the presented approach the Conversational Data Model has been removed and replaced by the Wave Content Model. Of course, the existing user interface layer is also removed (Fig. 5).

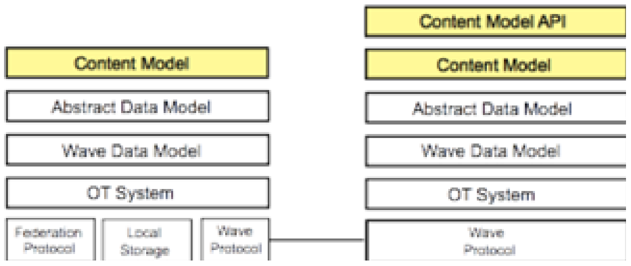


Fig. 5. New Apache Wave Architecture, including new content model

2) *The Wave Content API*

The new Wave Content Model allows to see Waves as real-time collaborative data structures. However, additional effort is required to expose this model to third-party applications in a handy manner.

According to the technology used in the Apache Wave implementation, just new Java or GWT web applications could use new content data model directly. With the aim of offering these new capabilities to any web application, 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 due to the following facts (among others): GWT-generated JavaScript, which is obfuscated by the compiler, does not provide references to objects with suitable names; GWT Exceptions do not flow out of the GWT code, so they must be translated and adapted to external code properly.

Java Script Native Interface (JSNI) and Overlay Types are features of GWT allowing to write arbitrary native JavaScript code integrated transparently within Java code. These features have been used to develop a native JavaScript layer which exposes functionality of the GWT-generated objects of the Wave Content Model. This is an implementation of the Proxy Pattern.

Additional functionality is also required in the JavaScript API. First, users no longer will use the former user interface to get registered or logged in. Therefore, the API provides replacement methods for making HTTP calls to create and authenticate users.

Management of the Wave life cycle now is provided through the API to clients. They can open or create Waves by calling API's methods. Moreover they can be aware of changes in the model registering callback functions in the API.

3) *Content Search Index*

Clients are able to query Waves stored in the Server Provider thanks to a new query service. Original Wave server implementation stores Wavelets as a sequence of OT's. This approach prevents to look into actual data of Documents to perform operations, for example executing search queries, regardless of the storage engine used.

A secondary storage is used now in order to provide a query service. Anytime the Server Provider commits a change to the main storage, an asynchronous indexing process takes care of the changed Wavelet: a full view of its Wave Content Model is generated in memory and a Visitor Pattern is used to transverse data objects generating an equivalent JSON document.

This process is optimised in two different ways: first, the number of times the indexing process runs is decreased by queuing committed changes sequentially and processing them in groups according their time closeness. Second, loading and transversing the full content model in memory is avoided by pruning. Each received change references to its target Document, which stores unequivocally one data object in the data model. This information is used to skip data model branches without changes in any of its data objects.

Finally, JSON documents are stored in the NoSQL database. The API encapsulates the database query interface and filters queries according to the current logged in user: a user cannot retrieve Wavelets where she is not a participant.

V. Discussion

This paper introduces the first federated platform for real-time collaboration available nowadays. However, using Wave involves some issues, mainly due to the limitations of the source code and its technologies.

There are several critiques concerning the complexity of the Wave OT system regarding two main issues: the complexity of the Operational Transformation system put in place [16] and the large length of the source code with around 500 thousand lines [27]. These facts together with the lack of good documentation causes the maintenance of the source code to be a tough task, requiring highly skilled developers in object-oriented programming with enough mathematical background. However, any OT system is inherently complex. To design a flexible and comprehensive set of operational transformations –such as Wave's– in order to provide an actually usable functionality is hard in any case. Besides, to implement control algorithms is a hard task, even if nowadays they are properly formalised.

Some existing OT implementations use a simpler approach. These OT systems are generally based in the JSON language, having a smaller set of OT operations just defined to operate at the language level. In contrast, Wave's OT system has significantly superior capabilities. It includes business logic operations in the system, such as add and remove participants to a Wavelet. But the most relevant features are to include XML tags and text annotations as part of the OT language. The first allows to handle any XML dialect, while the latter enables contextual meta data over that XML. These characteristics are used in the Wave's rich text format, which, for example, allows to embed arbitrary objects within the text, from images to widgets, just using new XML tags for them.

Operation's semantics and syntax of the introduced API follows the same style of the Google Drive Real-Time API: starting from a root map, new data objects must be created by a factory and then attached to the existing data tree. On the other hand, JSON based OT systems work seamlessly in JavaScript environments, allowing direct manipulation of the data. It is hard to conclude which approach is more appropriated, but the first seems more generic concerning the API implementation in different programming languages, as it is not as tied to JavaScript. Moreover, data structures of JSON documents and new Wavelet's inner structure are equivalent, so it would not be hard to develop adapters. However, currently there is no actual data about the developers preference, i.e. how comfortable are they with each approach.

Performance issues must be taken into account in the new Wave

Content Model. The first consideration is whether the new changes have a negative influence in the general performance of the platform in comparison with the original architecture. Regarding the client, no special impact in performance is expected as long as data objects of the new content model are created in memory only when access to them is required. On the server's side, no changes have been done affecting performance critical aspects of the OT system like in memory recreation of Wavelets and delta-based storage. However, current design of the JavaScript API duplicates some data structures of the underlaying data model to simplify the implementation. Internal improvements in this area could be performed, although they do not affect current or future use of the API.

The GWT development framework is sometimes seen as a disadvantage regarding efficiency and code complexity in comparison with development of native JavaScript software with modern native frameworks [28]. It is true that GWT was produced in a time when JavaScript tools and frameworks were not as advanced as today. However, it is a very stable and mature FLOSS project, and it is supported by Google. Moreover, the GWT compiler generates highly optimised code and it solves the issue of managing dual-language applications.

Client-Server communications relies massively on WebSockets [29] because changes in Wavelets are transmitted in both directions continuously. Protocol implementation is provided by an embedded Jetty HTTP server instance, a classic *Servlet* container which has been improved to support new HTTP features recently. It might be more efficient to use a non-blocking IO server [30] in order to improve vertical scalability. In addition, to use an embedded Jetty instance, prevents the deployment of the code into standard Java server containers.

Finally, it is necessary to assess the use of XMPP as a federated communication protocol among servers. It has been almost a standard for distributed communications in chat applications during more than a decade. However, the previous adoption from big players, such as Google and Facebook, has dropped. Moreover, it seems a heavy protocol to be used in small devices, and to support new features apart from chatting, especially in comparison with new decentralised protocols .

VI. Concluding Remarks and Future Work

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 of data and rich-text.

The provided API is a functional alternative to existing collaborative platforms. It provides a full-stack of software ready to be deployed, including functionalities only comparable with the proprietary Google Drive Real-Time API. Additional features such as the participation model, content storage and search index are part of the platform whereas they are missed in the rest of OT systems.

The API is offered in JavaScript and it can be used in any Web application. But thanks to the Java code base, it would be really easy to have versions for Java and Android applications. In such case, it would be an alternative to the lack of a Google Drive Real-Time API native client for Android.

From a wider perspective, this work opens new challenges in the context of decentralised collaboration:

In the introduced model, access and modification of content (and its structure) is granted to all participants in a Wavelet. However, this might not be enough for some sort of applications where read but not write permissions could be required for some users, e.g. a participant's profile information should not be written by anyone else whereas it must be readable by friend participants.

But also a fine-grain access control could be required beyond the current per-document access control. For instance, in a content Wavelet representing a poll, a user might be allowed to change her vote, but not to change others participants votes.

Under some circumstances integrity of the data model should be enforced, for instance allowing one and only one vote in the previous example. Or in a list of chess moves, enforcing the order and correctness of them.

Content Wavelets are highly flexible data entities for model application where the inner structure allows to define parent-child relationships of data elements. However, in any application, relationships among Wavelets or among inner objects of different Wavelets emerge naturally, so mechanisms to handle them must be explored, e.g. typifying Wavelets, object identification, etc.

Furthermore, in a scenario where several applications make use of the distributed data objects (for instance accessing profile information of users), the use of standard formats for data representation would be required. Technologies such as the Semantic Web [31] and Linked Data [32] provide an example of how distributed data can be organised and linked in a manner that allows further operations such as querying in a decentralised environment.

Current trends in software are driven by the mobile ecosystem. There, code and data are separated: *apps* running in devices, while retrieving data from a remote storage. Nowadays, it is easier to consider these apps managing data generated from different users and stored in different remote servers but eventually combining them in the device.

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 the reuse of parts of Apache Wave. The result is a platform ready to explore new challenges in decentralisation of data and services. We certainly hope this work will pave the way for other researchers and developers.

Acknowledgment

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] B. Leuf and W. Cunningham, The Wiki Way: Collaboration and Sharing on the Internet. {Addison-Wesley Professional}, 2001.

[2] "Collaborative communities for everyone! - Wikia." [Online]. Available: http://www.wikia.com/Wikia.

[3] B. Berliner, "CVS II: Parallelizing software development," USENIX Association., pp. 341–352, 1990.

[4] Google Inc. "Google Docs." [Online]. Available: https://docs.google.com.

[5] The Etherpad Foundation, "Etherpad." [Online]. Available: http://etherpad.org/.

[6] Sun, S. Xia, C. Sun, and D. Chen, "Operational Transformation for Collaborative Word Processing," in Proceedings of the 2004 ACM Conference on Computer Supported Cooperative Work, New York, NY, USA, 2004, pp. 437–446.

[7] C. A. Yeung, I. Liccardi, K. Lu, O. Seneviratne, and T. Berners-lee, "Decentralization: The future of online social networking," presented at the In W3C Workshop on the Future of Social Networking Position Papers, 2009.

[8] C. A. Ellis and S. J. Gibbs, "Concurrency Control in Groupware Systems,"

in Proceedings of the 1989 ACM SIGMOD International Conference on Management of Data, New York, NY, USA, 1989, pp. 399–407.

[9] "ACE - a collaborative editor." [Online]. Available: http://sourceforge.net/projects/ace/.

[10] A. Ferrate, Google Wave: Up and Running. O'Reilly Media, Inc., 2010.

[11] D. A. Nichols, P. Curtis, M. Dixon, and J. Lamping, "High-latency, Low-bandwidth Windowing in the Jupiter Collaboration System," in Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology, New York, NY, USA, 1995, pp. 111–120.

[12] Baxter, A. and Bekmann, J. and Berlin, D. and Gregorio, J. and Lassen, S. and Thorogood, S., "Google Wave Federation Protocol Over XMPP." Google Inc., 2009.

[13] G. Trapani and A. Pash, The Complete Guide to Google Wave. 3ones Inc, 2010.

[14] "Apache Wave Incubating." [Online]. Available: http://incubator.apache.org/wave/.

[15] Mozilla Labs, "TogetherJS." [Online]. Available: https://togetherjs.com/.

[16] J. Gentle, "ShareJS," Nov-2011. [Online]. Available: http://sharejs.org/.

[17] T. Chuanwu "Goodow - Google Docs–style collaboration via the use of operational transforms," GitHub. [Online]. Available: https://github.com/goodow.

[18] "Google Wave Protocol." [Online]. Available: http://www.waveprotocol.org/.

[19] R. Dewsbury, Google Web Toolkit Applications. Pearson Education, 2007.

[20] K. Chodorow, MongoDB: The Definitive Guide. O'Reilly Media, Inc., 2013.

[21] "Jasmine: Behavior-Driven JavaScript." [Online]. Available: http://jasmine.github.io/.

[22] P. Ojanguren, "SwellRT, a real-time federated collaboration framework." [Online]. Available: https://github.com/P2Pvalue/swellrt.

[23] A. North, "Wave model deep dive," 2010. [Online]. Available: https://cwiki.apache.org/confluence/display/WAVE/Wave+Summit+Talks

[24] G. North, A. J., "Google Wave Conversation Model," Oct-2009. [Online]. Available:

[25] http://wave-protocol.googlecode.com/hg/spec/conversation/convspec.html

[26] P. Saint-Andre, "Extensible Messaging and Presence Protocol (XMPP): Core," RFC Editor, RFC6120, Mar. 2011.

[27] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software Pearson Education, 1994.

[28] "The Apache Wave (Incubating) Open Source Project on Open Hub." [Online]. Available: https://www.openhub.net/p/apache_wave.

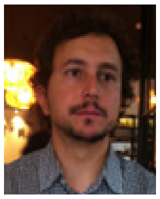
[29] T. Burnham, CoffeeScript: Accelerated JavaScript Development. Pragmatic Bookshelf, 2011.

[30] I. Fette and A. Melnikov, "The WebSocket Protocol," RFC Editor, RFC6455, Dec. 2011.

[31] Gregor Roth, "Architecture of a Highly Scalable NIO-Based Server" 2007. [Online]. Available: https://today.java.net/pub/a/today/2007/02/13/architecture-of-highly-scalable-nio-server.html.

[32] T. Berners-Lee, J. Ora, L. Ora and others, "The semantic web," Scientific american, vol. 284, no. 5, pp. 28–37, 2001.

[33] C. Bizer, T. Heath and T. Berners-Lee, "Linked data-the story so far," Semantic Services, Interoperability and Web Applications: Emerging Concepts, pp. 205–227, 2009.



Pablo Ojanguren (Oviedo, 1979) holds a Engineering degree in Computer Science (2003) and a MSc in Software Engineering (2006) from the Universidad de Oviedo (Spain). It is also a certified Project Manager Professional. He is currently senior software engineer and researcher in the EU-funded FP7 P2Pvalue project on the development of webtools for Commons-based peer production. He has been running different IT positions in international companies as Accenture, BBVA and YellowPages Group with special focus in content management systems, enterprise integration patterns and IT project management. Pablo's main research area is decentralised architectures in social issues as commons-based peer production, peer-to-peer participation, digital democracy and data privacy.

Antonio Tenorio-Fornés (Madrid) holds an Engineers's Degree on Computer Science (2012) and a Master in Computer Science Research (2013) by the Complutense University of Madrid (Spain). He is currently doing his PhD research on democracy tools for Commons-based Peer Production Communities and working as researcher and engineer in the GRASIA research group of Complutense University of Madrid as part of the EU-funded FP7 P2Pvalue project. His research interests include decentralized technologies, Commons-based Peer Production communities, Artificial Intelligence, Multi-agent Systems, Agent-Based Social Simulation and declarative programing languages among others. Antonio Tenorio-Fornés (Madrid) holds an Engineers's Degree on Computer Science (2012) and a Master in Computer Science Research (2013) by the Complutense University of Madrid (Spain). He is currently doing his PhD research on democracy tools for Commons-based Peer Production Communities and working as researcher and engineer in the GRASIA research group of Complutense University of Madrid as part of the EU-funded FP7 P2Pvalue project. His research interests include decentralized technologies, Commons-based Peer Production communities, Artificial Intelligence, Multi-agent Systems, Agent-Based Social Simulation and declarative programing languages among others.

Samer Hassan (Madrid, 1982) holds an Engineering degree in Computer Science (2006), a MSc in Artificial Intelligence (2007) and a PhD in Social Simulation (2010) from the Universidad Complutense de Madrid (Spain), together with a Diploma in Political Science (2006) from the Spanish National Distance Education University (UNED, Spain). He is currently Fellow at the Berkman Center for Internet & Society (Harvard University, US) and Assistant Professor at the Universidad Complutense de Madrid (Spain). He has carried out research in distributed systems, social simulation and artificial intelligence from positions in the University of Surrey (UK) and the American University of Science & Technology (Lebanon). Coming from a multidisciplinary background in Computer Science and Social Sciences, he has more than 45 publications in those fields. Engaged in free/open source projects, he co-founded the Comunes Nonprofit and the Move Commons webtool project, and has been accredited as grassroots facilitator. He's involved as UCM Principal Investigator in the EU-funded FP7 P2Pvalue project on the development of webtools for Commons-based peer production. His research interests include Commons-based peer production, online communities, distributed architectures, social movements & cyberethics. Dr Hassan currently belongs to the GRASIA research group, the Berkman Center for Internet and Society, the Editorial Board of the Society for Modelling & Simulation newsletter, and has belonged to the European Social Simulation Association and the Center for Research in Social Simulation. He has been member of Scientific or Organising Committees of 45 international conferences.