Project Brief

A Wavy Future - Project Brief

Overview

Apache Wave is a successor to Google Wave -> FedOne > Wave In A Box. Wave is a real-time communication tool. Wave is a server that hosts and federates waves, supports extensive APIs, and provides a rich web client. This project aims to rewrite the original implementation of Wave that is currently available at www.github.com/apache/incubator-wave. This rewrite is to ensure a scalable and universal OT server implementation to allow Apache Wave to be used for much more than just a communication-tool that is its only service available at the time. The client implementation of the currently available tool will be extended across multiple platforms to show solutions to OT editor implementations on varying devices.

please note the project proposal for a rewrite was given here: https://docs.google.com/document/d/1YnhcupFtReZyq5Y5QheIbYFO2epEhXGucNZE04r_oA4/edit.

Server:

The Apache Wave server is to be designed as such to be a authenticated server with inbuilt data redundancy to perform OT operations on Wave data structures. Wave data structures are customised xml-like documents in which custom tags can be made to allow for differnt uses of data manipulation in this video (https://www.youtube.com/watch?v=6ZqpeFydq4A) the decisions behind the xml-like language is explained. These structures are not exclusive to the conversational structure which can be viewed here: http://wave-protocol.googlecode.com/hg/spec/conversation/convspec.html but some projects outside wave have developed additional data structures like the Swellrt project. The development of these data specifications outside the servers implementation allow for a more general use OT (operational transform) server back end and allows clients to handle the implementations. This makes the server responsible for storage, authentication and consistency only.

Client:

Note: This will address the original use of Google Wave only.

Clients for Apache wave need some level of OT implementation to accept OT instructions from the server and change the document on the clients end. The current client is deeply intertwined in the server to reduce the duplication of code for this however it limits the possibility of native implementations on other clients. The goal is to produce a Rich text multi user panel & editor to allow display of read only documents and full read /write access documents that does **not** know OT and a OT layer which sits below the editor. In doing this it also allows mobile clients to match the compatibility of the web client, a desktop client could also be developed if needed.

With the client wave data structure being decoupled from the server this does open up other uses for wave outside the current client which with proper levels of documentation can allow many different applications be developed on top of the Wave server.

Project Objectives/Outcomes

Incubator Related:

The Apache Incubator is a place where young projects with basis can be further grown into top level projects at Apache, all new projects must go through the incubator before becoming top level. Most restrictions the incubator puts around graduation is about having a stable project with a community of developers and users.

- Build a community of developers and users, this is to ensure growth after the project has left the incubator and that the project doesn't lose traction.
- Have a stable source code repository.
- JIRA issues dealt with timely manner.
- Well documented, If no one can learn how to use Wave then how will it get used.
- License compliant. This refers to the legal issues with opensource licensing requirements.

Server:

- Basic Authentication.
- OAuth Authentication (Google, Facebook and so on).
- Account Authorisation Levels. (Admin, Basic, Guest and so on)
- Wave Document Authorisation Levels (Read & Write toggles per account type, except Admin level accounts)
- REST access to Wavelets.
- REST access to User Profile.
- WebSocket protocol (Wavelets and User Profile).
- Implement wave generic OT data structures.
- Apply OT operations.
- Apply Undo OT operations.
- Search.
- Federation Protocol.
- SSL validation for federation.
- Serve over http's independently (without nginx or apache sitting in front of it).
- request to ask server to hold updates for longer (bigger chunk of changes) when the server is struggling.
- Federation protocol between different external entities.
- Server-Server in-house (within same company) load balancing to deliver requests for a set of wavelets to be handled by server X instead of server Y.

Client:

- Define Document Structure. (if the original spec is to be changed, this may be done to define a document type like Apache-Conversation-v1)
- Editor
  - Read only mode (only cursors of editors and document updates).
  - Write enabled mode.
    - *Note: Ace editor has been recommended as an example of a good design and api (https://ace.c9.io/#nav=about).*
    - *Note: The editor is **not** to be OT aware.*
    - *Note: The editor is to be rich text aware.*
    - Copy, paste, undo interactions.
- Editor-OT-Layer
  - be able to translate OT operations to the editors API.
  - be able to translate editor callbacks to OT operations.
- Note: All Clients must stay relatively in sync development wise though the web client will generally be the most up to date.
- Display all active wavelets for the user.
- Display unseen wavelets (or ones with updates).
- Search wave's.
- Apply rich text (bold, italic, font)
- Add attachments
- Inline attachments (photos, videos)
- Responds to servers requests to have longer deltas to reduce over pinging the server.

Website/Documentation:

These goals only refer to the website which is currently available at https://incubator.apache.org/wave/index.html.

- New Design.
- Built using a static file generator or Apache CMS.
- Include Docs onto Website (html versions).
  - *note: https://jacobian.org/writing/what-to-write/ describes the types of documentation which should be written.*
- step-by-step tutorials.
- overviews, guides of OT and other concepts.
- reference material.
- Video tutorials.
- Wave related talk showcase page.
- Examples of Wave in the wild.

Project Depreciations

Due to Apache Wave coming from a already developed source code there are many features and other components that live within the current source. Other than the server and client there exists a few deprecated sub projects which will be addressed here:

PST Compiler:

The Protobuf String-Template compiler was used to generate json and protobuf builders for the wave project specifically. Though this project has uses outside the realm of wave and recently was separated in the source to remove it from Wave related source files in Protobuf's v3 a sub json module was implemented to reduce the need for this compiler.

Python API:

To the best knowledge of the original author of this document (Evan Hughes), I believe that the python client hasn't been functioning for some time and as such with a well made client-server API is unnecessary in the short term. Latter down the track making per language wrappers around the API would be a reasonable suggestion. As such the Python API should not be concentrated on till the server-client API is completed (or stable).

Old Client:

The current Wave client and other proposals which were made back when the project was initially open sourced had visions of the wave's being able to be floating windows within the browsers tab, this implementation would impact the rendering capabilities of the application. If the community wants the ability to open more than one wave within the same tab multiple columns may be the best approach instead of these floating windows. The original client also had support for gadgets which could be included into the interface, this currently isn't a priority for the core functionality so will not be included in this project brief.

Project Approach

Testing and Documentation is an important part of this project as the lack of documentation has hindered the previous source we must learn from those lessons when developing further to create a well structured and well documented system. An approach to this would be a code sprint -> documentation sprint cycle where features are implemented over a time period and after that period documentation is checked for inconsistencies and if so updated. This is to not promote not writing documentation while developing features (which is a need) but to make sure documentation on features does not vanish into the void. Larger pieces of documentation generally lags behind like the development of tutorials

Client:

Languages:

Typescript allows the developers to use high level object orientated principals and type safety for developing the client which then gets compiled down to plain JavaScript. https://www.typescriptlang.org/

React is the V in MVC and is used in large projects like the atom editor. https://facebook.github.io/react/index.html

For mobile each should be built in its own language for the platform (Android: Java, Iphone: Swift), a common shared c library could also be incorporated but may cause more issues than solutions.

Development Cycle:

The development of the client will lag behind the server in most cases, as such the initial development of the client should be focused on developing the panel/editor that is independent of the OT protocol layer. The OT protocol layer then can be developed after the editor has matured and will be matched with the development on the server to be able to translate OT ops to the editors ops. The editor should not be limited to wave only related uses and documentation and tutorials should be provided for the editor to be used in other examples, in essence treated as a separate entity to the project. The editor being built in house allows the project to maintain the rendering effects of the editor and how much memory it uses as on the wave's final client there may be a large amount of these editors on the screen at one time. This also allows the developers to gather skills which will be needed on the mobile platforms.

Server:

Languages:

Golang is a c like language which is garbage collected but features simple syntax and a beautiful standard library. It is built by Google and has a large growing community. Golang also has custom built debuggers for handling mutli-threading and great profiling tools.

Database: NOSQL database layer to sit below the server to store wave related data to allow for data consistency and structure, multiple db's can be supported but two main options should be Cassandra and MongoDB.

Development Cycle:

The server implementation of features will lead the client implementation to create a stable API before being implemented in the client. The servers only concerns are data integrity, authentication and document integrity; the database layer sitting below the server will be a nosql solution which can support its own data integrity. Suggestions for the database layer are Cassandra and Mongodb and a debugging file based solution which should not be used in production for performance and integrity concerns. Authentication should be developed to allow for Basic auth and external OAuth to be able to sit side by side or just one of the options.

Website/Documentation:

Languages:

Website: I (Evan Hughes) have no preference over which static website generator is used, a short explanation of some of the most popular can be found here: https://www.smashingmagazine.com/2015/11/static-website-generators-jekyll-middleman-roots-hugo-review/.

Docs: The current docs which can be found on github (https://github.com/apache/incubator-wave-docs) uses the Restructured Text Format and sphinx as the compiler. Sphinx is highly customisable and allows multiple outputs (pdf and html) to be generated which is a large benefit.

Development Cycle:

The more complex the project the more important full documentation is, with a consistent approach to delivering good quality documentation future developers and users can be drawn in to the Apache Wave Project.

Major Deliverables

Client:

- Web Editor read only mode (Panel mode).
- Web Editor read and write mode.
- Web Editor standard text (no rich text features).
- Web Editor rich text mode.
- Web Editor in line objects (images, videos, audio, links).
- Web Editor Spell checking.
- Web OT <-> Editor layer: translate OT ops to editor API calls.
- Web OT <-> Editor layer: translate Editor callbacks to OT ops.
- Android and Iphone Editors to follow same API calls or similar in design.
- Android OT <-> Editor layer: translate OT ops to editor API.
- Android OT <-> Editor layer: translate Editor callbacks to OT ops.
- Iphone OT <-> Editor layer: translate OT ops to editor API.
- Iphone OT <-> Editor layer: translate Editor callbacks to OT ops.
- Document structure design updated to include the document type.
- Basic Auth login.
- OAuth social login.
- Make websocket connection.
- Fetch list of available documents through rest and websocket.
- Register for updates for a document.
- Parse updates to OT <-> Editor layer.
- Send updates to server through websocket.
- Produce undo operations to send to OT <-> Editor layer.

Server:

- Connect to a database store.
- Load up a base config file and a optional override config file.
- Create necessary scheme in database system's.
- Multiple storage back ends through common interface (file, Cassandra, mongodb and others in the future).
- Basic authentication.
- OAuth authentication.
- OT ops.
- REST interface.
- WebSocket interface.
- Federation Protocol.
- Server-Server load balancing (Server X controls the operations on ABC wavelets, Server Y controls operations on DEF wavelets but each can have different clients but servers pass on messages).
- Profiling.
- Statistics.

Website/Documentation:

- New design.
- Transition current info on website to the new design.
- Transition to new build using a static web generator (probably middleware for extensibility, but see above).
- Add docs integration into the build system.
- Tutorials for Editor
- Tutorials for Server
- Tutorials for Client
- Document explaining how to create a OT<->Editor layer
- Developer tutorials for setup for development
- Developer code walk to explain directories and locations of items
- OT for XML like documents explanation and interactive tutorial on the website.
- Specific documents explaining each principal of the server
- Specific documents explaining each principal of the client
- Specific documents explaining each principal of the editor.
- Documents explaining the Wave data structures
- Documents explaining the conversation document format.

- Tutorial on how to design a new document format.

Time frames

Apache Wave has been in incubation state for many years and consistent progress needs to be made for the project to graduate from the incubator as a top level project. Depending on the amount of active developers will decide the amount of progress but A 2-3 year goal of completing all above requirements and deliverables may be achievable.