An approach to build P2P web extensions

Rodolfo Gonzalez[1], Sergio Firmenich[1,2] , Alejandro Fernandez[1], Gustavo Rossi[1,2]
Darío Velez[1]

[1]LIFIA, CIC, Facultad de Informática, Universidad Nacional de La Plata
[2] CONICET, Argentina
{rgonzalez, sfirmenich, casco, gustavo}@lifia.info.unlp.edu.ar

Abstract. Web extensions are currently the most frequently used mechanism for end-users to externally adapt and enrich the web. While most functionality offered by extensions runs on the browser, extensions that offer collaboration, complex computation, or massive storage rely on a centralized server. Relying on a server increases the cost of building, deploying, and maintaining web extensions (even small ones). This paper presents a novel P2P approach to build web extensions. It removes the need for a centralized server while hiding behind a framework the complexity of building P2P extensions. It uses a middleware to manage the resources offered by the browser so multiple P2P extensions can coexist, without degrading the browser's performance. This paper discusses the main challenges of building P2P web extensions, presents the approach, and shows its potential with a proof of concept.

Keywords: web extensions, peer to peer.

1. Introduction

From the perspective of the average web user, the web browser has not changed much since it was first introduced. Most of the browser's evolution over the years was aimed at handling richer and more interactive content, offering better security, and dealing with aspects of the web that most users are unaware of. Browser plug-ins (currently well-known as web extensions) were introduced to enable customization of the browser by third parties. Web extensions [1] are nowadays the *de facto* standard to customize the web browser and consequently augment the user's experience with the web. Web extensions can change the browser behavior, introduce changes to visited websites and also to provide new web pages delivered with the extension once installed in the Web browser. A web extension is made of a combination of Javascript, HTML, CSS, and configurations files. A well-defined API [1] governs the interaction of extensions with the browser and with the visited web pages. Extensions can communicate with external services (via HTTP requests), for example, to augment the visited web page with content from other sources, to store information on the cloud, to perform complex computation, and to support collaboration. In most cases, regardless of the complexity of the functionality they offer, such networked extensions are designed in a client-server architecture. This means that they depend on a server-side component/service, which increases the technical skills required to create the web extension. In addition, depending on a server component complicates maintenance and increases the costs associated with deployment for production.

We argue that building web extensions in a P2P style opens new opportunities to augment the web, especially when collaboration is involved, by removing the need for a server component. Following the P2P philosophy, web extensions are designed in order to allow users to collaborate by sharing the computation, storage and networking capabilities of their browsers, and by explicitly solving tasks for one another. In this article, we discuss the challenges that building P2P extensions present and outline our proposed approach based on a middleware that manages the resources offered by the browser so multiple P2P extensions can coexist, without degrading the browser's performance.

This article is structured as follows. Section 2 motivates the approach through an example. Section 3 outlines our approach, and Section 4 presents a proof of concept. Related works are discussed in Section 5. Finally, in Section 6 we conclude and reflect on future works.

2. Motivation: augmenting news portals with visualizations

Consider that we want to improve navigation in news portals by augmenting them with visualizations of its latest articles and related articles from other news portals. One alternative to creating such augmentation implies:

1. Harvesting the articles in the portals, starting from those featured on the homepage and recursively following the links to other articles in the portal, until a maximum level N has been reached.
2. Extracting for each article its author, date, media type, title, text, and links to other articles in the portal.
3. Characterizing each article by topic (for instance, using TF-IDF) and by sentiment (using an opinion lexicon), and then computing article similarity.
4. Building and presenting alternative visualizations of the articles (topics cloud, timeline, sentiment, etc.) that visitors can use to navigate.

The augmentation can be realized as a web extension. With the support of the extension, visitors annotate web sites' DOM to define how to harvest articles and their properties (for example, using the approach defined in [2]). They annotate the portal's home page (to identify entry points), and one article (that serves as a prototype). Upon visiting an annotated news portal, the extension automatically and transparently crawls those news portals for which an annotation is available, harvests the content, and builds or updates the visualization.

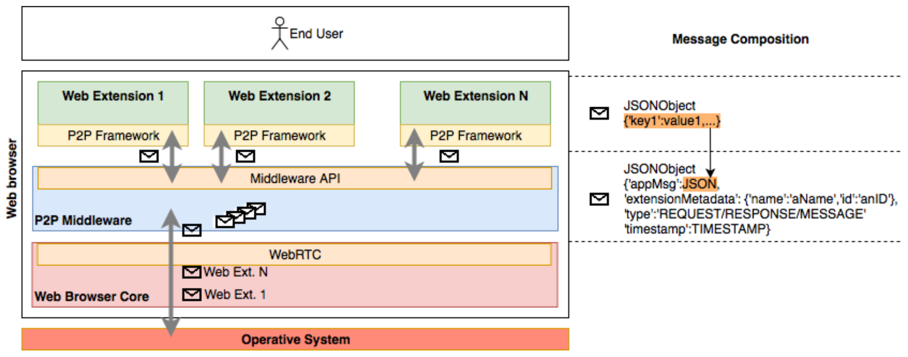
Simple as it sounds, building such an extension presents some challenges. For any mainstream news portal, the number of articles to process can easily be in the order of hundreds. On such scale, crawling and processing are both time-consuming and computation-intensive, rendering the visualization (and possibly the news portal) unusable.

Moreover, all this work would be performed by each visitor using the extension unless some collaboration mechanism to avoid unnecessary work duplication is used. To respond to these challenges, and assuming that many users will install the extension, we propose designing it in a P2P style. Doing so yields the following benefits:

- Extensions can collaborate to reduce time and workload. When an extension starts the process, it can delegate part of the crawling, harvesting and processing tasks to other available peers. In fact, performing these tasks would be unnecessary if a peer has already performed them and the results are available.
- Navigable visualizations of news articles as well as the results of harvesting (i.e., article's data), and processing (i.e., topic, sentiment, and similarity models) may be replicated and shared without the need of a central server.

3. The approach in a nutshell

To simplify development, and reducing development and execution errors, we separate different concerns into two supporting artifacts. First, a middleware that manages all P2P extensions installed in the browser, handling message exchange, and monitoring workload. Next, a framework that abstracts the key domain objects (such as message and peer), provides a clear interface to send messages and hides interaction with the middleware. Following this approach, developers do not require any other technical skill than those required to write any other web extension: JavaScript, HTML, and CSS and also the same kind of deployment process, i.e. just to install the extension in a Web browser.



```
JSONObject
{'key1':value1,...}
```

```
JSONObject
{'appMsg':JSON,
'extensionMetadata': {'name':'aName','id':'anID'},
'type':'REQUEST/RESPONSE/MESSAGE'
'timestamp':TIMESTAMP}
```

Figure 1: the approach in a nutshell

Figure 1 shows the overall approach. A web extension implements the P2P middleware and exposes the P2P API to any other web extension installed on the same browser. Other web extensions can directly execute functions in the middleware API via the message passing mechanisms [1]. However, the recommended way to interact with the middleware is via the P2P framework. Developers include the framework as a dependency of their extensions (it is JavaScript library). As it may be appreciated, all the messages pass through the middleware and are later routed to the corresponding web extension. For the concrete web extension, a message is a JSON object with the features the developers desire. When the messages pass through the middleware (using the behavior provided by our framework), it is encapsulated with the information required in the middleware layer (the type of message, timestamp, the extension's metadata among others), as Figure 1 shows at the right.

The P2P extension that delivers the middleware comes with a minimalist user interface, which allows the user to have control of the messages.

Communication between peers is currently based on WebRTC[1]. WebRTC brings real-time video and audio communication to the browser and can be used to transport other forms of content. It robustly solves the technical challenges in P2P communication. To establish a direct connection between peers in WebRTC, a discovery and negotiation method called *signaling* is used. It involves both parties connecting to a commonly agreed upon service to decide the mechanisms they will use to connect (as they may be located behind firewalls, in NAT'd networks, etc.). The signaling process can be implemented with any technology compatible with WebSocket/XHR. WebRTC depends on a commonly known signaling server that introduces a unique point of failure and turns the architecture into a hybrid P2P. However, it must be noted that our approach still removes the need for writing and deploying a specific server for each web extension. We consider this to be a good trade-off while we explore other alternatives.

3.1 Framework

The framework is packaged as a JavaScript library. Once included in the web extension project, the user must create a class that represents the application (for instance, *P2PNewsVisualization*), and make this class inherit from the extension point offered by the Framework, which is called *AbstractP2PExtension*. This extension point lets developers specify the behavior of their web extensions considering two communication modes: (a) to send a message to another peer without expecting a response, (b) to send a request message for which a response is expected and must be managed by the peer that made the request when it arrives. The following list presents the main aspects to be considered for using the *AbstractP2PExtension* extension point:

- The developer must instantiate the concrete class and send to the new instance the *connect()* message (it is inherited from the extension point), whose purpose is:
  - to send the *initialize()* message to the new instance, which is a method for which developers must offer concrete behavior in their classes, among other things, to set instance variables related to the extension's metadata (*name* and *id*), in order to uniquely identify the extension.
  - to initialize the P2P communication mechanism for the web extension.
  - to register the extension in the middleware.
- Developers may use other inherited behaviors to look for peers, and to send messages/requests to other peers:
  - *getPeers(callback)*: obtains the peers currently connected. Since this method is asynchronous, a callback function must be passed as a parameter.

[1] WebRTC. https://webrtc.org/ - Last accessed on January 5th, 2020.

  - *sendMessage(msg, peer)*: sends a message (first parameter) to a specific peer (second parameter).
  - *broadcasting(msg)*: send a message to all the peers.
  - *sendRequest(msg, peer)*: send a request message (first parameter) to a specific peer (second parameter). It is expected to receive a response.
  - *sendResponse(msg, peer)*: send a response using a message (first parameter), and to a specific peer (third parameter). In this case, the *msg* (a *JSON* object) should be populated with further information about the original request.
- To handle messages and requests according to its needs, the extension must implement some of the following methods (or all of them):
  - *receiveMessage(msg, peer)*: this method will be executed when a new message is sent to the extension. It is not expected to deliver a response. It receives the message as a first parameter and the peers who sent it as the second parameter.
  - *processRequest(msg, peer)*: this method will be executed when the extension receives a request. It is not expected to create and deliver a response during the method execution. This method is suitable for human (interactive) collaboration. Its response depends on the user's interaction which occurs asynchrously.
  - *automaticProcessing(msg, peer)*: this method will be executed when the extension receives a request and this request was marked as *automatic* (it is just a flag in the message). This method must return a JSON object intended to be used as a response, and the framework automatically delivers it when the method finishes. This method is specially designed for computing collaboration, that can be automated, i.e. without depending on user intervention.
- If the extension sends requests, it must implement the *processResponse(msg, peer)* method to manage the responses to the requests previously done.

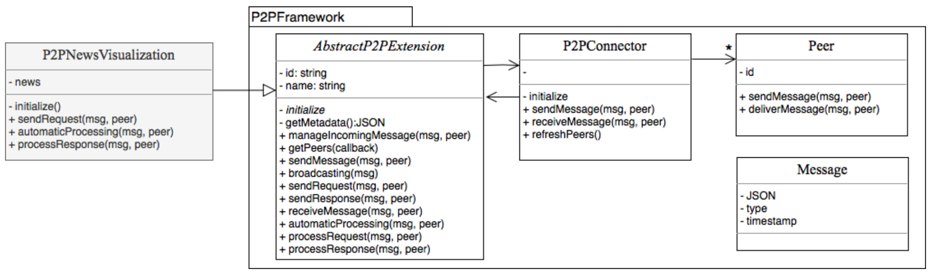


Figure 2. The framework and its extension point, the class AbstractP2PExtension

Figure 2 shows a simplified version of our framework plus another class showing how to inherit from the extension point, named AbstractP2PExtension. The P2PConnector class is the one that uses the middleware API. Two other classes provide simple abstractions for the peer and the message. The Message class is managed by the AbstractP2PExtension and the P2PConnector objects, meanwhile, the concrete class representing the web extension (P2PNewsVisualization) always work with the JSON Object defined by the developer.

4. Proof of concept

As proof of concept, we have developed a variation of the P2P extension discussed in Section 2. In this case, we have chosen to collaboratively perform the scraping and the processing of news articles. Note that in the P2P extension, the collaboration among peers could be organized with different granularity levels, which at the end depends on developers' decisions. For the sake of space, in this example, we use a coarse granularity to simplify the source code shown in Figure 3. The main idea is that the peer that starts the process (*pA*) requests another peer to handle an extraction template. The extraction template includes a news portal home's URL, the annotations to extract the news on this home page, and the annotations to extract information for a particular news article. In this way, each peer will be requested to start the scraping process from a particular news portal, applying the same extraction template to each news article crawled. As a result, each peer will follow a different crawling and scraping path. For each news article, the peer must also extract the topics (basically, calculating the most relevant words) and then compute the sentiment analysis. Then, the extracted and processed news articles are sent to *pA* together with their metadata.

Figure 3 shows an excerpt from the source code (focused on those aspects related to our approach), whose main class is shown in Figure 2, named P2PNewsVisualization. The source code example is illustrative, so we do not take into account in this excerpt the number of peers in relation to the number of news portals.

```
class P2PNewsVisualization extends AbstractP2PExtension{
    startCollaborativeScraping(){
        this.getPeers(this.collaborativeScraping);}
    collaborativeScraping(peers){
        templates = this.restoreExtractionTemplates();
        for (peer in peers){
            ...
            msg = {'extractor':templates[j], 'automatic':true};
            this.sendRequest(peer, msg);
            ...
        }
    }
    processResponse(msg, peer){this.storeNews(msg.news);}
    automaticProcessing(msg, peer){
        let news = this.processWebSiteExtractor(msg.extractor);
        return {'news':news};
    }
    processWebSiteExtractor(templateExtractor){
        let news = this.extractNewsFrom(templateExtractor);
        news = this.augmentNewsWithTopics(news);
        news = this.applySentimentAnalysis(news);
        return news;}
}
```

Figure 3. Source code excerpt for the news visualization scenario.

For the proof of concept, we defined extraction templates for three news portals. We execute scraping and processing of news considering two scenarios:

- Stand-alone: *pA* processes all the available extraction templates.
- P2P: *pA* processes one extraction template and delegates the rest of the available extraction templates to other peers (one extraction template per peer).

A total of 228 web page's DOMS were processed. For the stand-alone scenario, it took 298 seconds. For the P2P scenario, which was based on three web browser (*pA* plus two peers), it took 120 seconds. Although the difference is remarkable, we did not take the best advantage of the P2P approach, given that using more peers and more fine granurality in the tasks requested, probably would improve the total time.

5. Related works

To the best of our knowledge, there are two well-known applications of P2P in web browsers. First, there are approaches to support collaborative computing. For instance, Pando offers a platform in which a user must install a server and run it in his own machine. Then, other users may access this back-end application with their browsers to offer it for computing [5]. On the other hand, it has been proposed to use the browser as a distributed platform for content delivery [6][9]. Over this idea, a work studies the use of a communication protocol [7] that improves how to program over WebRTC. [10] proposes a generic distributed application server which is also currently supported by existing web browsers such as Beaker Browser [11]. Other approaches use P2P communication for specific aims, such as improving virtual environments [8].

Although these works show that decentralizing the Web is a current topic, these are far to be applicable to web extensions with the final goal of improving the overall user's web experience. Server-side support for web extensions was already studied and analyzed [4], in which authors propose a Model-Driven Web Augmentation approach to model back-end requirements. Although the complexity for developing, deploying and maintaining the back-end component is clearly better than using an ad-hoc approach, we believe that a P2P approach based exactly on the same technology required for programming web extensions is a more suitable and convenient way, at the same time that it removes any need of a centralized server application.

6. Conclusions and Future works

External Web structures (i.e. "defining hypermedia structures externally of the involved documents" [3]) are software artifacts that improve the overall Web experience. Web extensions are the most common and convenient way to develop and deploy this kind of software. Without an intermediate server, a web extension cannot communicate to the same web extension installed in another user's browser. Even more, when some communication between different web browsers is required, new technical barriers appear (for instance, dealing with back-end technologies beyond HTML, CSS, and JavaScript). Sever-side support has been very important for different reasons [4].

This paper presented an approach build P2P web extensions, which aims to eliminate the need for a centralized server to communicate web browsers and users. A signaling back-end service has been designed and implemented. It may connect peers for any web extension or for a specific one without requiring changes on it, neither on the P2P web extensions source code because it was conceived as a generic single-purpose (to connect peers) platform. Although we believe that our approach improves the potential of web extensions without requiring a centralized application, we still need to create and evaluate more scenarios. For instance, pervasive and distributed storage should be supported by the framework. However, we already could apply our approach in several scenarios. Besides future evaluations and experiments in this regard, it is also mandatory to study which is the power of a P2P web browser, as well as how to continuously measure and limit this kind of collaboration in order to not spoil the overall Web experience.

References

1. Browser Extensions. Draft Community Group Report 23 July 2017, https://browserext.github.io/browserext/. Last accessed on January 20th, 2020.
2. Bosetti, G., Firmenich, S., Rossi, G., Winckler, M., & Barbieri, T. (2016, June). Web Objects Ambient: an integrated platform supporting new kinds of Personal Web experiences. In International Conference on Web Engineering (pp. 563-566). Springer, Cham.
3. Bouvin, N. O. (2019, September). From NoteCards to Notebooks: There and Back Again. In Proceedings of the 30th ACM Conference on Hypertext and Social Media (pp. 19-28).
4. Urbieta, M., Firmenich, S., Bosetti, G., Maglione, P., Rossi, G. & Olivero, M. MDWA: A Model-driven Web Augmentation approach - Coping with Client- and Server-Side Support. Software and System Modeling, 2020, in press.
5. Lavoie, E., Hendren, L., Desprez, F., & Correia, M. Pando: Personal Volunteer Computing in Browsers. In Proc. of the 20th International Middleware Conference(pp. 96-109), 2019.
6. Kobusińska, A., Wolski, A., Brzeziński, J., & Ge, M. P2P Web Browser Middleware to Enhance Service-Oriented Computing—Analysis and Evaluation. In 2017 IEEE 10th Conference on Service-Oriented Computing and Applications (SOCA) (pp. 58-65).
7. Tindall, N., & Harwood, A. (2015, September). Peer-to-peer between browsers: cyclon protocol over WebRTC. In 2015 IEEE International Conference on Peer-to-Peer Computing (P2P) (pp. 1-5). IEEE.
8. Koskela, T., Vatjus-Anttila, J., & Dahl, T. (2014, March). Communication architecture for a p2p-enhanced virtual environment client in a web browser. In 2014 6th International Conference on New Technologies, Mobility and Security (NTMS)(pp. 1-5). IEEE.
9. Vogt, C., Werner, M. J., & Schmidt, T. C. (2013, October). Leveraging WebRTC for P2P content distribution in web browsers. In 2013 21st IEEE International Conference on Network Protocols (ICNP) (pp. 1-2). IEEE.
10. Jannes, K., Lagaisse, B., & Joosen, W. (2019, March). The web browser as a distributed application server: towards decentralized web applications in the edge. In Proceedings of the 2nd International Workshop on Edge Systems, Analytics, and Networking (pp. 7-11).
11. Beaker Browser, https://beakerbrowser.com, accessed: 1-29-2020.