November 30th, 2018

Chrome Proposed Feature Report

Authors
Brynnon Picard (15bdrp@queensu.ca - #20005203)
Roy Griffiths (18rahg@queensu.ca - #20137434)
Alex Galbraith (18asrg@queensu.ca - #20135646)
Sam Song (15shs1@queensu.ca - #10211857)
Bradley Kurtz (15bdk2@queensu.ca - #20020794)
Dongho Han (16dhh@queensu.ca - #20027554)

1.0 Abstract

This report describes our proposed function, its impact on the concrete architecture of Google Chrome and explains the rationale of choosing one of two approaches proposed. In our previous report, we obtained our final conceptual architecture and concrete architecture from the documentation of the open source analogue and Chromium source code. This allowed us to identify major subsystems and crucial design elements such as Chrome's multi-process architecture, use of implicit invocation, and Renderer/Browser decoupling. Through this process and running through use cases allowed us to identify the concrete architecture of Chrome.

In producing this report we examined the concrete architecture and how our proposed program can be implemented into the architecture. Our proposed program suggests predicted words once the user stops typing for the specified time. It will involve the Browser and Renderer subsystems for calculating the prediction algorithm and displaying a popup for the suggested words. We suggested two different approaches for predictive text, approach A and approach B, for implementing the function into the architecture. SAAM analysis was performed on both approaches, and approach A was chosen due to approach B's non-functional requirement violation.

Ultimately, our concrete architecture remained, and architecture is still Object-oriented, multi-process architecture supported by an implicit invocation for communicating between the Browser and Renderer. Despite all the additional functionality implemented, our architecture has no additional subsystems or dependencies.

2.0 Introduction

In this report, we set out to solve a long fought battle with productivity. We propose a new feature for Google Chrome that will increase user productivity by providing tailored text prediction and spell checking. Before we dive into the implementation, we must briefly discuss the architecture of Chrome which is displayed in Figure 1 below. Chrome has an Object-oriented, multi-process architecture, supported by an implicit invocation for inter-process communication. The most important subsystems for this report are the Renderer and Browser subsystems. The Renderer deals with the rendering of pages and includes Blink, a standalone engine that processes HTML and CSS, and V8 which processes Javascript. There can be multiple instances of the Renderer, each which run in their own process. The Browser manages the Renderers and all other components. Inside the Browser is the Kernel which is the brain of Chrome and manages data flow between components. Data persistence deals with local and remote data storage. Mojo is used to communicate via inter-process communication with the Renderers. Finally, the UI displays the user interface.

Our feature would appear in a popup above the user's caret (see Fig 2. below). The feature should not reduce the user experience by causing lag, nor should it endanger the user's security. We propose two approaches for implementing this feature. In the first, processing the current text the user is typing and producing suggestion is performed in the Browser kernel, while rendering the popup is performed in the Renderer's Blink module. In the second approach, the Renderer's Blink module performs both the word prediction and rendering (See Figure 1). In both approaches, Data Persistence in the Browser, and V8 in the Renderer are used for storing data and acting on the popup interactions respectively.

By performing a SAAM analysis, we were able to decide which approach was optimal. We analyzed each approach with respect to the stakeholders: Google, Google shareholders, Developers, and Users; and the NFRs: performance, portability, interoperability, management, and security. We found that while approach B produced better performance and had lower management costs, it also exposed a security risk which would be unacceptable to our stakeholders. Thus we settled on approach A.

After running through the primary use cases and creating some sequence diagrams, we found that neither approach had an effect on our architecture. The majority of the work would be in implementing the text prediction and data structures. In our selected approach, both of these functionalities would be located in the Browser subsystem, thus Chrome's browser team would be the most effective team to work on the project. Additionally, since Android (Owned by Google) already has text prediction, it may be beneficial for these teams to collaborate on text prediction.

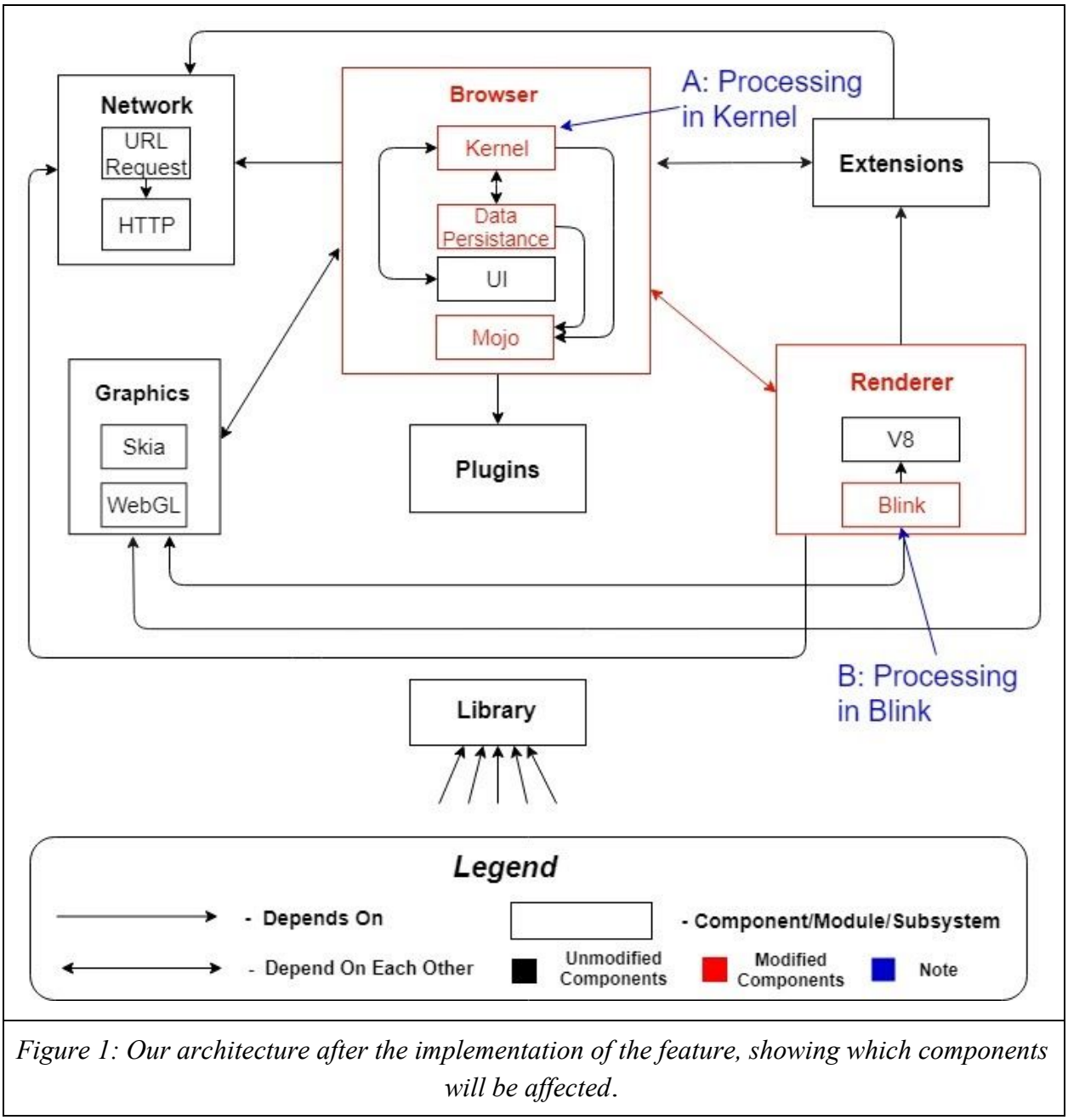

Figure 1: Our architecture after the implementation of the feature, showing which components will be affected.

3.0 Proposed Feature - Predictive Text

3.1 Overview

The feature we are proposing is a word suggestion popup which will appear when a user is typing into an input field. This is similar to what might be found on an Android or iOS device when the keyboard is open. Figure 2 shows a mockup image of what the feature would look like

when active. Here, the user has typed in some words and the popup is displaying some possibilities for words the user might want to choose, and if one is selected it will swap out the word being typed by the user. This feature would employ a local dictionary of words, which would be customized to the user based on the words they are most likely to use.

This feature will provide an enhanced user experience by improving user productivity when typing. It can help users to figure out what word they're thinking of but aren't sure how to spell or to choose word predictions while they're partway through typing a word to speed up their typing speed.

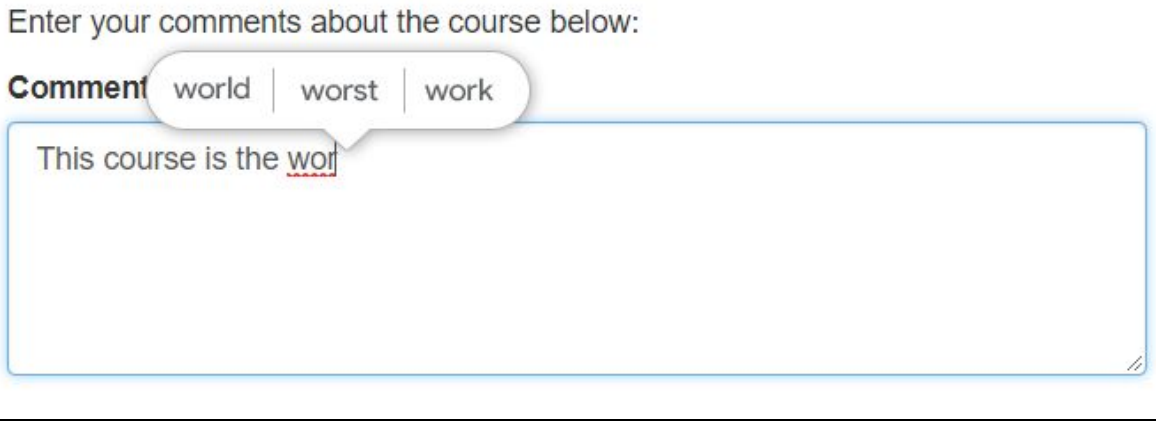

Figure 2: A mockup of the proposed feature.

3.2 Functional Requirements

We identified several functional requirements for our feature. An interactive popup should appear when a user begins typing in an input field which will appear above the user's cursor, displaying a maximum of 3 suggestions. It should move with the user's caret, and if one of the suggestions is selected, it should replace the current word that the user is typing. When the user pause for specified amount of time (< 0.5 seconds), the popup should suggest commonly used words. These word suggestions would be based on the spelling of the word, and would look for similarly spelled words or, if the user has misspelled what they're typing, the correct version of the word. The feature will maintain a customized prediction data structure based on the user's inputs across time, which would be synced with the cloud once per day to update the prediction network across all devices on the user's Google account.

3.3 Non-Functional Requirements

In terms of performance, the popup should populate with suggestions and be prepared to display in under 500 milliseconds. This will ensure the feature won't be too resource-intensive, but will still be useful when the user stops typing to consider their options. This is the same amount of time the popup waits to show anyway. For security, the new feature should not introduce any

new security vulnerabilities. From a managerial standpoint, a functional version of this feature should be possible to complete in under 12 business days by 2 people. Of course, there will need to be quite a bit of time to properly test this feature as well. For portability, this feature should work across all desktop versions of Chrome, while mobile is not necessary as iOS and Android already have a similar feature. And finally, for interoperability, user data should either be stored as or exportable as XML so that it can be easily synced across platforms.

Evolvability: Our program is by definition specifically intended to evolve well. The words that the feature will predict is based data structure which stores most used words and all the previous words typed by the user. As time goes on the prediction logic will increase in accuracy.

Testability: The predictive text and popup modules should be implemented as standalone functionalities and be glued together into our functionality via glue code. Each of the these modules can then be unit tested on their own, leading to good testability. The visual popup module is slightly harder to test as at least part of it requires a person to verify the visuals look correct.

3.4 Testing

Prediction Use Case: Train or populate a text prediction algorithm based on the first 75% of a large text source. This could be from a book, or perhaps private emails from Google's email servers. Next the predictive text is tested on the remaining 25% of the text, recording the percentage of correct predictions. The required rate of prediction depends on whether the word is partially complete, its position in the sentence and the frequency of the word in general.

Spell Check Use Case: Spell check is already implemented in Chrome, ensure that it is working as intended in the new feature.

Sync Use Case: Sign into two devices. On one device, populate the predictive text with an unusual set of data, e.g., the script of Shrek. Open Chrome on the second device. The text predictions on the second device should now resemble the text used to populate the first device. (E.g. when writing "Get out of my", one of the next words predicted should be "swamp!")

UI Use Case: Ensure popup functions correctly on a wide range of inputs. This should be tested on both simple input forms, and more complex Javascript based editors such as Google Docs.

4.0 Proposed Approaches

Initially, we had two approaches which would be able to achieve the functionality of our proposed extension. The two approaches we made were very similar and the differences only consisted of where a certain method would be applied. However, although they only differed in this aspect, the effects they had on the NFRs were significant. This will be described in great detail below in our SAAM analysis.

Approach A: Here we started with Blink sending the text being input into the system to the Kernel via IPC through Mojo. Then Kernel would retrieve prediction data from Data Persistence so it could apply this data on to our input text to create some suggestions to what is being typed. These suggestions would then be returned to the Blink via IPC through Mojo so that it is able to display these suggestions on screen to the user so they may select which is most appropriate. We have included a detailed sequence diagram later in this report.

Approach B: As mentioned before, the only difference between the two is where the prediction data is applied. So, in our first call from Blink to Kernel, instead of sending the input text to Kernel, it simply asks to retrieve the prediction data. Once the prediction data has been retrieved, Blink now applies this to the input text to create the suggestions itself and thus display them on screen to the users.

To describe what the prediction data stored within Data Persistence is, it is essentially a local dictionary. As well as the this, because our extension also considers the previous words typed in the sentence, the data must contain a hash table of all words which come after one another. As well as the hash table we decided to consider words which the user types often. To achieve this, the data contains a weighted graph according to how often a user has selected it. Thus if you type a certain word often then it is likely to offer that word as a suggestion if you start typing the letters contained in it. This will update once a user has selected a certain suggestion. We have described the process in greater detail in a sequence diagram later on.

4.1 Concurrency

Our feature will run concurrently with the main rendering thread. This is important as it will improve the overall performance of the system since it will not block the main rendering thread. And if the feature does slow down considerably the user's browsing experience will not be impeded. While the feature is enabled, it will wait for a pause in user typing and then a dedicated prediction thread will be stimulated to retrieve text suggestions. This thread will then be given a callback to the Renderer to display the text suggestions on the page.

5.0 SAAM Analysis

Below is a table analyzing the relationship between our NFRs and approaches:

Attribute	**Approach A**	**Approach B**
Performance	Performance of approach A is significantly affected due to the distribution of code, more IPC communication is required to perform prediction everytime	Due to the localized code, the performance of approach B will be better than approach A

Interoperability	Good interoperability	Good interoperability
Portability	Good portability	Good portability
Management	Worse management because Browser and Renderer are both a big part of the implementation, requiring teams to work on both areas.	Good management. Requires less effort than approach A because this implementation is mostly Rederer focused, requiring only one team.
Security	All critical user data is stored inside data persistence, which is safer compared to other programs	All code and data are stored inside the renderer subsystem. As the renderer will run any script given to them, approach B will have a higher chance of being compromised

We have displayed the related stakeholders according to each non-functional requirement in a detailed table down below:

Attribute	**Chrome User**	**Development Team**	**Google**	**Google Shareholder**
Performance	Performance is the most important metrics for general users, people will be frustrated	Performance of predictive text needs to be on par with other chrome function for consistency	Google is expected to make high-quality ware, the performance of predictive text needs to be above average to continuously meet the expectation	If Google creates low quality software, the market expectation will decrease which will cause a decrease in the price of share
Interoperability	N/A	Increased interoperability makes the feature easier to modify in the future and leverage for other projects.	Data is potentially useful for other evil privacy invading projects, which Google loves. Interoperability make the data easier to use for other projects.	Privacy invading projects made possible by data Interoperability could be great for profits. Shareholders love profits.

Portability	Some chrome users use multiple platform, chrome will be more convenient if functionality remains the same	The feature needs to be able to work on all devices of Chrome, including mobile versions, correctly.	Google is expected to make high-quality ware and need to give users a seamless experience across all versions of Chrome.	Google's approach of tackling all major OS will attract more users which will result in higher profit
Management	N/A	The Development team cares greatly about management. The more efficiently the project can be completed, the better.	Better management means more efficient production of the product, and thus better profit margins.	Better management means more efficient production of the product, and thus better profit margins.
Security	Chrome can hold private data and some users will regard security over performance	Google chrome is known for their excellent security, implemented function need to meet the expectation	Google has multiple privacy issues, if the program's security is breached, people will stop using their browser as thee are are multiple alternatives.	If Google creates software with low security and a major security breach occurs, the market expectation will decrease which will cause a decrease in the price of share

Approach B will have higher performance as it requires less IPC communication; however, it is not really a major issue as the user does not need instant feedback as long as their normal input is not affected. Approach B will be faster to implement and easier to maintain since all the code will be in one localized place and the Blink subcomponents of Renderer already contains the SpellChecker module thus it makes sense to add the predictive text in the same place as in Approach B. A compromised Renderer in Approach B might be able to access predictive text data such as word usage frequencies. This is a large violation of our NFR centred around security. Approach B requires each Renderer to maintain a cached copy of the predictive text data which, depending on how the prediction is implemented, could be a fairly large (e.g., dictionary + set of neural network weights). Both implementations will have to be wary of portability but should rely entirely on other abstracted code in the codebase, so portability shouldn't be an issue. Both approaches allow for the processing to be run asynchronously, preventing user experience from being affected. While Approach B will be faster, it directly violates one of our NFRs, specifically our security requirement. This violation outweighs the additional performance and development costs. An implementation that exposes security risks would be detrimental to users, and due to its effect of public opinion, would be poorly regarded

by shareholders and Google. Ultimately the additional performance and labour costs in A are outweighed by increased security, thus we have chosen A as our best approach.

6.0 Sequence Diagrams

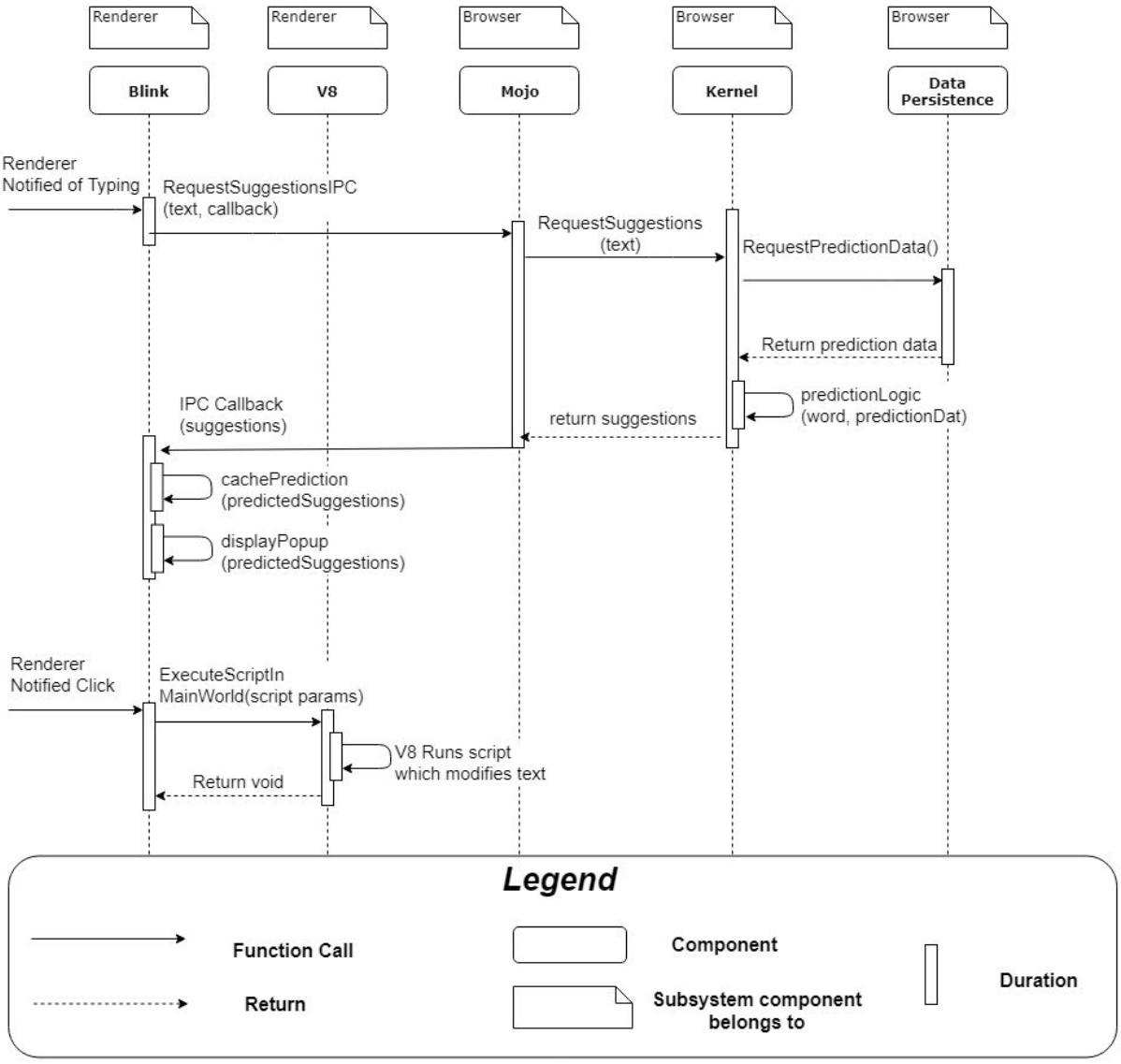

Figure 3: Sequence diagram of requesting, caching, and displaying suggested words, followed by a sequence in which the user has clicked a suggestion, and our feature modifies the typed text.

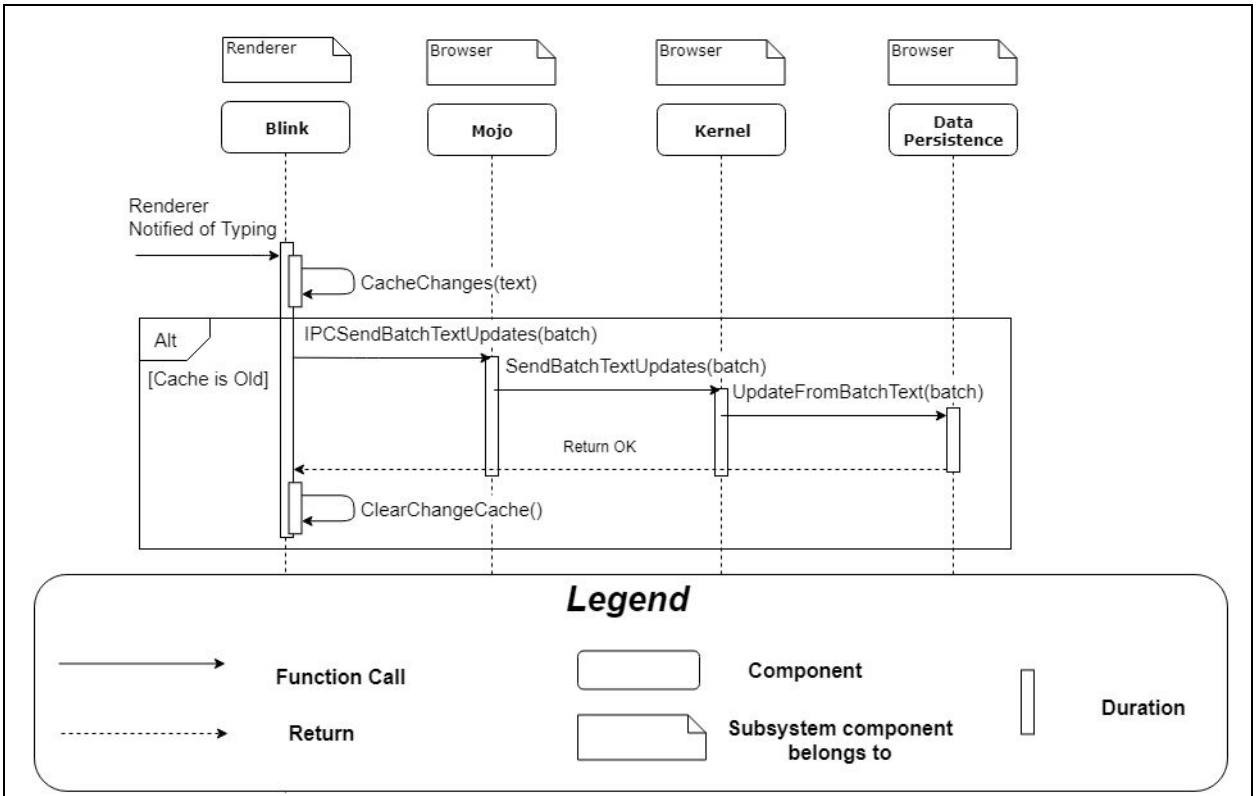

Figure 4:Sequence diagram of the Renderer caching and sending batched changes to the Browser to update prediction data.

The above sequence diagrams display the two major groups of sequences. In the first (Figure 3) we see the sequences that occur when the user types and uses the popup to modify their text. In the second (Figure 4) we see the sequence that relates to the caching of text modifications.

Most importantly, both sequences are consistent with our original architecture. No changes need to be made to the architecture in order to implement this feature.

7.0 Affected Directories

7.1 Rendering Popup

Implementing the popup would require modification to third_party/blink/renderer/core/input/ to handle input events and begin the chain of actions. Modifications to *third_party/blink/renderer/core/html/forms* may be helpful for streamlining the access of input related data. Blink's Mojo hooks should be modified to facilitate new messages regarding the prediction requests at *third_party/blink/renderer/core/mojo*. For the actual implementation of the popup, *third_party/blink/renderer/core/dom* and *third_party/blink/renderer/core/page* would be modified.

7.2 Data Format

While the data format must be carefully designed, Chrome has generic methods for storing abstract data so it is unlikely that there will need to be any modifications made to Data Persistence itself.

7.3 Text Prediction

A new module under *chrome/browser* would need to be added to encompass text prediction. This should be implemented as a standalone module. New mojo hooks would need to be added to handle IPC prediction requests.

8.0 Team Issues

Since we chose Approach A, the Browser subsystem will take the most work to implement and therefore it is likely the critical path of developing our feature. Thus, the optimal team to work on the implementation is the Browser team. Also, the developers working on this feature may want to communicate with the Google developers who worked on Android. The Android keyboard already has a similar predictive text feature, so those developers would have a wealth of knowledge in this domain.

9.0 Limitations and Lesson Learned

Throughout the duration of this project, our group encountered several limitations that needed to be overcome.

- We found that there was no publicly available reference architecture for typing suggestion systems, which meant we had to figure out the implementation details ourselves. This took valuable time and made it difficult to tell if we were going in the right direction.
- There were also only a couple of subsystems that would make sense to use for this feature, so there were a limited number of ways that we could possibly implement it.
- There was a limited amount of information online about conducting a SAAM analysis, which made it difficult to understand what we were meant to do on that front. There was no information on the slides nor in the readings which meant we had to do a great deal of research on how to conduct an analysis we were never taught.

That being said, we also learned some valuable lessons while working on this project that we believe could be applied for our careers in the future.

- We realized that in a well-developed architecture, implementing new minor features shouldn't require major architecture redesign. As we learned from the course, redesigning

an existing architecture is very expensive and time-consuming. This could also cause major stability issues which could compromise the whole architecture.

- Lastly, we also learned that there are always trade-offs in the possible implementations of any software feature as no one method will be the best in all cases.

10.0 Conclusion

We proposed a feature to increase user productivity by providing tailored text prediction and spell checking. These features would appear in a popup above the user's cursor. The feature should not reduce the user experience by causing lag, nor should it endanger the user's security. The popup will provide the user with word suggestions as they type, somewhat similar to the ones found on smartphones with the added capability of correcting spelling. We've created two possible approaches to our proposed feature for our extension. The two approaches we made were very similar and the differences only consisted of where a certain method would be applied. While Approach B will be faster, it directly violates one of our NFRs, specifically our security requirement. An implementation that exposes security risks would be detrimental to users, and due to its effect of public opinion, would be poorly regarded by shareholders and Google. Ultimately, we believe that this violation outweighs the additional performance and development costs.