

Team Rocket



CSCD01 Deliverable #3

March 22nd, 2020

Team Members: Yasir Haque, Adnan Shahid, Patricia Lee,

Sol Han, and Jacqueline Chan

Table of Contents

Feature 1	3
Description	3
Sample Design	3
Implementation Details	3
Why we choose this as our feature to implement:	4
Feature 2	5
Description	5
Implementation Details	5
Acceptance Tests For Our Selected Feature	7
Test Case 1:	7
Test Case 2:	7
Test Case 3:	7
Test Case 4:	7
Test Case 5:	7
Test Case 6:	8
Architecture	8
Relevant Architecture for Feature 1:	8
Relevant Architecture for Feature 2:	9
Discovered Design Patterns Explained	9
Adapter Pattern	9
Observer pattern	10
Singleton pattern	10
Appendix	11
Appendix A	11

Feature 1

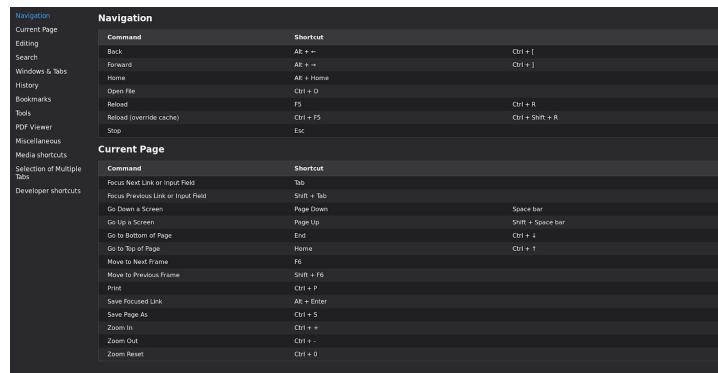
https://bugzilla.mozilla.org/show_bug.cgi?id=492557

Description

Create a keyboard shortcut page consisting of all the existing keyboard shortcuts for the Mozilla Firefox browser. At the moment, the requested implementation of this feature asks for the creation of an “about” page for shortcuts within Mozilla. The shortcuts span all sections of Mozilla, from navigation to developer shortcuts. This feature is listed as a student project within the Bugzilla repository.

Sample Design

The created “about” page would look like something similar to:



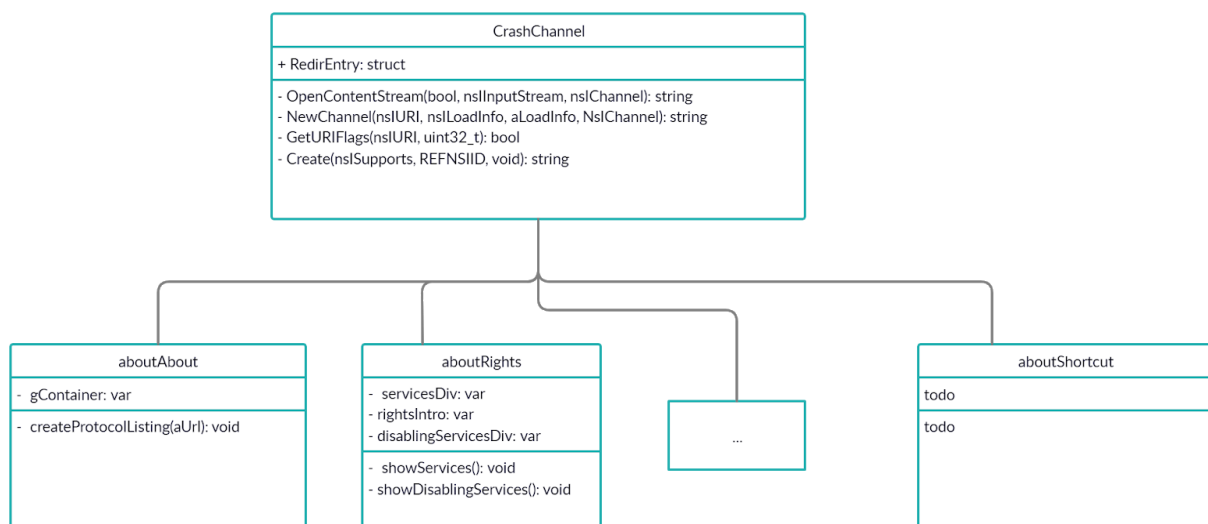
Command	Shortcut
Back	Alt + ← Ctrl + [
Forward	Alt + → Ctrl +]
Home	Alt + Home
Open File	Ctrl + O
Reload	F5 Ctrl + R
Reload (override cache)	Ctrl + F5 Ctrl + Shift + R
Stop	Esc

Command	Shortcut
Focus Next Link or Input Field	Tab
Focus Previous Link or Input Field	Shift + Tab
Go Down a Screen	Page Down Space Bar
Go Up a Screen	Page Up
Go to Bottom of Page	End Shift + Space Bar
Go to Top of Page	Home Ctrl + ↑
Move to Next Frame	F6
Move to Previous Frame	Shift + F6
Print	Ctrl + P
Save Focused Link	Alt + Enter
Save Page As	Ctrl + S
Zoom In	Ctrl + +
Zoom Out	Ctrl + -
Zoom Reset	Ctrl + 0

Implementation Details

Our team would need to create an “about:shortcuts” page following a similar design to what is listed above. It would also need to affect the “about:about” page on Mozilla and add the url to the paths. The code itself would follow the standards set for the existing “about” pages within Mozilla and any interactions for the new section would be the same as these existing pages. About pages have their URLs set within the ‘[docshell/base/nsAboutRedirector.cpp](#)’ (see the CrashChannel class in the UML diagram on the following page) directory, and thus the code within the C++ redirector file would need to be modified as well to include our new resources. This file allows a user to enter about: shortcuts in the url and load the page.

The directory mozilla-central/toolkit/content is where we would place our new about shortcut component (the (new) aboutShortcut class in the UML) and its related files to be able to statically serve on the url about:shortcut. So the new code changes here for this component would be making new entire files to make the design page (as shown above), and to be able to have the url about:shortcut redirect to it. Because this will be a new feature, much of our files/code would be expected to be new. We will also have to add the shortcut page to the about_pages variable in mozilla-central/source/docshell/build/components.conf so that the project builds with the new page.



Why we choose this as our feature to implement:

Working on feature 1 allows our team to easily separate the development tasks among developers as adding our page to the existing architecture and creating various segments within the page can be easily divided. Despite being more difficult to implement than our previous bug fixes in previous deliverables, this feature seems viable to do well and may allow our team to make a significant contribution to Mozilla. Also it would be nice to do a feature that would be frequently referred to and visually seen. There have also been subsequent listed tickets relating to this ticket, such as a popup page for devtools shortcuts specifically, (https://bugzilla.mozilla.org/show_bug.cgi?id=821359). Therefore, we thought it would be appropriate to tackle the ticket that had every shortcut and not just for the devtools project.

Feature 2

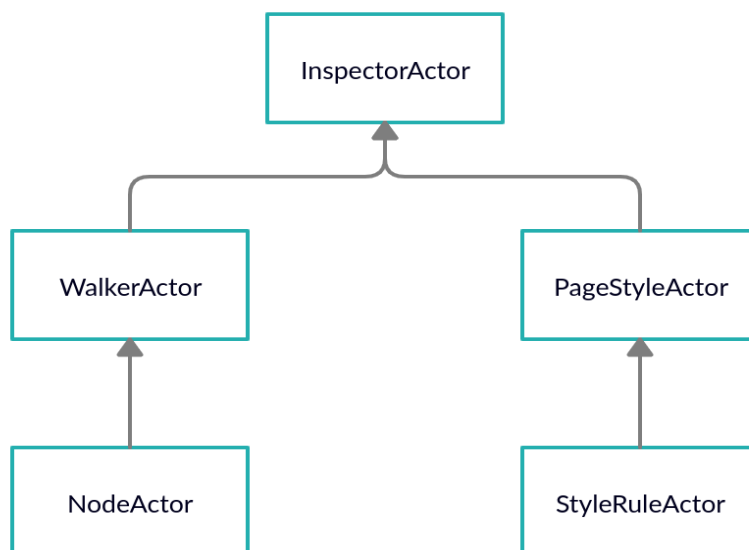
https://bugzilla.mozilla.org/show_bug.cgi?id=1528288

Description

Add the ability to create a tooltip which references the appropriate style rule when hovering over a number/property on the box model. For example, hovering the margin property on a box model will showcase a tooltip that displays the css class that is enforcing that styling.

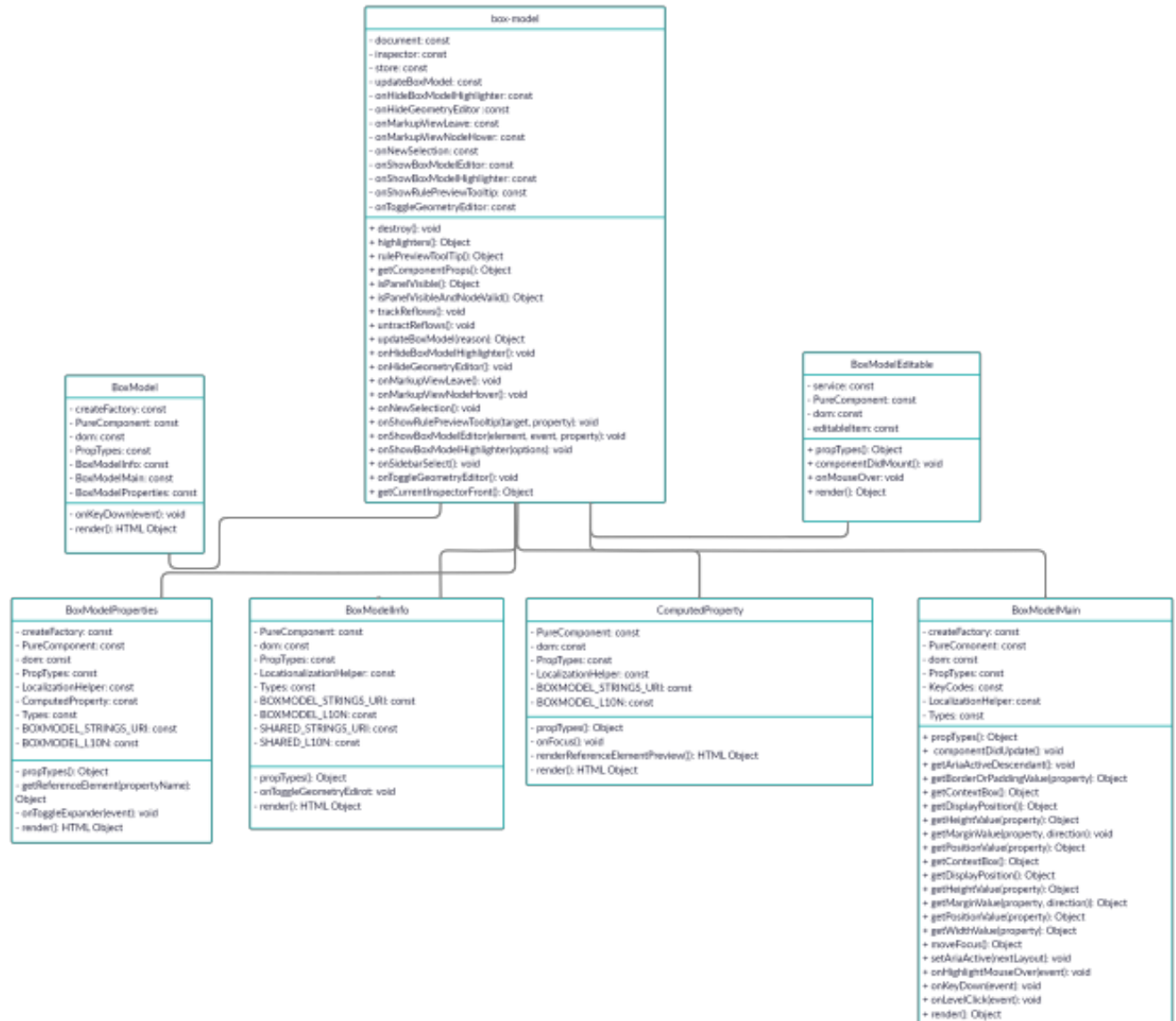
Implementation Details

When the box model object instantiated it takes in an instance of the inspector and window. Using the inspector we can access style rules which can be saved as a property within box-model.js at [devtools/client/inspector/boxmodel/box-model.js](#). We will then create a new component, like many that already exist within [devtools/client/inspector/boxmodel/components](#), and utilize the box-model controller instance to grab and populate the appropriate style rule. Furthermore, the tooltip should be formed such that it generates on a hover event on the right points within the BoxModelMain component.



The inspector holds onto information pertaining to DOM walking and CSS styling/rules. For this ticket we will have to tease out the css styling rules that the inspector object can hold.

Since this inspector actor component was talked about in detail when we first evaluated the devtools portion of the Mozilla Project back in deliverable 1, a more detailed UML outlining additional actors within the server side of the application has already been provided. Please see Appendix A.



We would be creating another component which utilizes the box-model.js controller to render the appropriate frontend UI elements as mentioned before.

Acceptance Tests For Our Selected Feature

Test Case 1:

1. Open a new instance of the firefox browser
2. Enter "about:shortcuts" in the url
3. Check that the page loads and looks similar to the sample design (listed above)

Test Case 2:

1. Open a new instance of the firefox browser
2. Enter "about:shortcuts" in the url
3. Click one of the sections listed in the sidebar
4. Check that the page scrolls to the correct section

Test Case 3:

1. Open a new instance of the firefox browser
2. Enter "about:shortcuts" in the url
3. Try using one of the shortcuts listed
4. Verify that the shortcut works as intended when pressing the keys listed

Test Case 4:

1. Open a new instance of the firefox browser
2. Navigate to any web page
3. Enter "about:shortcuts" in the url
4. Check that the page loads

Test Case 5:

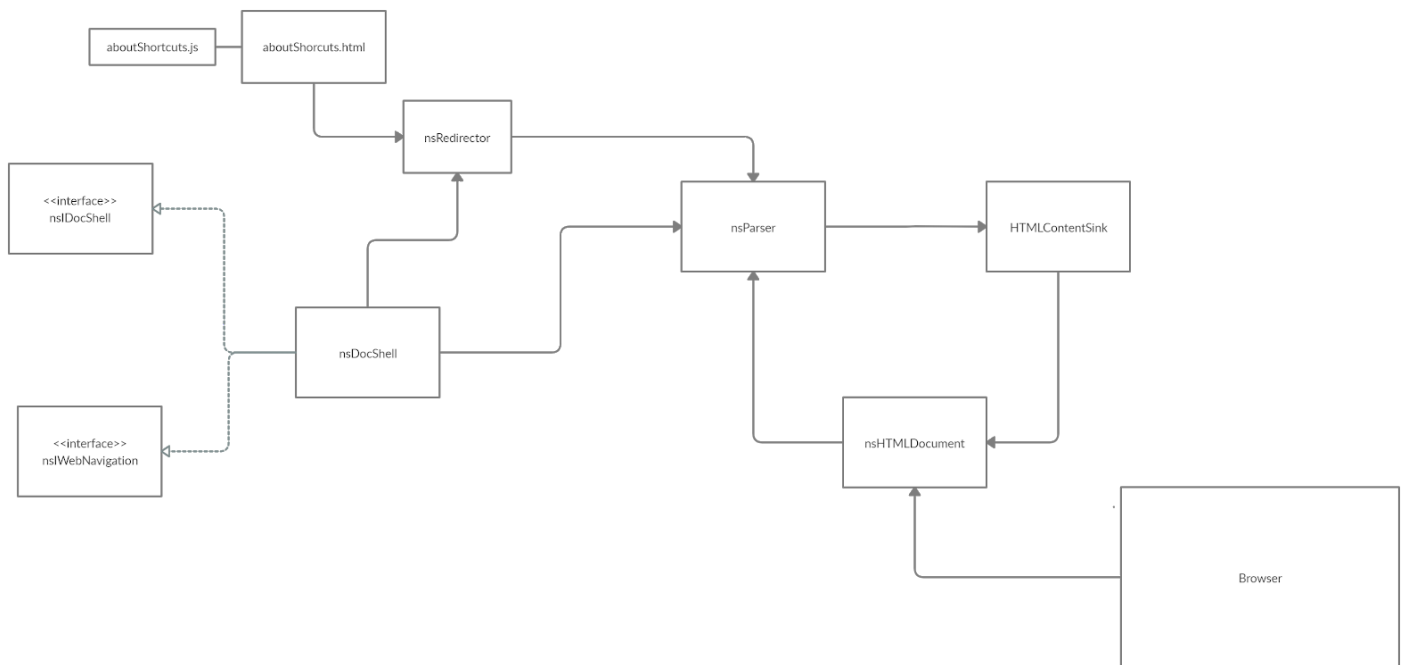
1. Open a new instance of the firefox browser
2. Enter "about:about" in the url
3. Check that the "about:shortcuts" link is listed

Test Case 6:

1. Open a new instance of the firefox browser
2. Enter “about:about” in the url
3. Click on the “about:shortcuts” link
4. Check that the browser then loads the “about:shortcuts” page

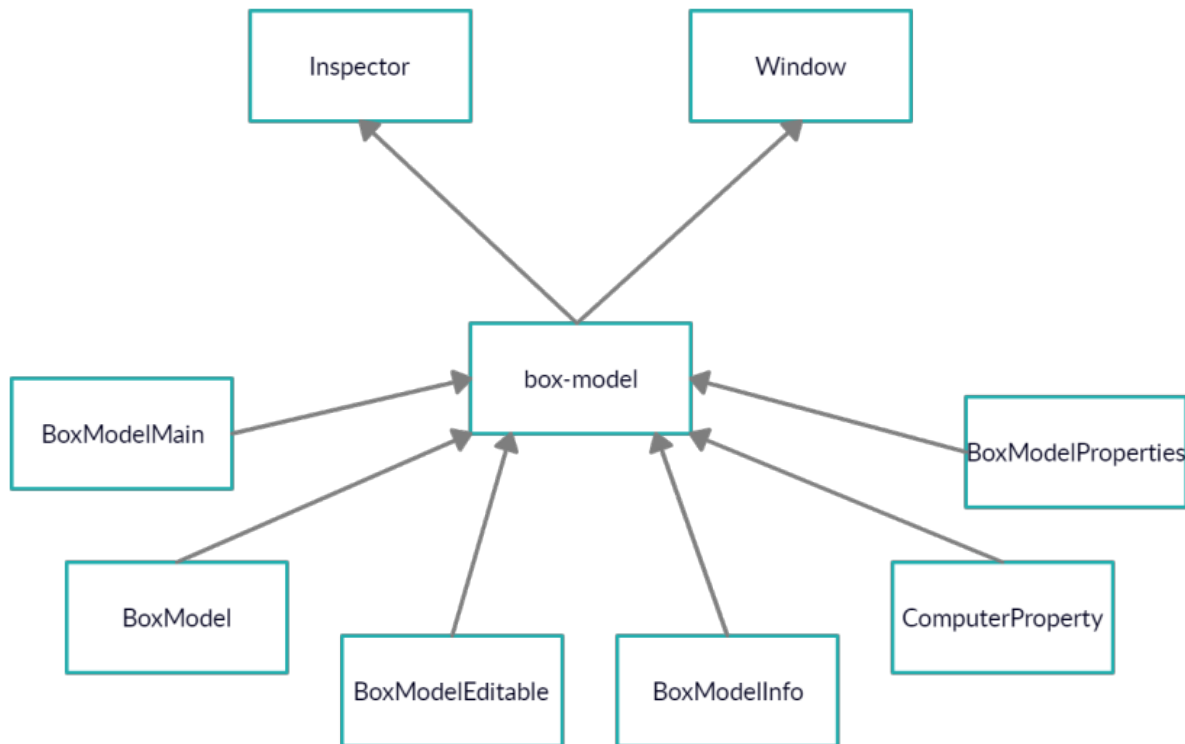
Architecture

Relevant Architecture for Feature 1:



The `nsDocShell` functions as the underlying brain/navigator for the project. When URLs are typed into the address bar the corresponding served pages are parsed by the `nsParser` and interpreted for the browser by `nsHTMLDocument`. To utilize the about protocol, “about” statements in the address bar are redirected using `nsRedirector` to serve the appropriate resource/page to be parsed.

Relevant Architecture for Feature 2:



The box-model controller takes in an instance of the inspector object and a window object to use when rendering its box-model related components. This is a prime example of dependency injection and the singleton design pattern.

Discovered Design Patterns Explained

Adapter Pattern

The adapter pattern is one of the design patterns used in the Mozilla Firefox system. It is a suitable design pattern for Firefox because it is designed to be usable on multiple platforms and operating systems. In particular, the adapter pattern is mainly used with Firefox's platform specific rendering and widgets. The adapter basically behaves as a bridge between Frame construction and the Display backend. Frame construction is the process of creating or

recreating frames when styles are changed in ways where frames need to be changed or when new nodes are inserted into the document, while the display backend is a platform specific rendering engine, which uses the adapter pattern as it expects inputs and outputs to be unvarying, regardless of the platform Firefox is on. Moreover, widgets carry out rendering necessary for specific platforms and provide input to the display backend.

Observer pattern

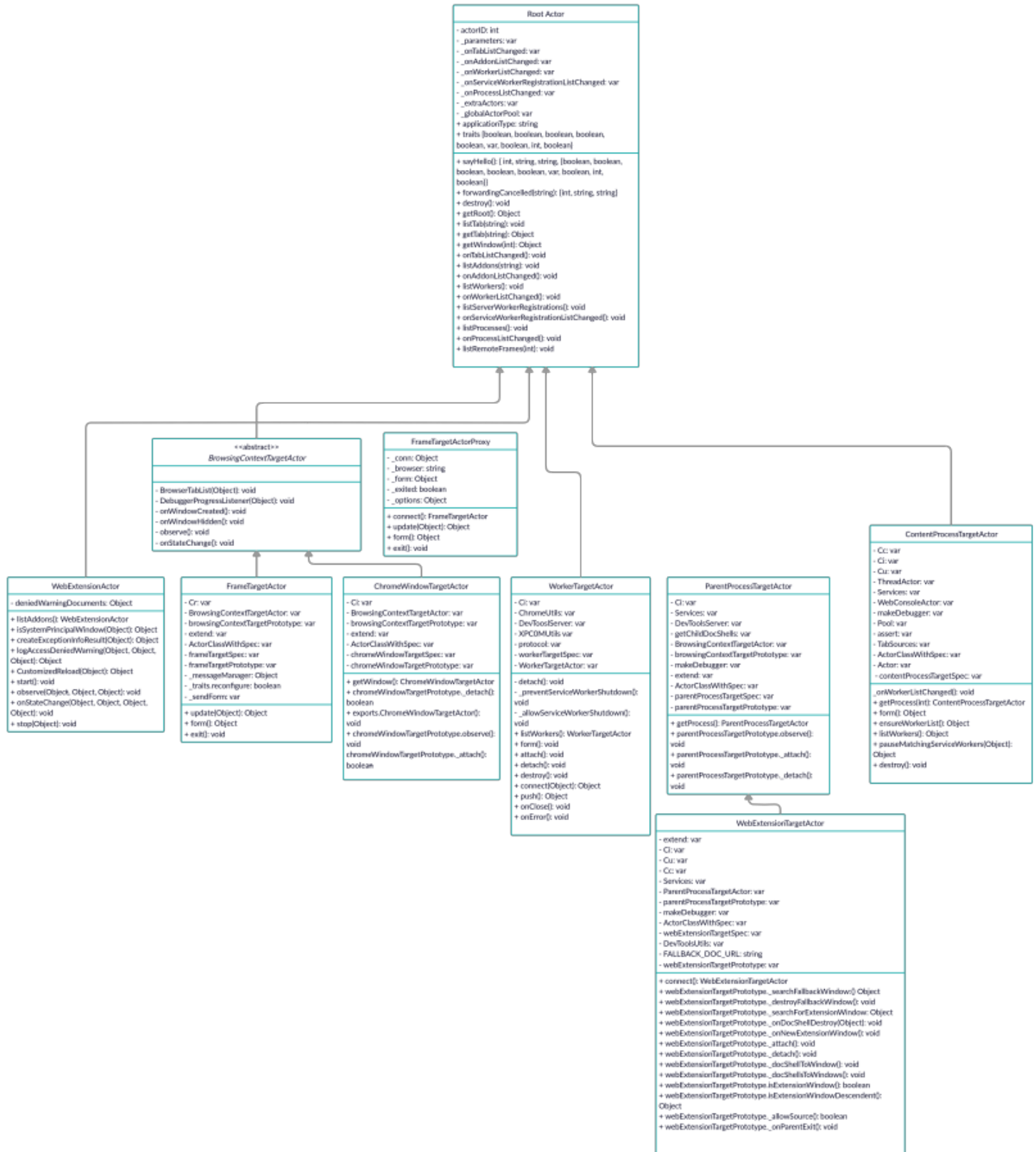
The observer design pattern can be seen in Firefox as well. In particular, there is a file called `EventStates` in the Content Model which is owned by the document. This file is responsible for updating the DOM tree, as well as indirectly updating the frame constructor when there are changes to the content, style, layout, or other components of a webpage. The observer pattern in this case allows for various components to easily be updated when there are changes to single components.

Singleton pattern

The singleton design pattern is used to ensure that a class has only a single instance, and provides a universal point of access to that instance. In Firefox DevTools, a single instance of the inspector and window objects are inserted into the box-model controller. This is because the inspector is made reference to throughout various points within the application. For example, various `PageStyle` actors or DOM walkers could make reference to the inspector as well. As these lower level actors make changes to the inspector, the updates should be reflected when the box-model controller makes use of it as well. This is why a single instance is inserted as a dependency into the box-model controller.

Appendix

Appendix A



The UML in Appendix A showcases the hierarchical actor structure found on the server-side of the application. The root actor functions as the genesis which gives birth to various other actors. When a parent actor is removed, so are its children. Actors are used for memory management regarding various front-end processes throughout the application. For example, "Target actors" represent objects being targeted by a toolbox such as tabs, frames and more. These actors then give rise to children for more specific processes as well. The above are a few examples of many possible actors and their relationships within the application.