Bilkent University

Department of Computer Engineering

Senior Design Project

Willow: Graph-Based Browsing

Low-Level Design Report

Efe Dağdemir, Tuana Türkmen, Sezin Zeydan, Can Cebeci, Cem Cebeci

Supervisor: Uğur Doğrusöz

This report is submitted to the Department of Computer Engineering of Bilkent University in partial fulfillment of the requirements of the Senior Design Project course CS491/2.

Table of Contents

1. Introduction

1.1 Object design trade-offs

Rapid Development over Platform Independence: Nearly all of the widely used browsers (including but not limited to Mozilla Firefox, Opera, Safari and Microsoft Edge) support extensions and add-ons. However, as this is a project with the time scope of less than a year, it requires rapid development. As a result, the decision was to develop the tool only for Google Chrome, as it is the most popular browser with the biggest active user base [1], making Willow platform (meaning, Google Chrome) dependent.

Usability over Functionality: Willow depends heavily on user experience so even though we will implement nice working and practical functionalities however the large quantity of functionalities does not always correspond to a better user experience. This is why intuitiveness of our user experience will always be more important to us. So our design favors usability over variety of functionality.

Compatibility over Simplicity: The design choices we explain throughout this document may sometimes seem to include a surplus of indirections, appearing unnecessarily complicated. This is due to Willow's inherent constraint that its design has to be compatible with Google Chrome. The insertion of multiple side panel elements and the indirect communication channels between content scripts are prime examples of this (described in section 3.2.1).

Robustness over Efficiency: In synchronizing the graphs in different pages, we choose to use Chrome API's messaging functionality to send a message to a background script and then distribute the whole cytoscape object to all content scripts. Another option would be to implement our own messaging system and only broadcast the changes. However, Chrome's messaging API is far more consistent and robust then ours ever will be and passing the whole cytoscape object enables us to make sanity checks. Our solution promotes robustness over some performance optimizations.

1.2 Interface documentation guidelines

We will use the following conventions for writing a low-level design report:

Class name

- The specific class name is given in bold followed by the description of the class and then attributes and methods of the class along with their definitions.

Attribute

- Given after description of the class, ‘+’ indicates that the attribute is public and ‘-’ indicates the attribute is private.

Method

- Given after attribute declarations, ‘+’ indicates that the attribute is public and ‘-’ indicates the attribute is private meaning that it can only be used within the class the method belongs to.

1.3 Engineering standards

In this report IEEE referencing style was used for citations [2] and Unified Modeling Language (UML) was used to visualize the design of our system [3].

1.4 Definitions, acronyms, and abbreviations

Session: Sessions are continuous instances of browsing activity. A new session starts every time the user launches the Chrome app. The session is terminated upon exit. Sessions can be saved and restored, allowing them to last across multiple runs of Chrome.

Session Graph: A visual graph structure that shows the relationships between nodes. The session graph contains an edge $u \rightarrow v$ if and only if the first instance of access to the website associated with node v was through a link contained in the website associated with node u.

Node: A vertex on the session graph which represents a web page visited within the session.

Active Tab: The tab whose content is currently displayed. At any time, the browser contains exactly one active tab and any number (possibly zero) of inactive tabs.

Willow Overlay / Graph Overlay: The hideable and extendable panel that is added to Chrome’s UI when Willow is installed. The overlay contains the session graph and is the main medium of interaction with Willow.

Cytoscape.js: A graph theory library for visualization and analysis [4].

Content Script: The class of files in Chrome extensions files that “run in the context of web pages”. These are injected into web pages, which match an indicated pattern that is specified in the extension manifest [5].

Background Script: The class of files in Chrome extensions files that run independently of the context of any webpage, i.e. in the background. Chrome generates a dedicated “background page” per extension which acts as a host for the content scripts the extension specifies [6].

2. Packages

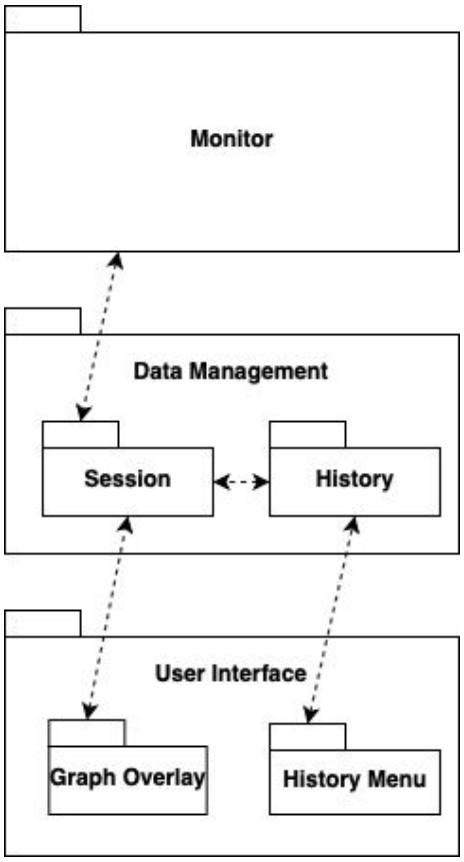

Figure 1: Package diagram of Willow

Data Management:

- Session: This package contains graph data classes such as Node and Edge and session management classes such as Session and SessionController.
- History:

User Interface:

- Graph Overlay: Maintains and synchronizes the data related to the HTML elements that take part in displaying the session graph.
- History Menu: Contains classes that interact with each other and exchange messages with the background pages to override Google Chrome's history page and display the browsing history as a list of sessions.

Monitor:

Classes in this package are responsible for monitoring the user's browsing behaviour and reporting certain events to the relevant system components.

3. Class Interfaces

3.1 Data Management

3.1.1. Session

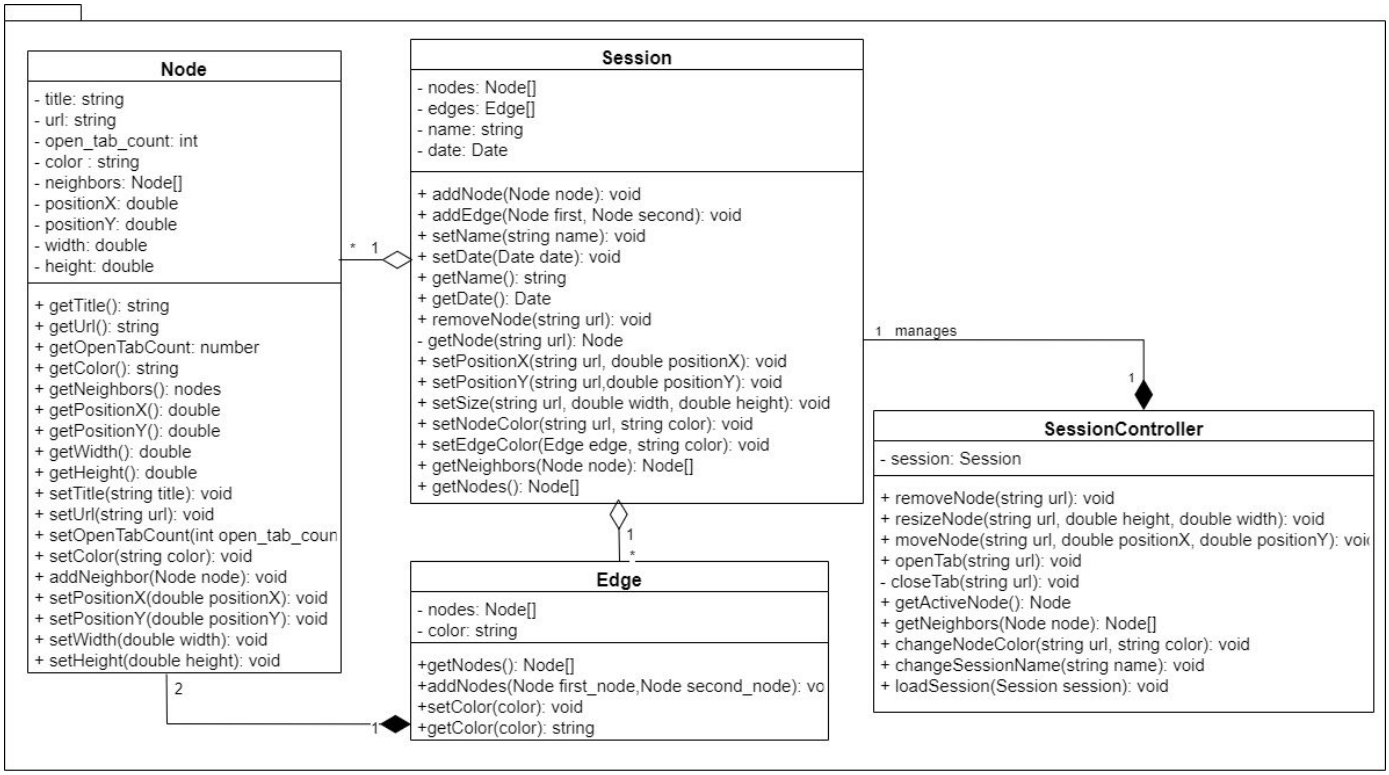

Figure 2: Class diagram of the classes of the Session package

Class: Node

Node class corresponds to the nodes in the graph. It has attributes to distinguish different cases of a node. Alongside the basic get and set operations, a function to add a neighbor to a node is presented.

Attributes:

- **title:** string
- **url:** string
- **openTabCount:** int
- **color:** string
- **neighbors:** Node[]
- **positionX:** double
- **positionY:** double
- **width:** double

- **height:** double

Functions:

- **addNeighbor(Node node):** is called whenever a neighboring node is added to a node. The added node becomes one of the neighbors.

Class: Edge

Edge class is the other part of the graph other than nodes. Used to link two nodes together.

Attributes:

- **nodes:** Node []
- **color:** string

Functions:

- **addNodes(Node firstNode, Node secondNode):** is used to attach the two end nodes of an edge.

Class: Session

This class represents the browsing session of a user. It has nodes and edges (representing a graph). Alongside the get and set operations, a node can be added to a session or can be removed from it.

Attributes:

- **nodes:** Node[]
- **edges:** Edge[]
- **name:** string
- **date:** Date

Functions:

- **addNode(Node node):** is called when a new node will be added to a session. The
- **removeNode(Node node):** is called when a node is removed from session.
- **getNode(string url):** is used to find the node in the graph which contains the provided url as its url.

Class: SessionController

This class only contains a session and is used to manage it. In order to make operations on a session, sessionController directs the operation via presenting the corresponding url and if needed additional information (such as color) to the session. Additionally it can open a Chrome tab with the given url and can load a session to be the current one.

Attributes:

- **session:** Session

Functions:

- **addNode(Node predecessor, Node node):** this method adds a new node to session.
- **removeNode(Node node):** is called when a node is removed from the session graph. This method subsequently calls close_tab(string url) method with the url of node to also close the tab from browser.
- **Private closeTab(string url):** this method is called from remove_node(Node node) method to close the tab that corresponds to removed node.
- **resizeNode(Node node, double height, double width):** is called when the size of node is changed.
- **moveNode(Node node, double positionX, double positionY):** is called when the node is moved on the session graph.

- **openTab(Node node):** is called to open a new Chrome tab which has the url of the provided node.

- **changeNodeColor(Node node, string color):** is called when a node's color will be changed.
- **changeSessionName(string name):** changes the current session's name to the provided name.
- **loadSession(Session session):** is called when a session will be loaded as the current session.

3.1.2 History

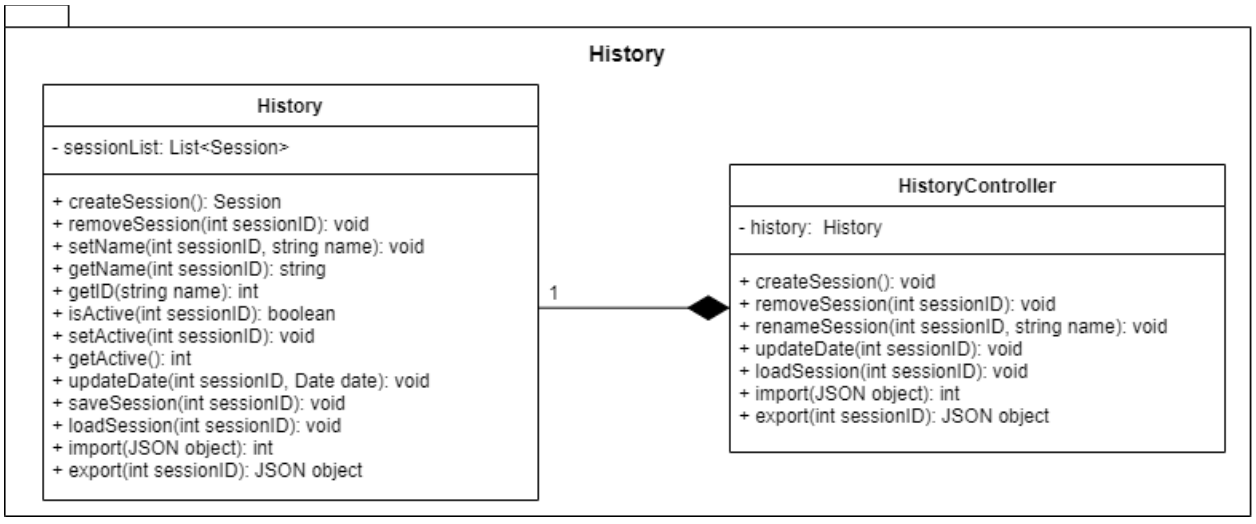

Figure 3: Class diagram of the classes of the History package

The HistoryService package consists of two classes that work together to provide the underlying structure that both deals with the storage of the history and the operations that can be done regarding the history. Classes in this package keep information about the sessions stored in the local machine as a list and perform several different types of actions starting from adding/removing sessions to importing/exporting sessions.

Class: History
History class is responsible for the storage of sessions and the various actions performed on the history, although the actions are designed to be not directly invoked on the object but rather from the controller object.

Attributes:

- **sessionList:** This is the list that holds all the session objects.

Functions:

- **createSession():** Gives the call to create a session and inserts the created session into the *sessionList*.
- **removeSession(int sessionID):** Removes the given session from the *sessionList*.
- **setName(int sessionID, string name):** Sets the name of the given session with the given name.
- **getName(int sessionID):** Gives the name of the session with the given sessionID.
- **getID(string name):** Gives the ID of the session with the given session name.
- **isActive(int sessionID):** Specifies whether the given session is active.
- **setActive(int sessionID):** Sets the given session as active.
- **getActive():** Gives the sessionID of the active session.
- **updateDate(int sessionID, Date date):** Sets the last accessed date of the given session.
- **saveSession(int sessionID):** Saves the session to the local machine.
- **loadSession(int sessionID):** Loads the given session as the current session.
- **import(JSON object):** Import a session outside of Willow's storage as the current session.
- **export(int sessionID):** Export the specified session as a JSON object.

Class: HistoryController

HistoryController class is the controller of the HistoryService package, it provides the useful actions that can be performed on the history as an interface.

Attributes:

- **history:** An instance of a *History* object, this is what this controller class will perform actions on.

Functions:

- **createSession():** Invokes the *createSession()* function on the History object to create a new session.
- **removeSession(int sessionID):** Invokes the *removeSession()* function on the History object to remove a session from the history.
- **renameSession(int sessionID):** Invokes the *setName()* function on the History object to rename the session.
- **updateDate(int sessionID):** Invokes the *updateDate()* function on the History object to update the last accessed date of the session.
- **loadSession(int sessionID):** Invokes the *loadSession()* function on the History object to load the selected session as the current session.
- **import(JSON object):** Invokes the *import()* function on the History object to import a session outside of Willow's storage as the current session.
- **export(int sessionID):** Invokes the *export()* function on the History object to export the selected session as a JSON object..

3.2 User Interface

3.2.1 Graph Overlay

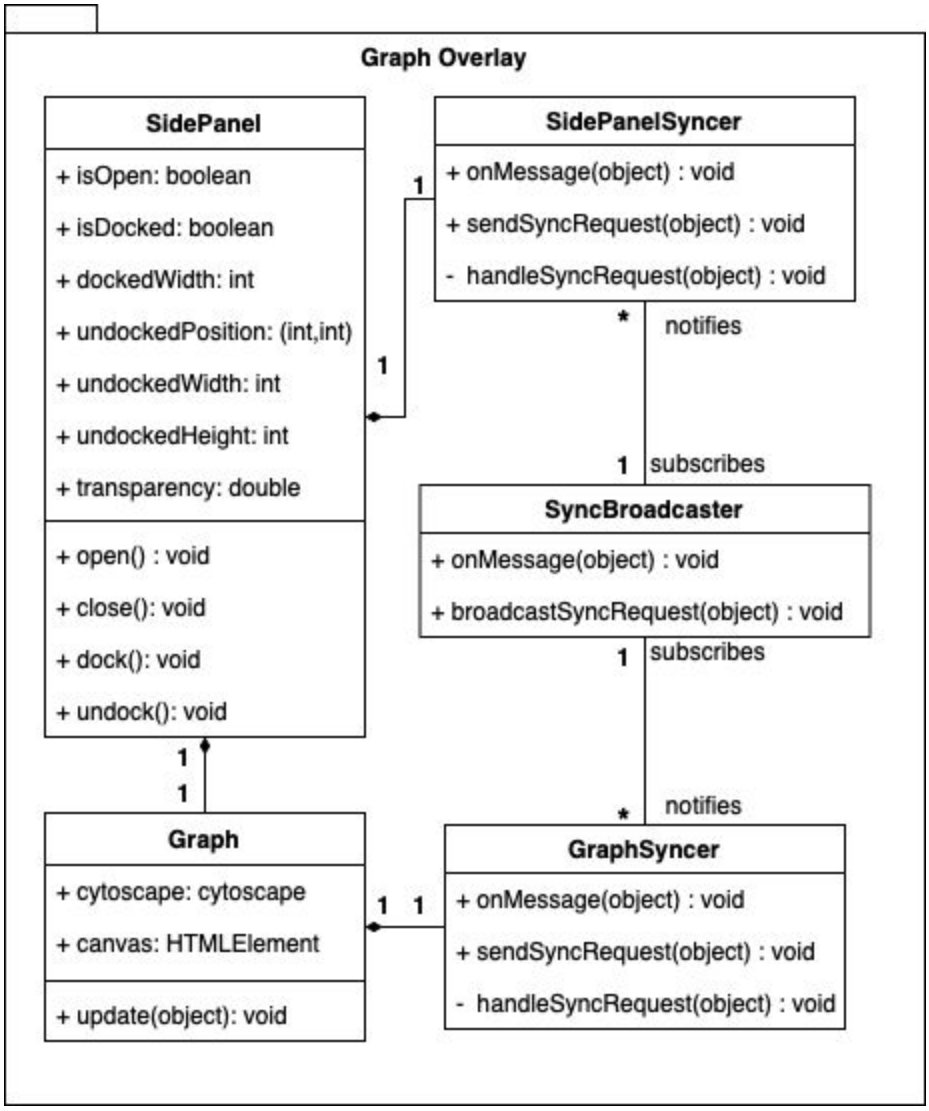

Figure 4: Class diagram of the classes of the Graph Overlay package

Class: SidePanel

A user's main method of interaction with Willow is a collapsible, undockable side panel. Due to the limiting boundaries of what a Chrome extension can do, this side panel is injected as an HTML element by a content script to the web page open in each tab in a Chrome window. This results in the proliferation of many instances of the side panel, each associated with an instance of the SidePanel class. This class encapsulates various properties regarding the state of the html element and defines methods which modify this state.

Attributes:

- **isOpen:** indicates whether the side panel is in its expanded (open) or collapsed (closed) form.
- **isDocked:** indicates whether the expanded form of the side panel resides along an edge of the window, in which case it is said to be in the docked state. An undocked panel, in contrast, can be moved anywhere on the page and scaled freely both in height and width.
- **dockedWidth:** the width of the panel in its docked state. Irrelevant when the panel is undocked. Retained for use to restore the panel to its width when it is docked again.
- **undockedPosition:** the pixel coordinates of the top left corner of the panel in its undocked state. Irrelevant when the panel is docked. Retained for use to restore the panel to its position when it is undocked again.
- **undockedWidth**: the width of the panel in its undocked state.
- **undockedHeight**: the height of the panel in its undocked state.
- **transparency**: the transparency of the panel. Can be configured by the user so that the page behind the panel is visible to a desired degree.

Functions:

- **open()**: expands the panel, bringing it to its open form, covering a portion of the page behind it and allowing it to display the session graph.
- **close():** collapses the panel, bringing it to its closed form, displayed as a small icon.
- **dock():** docks the panel, attaching it to an edge of the window and keeping it from being dragged.
- **undock():** undocks the panel, allowing it to be dragged and placed anywhere on the page.

Class: SidePanelSyncer

As described above, Willow injects a separate HTML element, associated with a separate instance of the side panel into the web page open at each panel. In the face of this non-singularity, the need to synchronize the state of the panel across all tabs arises. For example, when the panel in the active tab is opened, the panels in the other open tabs should be opened as well. This also applies to closing, docking, undocking, resizing and moving the panel. This type of synchronization hides the multiplicity of the panel and gives the impression of a single panel existing across all tabs. SidePanelSyncer instances reside in content scripts and ensure this synchronization by sending and receiving sync requests to/from the SyncBroadcaster.

Functions:

- **onMessage(object):** called whenever the page receives a message. Forwards the message to handleSyncRequest if the message is a sync request that was broadcast by the SyncBroadcaster.
- **sendSyncRequest(object)**: called when the state of a panel or any of its visible properties changes. Sends a sync request to the SyncBroadcaster so that the side panels in other tabs are notified of the change.
- **handleSyncRequest(object):** updates the associates SidePanel accordingly to the received syncRequest.

Class: SyncBroadcaster

The SyncBroadcaster is a singleton object that resides in a background script. Its purpose is to listen for sync requests and broadcast them. Such an intermediary object is needed since the SidePanel objects themselves reside in content scripts and due to security considerations, Chrome does not allow direct messages between content scripts.

Functions:

- **onMessage(object):** much like the mechanism in SidePanelSyncer, this function is the generic message receptor that is called by Chrome. It forwards the message to broadcastSyncRequest if it is a sync request.
- **broadcastSyncRequest(object):** sends the request to all SidePanelSyncer objects except the one that originated the sync request.

Class: Graph

The graph is displayed within the side panel and together they make up the graph overlay. To display the graph, we rely on cytoscape.js, which draws the graph on a dedicated HTML div, referred to as the cytoscape canvas. Just like SidePanel, the graph display is duplicated across open tabs and GraphSyncer instances are used to synchronize them.

Attributes:

- **cytoscape**: the cytoscape.js instance which displays itself of the canvas
- **canvas:** the dedicated HTML div element which contains the graph display

Functions:

- **update(object):** called by the associated GraphSyncer object. The input object indicates the parameters that need updating. Modifies the cytoscape instance so that it is synchronized with the instances in other tabs.

Class: GraphSyncer
Just like SidePanelSyncer, GraphSyncer acts as the communication channel between the Graph objects separately injected to each page and the singleton SyncBroadcaster. It sends sync requests when changes are made to its associated Graph object and likewise modifies the Graph object when sync requests are received.

Functions:

- **onMessage(object):** called whenever the page receives a message. Forwards the message to handleSyncRequest if the message is a sync request that was broadcast by the SyncBroadcaster.
- **sendSyncRequest(object):** called when the state of the related graph or any of its visible properties changes. Sends a sync request to the SyncBroadcaster so that the side graphs in other tabs are notified of the change.
- **handleSyncRequest(object):** updates the associated Graph accordingly to the received syncRequest.

3.2.2 History Menu

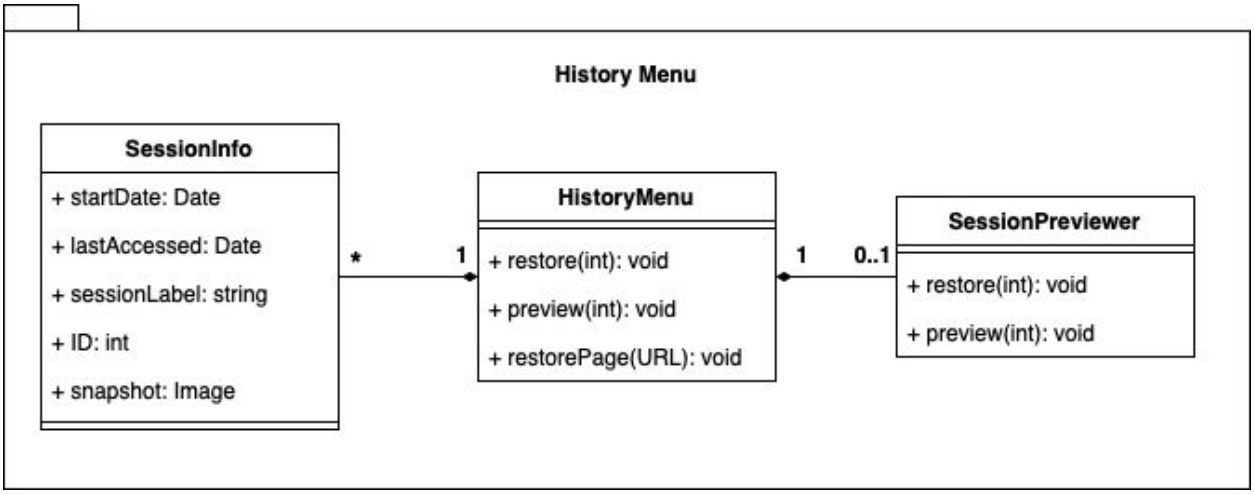

Figure 5: Class diagram of the classes of the History Menu package

The history menu package contains classes that interact with each other and exchange messages with the background pages to override Google Chrome's history page and display the browsing history as a list of sessions. The user may then preview some sessions to identify the one they are looking for and eventually choose to either restore a page and open it in the current session or restore a previous session and keep browsing in it. Classes in this package do not maintain data about the history or the previous sessions, they only read it from the History package described at section 3.1.2. Similarly, they do not perform the restore actions but request them from the History package instead.

Class: HistoryMenu

The class HistoryMenu shares the name of the package and predictably, it's the class that manages the rest of the classes in the package. It's responsible for initializing the history menu by retrieving the history data, organizing it and then presenting it to the user by overriding the Chrome history page. It's public functions correspond to the history menu use cases.

Functions:

- **void restoreSession(int):** restores the open tabs from the session with the given session id in the current Chrome window. Willow will start displaying the browsing graph of that session session as well. Any browsing actions done by the user will result in changes in the restored session. The function does not perform any changes on its own but requests them from other packages.
- **void restorePage(URL):** opens the page corresponding to the given URL in the current Chrome window. If a page corresponding to the node does not exist in the current session, a node is created with no incoming edges.
- **preview(int):** presents a preview of the session with the given session id. This request is received from one of the SessionInfo objects and passed to a SessionPreviewer object. HistoryMenu does not handle this request.

Class: SessionInfo

The class SessionInfo contains the information about a single session. A list of SessionInfo objects is maintained and the same list is represented visually to the user. Objects of this class are simply data items.

Attributes:

- **Date startDate**: the date at which the session started.
- **Date lastAccessed:** the date at which the session was last accessed.
- **string sessionLabel:** the label the user enters for the session.
- **int ID:** the unique id of the session.
- **Image snapshot**: a snapshot of the session's graph in image format.

Class: SessionPreviewer

The SessionPreviewer class's responsibility is to draw a preview of a session for the user to interact with. The user uses the session labels in conjunction with these previews to find the session they are searching for. The session previews support zoom and pan operations but no actual modifications to the session graphs.

Functions:

- **void restore(int)**: a request sent by the graphic elements for the session with the given id to be restored. This request is passed to the HistoryMenu object managing the SessionPreviewer instance.

- **void preview(int):** a request sent by the HistoryMenu for the session with the given id to be previewed. This class creates an appropriate HTML element to contain the graph representation and draws on it.

3.3 Monitor

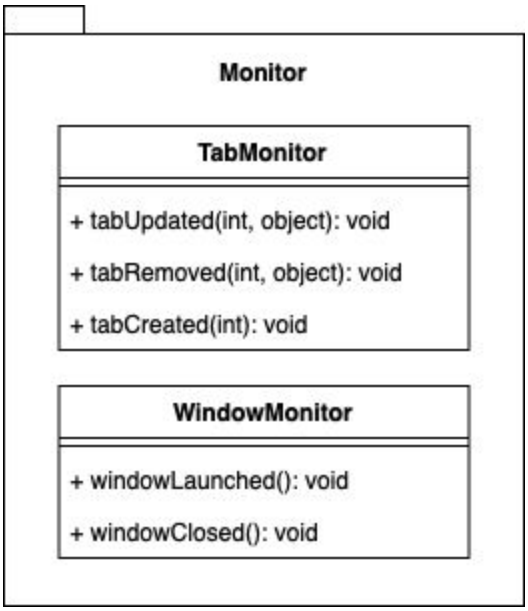

Figure 6: Class diagram of the classes of the Monitor package

Classes in the Monitor package are responsible for monitoring the user's browsing behaviour and reporting certain events to the relevant system components. The Chrome API's event listeners are used for this purpose. The monitored events are divided into two parts. User actions regarding tabs and user actions regarding windows.

Class: TabMonitor
Monitors user behaviour on tabs.
Functions

- **void tabUpdated(int, object):** A listener for the chrome.tabs.onUpdated event.
- **void tabRemoved(int, object):** A listener for the chrome.tabs.onRemoved event.
- **void tabCreated(int):** A listener for the chrome.tabs.onCreated event.

Class: WindowMonitor
Monitors user behaviour on windows.
Functions

- **void windowLaunched(chrome.Window):** A listener for the chrome.window.onCreated event.
- **void windowClosed(int):** A listener for the chrome.window.onRemoved event

4. References

[1] “Browser Market Share Worldwide,” *StatCounter Global Stats*. [Online]. Available: https://gs.statcounter.com/browser-market-share. [Accessed: 07-Feb-2021].

[2] “IEEE Referencing: Getting started with IEEE referencing,” *Library Guides*. [Online]. Available: https://libraryguides.vu.edu.au/ieeereferencing/gettingstarted#:~:text=%E2%80%9CIEEE%E2%80%9D%20stands%20for%20The%20Institute,paper%2C%20provided%20in%20square%20brackets.&text=This%20is%20known%20as%20an,of%20the%20work%20are%20provided. [Accessed: 07-Feb-2021].

[3] *What is Unified Modeling Language (UML)?* [Online]. Available: https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/. [Accessed: 07-Feb-2021].

[4] M. Franz, “Cytoscape.js: Graph theory (network) library for visualisation and analysis,” *Cytoscape.js*. [Online]. Available: https://js.cytoscape.org/. [Accessed: 07-Feb-2021].

[5] “Content scripts,” *Chrome Developers*. [Online]. Available: https://developer.chrome.com/docs/extensions/mv2/content_scripts/. [Accessed: 07-Feb-2021].

[6] “Manage events with background scripts,” *Chrome Developers*. [Online]. Available: https://developer.chrome.com/docs/extensions/mv2/background_pages/. [Accessed: 07-Feb-2021].