Intersection of Multiple Histories Through Transclusion

May - July '18

Arihant Parsoya[1]
Supervisor: Prof. James Eagan[2]

[1] Indian Institute of Technology Bombay
[2] Telecom ParisTech

Content

Abstract

Transclusion of documents helps us to easily reuse content from one document inside several other documents. Today, transclusions are mostly restricted to document without histories. We explore how transcluding documents with histories opens up new possibilities and use cases. Our tool enables exploring histories of multiple transclusions in the document. In this project, we discuss what are the different technological and design problems associated with intersection of multiple histories.

1 Research Problem

Understanding intersection of multiple histories through transclusion by reference. What are the technological and design problems related with intersection of multiple histories. What are the new use cases/scenarios where transclusions with histories can be helpful?

1.1 Document History

Accessing document history allows users to perform tasks such as:

- Undoing operations
- Restoring past version of the document
- Revisit older versions of the document

Histories of a document can be recorded as a sequence of states or sequence of operations.

1.2 Transclusion

Transclusion is inclusion of one document into another document by reference. Transclusion facilitates modular design by enabling use of content from one document inside other documents. When a change is made to a document, its changes are automatically updated in the documents which transcludes it. This helps in changing and updating information when a document is transcluded inside different other documents.

Transclusion generally is achieved using a template tag of the form {{template name}} where the tag includes the reference of the document which is to be transcluded. Transclusion on a webpage can be achieved by using a <iframe> tag which transcludes a webpage into another webpage. The <iframe> tag includes the link to the webpage as a *src* attribute which needs to be transcluded.

Sometimes it is possible to include only a part of the document rather than the entire document which is called partial transclusion. *"By using "noinclude", "onlyinclude" and "includeonly" markup, it is possible to transclude part of a page rather than all of it."* (Wikipedia)

2 Webstrates

Webstrate is a shareable dynamic media which enables collaborative real time editing of webpages. Shareable dynamic media are collections of information substrates. A substrates embody content, computation, and interaction.

2.1 Transclusion

"Transclusion is the process of inclusion of some or all part of the document into another document by simply referencing it, in such a way that any change made to the transcluding document is reflected in the document(s) transcluding it". (Clemens & James). This process of transclusion in webstrates is done using iframes. The iframe element embeds the webstrate client inside another webpage. This helps in keeping the webstrates synchronized across all clients. Any changes made to one client gets synchronized across all other clients.

2.2 Technical Details of Webstrates

Webstrate runs on NodeJS server and MongoDB database. Webstrates is build on top of ShareDB which uses Json0 transform types to store the operations of the webstrate.

2.2.1 Operation History

Each webstrate contains its operation history since the document began. Each operation contains the following elements:

- Version: the version of document on which the operation was applied
- Type: The JSON Operation Transform(OT)
- Data: The changes made in that operation. These can be insert, delete, move or replace of items. For more details see: https://github.com/ottypes/json0#features

Fig. Figure showing the history of a webstrate. Each blue circle represents an operation. When an operation is applied on a webstrate, its version changes simultaneously.

2.2.2 Versions

In webstrates, each operation creates a new version of webstrate. These versions are stored as list of operations. When the user wants to see the past version of a webstrate, the past version of the document is reconstructed using the operation history. The operation history can also be used to restore the document to previous version. A specific version of webstrate can be accessed by using the url "<domain>/<webstrateId>/<version>/".

2.2.3 Transcluding Version

Since a document in webstrates has its own history, we can either transclude the latest version of the document using the url "<domain>/<webstrateId>" or a specific version of the webstrate using the url "<domain>/<webstrateId>/<version>/". When we transclude a specific version of the webstrate, it remains static when changed are made to the webstrate. On the other hand when we transclude the webstrate without its version, the transclusion gets updated when the webstrate is updated.

2.2.4 Copy

A webstrate can be copied into a new webstrate. The copy operation copies all the assets from the old webstrate into the new webstrate. However, the operation history from the old webstrate is not copied into the new webstrate. The new webstrate contains the reference of the webstrate it was copied from in its operation history.

Fig. When an old webstrate is copied, its history exists independent of its parent webstrate.

2.2.5 Tagging

Webstrate supports tagging each version of webstrate. These tags are stored in the database and can be fetched when necessary.

3 Technical Challenges in Webstrates

To be able to navigate between different versions and copies of the webstrate, we need to have access to all the different version and copies of webstrates. This cannot be completely achieved with the current version of webstrates due to the following problems

3.1 Referencing Problem

When a child of webstrate is created, it contains a reference to the parent webstrate it was created from. However the child webstrate is not referenced in the parents operation history which makes it impossible to know the children of a webstrate by looking at its operation history. There are two possible solutions to this problem:

1. Reference the child in the parents operation history: Referencing the child in the operation history of the parent enables us to get the webstrateId of the child by traversing through its operation history. This could be an inefficient process when the operation history is long.
2. Search ops of all the webstrates and see find all the children of a webstrate
3. Metadata: A new attribute in the webstrate database can be added which stores list of all the children created from the given webstrate along with the version from which it was created. When a copy is created, both the child and parent reference each other in their metadata. This enables to know the webstrateId of the parent given its children and vice versa. The structure of the metadata can be:

```
{
        {
                'Version': 5,
                'Event': 'Parent', // parent or child
                'Data': {
                        webstrateId: webstrateId of the child
                }
        },
        ...
}
```

Solution: I implemented an algorithm which searches ops of all the webstrates to find the children and parent of a certain webstrate.

3.2 Timing of Operations

By default ShareDB does not return the timestamp from the operation database. To create the history tool we require timestamp of each operation so they can be compared across different webstrates. For example, if we want to load the past version of the webstrate, we need to load the transclusions also to older versions for which we require to get version of transclusions using the timestamp of the transcluding webstrate.

Solution: I implemented a new API which takes the ops directly from the MongoDB and obtains the timestamp of each operations.

4 User Scenarios

When a document is transcluded, its history exist independent of its transcluding document. Having independent history of a transcluded document allows users to easily manage the document and also create possible use cases. In the following sections, we have described scenarios which highlights new use cases when transclusions have histories.

4.1 Essay Writing

Alice is working on her essay which consists of two paragraphs (*paragraph1* and *paragraph2*), she first make changes in *pagagraph1* and then in *paragraph2*. Now, she realizes that the change she made in *paragraph1* is wrong and wants to undo that change. To undo the change she needs to undo the change she did in *paragraph2* and then in *paragraph1* due to linear history structure.

Now, consider that both paragraphs are independent transclusions. Having independent transclusions enables having independent histories for *paragraph1* and *paragraph2*. When Alice wants to undo the changes in *paragraph1*, she can undo the change without having to undo the change in *paragraph2*.

Consider another scenario where Alice has two different ideas both of which can be used in *paragraph1*. She is confused which one of the two ideas she should write in *paragraph1*. To solve her problem, she creates a copy of *paragraph1*, and writes her other idea in the new transclusion. She can now toggle between different versions of *paragraph1* depending on which version suits best in the essay.

4.2 Presentations

Bob is preparing a presentation for his lecture where each slide is a transclusion. While creating his slides, he wants to use a version of slide which he used few weeks back. Using the history tool, he goes back in history and finds the version of slide he wants to use. He then creates a copy of that version and starts editing it to his requirements. He now effectively has two version of the same slide which he can toggle between them if he wanted to.

Consider another scenario where Bob wants to know how his presentation has evolved over the years. When he uses the history tool to see how the presentation looked like at

any specific version he requires. He can also load the latest version of his slides in the old presentation.

4.3 Newspaper

Mary is a newspaper designer. The complete news page is a webstrate and each story in the newspaper is a transclusion. Here each story has different copies which can be switched using the history tool. She wants to know which combination of story looks best on the newspaper. Using the tool, she can switch between different versions of stories and find the best one which suits her purpose.

Consider another scenario where Mary wants to reuse an old template of newspaper with the new content. She uses the history tool to go back in history of the newspaper webstrate and stories in the newspaper are automatically changed to the version they were at that point in time to show how the newspaper looked like at that time. Now, she can change the version of stories to their latest version to understand how the newspaper looks with the latest context. She makes a copy of the webstrate to continue using the old template of newspaper.

5 Design Questions

Transclusion with histories poses several design challenges. These are some design questions which we faced while designing for transclusions with histories.

1. How can the the user navigate through transclusion history?
2. How to show difference between different history versions of a transclusion?
3. What are the different ways in which user can navigate between copies of a transclusion?
4. Viewing past version of a document containing transclusion

5.1 Version Navigation

Since each transclusion has different versions, the user needs to be aware which version of the document he is using and how to change version if he requires. These are the questions we considered while designing for version navigation of transclusions:

1. How does the user know which version he is using?
2. How does the user recognize he is using the right/wrong version of transclusion?
3. How does the user correct if he is using the wrong version of the transclusion?

Potential solutions to above problems:

1. Have an interface which displays history timeline of all the transclusions. The interface will allow the user to change the version of each transclusion.

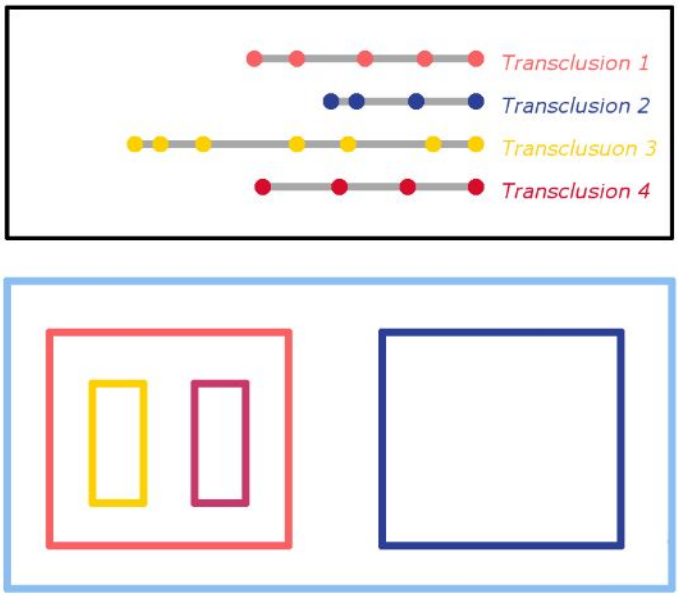


Fig. The panel at the bottom represents the rendered view of document. The panel on the top shows the history timelines of all the transcluded documents. The user can click on the history timeline to change the versions of the transclusions.

2. Creating a tree like representation of the transclusions and enable editing of histories.

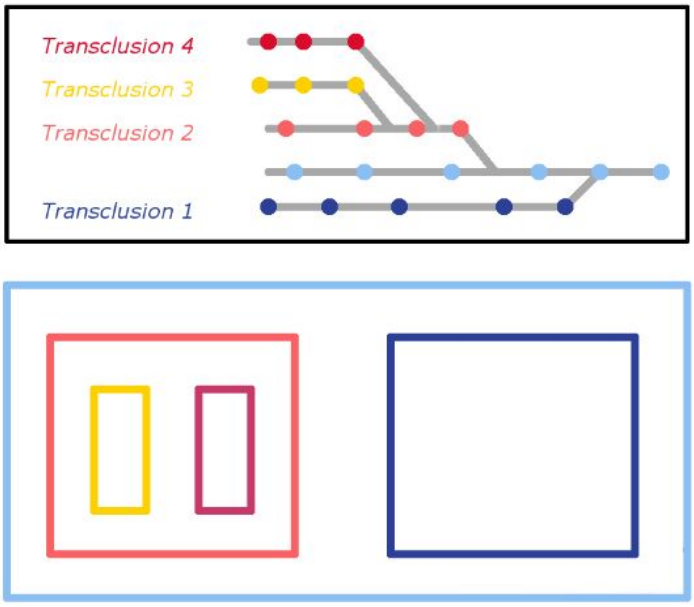


Fig. The panel at the bottom represents the rendered view of document. The panel on the top shows intersecting histories of the transclusions. The user will be able to navigate through history of the document by clicking on the top panel.

3. Create a transclusion plugin which allows to navigate the history of individual transclusions.

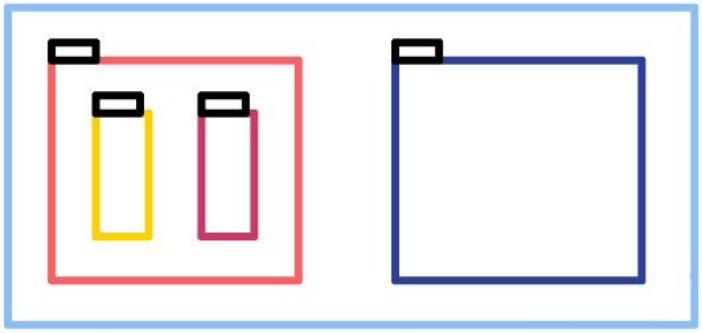


*Fig. A plugin (shown in **black**) is attached to all the transclusions in the document. The user can use the plugin to navigate through the history of the transclusion.*

For this project, we have followed the transclusion plugin approach because it suits our scenarios. However, we believe that other solutions should also be explored in other use cases.

5.2 Difference Between Versions

When the user navigates to alternate version or copy of a webstrate, he needs to be able to identify the differences between different versions. One potential solution to show differences between versions is to display the thumbnail of webpage. However, there are several problems associated with this approach:

5.2.1 Variable size of Webpages

A webpage can be of variable size. When a webpage has large height/width ratio, it becomes difficult to have a thumbnail which captures the entire webpage. When a large webpage is shrunk into a small thumbnail, the details of the webpage inside the thumbnail cannot not be visually distinguished.

5.2.2 Source/Surface Problem

The *source* of the webstrate is the HTML source code of the webstrate. The *source* controls the rendering (or the *surface)* of the webstrate in browser. To identify the difference between different versions, it does not always suffice to see the surface because a document can have same surface for different source. For example, a change in javascript code may not create visual changes in the DOM elements.

This problem can be divided into three scenarios:

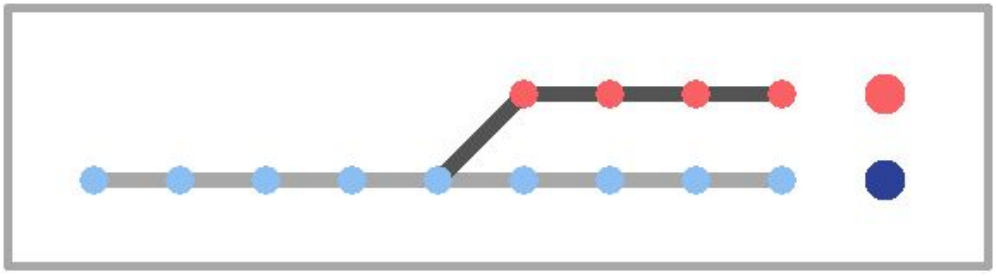
1. Only DOM elements change: When operation is done only on the DOM elements, the change will be reflected in the surface. In this case, it suffices to display only the surface to identify the difference between versions.
2. Only script changes and DOM elements remains unchanged: When operation is done only in the JavaScript code without changing the DOM elements, it suffices to show only source code because there is no visual change in rendering of document.
3. Script which changes the DOM of webstrate: When JavaScript is written to manipulate DOM elements. For example if someone working in D3 or P5.js require to see both the source and surface. In this case, the user require to see both the source and surface to identify differences.

From the above scenarios, we can conclude that user require to see both the source (code) and surface of a version during navigation. For general use case, we should show both source and the surface. However, other solutions can be used for specific cases.

5.3 Navigation Between Copies of Transclusion

As discussed in section 2.2.4, when a webstrate is copied, its child has its own independent history. When the user is using the copied transclusion, he cannot explore the history of copy beyond its creation. These are the few design options we explored to help the user to navigate to the copies of the webstrate.

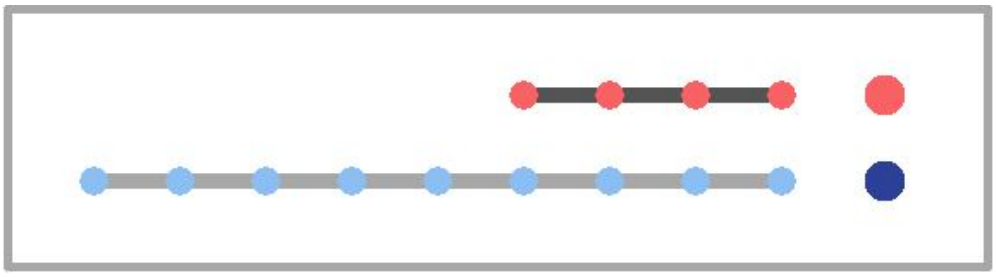
Design 1



Creating link between the history of the child and the parent. This gives an intuition that the **child** webstrate was copied from the **parent** webstrate. The link between the **child** and the **parent** webstrate allows the user to explore the history of **child** webstrate before it was created. This design can become cumbersome when we have many copies of the **parent** webstrate because we need to create ink between all of them.

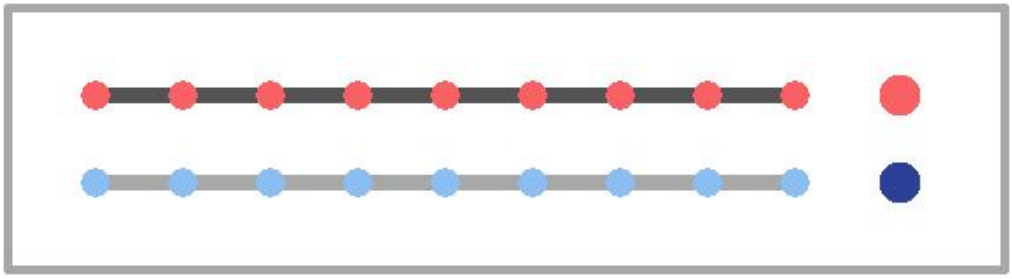
Another challenge with this design is how to show the link when the **child** webstrate was created before the history of **parent** webstrate is shown in the plugin.

Design 2



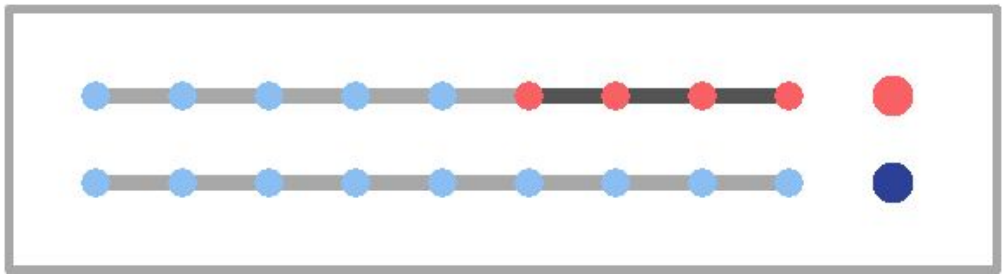
Show the copies of the webstrate and their histories without any links. This is a clearer design than Design 1 because the design will remain clear when there are many copies of the **parent** webstrate. However, the user will not be able to know that the **child** webstrate was created from the **parent** webstrate. Hence he will not be able to explore the history of **child** webstrate before it was created.

Design 3



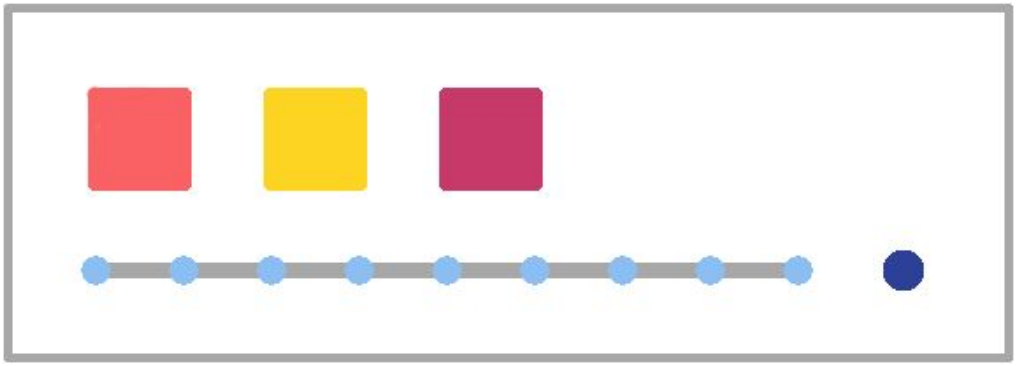
Merge the history of parent into the child. This enables the user to explore the history of the **child** webstrate before it was created. This can lead to problems because the webstrateId of the transclusion changes when the user tries to access the history of child beyond its creation.

Design 4



Extending the history of **child** webstrate to show this history of its parent before it was created. This enables the user to explore the history of the **child** webstrate before it was created.

Design 5



Do not show the history of copies of the webstrates. This design treats each copy as an individual entity. This design does not allow the user to know which webstrate was copied from which one.

5.4 Document History with Transclusions

When a document has its own history, it enables us to create new kinds of transclusions because we can transclude the different versions of the document. We can define two kinds of transclusions: *static* and *dynamic* transclusions.

Dynamic transclusions are the transclusions in which the content in the transcluding document gets updated when we make changes to the transcluded document.

S*tatic* transclusions are the ones in which the transclusion does not get updated when we make changes to the transcluded document. This happens when we transclude a specific version of the document in its history. When make changes to a document, its

individual versions remains unchanged hence its transclusions also remains unchanged.

*Fig. Dynamic transclusion: Figure on the left shows the **blue** document transcluding the **red** document. When new version of **red** document is created, the **blue** document is changed.*

*Fig. Static transclusion: Figure on the left shows the **blue** document transcluding the **red** document. When new version of **red** document is created, the **blue** document remains unchanged.*

5.4.1 Seeing Past Version

History of transcluded document exist independent of the transcluding document. This introduces new problems in viewing history of transcluding document. The following scenario explains the problem associated in viewing history of document with transcluding:

Consider a scenario where we have a **blue** webstrate which dynamically transcludes a **red** webstrate as shown in the image below.

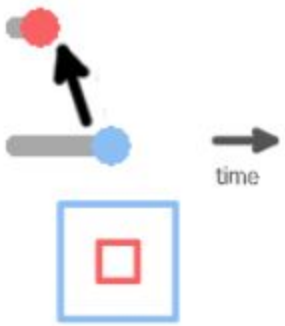


Fig. Figure showing blue webstrate transcluding the red webstrate. The square below depicts how the blue webstrate looks like at this point in time.

When a new version of blue webstrate is created (depicted green), both the blue and green versions of webstrates transcludes the same version of red webstrate.

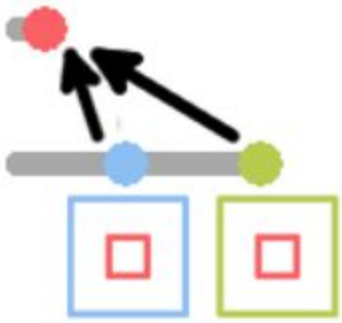


Fig. When a new version of blue webstrate is created (depicted green), both webstrates transcludes the same version of red webstrate.

When a new version of red webstrate is created (depicted yellow), the transclusion of blue version webstrate changes from red version to yellow version.

Fig. When new version of yellow webstrate is created, the transclusion inside blue version changes from red to yellow.

Now, when the user wants to see how the blue version of the webstrate looks like and opens the blue version, he will see the blue version which transcludes the yellow webstrate. However, the yellow webstrate was created after the green webstrate, hence he should actually see the blue webstrate transcluding the red webstrate.

Its is possible that between the blue and green versions there existed multiple versions of red webstrate. The problem is, how do we decide which version of red webstrate to show when the user wants to see how the blue webstrate looked like.

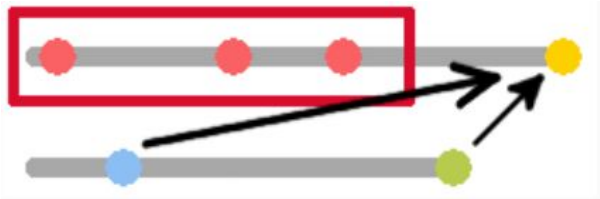


Fig. Between the blue and green version, there can be multiple versions of red webstrate.

Possible solutions to this problem:

- Show the first version of red webstrate after the blue webstrate was created.
- Render the version of red webstrate before the next version of blue webstrate was created.
- Give user the ability to choose which version he wants to transclude.

For now we have decided to show the version of red webstrate before the next version of blue was created as shown below.

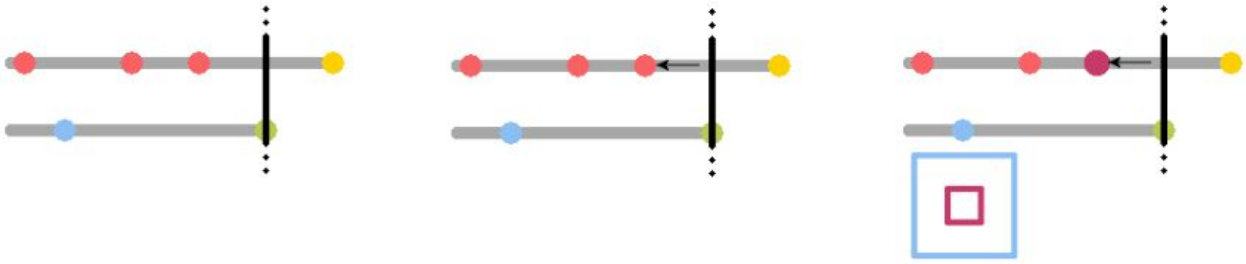


Fig. When when the user want to see how the blue webstrate looked like, we render the version of red webstrate before the green version was created.

6 Transclusion History Plugin

6.1 Design of Plugin

In the last section we have concluded that the user require to see both the source and surface of the webstrate for navigation. Our plugin allows the user to explore the history of each transclusion individually.

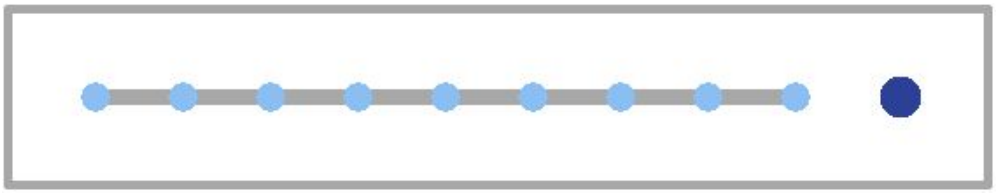


Fig. The the blue circles on the left depict the different versions of the webstrate in static mode of transclusion. The blue circle on the right shows the dynamic mode of transclusion of the webstrate.

In the last 5.5, we concluded that the user needs to see both the source and surface of the webstrate when navigating through different versions. Our plugin tool allows the user to see the source of the transclusion within the <iframe> itself which allows the user to explore the entire surface of the webstrate. Our plugin tool also allows the user to explore the source of the transclusion in <textarea> tag just below the transclusion.

6.2 Functionalities

Our plugin enables the following functionalities on the transclusions:

- Tagging a given version of a webstrate. This enables the user to easily navigate to past version of the webstrate.
- Copying webstrate. When the user is using static transclusion, he cannot edit the webstrate because each webstrate has a single history timeline. To enable editing of a static transclusion, the user can create a copy of the webstrate where he can make new edits to the webstrate.
- For easy navigation, the history panel remains open when the version of its parent is changed.

6.3 Technical Implementation

A *history* webstrate is created which can be used to explore document history of any given webstrate. These are the steps which are involved in exploring the history of webstrate:

1. Transclude the webstrate whose history needs to be explored in the *history* webstrate.
2. Wait for 1 second for all the transclusions to load. Since all transclusions are loading asynchronously and parallely, it is not possible to know exactly when all the transclusions have finished loading. We are assuming that after 1 seconds, all the transclusions are loaded.
3. Find all transclusions recursively inside <iframe>.
4. For all the transclusions add a <transient> tag before and after their <iframe>. This is done to prevent adding new operations in the transcluded webstrates. The UI elements for the plugin are added inside the <transient> tag. Event *OnClickListner* are attached to the UI elements to handle user interactions.
5. When a transclusions version is changed, repeat steps 2-4.

6.3.1 Drawbacks

- A webstrate cannot be transcluded twice in the document. Due to asynchronous loading, the the technical implementation becomes difficult.
- When a change has occurred in a transclusion, its changes events are not automatically notified in the parent history plugin, this prevents updating the history panel and script panel in the plugin automatically.
- The plugin tool creates new operations while navigating histories inside a dynamic transclusion.

6.3.2 Rendering Past Version

When the user wants to see past version of a document, all the dynamic transclusions needs to be converted to their static version. The history tool first creates the transclusion tree of the webstrate as shown below.

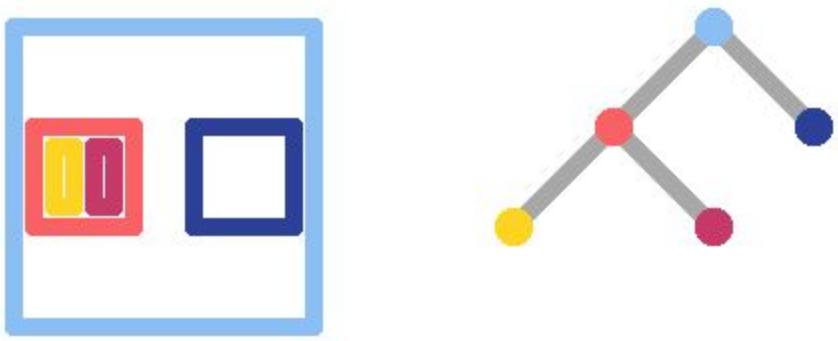


Fig. Figure on the left shows the rendered webstrate. Figure on the right shows the transclusion tree of the webstrate on the left.

When the version of a webstrate is changed, the timestamp of that version is sent to all its children transclusions recursively and their version is changed according to the timestamp.

7 Selective Undo/Redo in Context of Webstrates

Operation history in webstrates allow us to enable undo/redo functionality in webstrates because we can edit and apply new operations on the document. When the history of a document is edited, the changes will not be reflected in the dom in real time because webstrates stores the code for the latest version. To apply reflect the applied changes to the latest version, the document needs to be rebuilt.

Two prevalent selective undo/redo models are the inverse model and the script model. Both are discussed in the context of webstrates.

7.1 Script Model

Script model of selective undo consists of rebuilding the document while skipping a particular operation. As discussed in the last section, when a operation is skipped, the webstrate needs to be rebuilt from version 0.

Fig. In script modes, to undo an operation, the document is rebuilt by skipping that operation

An interesting question in script model is to explore if the change in history propagates to the children of the webstrate because conceptually a child is created from the parents history. This is a technical challenge as well as a design challenge. The user may or may not want to treat the copy of the webstrate independent of its parents.

7.1.1 Problem with Script Model

Similar to the problems of selective undo in Painting applications (Brad & Ashley), script model in webstrates can fail in certain scenarios. One of the scenario is when the operation of creating of element is undone. Then all the subsequent operation on that element will make no sense because it is impossible to apply a command to an object which isn't created.

7.2 Inverse Model

Inverse model consists of applying inverse of an older operation to the current version of the document. This approach is technically simpler and efficient because we won't need to rebuild the entire document again.

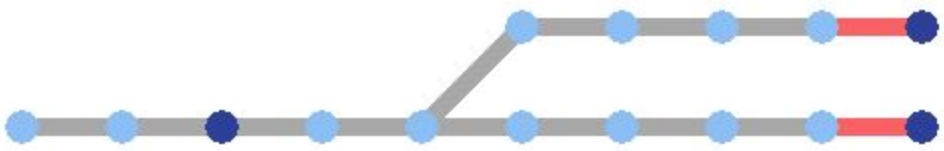


Fig. In inverse model, to undo an operation (blue), the inverse of that operation is applied to the latest version of the webstrate

However, this model fails in scenario when we are applying inverse operation to an object which is already deleted in the past. Similar to script mode, applying an operation to a deleted object makes no sense.

8 Future Work

- Understanding how the user conceptualizes transclusion. What are the different mental models the user can use to work with transitions.
- Doing user studies to see if transclusion with histories enhance/improve the workflow of the user in the scenarios?
- Exploring source/surface problem. How to show difference between two documents when they have source and surface?
- Exploring alternate conceptual models for the timeline. In this project we have used plugin for each transclusions which treats history of each transclusion independent. Alternate conceptual models can be explored which might help the user to easily understand and use transclusions with histories.
- Clustering of multiple history versions of a document. Due to constraints in the working memory of the user, the user does not remember older document versions. Clustering multiple history versions of a document might help the user when navigating through histories. As an example, all the versions can be clustered by the day they they were created, the user will be able to navigate history by date.
- Merging the operations of the transcluding document and transcluded document. Exploring if there are new possibilities which can be done when the operations of the transcluding and the transcluded document are merged.
- Parallel operations when different people are working on the same document. When two users are working on same document, it is better to maintain the operations of each user independent. This can be explored in context of documents such as Google Docs/Sheets/Slides.
- Keeping transclusions of same document synchronized when change is made. When a document is transcluded multiple times inside another document, should we keep its transclusions synchronized when the version is changed? What are the possible use cases for this functionality? One possible scenario is while using presentation, the slide and its thumbnail should be synchronised, when the user changed the version of slide, the version of its thumbnail should also change automatically.

9 References

Clemens & James (2015), Webstrates: Shareable Dynamic Media
Brad & Ashley (2015), Selective Undo Redo for Painting Applications