A Style Sheets Based Approach for Semantic Transformation of Web Pages

Gollapudi V. R. J. Sai Prasad(✉), Venkatesh Choppella, and Sridhar Chimalakonda

Department of Computer Science & Engineering, Indian Institute of Information Technology Tirupati, Tirupati, Andhra Pradesh, India
saigollapudi1@gmail.com, venkatesh.choppella@iiit.ac.in, ch@iittp.ac.in

Abstract. The goal of this paper is to propose a style sheet based approach for enabling semantic transformations of existing, already published web pages. Traditionally, web page transformations were largely driven by approaches such as XSLT that focuses on XML documents, and CSS that transforms the style of HTML content. However, despite their wide usage, XSLT is considered as too complex and rigid while CSS only focuses on form and the aesthetics of display. To address this major concern, we propose a new type of style sheet that is (1) applicable on existing, published web content, (2) able to perform semantic transformations, and (3) able to do some client-side processing of published web content. We present the design of the prototype and demonstrate the idea of using semantic style sheets by delivering a set of multiple transformations of a random web page from NASA website.

Keywords: Style sheet · Semantic transformation
Web page transformation · Web accessibility · Client side modification

1 Introduction

The goal of this paper is to propose a style sheet based approach for enabling semantic transformations of existing, already published web pages. This is motivated by our earlier work [19–21] in Renarration of web content. The approach of using style sheets for enabling semantic transformation of web pages is novel and it contributes to the larger work of Web Accessibility.

1.1 Background and Motivation

While the word *renarration* is uncommon, the concept itself is fairly prevalent and straightforward. For instance, as humans, whenever we are socially communicating an idea to somebody, we ensure that it is expressed and delivered in a way that is suitable to our audience. The idea in our head may be one, but its articulations may be many. For instance, a technical idea may be presented in a scholarly way to a researcher, but the same idea may be re-narrated in a more simplistic manner to a layperson. This variation in the content, delivery and expression ensures that we make our idea more sensible to our audience.

Examples of renarration of written content may also be found in such day-to-day things as commercial documents and academic literature. In teaching and learning of mathematics and sciences the variant versions of the original are labeled as MERs or Multiple External Representations [27]. We frequently use a diagram, a graph or a table to better illustrate a point that we may have already made elsewhere in text [22].

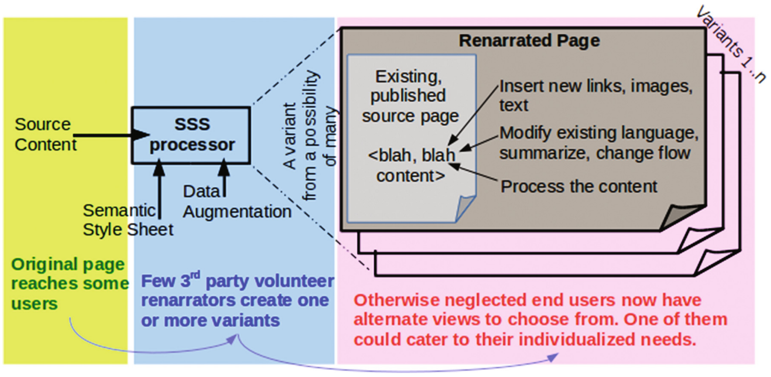

Source Content
SSS processor
Semantic Style Sheet
Data Augmentation
A variant from a possibility of many
Renarrated Page
Existing, published source page
<blah, blah content>
Insert new links, images, text
Modify existing language, summarize, change flow
Process the content
Variants 1..n
Original page reaches some users
Few 3rd party volunteer renarrators create one or more variants
Otherwise neglected end users now have alternate views to choose from. One of them could cater to their individualized needs.

Fig. 1. A web page may be said to have a scope and reach which may exclude some end-users with different needs. 3rd party volunteer renarrators can include them by creating alternative views.

1.2 Renarrating the Web

We are interested in taking this concept of renarration and applying it to existing, published web page content. See Fig. 1. Current material on the web, though prolific with lot of good information, it is not in itself equally accessible by all. Few segments of the user population, for instance, those with poor English skills or those with different or maybe even lower-order thinking skills, or those coming from a different socio-economic or cultural context may find some of the existing material foreign and incomprehensible [19]. For these minority groups, renarrating the original into a more simplified variant, or renarrating the original in a different language, or renarrating the source to have more diagrams and references may prove useful.

Simple renarrations of web pages could potentially include one or more of the following:

- **Augmenting:** adding more diagrams, links, videos; adding locally relevant examples, adding local information etc.
- **Modifying:** summarizing a complex piece of text, translating something into vernacular, removing some clutter, changing language etc.

- **Processing:** changing the numbering system from one representation (e.g. Arabic) to another (e.g. British), computing forex, changing the formats of dates, changing the representation of units from Kilograms to Pounds, or from Kilometers to Miles etc.

Dinesh et al. [7] talk about the difficulties that laborers may face in interpreting the online contents of a government labor law document. Renarrating it into some non-legalese language may help even some common users better interpret it. They also cite the example of a fire safety website being renarrated to include local fire safety standards and contacts.

1.3 Web Accessibility Problem

According to internetlivestats.com[1], a site dedicated to web analytics, there are over 1.2 billion websites and nearly 3.7 billion Internet users in the world today. Of this user-base, 48.4% are from Asia and 9.8% are from Africa. According to another web analytics site – W3Tech.com[2], English is the most dominant language on the web. That is, there are over 51.3% English websites out there on the web today. This is despite the fact that 1.5 billion users are from a non-English background.

In [21] we had already highlighted the Web Accessibility issue of these non-native English speakers. In a study of N = 372 college students [23] we observe that, just by adding instructions in local vernacular would help in improving accessibility. That is, just by adding guidance, by adding additional links to more locally relevant information, by giving tips & tricks and by giving more examples in vernacular to an already published web page, one could help these non-native English speakers make more sense out of that existing English content.

In addition to the language challenge, localization (i.e. changing information from one representational system to another), personalization (i.e. matching to the preferences of an individual user) and translation from one mode to another (i.e. moving text dominant content to visual information or even braille) could all help in improving Web Accessibility.

The bottom line is that to improve the accessibility of an already published web content, one needs to have a mechanism to alter the content of a web page at the semantic level. Renarration of a web page, is thus this ability to semantically transform web pages. In this paper, we propose the notion of *Semantic Style Sheets* as an approach to enable this semantic transformation (or renarration) of web pages. We believe that by developing a style sheet based approach will not only be resolving our challenge, but also be contributing to the larger goals of Translations, Personalization, Localization, and Customization as well.

[1] http://www.internetlivestats.com/internet-users/.
[2] https://w3techs.com/technologies/overview/content_language/all.

1.4 Notion of Style Sheets

History: The notion of style sheet is not new. It comes to us from the publishing industry where editors, designers and typesetters physically marked the author's printed manuscript with a blue pencil. The designers and typesetters came in after the editing phase. They specified things like the location and size of margins, the layout of the chapters, the fonts to be used in printing the text etc. The whole concept was based on the principle of Separation of Concerns (SoC) between content and its presentation, which continues even today.

The electronic publishing (or e-publishing) industry took the concept of markup and presentation and used it digitally [8]. They used the notion of style sheets to change the way a particular document appeared on screen, from how it appeared on print, and differentiated it from how it was recorded on a CD-ROM. SGML, the predecessor to XML (and HTML) used this in their definition of Document Type Definitions (DTD) [15]. Overtime, with the advent of HTML, the general variation in DTD was fixed to HTML versions and the styling aspect slowly moved out to CSS.

Popular Style Sheets on the Web: SGML initially started out with Document Style Semantics and Specification Language (DSSSL) but it was deemed too complex, and later on was not directly applied to web standards [25].

For the HTML users of the web, CSS (Cascading Style Sheets) has now become the defacto standard [4]. It was first proposed by Hakon W Lie [14], and it later became a CSS1 standard that was promoted by W3C[3]. Now with CSS3, there are even some preprocessors like *Less*, *Stylus* and *Sass* that are available for handling CSS content [16].

For the XML users, XSLT with XSL-FO has been offered as an option [3]. Currently, XSLT is more associated with data-rich, XML-marked, database-interacting web pages. While XLST does not enjoy as much success as its HTML cousin CSS, XSLT does provide a very wide range of transformation possibilities. But, when it comes to already published but not so "well formed" HTML pages, XSLT is less tolerant and less useful for semantic transformations.

Usage of Web Style Sheets: It was already mentioned that style sheets have been popularly used to deal with aesthetics and presentation. In particular they can be used to adjust color, size and style of font, layout, margins, spacing etc.

While XSL styling has the ability to re-assemble and transform documents, like XSL-FO, CSS is chiefly limited to look-and-feel only. Since CSS is the most popular of the style sheets, we restrict our attention to it only.

CSS for HTML: The directives of a CSS Style Sheet are articulated as rulesets. Their syntax is: *selector* {*property:value;... property:value;*}. The selector

[3] World Wide Web Consortium; A standards body for the Web; https://www.w3.org/TR/html4/present/styles.html.

is the HTML element tag. The property value pairs are unique and defined by the CSS specification released by W3C. Rule-sets are default and apply to the entire document, unless overridden by another applicable rule-set given elsewhere. Rule-sets can be defined inline or imported from another *.css* file.

Here is a snippet of HTML source code from a real NPTEL web page[4]. It shows how styles are integrated into a real web page.

```
<!DOCTYPE html>
<html>
<head><title>NPTEL</title>
   <meta charset="UTF-8">
   <link rel="stylesheet" href="...css">
   ...
   <link rel="stylesheet" href="...footer.css">
   <link rel="stylesheet" href="..css">
   <link href="http://fonts...Cookie" rel="stylesheet">
   ...
</head>
<body>
  ...
  <link rel="stylesheet" href="...lightslider.css"/>
  <link rel="stylesheet" href="...testimonial.css"/>
  ...
  h2 { text-align:top;
       font-weight:bold;
       color:#fff; }
  ...
</body>
</html>
```

Execution: The execution of the style declarations happens in the browser, in rendering engine, as part of the painting work[5]. A CSS processor interprets the rule-sets and produces a CSS Object Model called CSSOM and links it to the Document Object Model (DOM) for rendering. Currently the more popular browsers like Microsoft's Internet Explorer, Mozilla's Firefox and Apple's Safari mostly support CSS and only partially support XSLT[6]. This is yet another reason for our detailed focus on CSS.

1.5 Layout of the Paper

Thus far, in the **Introduction** section, we have already established (1) Web Accessibility and Renarration as the motivation and problem space for our work;

4 http://nptel.ac.in/.
5 http://taligarsiel.com/Projects/howbrowserswork1.htm#The_browser_main_functionality.
6 http://greenbytes.de/tech/tc/xslt/.

(2) we have discussed the background and context of style sheets, both as a notion as well as in practice. Also, we established the need for us to be able to do semantic transformation of web pages.

Going forward, we start with our **Design Consideration** for our Semantic Style Sheets (SSS). This will set the tone for what we are trying to accomplish with our style sheets. We do a literature survey to discuss the gaps with the current approaches and why we are motivated to select style sheets as an approach. We then proceed to discuss the actual **Design** where we give a indicative grammar and the structure of the SSS. Later in **Implementation** we discuss how the SSS has been actualized in code. In **Validation** section we apply the prototype we developed for demonstration to a NASA web page and show how it can be renarrated to suit the various needs of a few minority user communities. Finally, we finish with some reflections and insights in our **Discussion and Conclusion** section.

2 Design Considerations for SSS

At a high level we want to semantically transform a web page to make the renarrated web pages more accessible to a wider group of users. But, more specifically, here are our requirements:

1. It should allow for making changes to the semantics of the page. That is, user should be able to add new content or replace some existing information present in a given published page. We call this **Augmentation** and **Modification** respectively.
2. It should allow for changes to be made to the flow of material in the given page as well. That is, order of nodes should be reconfigurable. This is also part of our **Modification** requirement.
3. It should facilitate the processing or the computation of some existing content to create new values. We call this **Processing**.

3 Literature Survey of Techniques to Modify Web Content

Network and Proxy Solutions: Many options exist for manipulating already published web content. For instance, proxy assisted network based transcoding options exist for changing web page content to fit into the display requirements of a smaller-screen device [13,28]. Network side solutions use separate app servers and backend development. Proxy based solutions intercept and modify content before it reaches the browser [10]. However, these servers require to be configured into the browser flow. That is, the IP address of the proxy needs to be input into the browser. Such imposition may pose some concerns: For instance,

(1) users may perceive configuration as a 'technical' task; or, (2) the users may have some other mandatory institutional proxy that they are compelled to use which forbids them from using ours; or, (3) the end users may have security concerns. Such conditions have been known to dissuade users similar to our target users from using a proxy based solution [12]. It is for these reasons that we did not opt for a network side, proxy based solution.

Web Augmentation Solutions: Web Augmentation techniques exists to allow for the modification of content on the client side [6]. Client-side scripting tools like *GreaseMonkey* [18] or testing tools like *Selinium* [2] exist to enable modifications of published content at the browser level. While the browser is now-a-days quite powerful and capable of running complex *JavaScript*, it is still a programming option. Also, now-a-days security concerns are forcing people not to opt for enabling *JavaScripts* [1]. So, this discouraged us from going with this choice.

Style Sheet Based Solutions: As already indicated, style sheets have also been used to make modification to (the style of) a page. And, when it comes to exploring the choices available to the user, there are mainly two prominent options, namely CSS and XSL.

In our case, for renarration we are opting for a style sheet based choice (over the above listed options) because we wish to empower the end-user and her agents. From a Web Accessibility point of view, we notice that it is often necessary to modify the content for a diverse set of (and also an individualized set of) needs of only a minority of end-users [6,19]. For such low volume users, a complex, coding-intensive, back-end solution may not be appropriate. What they may prefer are simple environments that can be run on the client side, by themselves, or by 3rd party volunteer supporters. Having such a nimble client side environment would assure them more solutions, and also quicker development time. We opted for a style sheets based solution because they offer us this promise. Also [12] suggest that for web accessibility their users preferred a style sheet based approach.

Challenges with the Style Sheet Option: The style sheet based approach for making semantic transformation essentially presents three choices: Option one, go with XSLT; Option two, go with CSS based approach; Or option three, create own style sheet.

Challenges with option one is that XSLT is not supported by all browsers. It is also quite complex to work with[7]. Tool support is also dwindling. Moreover, its popularity has been restricted to XML or data-rich content only. Also, there has been lot of criticism on XSLT, claiming it to be too rigid and cumbersome. This discouraged us from going with this option.

[7] Criticism on XSLT can be found here on Stack Overflow. Refer: https://stackoverflow.com/questions/78716/is-xslt-worth-it.

CSS based option two has its own challenges as well [26]. While CSS is quite popular, it appears to be predominantly focused on styling and aesthetics only. While it is popular and tightly integrated with HTML, the syntax orientation – i.e. it works with selectors and not semantics of the content – makes it difficult for us to do higher order semantic operations. To meet our requirements, we need control at the node level or the concept level, not selector level. Moreover, it lacks the power to compute or process content information. Due to these limitations, we opted out of this as well.

Option three has to do with developing our own style sheet. This task can indeed be quite complex and large. However, selecting this option allows for us to custom design a semantics based "style" declaration that allows for Augmentation, Modification and Processing, which are now missing in CSS. By creating a new style sheet we are not intending to replace either XSL or CSS. Instead, this new proposed style sheet will only add to that portfolio.

4 Design of Semantic Style Sheet

Design Goals: There are two areas in which our proposed Semantic Style Sheet (SSS) differs from the popular CSS. One is in concept – ours is focused on semantics and not just style – and the other is in its application – we empower the end-user or her agent instead of the author or the publisher. By empowerment we mean that the creation of a new style sheet, the potential co-existing of multiple style sheets for the end user to select from, the definition of criteria of when to apply which style sheet etc. are all done without the involvement of either the author or the publisher of the original source page. We summarize our conceptual goals as:

1. **Augment** (or add) new content (like text, links, images etc.)
2. **Modify** existing content (i.e. change or replace content form an existing page)
3. **Process** content (e.g. compute over number, currency, units of measure; carry intelligent processing over text etc.)

Our application or usage goals include:

1. **Meta data:** SSS must have information on who created it, when etc.; there should also be information on the SSS, its label, ID, description etc. And, finally, the web page URL for which the SSS applies.
2. **Selection Criteria:** There should be some guidance on when this particular SSS must be applied when there are other choices available and co-existing for a given web page (i.e. URL).

We present indicative grammar to articulate our conceptual goals. And, we have couched the semantic directives which augment, modify and process, in a larger structure to ensure we meet the application goals.

4.1 Indicative Grammar

We articulate our design for the SSS directives by using a simple grammar. We use the following EBNF notation (as defined in [24]):

= for definition,
, for concatenation,
; for termination,
— for alternation,
[...] for optional,
{...} for repetition, and
e for null

```
SSS = {Directives}
Directives = (SeqNo, Operation)
SeqNo = <integer>
Operation =
    (Add (src-location:<xpath>, add-loc:(above | below),
        (new-url:url | new-nod:<xpath> | new-text:<string> |
         new-tool-tip-text:<string> | new-link:<url>) )) |
    (Repl (src-location:<xpath>,
        (new-nod:<xpath> | new-text:<string> |
         new-tool-tip-text:<string> | new-link:<url>) )) |
    (Process [xform-crncy-inr, xform-unit-british, xform-temp-celsius])
```

4.2 The SSS Document

The SSS Document is the larger structure containing the prior described SSS directives. This document consists of three segments: (1) some meta data (2) a list of semantic transformation directives, and (3) a selection criteria.

Segment 1 of the SSS document (Meta-data) is to contain the URL of the web page which is being transformed, the creator's information, and, label, description and ID for this SSS. Segment 2 of the SSS document (SSS Directives) is to contain the declaratives having a sequence number, a operation and some attributes as indicated by the indicative grammar. Segment 3 (Selection Criteria) is about identifying the criteria that is to be used for selecting this particular SSS from a choice of many. It consists of rules for selection.

Here is a snippet of a basic SSS document structure which we are using to renarrate a web page.

```
{"seg1":{  "sssName":"S1",
           "description":"Created by R1 for U1 community",
           "wp-url":"https://www.grc.nasa.gov/...html" },
 "seg2":[ { "seqNo":"1",
           "operation":"Add",
           "new-text":"<div><img
                src='http://...steam-engine-works.png'></div>",
            "src-location": "//",
            "add_loc": "above" },
          { "seqNo":"2",
            "operation":"Repl",
            "src-location":"",
            "new-text":"<p>Thermodynamics is the ...energy.</p>", } ],
 "seg3":{ "communityId":"U1" } }
```

5 Implementation of Prototype

5.1 Architecture of Prototype

We build a simple web application to demonstrate our notion of SSS. The front-end (FE) of the application was developed using *AngularJS* [5]. The back-end (BE) runs *Flask* on *Python 2.7* in a *Virtual Environment* [9]. The processing of the SSS happens at the server level, in the BE, in Python. See Fig. 2. To demonstrate the application of an SSS on a given web page, the FE supplies three things to the BE:

1. **URL** - the location of the web page which is to be renarrated (or semantically transformed)
2. **SSS** - the choice of SSS which is to be applied to this web page
3. **UserID** - the identity of the person accessing the web page

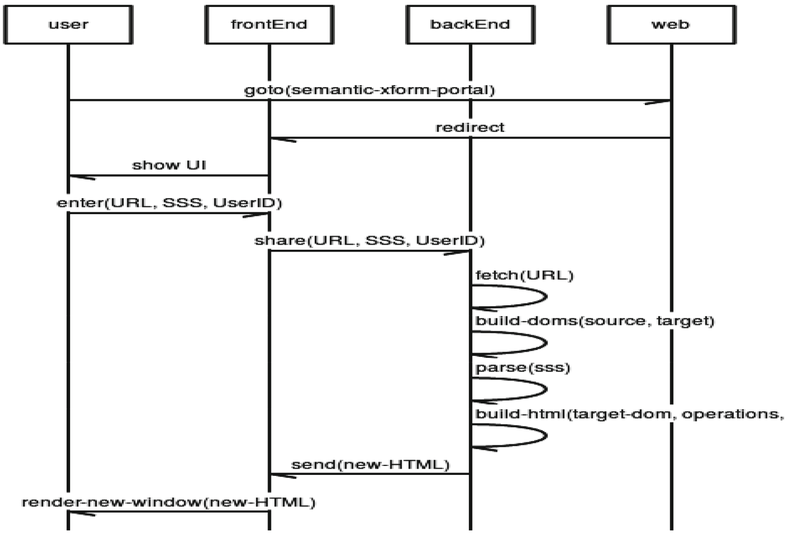
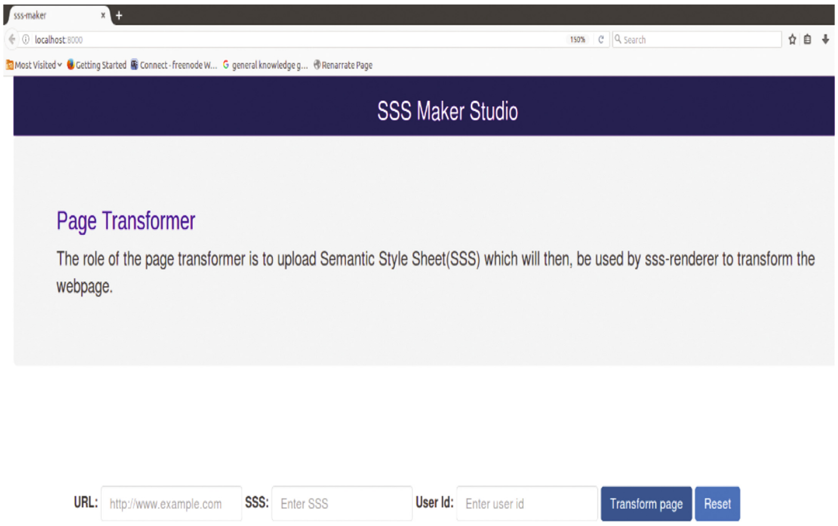


Fig. 2. The sequence chart for the processor application of our SSS.


SSS Maker Studio

Page Transformer

The role of the page transformer is to upload Semantic Style Sheet(SSS) which will then, be used by sss-renderer to transform the webpage.

Fig. 3. The front end of the web app prototype which implements SSS.

See Fig. 3. The BE takes the first parameter, the URL, and uses it to fetch the web page which is to be semantically transformed. It uses the second parameter - SSS - to process the change. And, finally, it uses the UserID to see how the SSS applies to the user. Here is the basic algorithm behind the SSS processing in the BE.

```
fetch-page (URL)
source-DOM = construct-DOM(fetched-page)
target-DOM = source-DOM
sss = parse(JSONobject-sss)
REPEAT
   d = get (next directive)
   a = read d.operation.attributes
   SWITCH (d.operation)
      /* AUGMENTATION handling */
      CASE (add):
          target-DOM = add(a.location, a.new-node, source-DOM)
      /* MODIFY handling */
      CASE (replace):
          target-DOM = replace(a.location, a.new-node, source-DOM)
      /* PROCESS handling */
      CASE (process):
          p = d.operation.attribute [1]
          SWITCH (p)
              CASE (xform-crncy-inr):
                 target-DOM = xform-crncy(p, inr)
              CASE (xform-unit-british):
                    target-DOM = xform-units(p, british)
                 CASE (xform-temp-celsius):
                    target-DOM = xform-temp(p, celsius)
                 DEFAULT: Log (operation, unknown)
                 Print ("Error: unknown Process operation attempted", p)
        DEFAULT:
            Log (operation, unknown)
            print("Error: unknown operation attempted", operation)
UNTIL all directives done
new-page = construct-HTML(target-DOM)
front-end = send(new-page)
```

5.2 Implementation of SSS

The SSS document has been implemented as a JSON object. This data structure has the three segments that were previously identified.

Processing of SSS: Typically a style sheet has a processor associated with it. For web documents, the browser contains this processor. In our prototype implementation, we have positioned the code in the BE, in the network side. That is, we transform the source web page with SSS before it comes to the browser.

In the server side, the processor could parse and interpret the SSS directives as if it were processing a Domain Specific Language (DSL) [17]. However, in the prototype, we do not do parsing and interpretation, which could easily be added later. Instead, we focus on the transformation algorithm. The pseudo-code for this algorithm has already been shared. This algorithm simply transforms source HTML into a target HTML by way of the SSS directives.

Accessing DOMs and XPaths on the Server: To augment or modify a portion of an HTML, we use DOMs and/or XPaths. But, in current web implementation, the DOMs and XPaths for a given web page are constructed within a browser. Since the processing for our implementation is now in the server side (and not the browser), we lack access to the DOM of both the source and the target pages. To overcome this problem, we use a headless browser, which in our case is *WebKit* based *PhantomJS* [11]. Using the headless browser, on the server side, we create two instances of it to access the source and target DOMs. In this way we gain access to the XPath on the server side.

Additions, replacements or just extractions of node elements from a page (or to a page) are now done in this headless browser using DOM API. Once a target DOM is created, we have our HTML to push to the front-end. See Fig. 2. Completing the target DOM requires processing all the directives in the SSS. Which, in turn, is equivalent to completing the semantic transformation of the source web page.

Implementation of Selection Criteria: Selection criteria can consist of rules which are then addressed by a rule engine like *npm*[8]. However, for our initial implementation, we did not go this route. As we were only trying to demonstrate the feasibility of this aspect, we simply carry out a regular expression match on one of two variables: Community ID or a User ID.

The idea is that one single page may have multiple, co-existing, alternative views. Each of these is a renarration of the original. And, each requiring its own SSS document. The selection of which view to present turns into a selection of which SSS to implement. This selection criteria section enables that choice.

A document meant for the blind users my have their community ID as its selection criteria. Similarly, the same document could also be meant for people of a different language. Their version may have a different SSS, with its own selection criteria indicating the different language speakers. During rendering, the selection criteria checks to see who is viewing the SSS. If it is a registered blind user, then blind SSS is processed. Else, if the user is of a different language, then the diff-language SSS is processed. The community ID and/or user ID are obtained through login and registration processes.

Selection criteria and algorithm can indeed be made more sophisticated to handle complex cases. For example, a rule engine may be established to optimize this facet of SSS processing. But this is seen as future work and is considered out of scope for this paper.

6 Validation of SSS

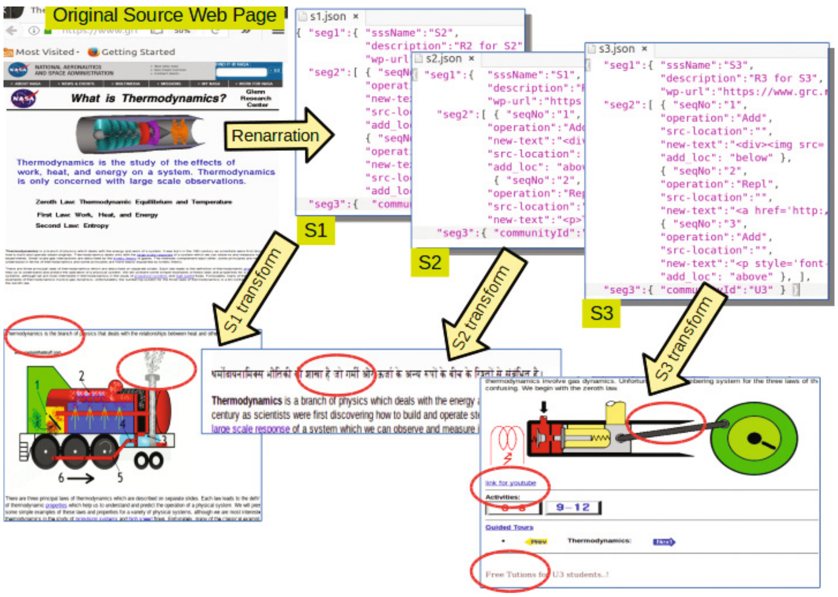
We validate our idea of SSS by applying our prototype to a real web page and renarrating it. For our test, we arbitrarily chose a page that was focused on a 'complex' topic of Thermodynamics from NASA's website[9].

For our demonstration, we made the assumption that the NASA site is confusing and inaccessible to certain non-native English speaking users. We took the roles of three independent, unrelated renarrator volunteers (R1-3), having to develop 3 different SSS (S1-3) for our respective target audience (U1-3).

The assumed objective of R1 was to reduce the complexity of NASA's page for U1. She thus creates an S1 for it. Similarly, R2's objective in creating S2 was to meet the vernacular needs of U2. That is, U2 is Hindi speaking audience, and they need guidance in it. R3's objective was to insert advertisements into a page for all U3. The general idea here is that renarrations can be many, and they can co-exist, but they have to be targeted to a community or a user. For instance, when U1 logs in, they should only see the S1 renarrated NASA site. Similarly U2 should get the S2 renarration, and U3 should get S3.

See earlier given SSS doc snippet for S1. See Fig. 4 for the process of converting original source web page (top left), to S1–S3 (top right), finally yielding three different outputs (bottom). The bottom portion showcases the actual transformed output of our web application. The S1-3 are also original. They were

[8] https://www.npmjs.com/package/json-rules-engine.
[9] https://www.grc.nasa.gov/www/K-12/airplane/thermo.html.


Original Source Web Page
Renarration
S1
S2
S3
S1 transform
S2 transform
S3 transform
What is Thermodynamics?
Thermodynamics is the study of the effects of work, heat, and energy on a system. Thermodynamics is only concerned with large scale observations.
link for youtube
Activities:
9 - 12
Guided Tours
Thermodynamics:
Free Tutions for U3 students..!

s1.json

```
{ "seg1":{ "sssName":"S2",
           "description":"R2 for S2"
           "wp-url
  "seg2":[ { "seqN
           "operat
           "new-te
           "src-lo
           "add_lo
           { "seqN
           "operat
           "new-te
           "src-lo
           "add_lo
  "seg3":{  "commu
```

s2.json

```
{ "seg1":{  "sssName":"S1",
            "description":"
            "wp-url":"https
  "seg2":[ { "seqNo":"1",
            "operation":"Add
            "new-text":"<div
            "src-location":
            "add_loc": "abo
             { "seqNo":"2",
            "operation":"Re
            "src-location":"
            "new-text":"<p>"
  "seg3":{ "communityId":"
```

s3.json

```
{  "seg1":{ "sssName":"S3",
            "description":"R3 for S3",
            "wp-url":"https://www.grc.
   "seg2":[ { "seqNo":"1",
            "operation":"Add",
            "src-location":"",
            "new-text":"<div><img src=
            "add_loc": "below" },
            { "seqNo":"2",
            "operation":"Repl",
            "src-location":"",
            "new-text":"<a href='http:
            { "seqNo":"3",
            "operation":"Add",
            "src-location":"",
            "new-text":"<p style='font
            "add_loc": "above" }, ],
   "seg3":{ "communityId":"U3" } }
```

Fig. 4. Three renarrated views being constructed out of three SSS. Real output of a working prototype.

designed to semantically transform the content by adding new content (augmenting), altering existing content (modify), and computing some numbers (processing). The output clearly shows how the content is now semantically transformed to meet the individualized needs of U1-3.

7 Discussion and Conclusions

The prototype demonstrates our proposed idea for our SSS. The intent of SSS is not to replace an existing style sheet like CSS or XSLT. Instead, our intent here is to complement the power of styling the CSS provides with an additional control on semantics.

In developing our renarration idea for addressing the Web Accessibility problem, we discovered that semantic transformation of existing, already published web sites is necessary. Amongst the various approaches out there, we opted to work with a style sheet based approach because it offered non-technical involvement of the end-user. And, it empowered the end-user. Investigating the needs we discovered that our need to Augment, Modify and Process content was not being sufficiently addressed by the current defacto standard - CSS. To facilitate semantic level transformation of web page content we proposed a new style sheet called Semantic Style Sheet. We proposed a grammar for it and developed a simple processor for it. Its utility was finally demonstrated by way of a prototype application.

Through this exercise we realize that

1. style sheets can indeed be developed for doing more than just style adjustments
2. the concept of a semantic oriented style sheet shows promise and can be further developed; perhaps it can be turned into a DSL and be linked with a rule engine
3. Web Accessibility needs of the non-body-disabled user can, to some degree, be met by renarrating existing, already published web content.

Upon reflecting, we also realize that due to the sheer volume of web content that is out there, as a next step, we need to move from a manual renarration process to an automated one. This requires a set of standard semantic structures in specific domains, a set of renarration needs and techniques, which can then be analyzed by developing a set of tools that can automatically process and apply renarration techniques. We leave the task of automation of renarration as a logical next step or a future activity. Finally, we see the work in this paper is a first step towards a major research direction spinning off multiple research areas in the space of semantic web, renarration and web accessibility.

Acknowledgements. The authors are grateful to the effort by Ms. Sadhana Virupaksha, and Mr. Raghav Mittal for developing the prototype for SSS.

References

1. Bandhakavi, S., Tiku, N., Pittman, W., King, S.T., Madhusudan, P., Winslett, M.: Vetting browser extensions for security vulnerabilities with vex. Commun. ACM **54**(9), 91–99 (2011)
2. Bruns, A., Kornstadt, A., Wichmann, D.: Web application tests with selenium. IEEE Softw. **26**(5), 88–91 (2009)
3. World Wide Web Consortium, et al.: XSL transformations (XSLT) version 2.0 (2007)
4. World Wide Web Consortium, et al.: Cascading style sheets level 2 revision 1 (CSS 2.1) specification (2011)
5. Darwin, P.B., Kozlowski, P.: AngularJS Web Application Development. Packt Publishing, Birmingham (2013)
6. Díaz, O.: Understanding web augmentation. In: Grossniklaus, M., Wimmer, M. (eds.) ICWE 2012. LNCS, vol. 7703, pp. 79–80. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35623-0_8
7. Dinesh, T., Uskudarli, S., Sastry, S., Aggarwal, D., Choppella, V.: Alipi: a framework for re-narrating web pages. In: Proceedings of the International Cross-Disciplinary Conference on Web Accessibility, p. 22. ACM (2012)
8. Goldfarb, C.F.: SGML: the reason why and the first published hint. J. Am. Soc. Inf. Sci. (1986–1998) **48**(7), 656 (1997)
9. Grinberg, M.: Flask Web Development: Developing Web Applications with Python. O'Reilly Media, Inc., Sebastopol (2014)
10. Gupta, S., Kaiser, G., Neistadt, D., Grimm, P.: DOM-based content extraction of HTML documents. In: Proceedings of the 12th International Conference on World Wide Web, pp. 207–214. ACM (2003)

11. Hidayat, A.: PhantomJS: headless webkit with Javascript API. WSEAS Trans. Commun. (2013)
12. Kurniawan, S.H., King, A., Evans, D.G., Blenkhorn, P.: Personalising web page presentation for older people. Interact. Comput. **18**(3), 457–477 (2006)
13. Laakko, T., Hiltunen, T.: Adapting web content to mobile user agents. IEEE Internet Comput. **9**(2), 46–53 (2005)
14. Lie, H.W.: Cascading HTML style sheets-a proposal. World Wide Web Consortium (W3C) (1994)
15. Maler, E., Andaloussi, J.E.: Developing SGML DTDs: From Text to Model to Markup. Prentice Hall PTR, Upper Saddle River (1995)
16. Mazinanian, D., Tsantalis, N.: An empirical study on the use of CSS preprocessors. In: 2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER), vol. 1, pp. 168–178. IEEE (2016)
17. Mernik, M., Heering, J., Sloane, A.M.: When and how to develop domain-specific languages. ACM Comput. Surv. (CSUR) **37**(4), 316–344 (2005)
18. Pilgrim, M.: Greasemonkey Hacks: Tips & Tools for Remixing the Web with Firefox. O'Reilly Media Inc., Sebastapol (2005)
19. Prasad, G.V.S.: Renarrating web content to increase web accessibility. In: Proceedings of the 10th International Conference on Theory and Practice of Electronic Governance, pp. 598–601. ACM (2017)
20. Prasad, G.V.S., Chimalakonda, S., Choppella, V., Reddy, Y.R.: An aspect oriented approach for renarrating web content. In: Proceedings of the 10th Innovations in Software Engineering Conference, pp. 56–65. ACM (2017)
21. Prasad, G.V.S., Dinesh, T., Choppella, V.: Overcoming the new accessibility challenges using the sweet framework. In: Proceedings of the 11th Web for All Conference, p. 22. ACM (2014)
22. Prasad, G.V.S., Ojha, A.: Text, table and graph-which is faster and more accurate to understand? In: 2012 IEEE Fourth International Conference on Technology for Education (T4E), pp. 126–131. IEEE (2012)
23. Prasad, V.G.S., Choppella, V.: Descriptive study of college bound rural youth of AP, India. In: 2013 IEEE Fifth International Conference on Technology for Education (T4E), pp. 76–79. IEEE (2013)
24. Scowen, R.: Extended BNF - a Generic Base Standard. Technical report 14977 (1998)
25. Sperberg-McQueen, C., Goldstein, R.F.: HTML to the max: a manifesto for adding SGML intelligence to the world-wide web. Comput. Netw. ISDN Syst. **28**(1–2), 3–11 (1995)
26. Tidwell, D.: XSLT. O'Reilly Media Inc., Sebastopol (2008)
27. Wu, H.K., Puntambekar, S.: Pedagogical affordances of multiple external representations in scientific processes. J. Sci. Educ. Technol. **21**(6), 754–767 (2012)
28. Zhang, D.: Web content adaptation for mobile handheld devices. Commun. ACM **50**(2), 75–79 (2007)