Towards a Domain-Specific Language for the Renarration of Web Pages

Gollapudi VRJ Sai Prasad
International Institute of Information Technology
Hyderabad, Telangana
saigollapudi1@gmail.com

Sridhar Chimalakonda
Indian Institute of Technology Tirupati
Tirupati, Andhra Pradesh
ch@iittp.ac.in

Venkatesh Choppella
International Institute of Information Technology
Hyderabad, Telangana
venkatesh.choppella@iiit.ac.in

ABSTRACT

We are interested in the problem of enabling transformation of existing, already published web pages. We call this Renarration of web content. In our earlier work, we had already established the role and importance of renarration for improving Web Accessibility. There are nearly a billion websites on the web, making transformation of pages a domain on its own. In this paper, we present the development of a Domain-Specific Language (DSL) for the purpose of web page transformation. We show how the design and implementation of our DSL is driven by our problem domain, its terminology and its unique requirements. We take up an existing online video-course delivery system, which has accessibility challenges, as a specific case to demonstrate our DSL. We end with insights and reflections for future work in both DSL and web page transformations.

CCS CONCEPTS

• **Software and its engineering → Domain specific languages**;
• **Information systems** → *Document structure*; Personalization;

KEYWORDS

Domain Specific Language (DSL), Web Page Transformation, Renarration

ACM Reference Format:
Gollapudi VRJ Sai Prasad, Sridhar Chimalakonda, and Venkatesh Choppella. 2018. Towards a Domain-Specific Language for the Renarration of Web Pages. In *ISEC '18: Innovations in Software Engineering Conference, February 9–11, 2018, Hyderabad, India.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3172871.3172873

1 INTRODUCTION

World Wide Web has over a billion websites[1]. *W3Techs*, a web statistics site[2], reports that over 54.2% of those billion websites are currently in English language. This is despite the fact that the majority of the global population is non-English speaking. Of the 3.3 billion Internet users, *Internet World Stats*[3] reports that nearly 1.5 billion are from Asia, a region where English is a foreign language. Europeans too constitute 18.1% of that 3.3 billion. These numbers not only highlight a potential language gap between the presented web content and the browsing user, it also highlights, the various barriers that may arise due other socio-cultural issues as well. In [21] we suggested that these are legitimate web accessibility challenges that are plaguing end-users.

In our earlier work [3, 19–21] we had suggested that existing, published web content and web pages, can be made more accessible by a concept we termed as *Renarration*. Renarration is essentially the act of transforming the original source content into a new target representation. Transforming a source into a new renarrated target could involve simplifying some text, translating it, elaborating on something, adding images, adding new links etc. Such transformations are renarrations. A source may be renarrated into many such variants, which may all co-exist as alternative views of the original.

A source may be transformed into a new alternate representation by a renarrator. A renarrator is an agent which can be automatic, or manual. In human approaches, we see this to be the work of a third-party-user volunteer who mediates the relationship between the content author and the browsing end-user. Multiple renarrations of a single source can thus be created either by having multiple renarrators or by having one renarrator producing different variants.

In this paper, we treat this renarration as a transformation of a web page. There are several existing approaches to web page transformation such as Transcoding, Web Augmentation and Presentation layer modifications, but most of these conventional approaches tend to be too technical and programmer oriented. Transcoding is typically related to power savings, color modifications or resizing views for small screen devices. The location where the changes are executed is typically within the network (as opposed to doing it at the client side) [9]. This requires access to content which is mostly not available for end-user renarration. Web Augmentation techniques (e.g. *GreaseMonkey*) tends to be more like a platform, providing users the ability to write their own *JavaScript* code for client side modification [2]. Though powerful, it still requires programming competency. Presentation layer modifications are typically facilitated by stylesheet paradigm. Here XSLT[4] and CSS[5] have gained in prominence. CSS predominantly tends to be style and layout oriented [18]. XSLT, on the other hand, does facilitate transformation, but it is often perceived to be rigid, cumbersome and complex [1] for end users.

[1] http://www.internetlivestats.com/total-number-of-websites/
[2] http://w3techs.com/technologies/overview/content_language/all
[3] http://www.internetworldstats.com/stats.htm
[4] (eXtensible Stylesheet Language) XSL Transformations
[5] Cascading Style Sheets

ISEC '18, February 9–11, 2018, Hyderabad, India

ACM ISBN 978-1-4503-6398-3/18/02…$15.00
https://doi.org/10.1145/3172871.3172873

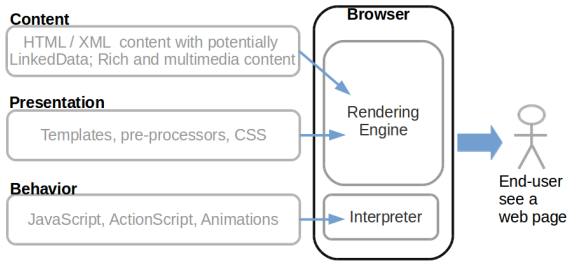


Figure 1: Components of a web page include Content, Presentation and Behavior. The web page perceived by an end-user can be a temporal artifact produced by browser by mixing the components, as specified by the publisher of the page.

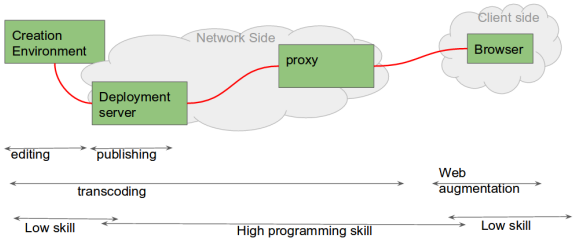


Figure 2: Web page transformation can be done either on the network side, or the client side.

1.1 Goal of this Paper

We are interested in the design of a Domain-Specific Language (DSL) [5] that allows transformation of web pages to facilitate renarration. Having a DSL to renarrate web pages could then help in overcoming some of the Web Accessibility challenges that are now plaguing a majority of the billion websites that are on the web.

The exploration of the design for a DSL is done through the development of a concrete system, aimed at renarrating online video courses. We have chosen to renarrate video-courses from NPTEL[6] for four main reasons (i) videos and course content are one of the most commonly used form of web content (ii) it has substantial amount of video content and user base within India (iii) millions of students who rely on NPTEL come from diverse cultures and backgrounds within India (iv) it is relatively easy for end users to renarrate video course content as it involves the simple operations of adding or removing or positioning video content for different purposes. While the NPTEL video course rennarations, that is used to illustrate the ideas in this paper seems simplistic, we see that it is a strong case for transformation of web pages to provide better accessibility through renarration, owing to the reason that there thousands of videos available and not accessible, which could be overcome by our DSL.

1.2 Layout of Paper

We begin by providing a working definition of a web page. We follow it up with a literature survey. Our survey explores the current approaches used in conducting web page transformation. After a brief explanation of NPTEL and the context of video-course renarration, we focus on this case and instance of web page transformation. Subsections on analysis, design and implementation on NPTEL case study are followed up by insights drawn from this example to further extend our DSL for an extensive general web page transformation. We end the paper with a section on conclusions.

1.3 Context: What is a Web Page?

A web page can be interpreted as having three components: Content, Presentation and Behavior. See Figure 1.

Content can be HTML tagged text, images or video. It can also be XML or other web accessible variant. It is essentially rich media tagged for the current web. Presentation is about aesthetics, style, layout, delivery-layer [18]. Typically this is handled by CSS or some pre-processed version of CSS. Behavior is something that is experienced by users. This is generally achieved through JavaScipt code associated with the web page. Even ActionScipts[7], HTML animations etc. can also create a behavior of a specific page.

What is experienced as a specific web page by the user is no longer some static content written up in HTML. While such material still exists on the web, the current web is becoming more and more of a confluence of the content, presentation and behavior being manifest by the rendering engine and the language interpreter present in a browser. While browsing, the end-user thus experiences the temporal artifact of such a process as a web page.

When we propose renarration through web page transformations, we are essentially talking about tampering with either the content, presentation or behavior of some earmarked source, to ultimately influence the view experienced by the end-user. Our intent in this paper is to create a minimalistic DSL that would enable such a web page transformation.

2 LITERATURE SURVEY

Currently there are multiple points at which published web content can be intercepted and transformed: It can either be done on the network side, or on the client (browser) side. See Figure 2. Network side modifications can be handled at three different locations: 1) at the application level, 2) on the cloud, by some other application which does back-end calls, and 3) on a proxy, as the content is in transit to the destination.

2.1 Network Based Approaches

At the application level, content can be modified before publishing. This may be termed as editing work. An author would be responsible for such changes. Influencing a change at editing time is routine and easily understood. However, this approach requires access to content authoring, which our users lack, as we work with existing and already published work. Therefore, we do not consider this editing scenario.

[6] A large repository of online video courses funded by Indian government. See http://nptel.ac.in/

[7] Scripts for Adobe Flash animation

Changes to content after publication are typically called annotation. Hypothes.is recommendation by W3C[8] promotes this annotation work. However, annotations are seen as extra information on top of a web page. They are not considered fundamental changes to the original source page.

The other scenario at the application level is when application itself modifies what is presented. This is the case with Adaptive Systems. Personalization, Customization solutions are also focused on modifying content at the application level.

Cloud based approaches tend to interact with the original source at the back end, negotiate a change before sending it out to the end-user. Transcoding systems tend to take this route [9].

Proxy based solutions are usually at the edge of the network. They intercept user-server interactions and interject in between. Again, some transcoding solutions [9], some personalization solutions tend to take this approach.

In all of the above modification methods, programming and system development expertise is required. In our effort, we are looking for ways and means to create web page transformation by end users on already published and publicly accessible content. Adopting above discussed approaches to enable such changes will be too cumbersome and complex for our earmarked renarrators.

2.2 Browser Based Solutions

As an alternative, browser based (client side) solutions to web page transformation tend to be relatively less complex.

A web page is structured as a Document Object Model (DOM) inside the browser. Accessing the DOM for the purpose of transforming a web page would mean that control is executed within the browser. Conventional solutions for such browser level DOM modification work would require knowledge of browser add-on or browser plug-in development work. This would mean that programming skill in *JavaScript* and knowledge of browser specific development would be required.

Existing web page code could also be modified by injecting *JavaScript* code either through a bookmarklet function, or through a external tool like *Selenium*. These injected scripts could then leverage existing code libraries (e.g. *jQuery*) to modify the page. Again, this would require programming and tool level expertise.

Taking the view that a renarrator need not be a programmer or a system developer, such a *JavaScript* and browser level development work would be too complex for them. Our goal thus becomes developing a simpler language and mechanism for enabling our web page transformation.

2.3 Stylesheet Based Modification

Stylesheets separate data from control, and content from presentation. They enable relatively novice end-users to modify web page presentation by way of stylesheet commands. In this regard, [5] calls stylesheets (as well as other web markup languages) as DSL as well. We are interested in this type of stylesheet based approach. However, the currently popular web stylesheet language (CSS) lacks explicit and simple techniques for semantic manipulation. CSS is largely a style, presentation or visual oriented control. From a renarration point of view, our intent is to initially propose a simple DSL which subsequently has the caliber to grow to handle semantic manipulation of content as well. It is in this regard that we find CSS, though immensely useful and popular is insufficient for renarration.

XSLT is an XML oriented stylesheet for tree transformation. While versatile, the stylesheet does require well-formed documents and lacks in popularity. Criticism against XSLT has often been with respect to its rigidity and complexity [1]. It is less likely to be perceived as a end user's tool.

[8]https://web.hypothes.is/blog/annotation-is-now-a-web-standard/

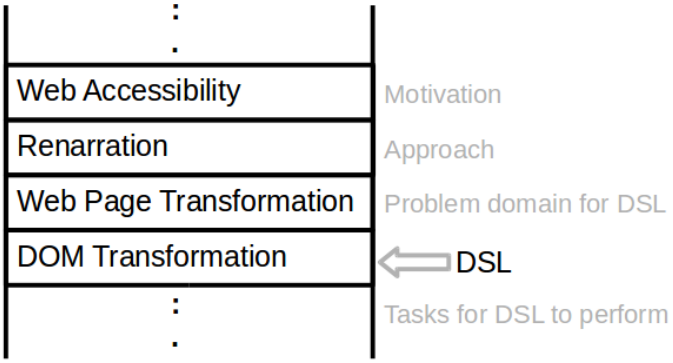


Figure 3: A Ladder of Abstraction for the design of DSL.

2.4 Existing DSL Based Approaches

The intent of a DSL is to focus on some specific domain, use the terminology and semantic models from that space, while hiding from the user, the design and implementation complexity in performing the operations. Many domains have enjoyed the benefits of DSL. Specifically in the web space, there is a DSL for validating data inputs to web applications [8], a DSL for dealing with web APIs and their service mashups [15], a DSL for creating dynamic web applications [7], a DSL for type-safe server-side scripting [27], a DSL for form based services, a DSL based on CSS for Hypertext adaptation [17] etc. Even markup languages like HTML and XML are also considered DSLs [5].

But, despite this abundance of DSLs in the domain of the web, there do not seem to be a DSL for specifically transforming web pages. While stylesheets like CSS and XMLT are also considered as DSLs[5], they have been criticized as being style oriented and complex. Thus this motivates our development effort of a DSL applicable to end users who could create web page transformations.

The work involved in creation of DSLs is detailed in [12, 13, 16]. [26] focuses on delivering an implementation from a design. And, in general, an annotated bibliography for DSL is presented in [28]. We rely on some of these earlier efforts to build our web page transforming DSL.

3 PROBLEM DOMAIN: WEB PAGE TRANSFORMATION

Research in Web Accessibility has compelled us to consider renarration as an approach for addressing the accessibility problem [19]. From a Computer Science and Web point of view, renarration translates into a problem in web page transformations. A transformation of a web page, in the context of a browser, may be discussed as a

DOM level transformation: which converts a source DOM into a target DOM (Figure 3).

To transform a webpage would mean to modify the content, presentation or its behavior, or some combination of them. Content of a page may be kept same, modified or deleted. This can apply to all types of *MIME* content found on the web. Modification to presentation of a webpage can come from changes made to the layout, color, style, flow etc. Similarly behavior level modifications can be from changes made to the *JavaScript* that is in the page. Also, new *JavaScript* could be included. So, again, CRUD[9] operations can be made to text and scripts as well.

It has already been stated that 1) renarrators need not be programmers or system developers, and that 2) there could be one or more renarrators out there producing potentially multiple variants (or renarrated views) for a given page.

Taking all these into consideration, our goals for a DSL that would transform web pages, would be:

(1) The DSL should allow end user renarrators to specify web page transformations.
(2) The DSL should allow renarrators to create a target DOM for rendering. This may be built from the source web page. To effect content level and behavioral changes, CRUD operations should be allowed on content or script. To effect presentation level changes, the aesthetics, layout, order of the source may be changed.
(3) The DSL should allow renarrators to additionally incorporate some meta-data and selection criteria (which could be in the form of rules, conditions), for subsequent use by the system or the end-user in selecting one renarrated variant over the others.

The intent of this paper is to design a preliminary DSL as an option to meet the above requirements. Our goal is not to exhaustively propose a formal language, but to initiate the idea of using DSLs for web page transformations. We explore the idea by developing a simple DSL to renarrate some already published online video courses.

4 NPTEL VIDEO-COURSE RENARRATIONS

To further explore the notion of DSL, to anchor some of our ideas, and to create a concrete starting point for our web page transformation task, we focused on a specific case-and-point activity: We focused on developing a simple DSL system for renarrating online NTPEL video-courses.

4.1 What is NPTEL?

NPTEL is a Indian government sponsored repository of over 355 video courses [14]. It has more than 1.5 million enrolled students and thousands of hours of recorded content. Indian government's intent is to freely make available the course content of their top colleges to various students all over India. A typical course like "Data Structures" within Computer Science tends to be a collection of several videos, each consisting of hour long classroom recordings. Typically, these videos feature "talking heads" of professors running through topic focused power-point slides. See Figure 4

[9] A term inherited from the Database domain. It stands for typical Create (C), Read (R), Update (U) replace, and Delete (D) operations

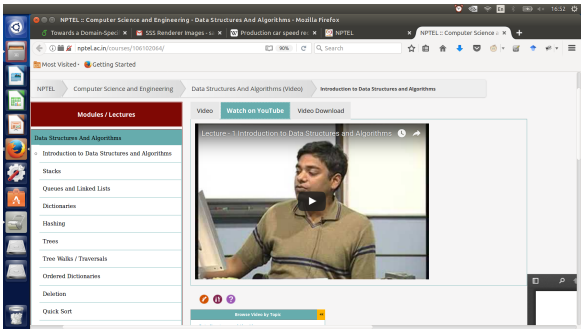


Figure 4: A sample of NPTEL video lecture page from the NPTEL repository.

In an earlier unpublished simple ethnographic study of few students, we discovered that most of the recordings were deemed useful [22] but were considered 'boring' to watch. Students reported speeding up the videos, jumping to different sections, turning on subtitling etc. to increase their derived value from the videos.

4.2 Goal for NPTEL Renarration

Our goal for this exercise was to take one such video course – for example, "Data Structures and Algorithms" – and renarrate it for our otherwise bored end-users.

The target (or end-product view) of our renarration is to produce a new narrative for the Data Structures video-course. One idea for the new renarration is to replace the hour long video source, with a string of media items like video segments, PPTs, and PDF documents, presented in some predefined order. That is, the one hour long message of the source video can now be re-presented as a collection of PDF material, few slides, some portions of video etc.

The previously bored end-user can now experience the same content in a novel way: Perhaps an introduction slides are presented first. Then, a key portion of the NPTEL video is shown. After the allocated time, a PDF is rendered. The videos, PPTs and PDFs for the new rendering can come from some URI located on the web. That is, pre-existing material are now being stitched into a multimedia presentation, and being rendered for the end-user. In such situations, the video content and the slides need not be shown in their entirety. They can be shown from some start to some pre-set end point.

4.3 End-user Experience

From a end-user's point of view, navigation buttons may be used to move linearly from one event to the next in the series. The expectation is that by breaking the video into chunks, and then stringing the content with other online resources (which may simplify or elaborate the messages present in the video) may positively enhance the user-experience.

In this context, a professor could be the renarrator. Her students for a particular class may be the target end-users for whom the renarration is being captured. The new narrative may be the original video source, chunked, pruned and augmented with other online PPTs, PDFs, videos etc.

For our example course on Data Structures, the one hour long video could now be replaced with a single intro PPT slide, followed

by 3 minute video, then followed by few more slides, a PDF and yet another 20 minute video etc.

4.4 Why this Exercise?

We use this video-course renarration as a case of web page transformation towards creating different elements of our DSL.

We now present our analysis of this NPTEL video-course renarration problem. Despite the concrete problem, often we have considered the broader challenge of web page transformations and not video renarration. We follow up our analysis, design and implementation discussion with lessons learnt and insights derived. We conclude by providing inputs to the design of our DSL for web page transformations.

5 EXPLORING DSL FOR VIDEO-COURSE RENARRATION

Our DSL development is influenced by the work presented in [12, 16].

5.1 Decision

Our decision to develop a purpose built DSL is motivated by the following two drivers:

(1) Web Accessibility is a *end-user* challenge. While our end-users have browsing skills, they may not necessarily be programmers. In keeping with this, we wanted an interface which was not programming intensive, but would still enable simple renarrations.
(2) We wanted the interface model to match the mental model of the renarrator and not burden them with constraints and requirements that a general purpose language would impose.

5.2 Analysis

According to [16], in this analysis phase, one must gather knowledge about the domain, extract relevant terminology, study the back-end models that animate it and look for patterns. The proposed formats for doing domain analysis include: Domain Analysis and Reuse Environment (DARE), [6], Domain Specific Software Architectures (DSSA) [25], Family-Oriented Abstractions, Specification, and Translation (FAST) [29], Feature Oriented Domain Analysis (FODA) [11], Ontology-based Domain Engineering (ODE) [4], and Organization Domain Modeling (ODM) [24]. We follow a simplified version of DARE during our analysis phase owing to the goal of a simplified DSL.

For the consideration of domain oriented terminology, we observed that for the scenario of web page transformation, on client (browser) side, the terms related to 'DOM;, and the term 'nodes within a DOM are common. For video renarration context, terms related to media (its type, its source and target URI, its attribute) are common. In web terminology source and target URIs; in stylesheet terminology, selectors with attributes are also commonly used.

For the video, we related the semantic of rendering a page from a target DOM to the semantic of a media player executing a *playlist*. Similarly, we related the selection algorithm for picking one renarration variant from a potential choice of many to affirming a rule in a *rule-engine* metaphor.

5.3 Considerations for Design

Our DSL is proposed as a special purpose language, intended to only meet very specific domain related goals, and use only problem-domain relevant vocabulary. Ultimately, the DSL has to enable a user to express a set of nodes, sourced from various web locations for rendering them in a given sequence (like a *playlist*). Here are some of our considerations in the design of our DSL:

- **domain terminology vs programming terminology:** Concepts of renarration of some source to some target, or notions of web page transformation of DOM flow do not just directly map to any language abstractions. While languages like *JavaScript* (using *jQuery* library) can support DOM manipulation, the implementation would be awkward and not too intuitive to the domain. So, we choose to focus on domain terminology.
- **descriptive Vs functional:** A *playlist* consists of 1) what to play and 2) the order in which to play. We wanted to port the same concept into our DSL. Here we wanted to only describe what we wanted to render and not *how* the rendering has to happen. Therefore, we opted for descriptive language.
- **data vs algorithm:** Again, using a *playlist* metaphor, we wanted to have the video order to be separated from the media player that displayed them. Similarly, for a web page transformation, we want to separate out our node data from the algorithm that executed and rendered them. So, Separation of Concern (SoC) technique is chosen.
- **simple domain specifics vs powerful generic control** Wherever possible, if there was a choice in design, we wanted to opt for a simple domain specific implementation over powerful generic but nimble solutions. Maintaining a domain focus may at times restrict us, but it could also give us the benefit of allowing a end user to create and also analyze, verify a web page transformation code. Here our choice is for domain oriented terminology.
- **input vs programming** As our end users are programming illiterate, we seek a declarative DSL over an executable language. Declarative language can not only map easily to the node or DOM type constructs in the problem-domain, but they can also be turned into an input to some user interface during the web page transformation. Here we opted for input driven design.

In general, we want a DSL only for meeting the needs of our niche application. In this limited domain, however, we are still looking for more expressiveness and improved ease-of-use for end users. The benefits we are seeking are: 1) ease in description of task and detailing it, 2) a non-programming work environment 3) more domain-specific terminology so that learning curve is small and end user (but domain literate persons) are immediately productive.

5.4 Models used for Design

The operations that need support in our DSL can be inferred from the requirements themselves. Requirement 1) indicates that DSL has to be for end users. This implies that our DSL will be more declarative and data structure oriented. This data-structure would map to a intuitive *playlist* metaphor. Thus allowing easy navigation

Segment 1: Metadata						
	User Info	Name:	userID:	pwd:	email:	institute name:
	SSS Info	Name:	ID:	Desc:		
Segment 2: Node List						
	Node Description		**Node Action**			
S.No.	**Node Type**	**Source URI**	**Operation**	Attribute **new URL**	**Attr 1**	**Attr 2**
1	html	http://.../xpath	Insert	http://...		
2	Video	https://.../xpath	Replace	https://		
3	Image	http://.../xpath	Delete	http://...		
4	html	https://.../xpath	tReplace	https://	lang: Chinese	
5	html	http://.../xpath	hReplace	http://...	color: Yellow	
	Stop					
Segment 3: Selection Criteria						
	Test-condition:	User-param:	Env-param:			

Figure 5: A renarration script, essentially a DSL program, contains 3 segments. Segment 1 is about meta data, 2 is about the actual playlist, and 3 is about conditions & rules which dictate selection of this variant.

Event List						
	Event Description		**Event Action**			
S.No.	**Media Type**	**Source URI**	**Operation**	Attribute **new URL**	**start**	**stop**
1	PPT	http://...	Show	http://...		
2	Video	"	Show	https://	Start 09:05	Stop 12:15
3	PPT	"	Show	http://...	Slide 5	Slide 45
4	PDF	"	Show	https://		
5	Video	"	Show	http://...	Start 09:05	Stop 12:15
6	PDF	"	Show	http://...		
	Stop					

Figure 6: Segment 2 or Node List part of the renarration script.

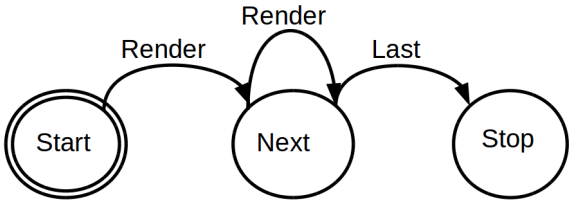


Figure 7: Abstract State Machine for our DSL.

for renarrators. Requirement 2) indicates that nodes must be rendered in a sequence to form a target DOM. And these nodes could be new injections or deletes or replacement. Again, we use the *playlist* metaphor for this. Requirement 3) suggests that one renarrated variant must be selectable over a set of potentially multiple alternative renarrations. For this we use the rule-engine metaphor.

5.4.1 Playlist metaphor. While *playlist* is related to songs and videos, we felt that the same notions would apply to DOM nodes and to their order of rendering. Moreover, even novice web users are now-a-days familiar with the *playlist* metaphor - therefore, we adopted this notion for our renarration work. The semantic model for our *playlist* is given in Figure 7.

Our semantic model for the *playlist* is simplistic. The assumption is that there are several nodes in a list that one needs to trickle through (one by one) till the last for rendering a target DOM. This has been depicted as start state (as the first node) and then going through a sequence of Next states and finally reaching the last node (depicted as Stop state). This model simply emphasizes the importance of sequencing. One reason for having a simplistic model is that, as we develop our DSL, we will have a start point for our semantic model that we can progressively improve.

5.4.2 Rules & Rule-Engine metaphor. This metaphor emphasizes the need to separate data from algorithm. We propose to use this

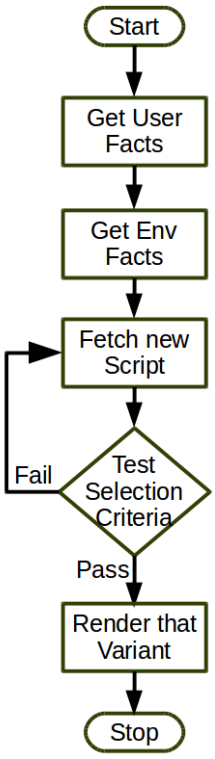


Figure 8: Stylesheet selection algorithm.

notion for the selection of one variant from a set of several possible alternatives (which may be present for a given web page). The algorithm that we use for this is depicted in Figure 8.

The rule or condition needed for selection of an variant is to be defined at the time of its creation by the renarrator. This condition can be based on either user parameters or environmental conditions. For instance, some rules may be set up to be affirmed only when a particular user is logged on. Another rule can be setup to trigger only when it is a weekend or after certain date. The former is a user parameter, the latter is an environmental parameter.

The user parameters can be used to select an individual, a community of users or a institute. More user profile parameters can be used to increase the control at the user level.

Environmental related information are things that system can automatically infer (or system should know) during the course of its operations. Time of day, type of browser used, IP address are all examples upon which a selection condition (or rule) can be based.

A rule can also involve a combination of both user information and environmental information.

As shown in Figure 8, the rule gets checked against some known facts. The facts for a particular session are known by the rule engine. The known facts are user's name, IP address (location), host institute etc. These maybe known through some sort of registration process. Similarly, environmental facts can be derived through simple queries to the operational environment.

Given the facts and the selection condition, the rule engine can simple run a test and declare a pass or fail in operation. A pass could imply selecting that variant and a fail could imply fetching a new variant from the available set. This process can be repeated till the first renarration variant is selected.

5.5 Design Details

A script containing the DSL for the video renarration needs to handle three components: The first component is the meta data of the renarration script. Here is where the description of the variant, the creator information are detailed. We call it the Segment 1 of the script. See Figure 5 for a depiction of the overall script.

The second component is the node-list (or the playlist). Here the nodes are presented in some sequence of a Target DOM for rendering. We call this Segment 2 as shown in 6 The information here is a series of nodes, where each node is a renderable entity within a DOM of a page. Here is where the new or replacement DOM for the target is inject, or the old one modified.

The third and the last component in this renarration script, called Segment 3, has to do with selection criteria. Here is where the renarrator can embed the rules or conditions for selection.

5.5.1 Grammar. We articulate our design for the Segment 2 or node-list portion using a simple grammar. We use the following EBNF notation (as defined in [23]):

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

We present an indicative grammar as our focus is on a simplistic DSL aimed at facilitating renarration of web pages from end users perspective. Our notaion, see Listing 1, uses a top-down approach. We start with a start symbol âĂIJtarget DOMâĂİ and reduce it down to terminals by using intermediate productions.

```
target_DOM = e | node_list
node_list = {sequence_number, node_info}
node_info = node_description, node_action
node_description = media_type, source_URI,
media_type = âĂpptâĂİ | âĂvideoâĂİ | âĂpdfâĂİ | âĂhtmlâĂİ
node_action = (ppt, (start slide, end slide)) | (video,
     (start time, end time)) | (pdf, ()) | (html, ())
```

Listing 1: Indicative grammar for a simple DSL

6 IMPLEMENTATION

With a *playlist* like representation our DSL may now be used as a front-end device which provides input to our renarration oriented applications. In this sense, the DSL is very much declarative. Since it is not intended as a functional executable, our DSL code, now, need not compile to any byte level code.

The DSL for a DOM structure can be built either textually or graphically. That is, graphic icons could be used as an input language to define our DSL. As we are initially focusing on creating a core DSL for this first iteration, we kept our DSL implementation simple and text based. Design phase has already indicated that the

terminology for the text of the DSL must be domain oriented. For notation, we have relied on JSON format for this version of DSL. Thus, a program in our DSL would be text based, small, declarative, use domain-oriented language, resemble a *playlist*, and would be formated in JSON notation. See Listing 2.

```
[ { // --------------------
    // segment 1: meta-data
    // --------------------
    "seg1": {
      "username": "naveen garg",
      "user-id": "naveen",
      "ren-url": "http://...",
      "sss-desc": "This renarration is about...",
      "sss-name": "data structures renarration" },

    // --------------------
    // segment 2: Node-list
    // --------------------
    "seg2": [ {
        "media-type": "video",
        "media-url": "https://...",
        "start": "1020",
        "end": "2020" },
      {
        "media-type": "pdf",
        "media-url": "https://..." },
      {
        "media-type": "ppt",
        "media-url": "https://...",
        "start-slide": "2" },
      {
        "end": "1020",
        "media-type": "video",
        "media-url": "https://...",
        "start": "500" } ],

    // ----------------------------
    // segment 3: Selection Criteria
    // ----------------------------
    "seg3": {
      "env-info": {
        "browser": "firefox",
        "time-of-day": "" },
      "user-info": {
        "degree-program": "cse",
        "user-college": "iiith",
        "user-name": "sadhana virupaksha",
        "year": "03" } } } ]
```

Listing 2: A sample script file representing a DSL program.

6.1 Elements Descriptions

In this section we describe the elements that we have used for our JSON structure. The types can be associated with LinkedData [10]. In such a case our DSL structure could become JSON-LD. We see that as a natural next step progression from this fledgling effort.

6.1.1 Segment 1: Meta-data. This section captures meta data of the variant as well as SSS creator's (user's) information. The elements that are present here include

"username": (string) name of creator
"user-id": (string) user ID
"ren-url": (URI) points to source page being transformed
"sss-desc": (string) contains description of the variant
"sss-name": (string) name given to the variant

6.1.2 Segment 2: Node-list. This section captures the list of nodes that the renarrated page must render as Target DOM.

"media-type": (one of "video", "ppt", "pdf", "html")
"media-url": (URI) an expath to the location of the resource
"start": (number) start time or start slide
"end": (number) start time or start slide

6.1.3 Segment 3: Selection Criteria. This section captures the conditions (rules) needed for selection of this particular variant. We focus on two categories of information that we use in the selection test: first category is environmental information. The second category is user information.

In our implementation we could have used a rule engine directly. However, for our first effort, we kept it simple and only utilized a simple algorithm (ie. string match) to enable our selection. However, in the future, we can enhance this to include a more powerful selection engine.

"env-info": (set) environmental info
"browser": (browser type)
"time-of-day": (time)
"user-info": (set) user information
"degree-program": (string) degree abbreviation
"user-college": (string) institute name
"user-name": (string) user name
"year": (number)

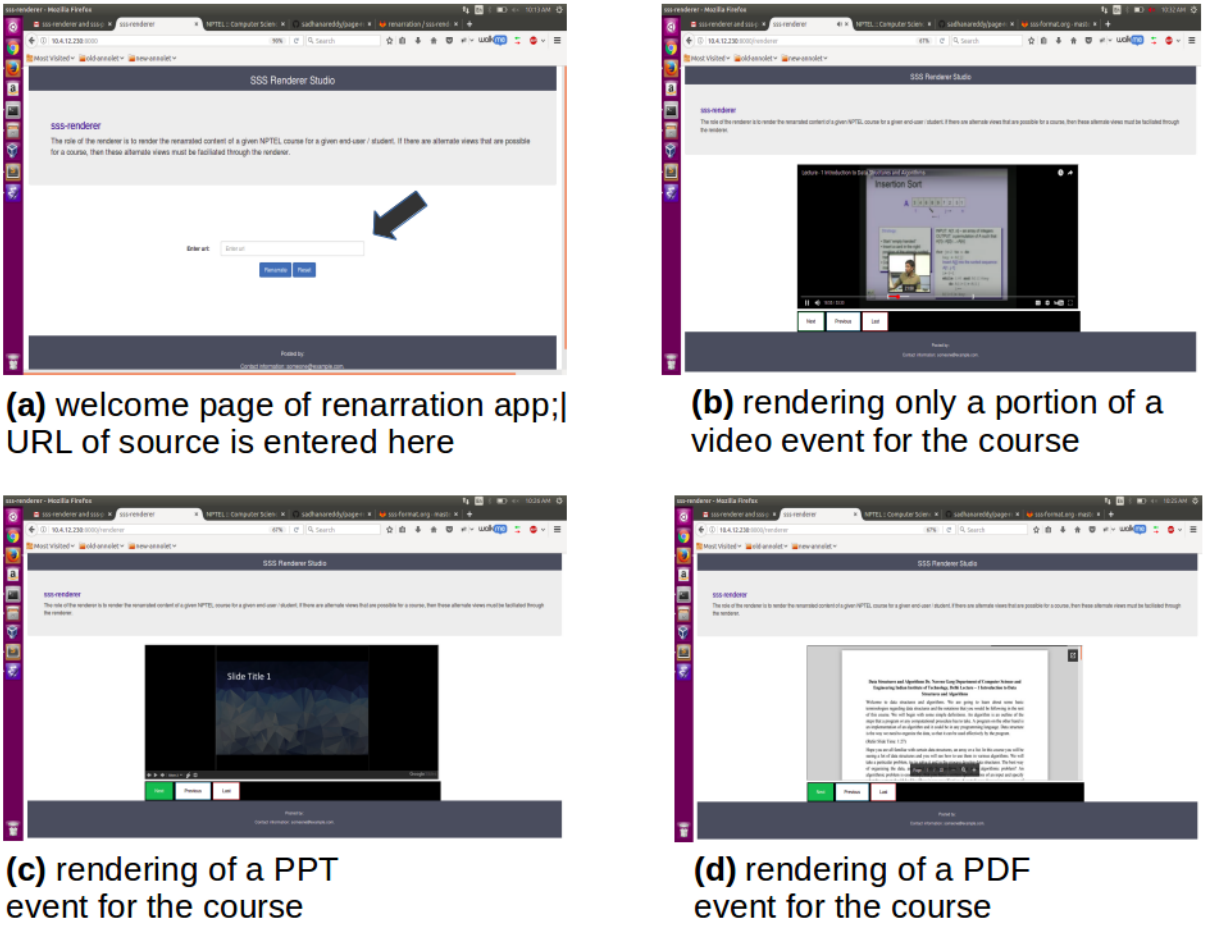
7 PROTOTYPE

To validate our idea, we built a simple web page transformation (renarration) application. The aim of the application is to renarrate an existing online video course into a new narrative. The use case we assumed was that a college lecturer wishes to take an existing online video course (which is presumably quite long) and wishes to renarrate that into a collection of events, which are now played on the screen instead. The series of events forms the *playlist* (or node list). Each event is a rendering of PDF, or a PPT, or a Video. In our DSL vocabulary, these are *media_types*, sourced from a *media_url*. For some media types (e.g. Video and PPT) we have some attributes like start and stop times or slide numbers. This event-list forms the segment 2 node-list of our renarration script.

For segment 1, we captured the renarrator's name, id, institute name etc. as user information; and, description, id, name as meta-data for the script.

For segment 3, we specified a user name and a time of day item for selection criteria.

With such a renarration script, our video online course "Data Structures" was turned into a short *playlist* of video, ppt and pdf events. The renarration was specified through the script. The rendering was integrated with processing logic of the DSL. The rendering



(a) welcome page of renarration app;| URL of source is entered here

(b) rendering only a portion of a video event for the course

(c) rendering of a PPT event for the course

(d) rendering of a PDF event for the course

Figure 9: A prototype application that renarrates a online video course.

was a simple display of the content on a iFrame of a web application. Listing 3 showcases a pseudo code sample of our rendering logic.

```
function renderMedia(){
   var new_src;
   if(index >= 0 & index < event_list.length) {
     var cur_event = event_list[index],
     media_type = cur_event["media-type"],
     media_url = cur_event["media-url"];
     switch ( media_type ) {
        case "video":
           render(video_player, media_url + '&start=' +
                cur_event["start"] + '&end=' +
                cur_event["end"];
           break;
        case "pdf":
           render(pdf_viewer, media_url);
            break;
        case "ppt":
           render(ppt_viewer, media_url, start_slide);
           break;
        default:
           alert("ERROR: Unkown Event was asked to be
                rendered");
   } } } }
```

Listing 3: Pseudo code representing the rendering logic.

Figure 9 shows the renarration application that we had built for validating our DSL. Part (a) shows the welcome page where the URL of the source page is entered. Renarrations for this URL are sought. (b) shows the rendered view. What is depicted is the first event in the node-list. In this case it is a video. The video has start and stop times. Also, along with the player buttons, there are also previous, next, last buttons available for user. These buttons navigate the user from current to next to previous events respectively. (c) shows ppt being rendered. (d) shows PDF event being rendered. The collection of all (b)-(d) constitutes the node-list given by the renarration script.

8 DISCUSSION & INSIGHTS

Through the development of a core DSL for web page transformation that was instantiated for NPTEL online video course renarration, we learnt the following lessons:

(1) A domain oriented non-programming user interface is possible, but we need some common terminology for handling of content, presentation and behavior. Focused studies and further analysis may reveal more conventional but sticky terms used in this area.
(2) Currently we used locally declared types for the definition of our elements. *Name*, for instance, has been defined as a *String*. Going forward, for a more robust and higher level implementation, we need to work with more structured data [10] Schemas[10]. For example, FOAF[11].
(3) A *playlist* is a good metaphor for sequencing nodes. However, when it comes to web page renarration, we need to deal with node hierarchies. A *playlist* metaphor may not be able to handle this. Going forward, a tree model for a DOM structure also may not suffice. A graph or an ontology type structure should also be considered. This would require a more sophisticated semantic model that is robust enough to handle non-linear DOMs.
(4) In current implementation, we had integrated rendering logic with DSL element processing. Going forward, we may need to separate the logic into a DSL layer which is different from execution of the DSL code.

9 CONCLUSIONS

Our aim was to explore the design of a DSL to facilitate renarration by way of web page transformation. We presented the notion of a web page as a set of components emphasizing the semantic view of web content that emerged from unique our goals of rennaration. We elaborated the domain of web page transformation, its elements and presented the design of our DSL through a simple grammar. We then presented an instance of the DSL for video-course renarration from NPTEL video repository. The application of our DSL for NPTEL video courses resulted in multiple renarrations of the same course eventually leading to better accessibility. Finally, we see our DSL as a core work leading to several extensions in web page transformations including a suite of frameworks and tools.

ACKNOWLEDGEMENTS

We thank V. Sadhana Reddy, M. Raghav and MS Soumya for their help in the implementation of this DSL work.

REFERENCES

[1] Eric Bae and James Bailey. 2003. CodeX: an approach for debugging XSLT transformations. In *Web Information Systems Engineering, 2003. WISE 2003. Proceedings of the Fourth International Conference on*. IEEE, 309–312.
[2] Oscar Díaz. 2012. Understanding web augmentation. In *International Conference on Web Engineering*. Springer, 79–80.
[3] TB Dinesh, S Uskudarli, Subramanya Sastry, Deepti Aggarwal, and Venkatesh Choppella. 2012. Alipi: A framework for re-narrating web pages. In *Proceedings of the International Cross-Disciplinary Conference on Web Accessibility*. ACM, 22.
[4] Ricardo De Almeida Falbo, Ana Candida Cruz Natali, Paula Gomes Mian, Gleidson Bertollo, and Fabiano Borges Ruy. 2003. ODE: Ontology-based software development environment. In *IX Congreso Argentino de Ciencias de la Computación*.
[5] Martin Fowler. 2010. *Domain-specific languages*. Pearson Education.
[6] William Frakes, Ruben Prieto, Christopher Fox, et al. 1998. DARE: Domain analysis and reuse environment. *Annals of software engineering* 5, 1 (1998), 125–141.
[7] Danny M Groenewegen, Zef Hemel, Lennart CL Kats, and Eelco Visser. 2008. Webdsl: a domain-specific language for dynamic web applications. In *Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications*. ACM, 779–780.
[8] Danny M Groenewegen and Eelco Visser. 2013. Integration of data validation and user interface concerns in a DSL for web applications. *Software & Systems Modeling* 12, 1 (2013), 35–52.
[9] Richard Han and John R Smith. 2000. Transcoding of the Internet's multimedia content for universal access. *Communications, Networking, And Multimedia, Multimedia communications: directions and innovations* (2000), 261–296.
[10] Tom Heath and Christian Bizer. 2011. Linked data: Evolving the web into a global data space. *Synthesis lectures on the semantic web: theory and technology* 1, 1 (2011), 1–136.
[11] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, and A Spencer Peterson. 1990. *Feature-oriented domain analysis (FODA) feasibility study*. Technical Report. Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst.
[12] Gabor Karsai, Holger Krahn, Claas Pinkernell, Bernhard Rumpe, Martin Schindler, and Steven Völkel. 2014. Design guidelines for domain specific languages. *arXiv preprint arXiv:1409.2378* (2014).
[13] Anneke Kleppe. 2008. *Software language engineering: creating domain-specific languages using metamodels*. Pearson Education.
[14] Mangala Sunder Krishnan. 2009. NPTEL: A programme for free online and open engineering and science education. In *Technology for Education, 2009. T4E'09. International Workshop on*. IEEE, 1–5.
[15] E Maximilien, Hernan Wilkinson, Nirmit Desai, and Stefan Tai. 2007. A domain-specific language for web apis and services mashups. *Service-oriented computing–ICSOC 2007* (2007), 13–26.
[16] Marjan Mernik, Jan Heering, and Anthony M Sloane. 2005. When and how to develop domain-specific languages. *ACM computing surveys (CSUR)* 37, 4 (2005), 316–344.
[17] Alejandro Montes García, Paul De Bra, George HL Fletcher, and Mykola Pechenizkiy. 2014. A DSL based on CSS for hypertext adaptation. In *Proceedings of the 25th ACM conference on Hypertext and social media*. ACM, 313–315.
[18] Lee Naylor. 2016. Restyling the Web Site: An Introduction. In *ASP. NET MVC with Entity Framework and CSS*. Springer, 467–484.
[19] Gollapudi VRJ Prasad. 2017. Renarrating Web Content to Increase Web Accessibility. In *Proceedings of the 10th International Conference on Theory and Practice of Electronic Governance*. ACM, 598–601.
[20] Gollapudi V. R. J. Sai Prasad, Sridhar Chimalakonda, Venkatesh Choppella, and Y. Raghu Reddy. 2017. An Aspect Oriented Approach for Renarrating Web Content. In *Proceedings of the 10th Innovations in Software Engineering Conference, ISEC 2017, Jaipur, India, February 5-7, 2017*. 56–65. http://dl.acm.org/citation.cfm?id=3021466
[21] Gollapudi Vrj Sai Prasad, TB Dinesh, and Venkatesh Choppella. 2014. Overcoming the new accessibility challenges using the sweet framework. In *Proceedings of the 11th Web for All Conference*. ACM, 22.
[22] Jayanti Ravi and Haresh Jayantilal Jani. 2011. A critical study of nptel. In *Technology for Education (T4E), 2011 IEEE International Conference on*. IEEE, 35–42.
[23] R.S. Scowen. 1998. *Extended BNF - a Generic Base Standard*. Technical report 14977.
[24] Mark Simos and Jon Anthony. 1998. Weaving the model web: A multi-modeling approach to concepts and features in domain engineering. In *Software Reuse, 1998. Proceedings. Fifth International Conference on*. IEEE, 94–102.
[25] Richard N Taylor, Will Tracz, and Lou Coglianese. 1995. Software development using domain-specific software architectures: CDRl A011âĂŤa curriculum module in the SEI style. *ACM SIGSOFT Software Engineering Notes* 20, 5 (1995), 27–38.
[26] Scott A Thibault, Renaud Marlet, and Charles Consel. 1999. Domain-specific languages: From design to implementation application to video device drivers generation. *IEEE Transactions on software Engineering* 25, 3 (1999), 363–377.
[27] Peter Thiemann. 2005. An embedded domain-specific language for type-safe server-side web scripting. *ACM Transactions on Internet Technology (TOIT)* 5, 1 (2005), 1–46.
[28] Arie Van Deursen, Paul Klint, Joost Visser, et al. 2000. Domain-specific languages: An annotated bibliography. *Sigplan Notices* 35, 6 (2000), 26–36.
[29] David M Weiss et al. 1999. *Software product-line engineering: a family-based software development process*. Addison-Wesley Professional; Har/Cdr edition.

[10] http://schema.org/ and http://dublincore.org/ are couple of good structured data initiatives
[11] Friend Of A Friend initiative structures human related data for use in social networks. See http://www.foaf-project.org/