The Navajo Language Literature Project: A Case Study in Client-side Design Patterns Using Asynchronous Requests

Kip Canfield
Department of Information Systems ITE 425, University of Maryland, UMBC, 1000 Hilltop Circle, Baltimore MD 21250, USA

Correspondence:
Kip Canfield, Department of Information Systems
ITE 425, University of Maryland, UMBC, 1000 Hilltop Circle, Baltimore MD 21250, USA.
E-mail: canfield@umbc.edu

Abstract

The Navajo Language Literature Project was established to create and deliver a web-based, digital library of Navajo language texts. The current focus is to deliver the texts on the web and allow collaborative editing for linguistic detail such as word parses and glosses. The original implementation of the project used a server-side design for the Internet applications. The addition of asynchronous update to the server for this project gave the client web application more responsibility and started a line of inquiry into how much processing can be pushed to the web browser client. A major advantage of this move is a simplification of deployment that can be beneficial for small and unfunded projects in the humanities. The case study below defines and parameterizes a model for this client-side pattern.

1 Introduction

The Navajo Language Literature Project was established to create and deliver a web-based, digital library of Navajo language texts. Navajo is not originally a written language, but there have been many written texts created from the 1800s forward. These texts exist in a variety of orthographies, many of which predate the current standard one. All of the texts are not in digital format and so the first task in this project was to develop a methodology to acquire these texts in digital format and transform them to the standard orthography (Canfield, 2005). The next focus was to deliver the texts on the web and allow collaborative editing for linguistic detail such as word parses and glosses.

There are several options for the web application interface. The traditional interface requires a complete web page reload synchronously for every update.[1] This is the way that most updates are currently done for web applications. Recently, web developers have started to use XMLHttpRequest for asynchronous update of web pages.[2] This ability has long been included in JavaScript, but has only recently become popular due to standards compliance by most browsers and increasing bandwidth available to most web users. This interaction design has been called Ajax for Asynchronous JavaScript + XML and this term was introduced in Garrett (2005). Using this technique allows a web application to appear more like a traditional non-web client application due to lack of disruptive page reloads and the ability to both gather information for use in the web application and update server resources while the user is engaged in other tasks on the web page. Figure 1 shows the concept of

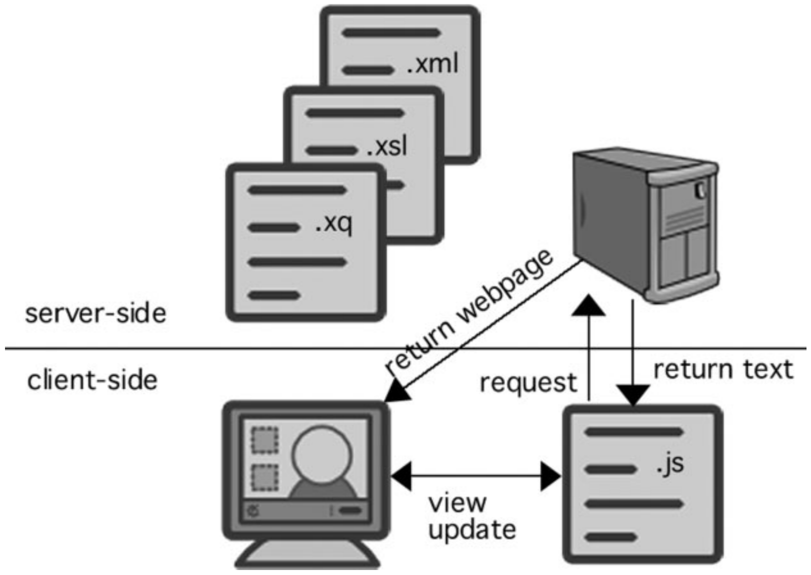


Fig. 1 Asynchronous versus Synchronous request

asynchronous request for this project. The client makes a request to the server for processing or information and the server responds synchronously with a complete web page or asynchronously with either simple status information or the requested text. In the case of a requested file, the JavaScript-based Ajax engine integrates it into the interface rather than through a page reload.

This project has experimented with several models for implementing web-based delivery of the project texts. The Navajo language texts are scanned and a custom optical character recognition program is applied resulting in plain text using characters used in a standard Navajo font. Then the texts are marked up to be compliant with TEI Lite. Since only the Navajo language portion of the text is stored as XML, the original Navajo language text images, English translations, and footnotes are delivered as images along with the marked up text. This requires an interface that can present the interrelated pages in an organized manner and a server-side component that can handle serving these related pages and also allows collaborative update. The images of the original text have value both because all images have not been recognized to text and the original images include information that is lost with regularization to the standard orthography. For example, the idiosyncratic manner in which some morphemes are put together by the skilled Navajo transcribers could be mined for evidence for morpheme boundary issues in Navajo and Athapaskan languages in general (Rice, 2000).

The original implementation of the project used a server-side design where most of the application work was done on the server-side. The TEI texts were stored in a native XML database and the TEI markup contained all the information for the interface. The markup included the word parses and glosses, the links to the related images, and XSLT was used to dynamically create the interface by transforming those TEI documents. The XML format proposed in Bird *et al*. (2003) was used to add interlinear gloss text to the original texts and links were inserted for reference to the corresponding images from the original. Since this resulted in a rather large document, page reloads were reduced by using asynchronous updates for the collaborative updating of that interlinear text. These requests would use XUpdate to update the TEI in the database via REST-style web services (Fielding, 2002) using XQuery. This works quite well and is very suitable in doing searches of large document collections. It is complex to develop and maintain, however, and this can be a barrier to especially small and unfunded projects in the humanities. The addition of asynchronous update to this project started a line of inquiry into how much processing can be pushed to the web browser client and the advantages and disadvantages of such a move. The case study subsequently defines and parameterizes a model to explore this question.

Complex	• Asynchronous • DOM • Google	• Asynchronous • DOM • Database
Simple	• Asynchronous • Frames • Google	• Synchronous • Frames • Database
	Client-side	**Server-side**

Fig. 2 The model for design patterns

2 Design Pattern Model

The model used for this discussion can be summarized in Fig. 2. This matrix has two axes for the complexity of the interfaces (simple or complex) and the location of the processing (client-side or server-side). Each model quadrant is described with parameters: the HTTP request type, the client HTML design type, and the collaborative database component.

The HTTP request type is either synchronous or asynchronous. Complex interfaces require asynchronous requests to reduce large page reloads and client-centric processing requires it in order to receive data files that would ordinarily be processed at the server. The HTML design type is simple in the case of HTML frames and complex for single Document Object Model (DOM) implementations. Frames are simple but reduce the options available in the interface while DOM implementations can rival the richness of traditional non-web client applications at the cost of complexity. The final parameter is where the collaborative update takes place. This logically requires server-side processing. In server-side design patterns, this would be done with database updates via XUpdate or SQL. For a client-centric design pattern, an outsourcing option from Google spreadsheets is explored subsequently that reduces the burden on small workgroups.

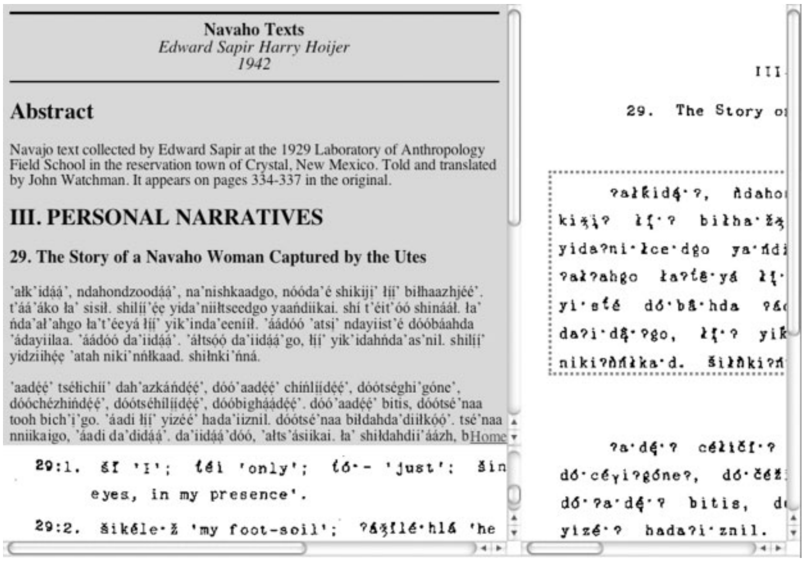
3 Model Axis: Interfaces

Figure 3 shows the simple frames-based interface.[3] The displayed text is from Sapir and Hoijer (1942) using the Times New Roman Navajo true type font for the TEI and the scanned images for the other documents.

Each document is presented in a separate frame. The text in the dark frame is transformed to HTML from the XML using a minor modification of the standard TEI Lite style sheet. The right-hand frame shows the image of the original text in the non-standard orthography. The lower frame shows the original footnotes image for the Navajo text. Since the simple frames interface has limited ability to add menu items, the user can replace the Navajo language information in the right-hand and lower frames with the images of the original English translation and associated English footnotes by alt-clicking in the dark text. Note that clicking (or alt-clicking) in the dark text also draws the rectangle around the corresponding text in the Navajo or English images.

The coordination of these documents needs to be stored someplace on the server. In the original server-side processing project application, this linking information was stored in the TEI markup and a style sheet dynamically created the frames interface. The client-side version of the simple interface uses JavaScript Object Notation (JSON) to deliver the linking information to the client where the processing takes place. The JSON for the linking shown in Fig. 3 is given subsequently in Fig. 4.

This JSON format can be easily stored in XML format as standoff markup (DeRose, 2004) for archival purposes and batch transformed to JSON



Navaho Texts
Edward Sapir Harry Hoijer
1942

Abstract

Navajo text collected by Edward Sapir at the 1929 Laboratory of Anthropology Field School in the reservation town of Crystal, New Mexico. Told and translated by John Watchman. It appears on pages 334-337 in the original.

III. PERSONAL NARRATIVES

29. The Story of a Navaho Woman Captured by the Utes

'ałk'idą́ą́', ndahondzoodą́ą́', na'nishkaadgo, nóóda'é shikiji' łįį' biłhaazhjéé'. t'áá'áko ła' sisił. shilįį'ę́ę yida'niiłtseedgo yaańdiikai. shí t'éit'óó shináál. ła' ńda'ał'ahgo ła't'éeyá łįį' yik'inda'eeníił. 'áádóó 'atsį' ndayiist'é dóóbáahda 'ádayiilaa. 'áádóó da'iidą́ą́'. 'áłtsǫ́ǫ da'iidą́ą́'go, łįį' yik'idahńda'as'nil. shilįį' yidziihę́ę 'atah niki'nńłkaad. shiłnki'ńná.

'aadę́ę' tsélichii' dah'azkáńdę́ę', dóó'aadę́ę' chińlįįdę́ę', dóótséghi'góne', dóóchézhińdę́ę', dóótséhilįįdę́ę', dóóbighą́ądę́ę'. dóó'aadę́ę' bitis, dóótsé'naa tooh bich'į'go. 'áadi łįį' yizéé' hada'iiznil. dóótsé'naa biłdahda'diiłkǫ́ǫ'. tsé'naa nniikaigo, 'áadi da'didą́ą́'. da'iidą́ą́'dóó, 'ałts'ásiikai. ła' shiłdahdii'áázh, bHome

29:1. šI 'I'; ƚéi 'only'; ƚó·- 'just': šin
eyes, in my presence'.
29:2. šikéle·ž 'my foot-soil'; ʔáǯIlé·hlá 'he

III
29. The Story o

ʔałkidą́·ʔ, ńdaho
kiǯį̨ʔ łį̨·ʔ biłha·žǯ
yidaʔni·łce·dgo ya·ńdi
ʔałʔahgo łaʔƚę·yá łį̨·
yi·sƚé dó·bą̄·hda ʔá
daʔi·dą̄·ʔgo, łį̨·ʔ yik
nikiʔńńłka·d. šiłńkiʔń

ʔa·dę́·ʔ céłičI·ʔ
dó·céγiʔgóneʔ, dó·čéž
dó·ʔa·dę́·ʔ bitis, d
yizé·ʔ hadaʔi·znil.

Fig. 3 The simple frames-based interface

```
{"bindings": [
         {"nifile": "sapir1a.html","eifile":"sapir1c.html","nfifile":"sapir1d.html","efifile":
"sapir1b.html"},
      {"nifile": "sapir2a.html", "eifile": "sapir2c.html", "nfifile": "sapir2d.html", "efifile":
"sapir2b.html"},
      {"nifile": "sapir3a.html", "eifile": "sapir3c.html", "nfifile": "sapir3d.html", "efifile":
"sapir3b.html"}
   ]
}
```

Fig. 4 JSON format

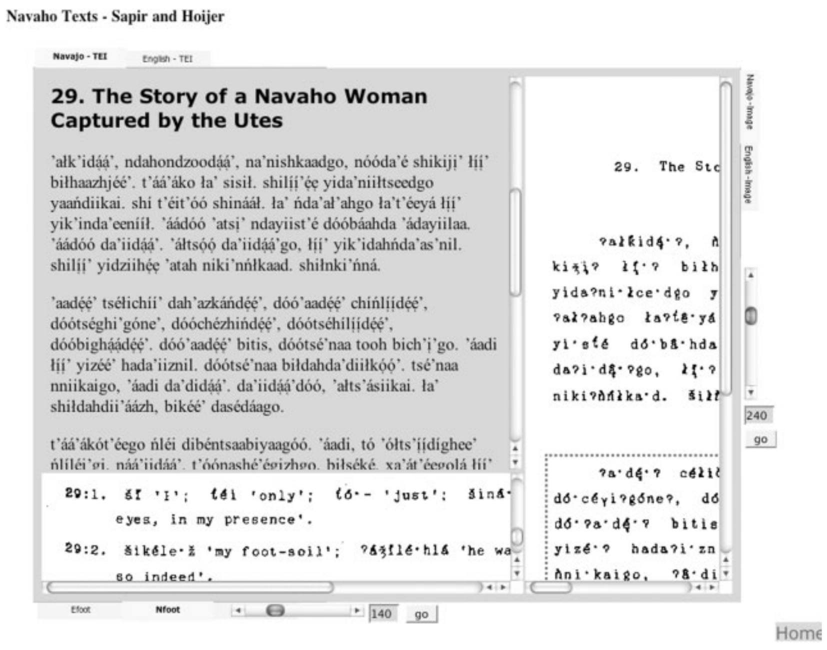
for the application. The 'bindings' specify the linkages between all the TEI related files for the interface: the Navajo image, the English translation image, the Navajo footnote image, and the English footnote image. There is only one TEI file for each text with its multiple related image files. Asynchronous request is required to use these JSON files where each file is asynchronously downloaded and evaluated natively as a JavaScript object in the client code. The use of JSON is a consistent component of the client-side design pattern. It allows dynamic delivery of information to the JavaScript processing engine for the application.

Figure 5 shows the complex interface. This interface is tabs-based and the entire page with all the associated regions is under one DOM.

This allows a much richer interface at the cost of complexity of the client application. The tabs can let the user explicitly choose regions and associated applications rather than the indirect keystrokes. For the complex DOM implementation, asynchronous request is needed for both avoiding large page reloads and delivery of the JSON files.

4 Model Axis: Processing

The processing axis is best illustrated with the update application added to the basic viewer application described before. In the original application design, interlinear information was added directly to the TEI for the Navajo language texts.



29. The Story of a Navaho Woman Captured by the Utes

'ałk'idąą', ndahondzoodąą', na'nishkaadgo, nóóda'é shikijį' łįį' biłhaazhjéé'. t'áá'áko ła' sisił. shilįį'ęę yida'niiłtseedgo yaańdiikai. shí t'éit'óó shinááł. ła' ńda'ał'ahgo ła't'éeyá łįį' yik'inda'eeníił. 'áádóó 'atsį' ndayiist'é dóóbáahda 'ádayiilaa. 'áádóó da'iidąą'. 'áłtsǫǫ da'iidąą'go, łįį' yik'idahńda'as'nil. shilįį' yidziihęę 'atah niki'nńłkaad. shiłnki'ńná.

'aadęę' tséłichii' dah'azkáńdęę', dóó'aadęę' chíńlįįdęę', dóótséghi'góne', dóóchézhińdęę', dóótséhilįįdęę', dóóbighąądęę'. dóó'aadęę' bitis, dóótsé'naa tooh bich'į'go. 'áadi łįį' yizéé' hada'iiznil. dóótsé'naa biłdahda'diiłkǫǫ'. tsé'naa nniikaigo, 'áadi da'didąą'. da'iidąą'dóó, 'ałts'ásiikai. ła' shiłdahdii'áázh, bikéé' dasédáago.

t'áá'ákót'éego ńléi dibéntsaabiyaagóó. 'áadi, tó 'ółts'įįdighee' ńlíléi'gi. náá'iidáá'. t'óónashé'égizhgo. biłséké. xa'át'éegolá łíí'

```
29:1.  šI 'I';  łéi 'only';  łó·- 'just':  šiná·
       eyes, in my presence'.
29:2.  šikéle·ž 'my foot-soil';  ?áʒIlé·hlá 'he wa
       so indeed'.
```

```
        29.  The Sto

     ?ałkidą́·?,  ń
kiʒį?  łį́·?  biłh
yida?ni·łce·dgo  y
?ał?ahgo  ła?tę́·yá
yi·sté  dó·bą́·hda
da?i·dą́·?go,  łį́·?
niki?ńńłka·d.  šiłń

     ?a·dę́·?  céłič
dó·céγi?góne?,  dó
dó·?a·dę́·?  bitis
yizé·?  hada?i·zn
ńni·kaigo,  ?ą́·di
```

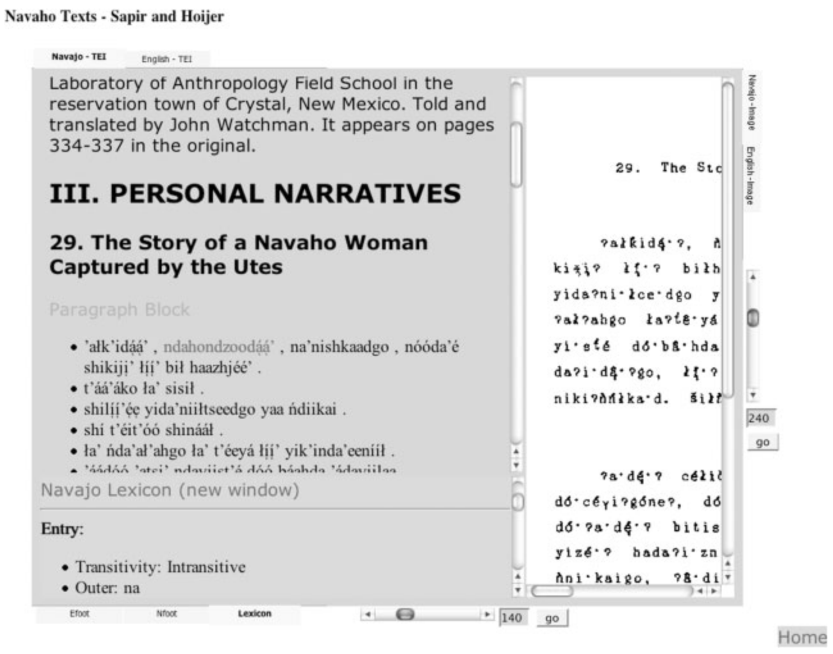
Fig. 5 The complex tabs-based interface

Either a synchronous or an asynchronous request carried an XUpdate or SQL update to the server in the traditional manner.

A later and more sophisticated on-line lexicon design (Canfield, 2007) led to a major change for the project client-side design pattern. Rather than place the interlinear information into the main document, each Navajo word in the TEI now contains only an ID. A separate standoff document relates each word ID to the relevant ID in the Navajo lexicon document. Both this standoff word document and the lexicon document are transformed to JSON for delivery to the client. This allows the client to manipulate both sources of information as native JavaScript objects. It also allows dynamic delivery of different sets of this information to the client. Figure 6 shows the complex interface with an added tab at the bottom for the lexical lookup.

The user clicks on any word in the TEI region and the lexical entry for that word is displayed in the bottom region. One can also see the entire searchable lexicon open in a new window by clicking on the link in that region. No further round trips to the server are required to display the lexical entry of a word in the text or to see and search the entire lexicon because that information has already been delivered to the application via asynchronous request and JSON files. Note that the TEI region is displayed differently, but this is the result of a different style sheet for the same TEI file that allows easier line-by-line display while requesting lexical information.

Both of these documents that come down as JSON files need to be updated collaboratively. Each word of the TEI text needs to be linked to the lexicon file and the lexicon itself needs updates. The collaborative nature of this update requires central server-side processing and handling of concurrency issues. The traditional methods for native XML or relational databases are effective and will continue to be the standard, but this poses a problem for a strictly client-side design pattern. This logically server-side function can be adapted to the client-side pattern by outsourcing it to the emerging public access applications such as Google Docs.[4]



Laboratory of Anthropology Field School in the reservation town of Crystal, New Mexico. Told and translated by John Watchman. It appears on pages 334-337 in the original.

III. PERSONAL NARRATIVES

29. The Story of a Navaho Woman Captured by the Utes

Paragraph Block

- 'ałk'idą́ą́' , ndahondzoodą́ą́' , na'nishkaadgo , nóóda'é shikiji' łį́į́' bił haazhjéé' .
- t'áá'áko ła' sisił .
- shilį́į́'ę́ę yida'niiłtseedgo yaa ńdiikai .
- shí t'éit'óó shináál .
- ła' ńda'ał'ahgo ła' t'éeyá łį́į́' yik'inda'eeníił .

Navajo Lexicon (new window)

Entry:

- Transitivity: Intransitive
- Outer: na

29. The Stc

ʔałk̔idą́·ʔ, ń
kiʒįʔ łį́·ʔ biłh
yidaʔni·łce·dgo y
ʔałʔahgo łaʔt̔ę́·yá
yi·sť̔é dó·bą́·hda
daʔi·dą́·ʔgo, łį́·ʔ
nikiʔńńłka·d. šiłł

ʔa·dę́·ʔ céłił
dó·céγiʔgóneʔ, dó
dó·ʔa·dę́·ʔ bitis
yizé·ʔ hadaʔi·zn
ńni·kaigo, ʔą́·di

Fig. 6 The update application on the complex interface

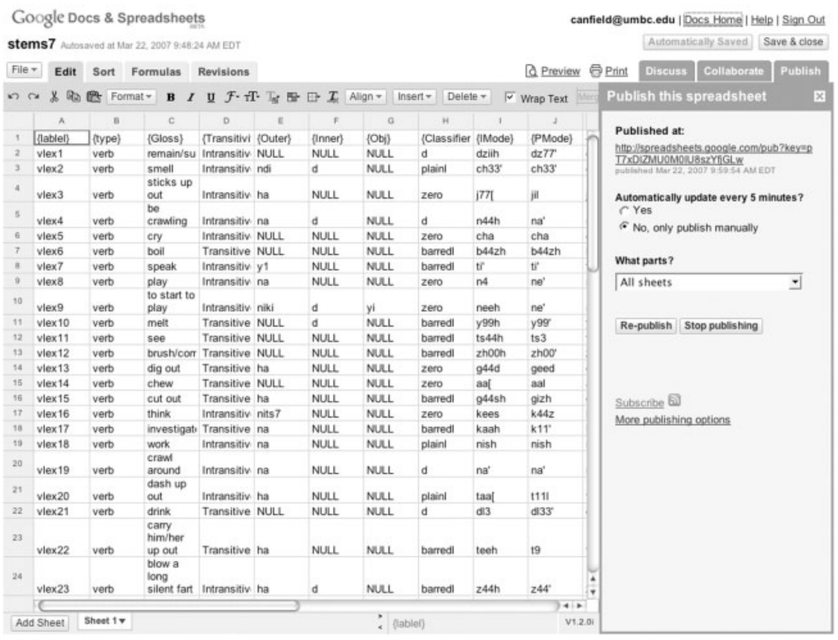
Figure 7 shows the Google spreadsheet collaborative interface for the Navajo lexicon file.

Each of these JSON files can be stored in a Google spreadsheet. Google spreadsheets allow instant publishing on the web with access control by email address for editors. Concurrency is handled by notification of all simultaneous on-line editors of a document with a chat function for resolving updates. Since projects that would use such a resource are small, this is a very effective control. These spreadsheets can be exported periodically to CSV format that can be easily transformed to XML or JSON format.

5 Discussion

There are various permutations possible for this design pattern model, but the most client-centric one from the upper left-hand corner is described as follows. The Navajo language texts are acquired and put in TEI form. XSLT style sheets transform these documents to HTML that can be published on the web server. The standoff markup that describes the interlinear text and the document linking structures is transformed to JSON rather than HTML and these are published on the web server. Finally the lexicon document is transformed to JSON and published. When a user requests the main application over the web, all of these files are dynamically integrated to produce a fairly sophisticated interface with no significant server-side processing.

The only maintenance for this kind of implementation is update and organization of files on the web server using typical client-side tools. The web application is a framework that can be customized for a particular project and then no further changes are required unless requirements change. All the files for the project can be created in XML and then transformed to either HTML or JSON in batch form and then placed into the appropriate directories on the web server. Updates to the interlinear text or lexicon can be collaboratively done using Google spreadsheets and



stems7 Autosaved at Mar 22, 2007 9:48:24 AM EDT

✓ Wrap Text

	A	B	C	D	E	F	G	H	I	J
1	{lablel}	{type}	{Gloss}	{Transitivi	{Outer}	{Inner}	{Obj}	{Classifier	{IMode}	{PMode}
2	vlex1	verb	remain/su	Intransitiv	NULL	NULL	NULL	d	dziih	dz77'
3	vlex2	verb	smell	Intransitiv	ndi	d	NULL	plainl	ch33'	ch33'
4	vlex3	verb	sticks up out	Intransitiv	ha	NULL	NULL	zero	j77[	jil
5	vlex4	verb	be crawling	Intransitiv	na	d	NULL	d	n44h	na'
6	vlex5	verb	cry	Intransitiv	NULL	NULL	NULL	zero	cha	cha
7	vlex6	verb	boil	Transitive	NULL	NULL	NULL	barredl	b44zh	b44zh
8	vlex7	verb	speak	Intransitiv	y1	NULL	NULL	barredl	ti'	ti'
9	vlex8	verb	play	Intransitiv	na	NULL	NULL	zero	n4	ne'
10	vlex9	verb	to start to play	Intransitiv	niki	d	yi	zero	neeh	ne'
11	vlex10	verb	melt	Transitive	NULL	d	NULL	barredl	y99h	y99'
12	vlex11	verb	see	Transitive	NULL	NULL	NULL	barredl	ts44h	ts3
13	vlex12	verb	brush/con	Transitive	NULL	NULL	NULL	barredl	zh00h	zh00'
14	vlex13	verb	dig out	Transitive	ha	NULL	NULL	zero	g44d	geed
15	vlex14	verb	chew	Transitive	NULL	NULL	NULL	zero	aa[	aal
16	vlex15	verb	cut out	Transitive	ha	NULL	NULL	barredl	g44sh	gizh
17	vlex16	verb	think	Intransitiv	nits7	NULL	NULL	zero	kees	k44z
18	vlex17	verb	investigat	Transitive	na	NULL	NULL	barredl	kaah	k11'
19	vlex18	verb	work	Intransitiv	na	NULL	NULL	plainl	nish	nish
20	vlex19	verb	crawl around	Intransitiv	na	NULL	NULL	d	na'	na'
21	vlex20	verb	dash up out	Intransitiv	ha	NULL	NULL	plainl	taa[	t11l
22	vlex21	verb	drink	Transitive	NULL	NULL	NULL	d	dl3	dl33'
23	vlex22	verb	carry him/her up out	Transitive	ha	NULL	NULL	barredl	teeh	t9
24	vlex23	verb	blow a long silent fart	Intransitiv	ha	d	NULL	barredl	z44h	z44'

{lablel}

Publish this spreadsheet

Published at:
http://spreadsheets.google.com/pub?key=pT7xDlZMU0M0lU8szYfjGLw
published Mar 22, 2007 9:59:54 AM EDT

Automatically update every 5 minutes?
Yes [✓ No, only publish manually]

What parts?
All sheets

Re-publish | Stop publishing

Subscribe
More publishing options

Fig. 7 Google spreadsheets for collaborative editing

frequently saved as CSV files by a project individual with the publishing responsibility. Then these files can be batch transformed to JSON and published to the project directories. This design pattern is well suited for small work groups in the humanities where maintenance-intensive server-side programs can be a barrier for web publishing of projects that require more user interaction than just viewing a static web page. This pattern allows that interaction and collaboration with only a web server on the server-side. Furthermore, this is a pattern that is easily setup for a work group by a specialist in humanities computing and then that group can take over the project.

6 Previous Work

The concept of a design pattern was introduced by Alexander *et al.* (1977) in the area of architecture. It was used for software patterns in Gamma *et al.* (1995) and became a popular abstraction as a general and repeatable solution to a commonly occurring problem in software design. This abstract concept of pattern was then used for software frameworks in object-oriented design. A framework (Larman, 2002) is a reusable design pattern that is expressed as a set of abstract classes and instances that collaborate. Both of these concepts were used in a research area for hypertext theory called Structural Computing (Nurnberg *et al.*, 2003). Kenneth Anderson defines 'Structural computing is a new paradigm of computation that asserts the primacy of structure over data. The field of structural computing is working to produce a set of principles, techniques, and technologies to ease the task of developing domain-specific application infrastructure.'[5] The current project attempts to create a domain-specific framework that uses a client-side design pattern to simplify the application infrastructure. The ARCHway project has a non-web based framework for creation of image-based electronic editions of Old English texts (Kiernan *et al.*, 2005). A goal of this project is to try and approach the richness of such a non-web application using internet application technology. The MIT Exhibit project (Huynh *et al*., 2007) creates a lightweight structured data publishing framework that allows publishing at very low cost and provides structure to what would ordinarily be static HTML.

7 Future Work

The use of the JSON files for delivery of information to the client as JavaScript objects promises to be a useful area for further work. Client-side applications dynamically receive the JSON-based templates that specify the static and behavioral aspects of domain-specific structure abstractions (Tzagarakis *et al*., 2006). A related topic for research is the ability for asynchronous methods to accumulate information from the web server(s) for 'just-in-time' delivery to the client in a more seamless way. For this Navajo language application, an example use of this is for the application to continuously request lexical information from the on-line dictionary for delivery as JSON so that it is more likely to be immediately available to the user when needed.

The format used for the lexicon for this project exports an RDF ontology that describes the terms in the associated lexicon. This ontology is being converted to the OWL language (Antoniou *et al*., 2003) and linked to the GOLD linguistic ontology using the owl:sameAs statement that indicates that two URI references actually refer to the same thing. This will allow for more global linguistic research (Simons *et al*., 2004).

The Navajo Language Literature Project continues to acquire texts for the digital library while experimenting with the software for delivery of the texts. Additionally, the on-line Navajo language lexicon based on the work of Young *et al*. (1992) continues to evolve for closer integration with this digital library. The client-centric design pattern described in this article allows a sophisticated and interactive application without the need for extensive server-side processing.

References

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., and Angel, S. (1977). *A Pattern Language: Towns, Buildings, Construction*. New York: Oxford University Press.

Antoniou, G. and Frank van, H. (2003). Web Ontology Language: OWL. In Staab, S. and Studer, R. (eds), *handbook on ontologies in information systems*. Heidelberg: Springer-Verlag, pp. 115–150.

Bird, S., Bow, B., and Hughes, B. (2003). Towards a General Model of Interlinear Text, EMELD Language Digitization Project, Michigan State University, July 11–13. http://emeld.org/workshop/2003/bowbaden bird-paper.html (accessed 24/1/2006).

Canfield, K. (2005). A Pilot Study for a Navajo Textbase, *In proceeding of ACH/ALLC Conference*. University of Victoria, CA, June 15–18. http://mustard.tapor.uvic.ca/cocoon/ach_abstracts/xq/xhtml.xq?id=35 (accessed 24/1/2006).

Canfield, K. (2007). *Issues in Presentation and Representation for the Navajo Lexicon, Conference on Endangered Languages and Cultures of Native America*, CELCNA April 13–15, Salt Lake City, Utah.

DeRose, S. (2004). *Markup Overlap: A Review and a Horse*, In *Proceedings of Conference on Extreme Markup Languages*. http://www.mulberrytech.com/Extreme/Proceedings/html/2004/DeRose01/EML2004 DeRose01.html (accessed 24/3/2007).

Fielding, R. T. and Richard N. T. (2002). Principled Design of the Modern Web Architecture. *ACM Transactions on Internet Technology (TOIT)*, **2**(2).

Gamma, E., Richard, H., Ralph, J., and John, V. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Upper Saddle River, NJ: Addison-Wesley.

Garrett, J. J. (2005). Ajax: A New Approach to Web Applications, http://www.adaptivepath.com/publications/essays/archives/000385.php (accessed 24/1/2006).

Huynh, D., Robert M., and David K. (2007). *Exhibit: Lightweight Structured Data Publishing, 16th International World Wide Web Conference*, May 8–12. http://people.csail.mit.edu/dfhuynh/publications.html (accessed 24/3/2007).

Kiernan, K., Jerzy, W. J., Alex, D. *et al.* (2005). The ARCHway Project: Architecture for Research in Computing for Humanities through Research, Teaching, and Learning. *Literary and Linguistic Computing*, **20** (Suppl 1) 69–88.

Larman, C. (2002). *Applying UML and Patterns: An Introduction to Object-oriented Analysis and Design and the Unified Process*. Upper Saddle River, NJ: Prentice Hall PTR.

Nürnberg, P. J., Wiil, U. K., and Hicks, D. L. (2003). *A Grand Unified Theory for Structural Computing*. In *Proceedings of the 2nd Metainformatics Symposium*, Lecture Notes in Computer Science. Springer Verlag.

Rice, K. (2000). *Morpheme Order and Semantic Scope: Word Formation in the Athapaskan Verb, Cambridge Studies in Linguistics 90*. Cambridge: Cambridge University Press.

Sapir, E. and Hoijer H. (eds) (1942). *Navaho Texts*. Iowa City: Linguistic Society of America.

Simons, G., Fitzsimons, B., Lanham, A., *et al*. (2004). A Model for Interoperability, EMELD Language Digitization Project Conference, Wayne State University, Detroit, MI, July 15–18. http://emeld.org/workshop/2004/langendoen-paper.html (accessed 24/1/2006).

Tzagarakis, M., Vaitis, M., and Karousos, N. (2006). Designing Domain-Specific Behaviors in Structural Computing. *New Review of Hypermedia and Multimedia*, **12**(2): 113–42.

Young, R. W., Morgan, W. Sr, and Sally, M. (1992). *Analytical Lexicon of Navajo*. Albuquerque: University of New Mexico Press.

Notes

1 Synchronous behavior means that the client must wait until the server responds to do any further processing. Asynchronous behavior allows the client to continue processing and then deals with the return as a callback interrupt.

2 The XMLHttpRequest JavaScript object allows web clients submit and retrieve data directly in the background without a web page reload. The asynchronous return must be handled as a callback that interrupts and updates the page dynamically.

3 The reader can get a better idea of how all the interfaces work by visiting the demonstration site created for this article at http://zaad.umbc.edu/nllp/ (accessed 26/03/2007). It includes screencasts and software demonstrations.

4 See http://docs.google.com (accessed 22/03/2007).

5 http://www.sigweb.org/community/labs/colorado04.shtml (accessed 24/3/07).