

# Sapphire: Composite Hypertext Document Structures

Peter Bergström

Baskin School of Engineering  
University of California, Santa Cruz  
Santa Cruz, CA 95064

pbergstr@soe.ucsc.edu

Melissa Chan

Baskin School of Engineering  
University of California, Santa Cruz  
Santa Cruz, CA 95064

melissac@soe.ucsc.edu

Michael Dale

Baskin School of Engineering  
University of California, Santa Cruz  
Santa Cruz, CA 95064

dale@ucsc.edu

## ABSTRACT

Inspired by Theodor Nelson's Xanadu and vLit concepts, Sapphire implements a composite document structure in which formatting is specified external to the text. Formatting overlays are rendered over the text, so multiple overlays may be stacked according to user preference. Sapphire supports the viewing of a document composed from multiple fragments, and viewing the original source of each fragment. Other features include annotation on the composite documents and the storing and restoring of document trails in the spirit of the Memex. An editor function permits the creation and editing of Sapphire composite documents. This project served as an exploration of the potential of alternative hypertext implementations to the pervasive HTML standard in use today.

## Categories and Subject Descriptors

H.5.2 [User Interfaces]: navigation, prototyping.

H.5.4 [Hypertext/Hypermedia]: architectures and navigation.

## General Terms

Design, Experimentation, Human Factors, Theory.

## Keywords

Xanadu, format and data separation, overlays, spans, hypertext system, composite document, multiple text fragments, trails, Memex.

## 1. INTRODUCTION

With inspiration from the Xanadu model [4] authored by Theodor (Ted) Nelson, the goal of the Sapphire project was to create a server side viewer and editor of a composite document structure. The project resulted from working closely with hypertext visionary Ted Nelson, author of Literary Machines and Computer Lib, to develop the unique .saf file specification and its behavior, along with the desired behavior of the Sapphire viewer and editor. The .saf file specification allows for the separation between text information and the transformations (such as bold or links) into a file that does not have inline markup.

Sapphire supports annotations, trails, and non-linear collection of textual information in the form of spans and this allows for a basic, Web-based implementation of Vannevar Bush's Memex.

## 2. SAF format

### 2.1 File Format

The .saf format was developed to separate text from overlay formatting directives. According to the .saf format, span locations are specified as a URL. The URL may be a link to an external site such as a tprints server, or from a local redirect to the sapphire

database. The metacore identifies the spans, stating that the first span can also be referenced as "History." This step subsequently renames the title of span. Overlays are stored as groupings, such as "Pictures," which can contain both links and text formatting. The benefit behind developing the .saf format has been the ability to see where the original text is located through the URL and multiple levels of applied overlays.

```
#span references as URLs
http://sapphire.quadfusion.net/local.php?span_id=3
http://tprints.ecs.soton.ac.uk/archive/00000005/01/seclife.txt?mode=te
xt&locspec=charrange:661/348
#overlays
overlay(metacore){
  identify("History", 1 ); #Identify spans
  identify("Experiment", 2 );
}
overlay(Monkey Pictures){
link("History", "57/14",
"http://www.nasm.si.edu/galleries/images/S81726.f.jpg");
italic("History", "204/9"); #formatting overlay
bold("History", "380/93");
,
```

Figure 1: Sample .saf file

## 2.2 Rendering Method

The .saf file specification allows for overlay information to be associated with spans. Text effects or links can be added on top of the basic text. There can be multiple overlay layers applied to each span and each overlay layer can have many effects on the text. In order to make this manageable, each overlay layer is applied one after each other in the parser.

## 3. VIEWER FUNCTIONS

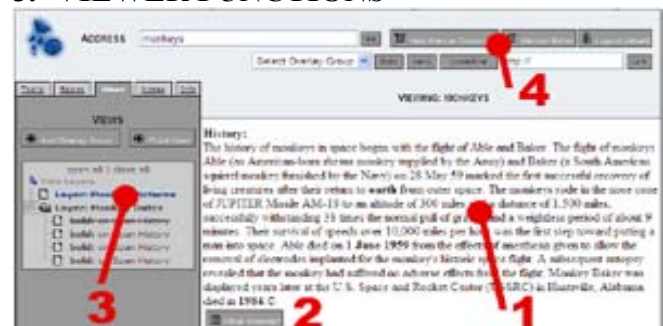


Figure 2: Sapphire application

1. Content area.
2. An annotation attached to the span.
3. The views tab is displayed. A tree structure with the layers and the individual overlays are shown. One can toggle the layers on and off in a similar way to Adobe Photoshop layers.
4. Buttons to apply overlays to specific layers. Effects are bold, italics, underline, and links.

### 3.1 Trails

When someone uses the Sapphire viewer, each .saf file that is viewed is added to a trail. The trail represents the browsing history of the user during the session and this trail can be saved for future use. It allows a user to save his trail at one point in time and start a new branch where he takes his research into a specific direction. He can save this branched trail then reload the previously saved trail before the branch. This then allows him to then start another branch. Then, using the saved trails the user can recollect his research processes when, for example, is writing a research paper.

### 3.2 Spans

*Document Spans* Each span of a document is saved in the database after a .saf file is parsed. When the document is then viewed by the user, the spans can quickly be retrieved by the client through the database, in order. The user has the option to collect spans as he goes along his trail of documents. By clicking on the span, the user is presented with a dialog box where he can enter in his description of the span and then it is collected into his span bin. The span bin is associated with the trail that the user is browsing along.

*External Spans* Spans can also be collected externally from a tprints server [2]. The user can browse to a desired tprints server and then highlight the text that he wants to collect and click a button to capture it. When it is captured, it is added to the user's span bin.

### 3.3 Annotations

A user may want to add annotations as he reviews a composite document. By clicking on a span that he wants to associate the annotation, he is presented with a dialog box where he can add his text. After then, each time the user reviews that document, it will have his annotation attached to the span that he annotated. The annotation is local to the user until he chooses to publish the document where the annotation is converted into a span.

### 3.4 Overlay Layers

*Viewing Options* In the Sapphire viewer, the user has the option to turn on or off whole overlay layers or individual text effects within a specific overlay layer. As the user toggled on or off the overlays, the page is refreshed and the effects are rendered appropriately.

## 4. EDITOR FUNCTIONS

A user can edit an existing .saf file by modifying its source. The source is built from the database information with all the modifications that the user may have done through adding annotations and adding overlays. The user can choose to save the document back to the database or export it to a .saf file.

## 5. BACKEND CONFIGURATION

### 5.1 Database

A database stores the span identification, overlay specifications, document trails, and user preferences. The three main tables are the document, span, and overlay tables. The document table contains: document name, owner, description, metacore, span order, and overlay order. The spans and overlays are stored in a simple list in a field. The span table contains: span name, description, and document id. The overlay table contains: overlay type (either a metacore or a layer), user id, name, description, overlay commands, and a field to toggle on and off the overlay groups. Since file data is parsed into the database, the manual editor is able to extract and display the data in the .saf format.

### 5.2 Parser

A parser was created to extract the relevant text in the address bar and store the information in the database. The parser is also used to update changes made in the manual editor. After the text is checked against the database for duplicates, the document, span, and overlay information is stored in the database. The parser, manual editor, and database are closely integrated to produce the .saf file format.

### 5.3 Integration

In the construction of all three backend elements, the aim was to separate the text from formatting when compositing a document. This was primarily achieved by 1) creating a file format that distinguished between text locations and overlay formatting and 2) storing the data in separate tables in the database.

## 6. CONCLUSION

The overall design of Sapphire aimed to integrate the goals of Ted Nelson's Xanadu and vLit projects within a server side editor and viewer. With the functional aspects of spans, annotations, overlays, and editing tools, Sapphire has become both a composite document viewer and editor. With the backend system design, document text is independent of formatting tags. Overlay layers may be toggled on and off according to user preference. Overall, Sapphire has achieved both its functional and design goals. Future development ideas may include incorporating text spans that are loaded completely dynamically from external resources and allow for images and other types of media.

## 7. ACKNOWLEDGMENTS

Thanks to our advisor, Professor Jim Whitehead at UC Santa Cruz for his guidance during this project. Thanks to Martin Bogomolni for his contributions during the initial stages of this project. Our deepest thanks goes to Ted Nelson for giving us many hours of his time in explaining his ideas for Xanadu, vLit and other projects.

## 8. REFERENCES

- [1] T. H. Nelson, *Literary Machines*, 93.1 ed. Sausalito, CA: Mindful Press, 1981.
- [2] T. H. Nelson, *Tprints*, <http://tprints.ecs.soton.ac.uk/vlit.html>.
- [3] T. H. Nelson, *Xanadu Australia FAQ*, <http://xanadu.com.au/general.faq.html>.
- [4] T. H. Nelson, *The Xanadu Model*. <http://www.xanadu.net>.