A Formal Language for Digital Libraries using Denotational Semantics

Neill A. Kipp, Edward A. Fox

Virginia Polytechnic Institute and State University

{nkipp,fox}@vt.edu

DRAFT: January 16, 1998

ABSTRACT

We formalize the operation of a digital library using denotational semantics for a language that expresses interaction with a digital library. The formalization defines digital objects and their relationship to their metadata, how digital objects are added and removed from digital library collections, how searching is performed on the collections, and how authored interfaces to collections are created and presented to users. In so doing, we provide an ontology for discussion of the construction and operation of digital libraries—to provide a scientific basis for their design evolution and a tool for their pedagogical manipulation—and compare it to existing digital library designs. Finally, we show how a multi-dimensional digital library interface may be built from the formal semantics using 4*S*: sets, streams, spaces, and scenarios.

KEYWORDS: Digital libraries, Open Systems, International Standards

Knowledge and Language

While digital library projects continue to grow and flourish, we require formal treatments to coordinate our digital library efforts, especially for the purposes of interoperability. We begin with an exploration of key concepts.

Knowledge is useless without language. This Vygotskian axiom roots our lifelong education. Language enables us to express what we know, and it effects the way we seek to know more [4]. Both question and answer are part of our language. We approach a library—the organized storehouse of knowledge—with a question, and we return with information in the form of an answer, or documents possibly containing answers. It is necessary, therefore, to treat interactions with a digital library the same way—with language.

The operation of a library, therefore, can be defined in terms of the language of conversation between a user and the library as the user seeks information from the library [1]. The conversation between a user and a library is analogous to the interactions between a software program's statements and the computer's memory and storage devices. Therefore, we model the language of a digital library with a formal programming language, and show how each operation in the formal language relates to our own understanding of the way a library operates using denotational semantics.

To begin this model, we define digital library in a 'denotational' fashion, structured so that the meaning of the whole is expressed in terms of its immediate constituents [19]. A digital library, then, is the combination of terms 'digital' and 'library.' Amending one noted librarian's definition of library, we have a "collection of [digital] graphic materials arranged for relative easy use [i.e., online access], cared for by an individual or individuals [in combination with computer software routines] familiar with the arrangement, and accessible to at least a limited number of persons [or other computers]" [7]. Below we see how digital libraries are the denotational combination of their collections, metadata, search methods, and presentation.

The "collection of digital materials" is the set of electronic information objects contained within the digital library. Objects can be acquired and added to the library collection, and, in reverse, objects can be retired and removed from the collection. Digtal migration and integrity checking are part of maintaining the digital collection [6]. This aligns with Kahn and Wilensky's definition of 'repository,' deposit, access, and mutability. While objects in their repository, however, may never be removed, our definition of collection will allow objects to be removed.

An item in the collection is accessed by position, attributes (name, keywords, color, size, or other features), or through searching or matching. For our purposes, though, a collection is a mathematical relationship: a set of ordered pairs whose first item is an identifier (extended, perhaps, by passwords or receipts indicating payment of royalties) and whose second is the stored digital object (extended perhaps by terms and conditions). By "position or attributes" we mean the sometimes complex combinations of comprehensive attributes that are required to identify an object fully. The set of all attributes of a collection item is its **metadata**. This is consistent with Kahn and Wilensky, whose definition of digital object includes the object's metadata [11]. To help support a plethora of usage patterns, characterization of library materials can become verbose (perhaps because indexers themselves agree on terms only half the time! [14]). We see that richly structured metadata is vital for objects in the digital library.

A variety of **search mechanisms** may be built that operate on the collection to facilitate retrieval of the digital objects. Furthermore, performance of the search mechanisms may be improved by the creation of indexes. Each object type must provide its own search function (or, by default, delegate that responsibility to the library). The digital library aggregates the variety of search functions as part of its interface.

Libraries always have relied upon a coherent, inherent **organization** and an aesthetic presentation of the information they contain. Indeed, librarians, for 4000 years, have organized and presented information for public use. Librarians rely on both art and science to achieve their ends. High ceilings, large windows, ornate fixtures, controlled vocabularies, and human ergonomics have always provided an altruistic context to efficient information reception in a library [7].

Our goal in designing the collection, metadata, searching, and presentation is to achieve a digital library that is useful and easy to use.

Formal System

In the following, we present a formal system as an abstraction for digital libraries and show how manipulations of statements within that system strictly specify how the abstract digital library controls its collection, metadata, searching, and presentation.

The operation of the interpreter of our abstract language is as follows. Using a browsing device that is connected to the digital library, the user submits a statement to the digital library to be evaluated. The library replies with some information and perhaps some alternatives. The user chooses an alternative by submitting a statement to the digital library, to which the library replies, and the cycle continues. This process is analogous to when a user asks the physical library: 'Where are the science books?' and the library replies with these alternatives: 'Lobby: 1st floor, Art books: 2nd floor, Science books: 3rd floor.'

A formal system is made up of a formal language and a deductive apparatus. The formal language consists of an alphabet of symbols and the rules for construction of well-formed formulas (wffs) from the alphabet. The deductive apparatus consists of some wffs that are designated as axioms and a two-place relationship between wffs and wffs. Note that semantic attachment is *not* part of the formal system [8].

We must, however, associate semantics to the parts of a formal system to obtain a formal model for systems of real-world phenomena. If all the statements obtainable by the deductive apparatus "are truths" in the real world, then the formal system is consistent. If the formal system is consistent, and if all semantic notions in the real world can be represented in the model, then the system is complete [13]. Consistency can be proved syntactically, but to have a completeness proof of this formal system, we must first have consensus in the real world regarding the definition of digital libraries.

Formal Language and Deductive Apparatus

The alphabet for our formal language is the set {0,1}. We therefore include everything that can be represented by a digital computer. For kindness in presentation, however, we will use as our alphabet the symbols available on a common typewriter and assume that they can be mapped uniquely onto our alphabet (using ASCII or Unicode, for example). We define a well-formed formula (wff) to be any string of members of of the above alphabet.

The deductive apparatus is as follows. Symbol D is the only axiom. Anything derivable from D by the following context-free grammar is a theorem. Nothing else is a theorem. Because languages generated by context-free grammars are decidable, we can always determine whether a string of alphabet symbols is a theorem in our formal system. [15]

Grammar

The following is a context-free grammar in Backus-Naur Notation (BNF) [18]. Lowercase epsilon (ϵ) denotes the empty string.

$$
\begin{aligned}
A &::= \epsilon \mid KA \\
V &::= A \mid Q \mid I \mid L \\
Q &::= () \mid (Q') \\
Q' &::= E \mid E\ Q'
\end{aligned}
$$

$$
\begin{aligned}
E &::= V \mid \textbf{search } E \mid \textbf{alt } E\ E \\
C &::= E \mid I = E \mid \textbf{add } E \mid \textbf{remove } L \mid \\
&\quad\ \textbf{interface } I\ Q \mid C\ C \\
D &::= C
\end{aligned}
$$

Semantic Domains

Each component of the denotional semantics is below. We begin by associating semantics with each of the syntactic symbols.

First we give meaning to the various symbols in the language.

$$
\begin{aligned}
K &: k\textbf{-tuple} = \{0,1\}^k, k \geq 1 \\
A &: \textbf{String} \\
V &: \textbf{Value} \\
Q &: \textbf{Stream} \\
I &: \textbf{Identifier} \\
L &: \textbf{Location} \\
F &: \textbf{Interface} \\
E &: \textbf{Expression} \\
C &: \textbf{Command} \\
D &: \textbf{Digital Library}
\end{aligned}
$$

Values

A character K is a k-tuple of bits. A string A is a list of characters. A value V is either a string, a stream, an identifier, or a location. An identifier I denotes a named storage location. A location L is the unique identifier of a storage location in the digital library's collection, as it is assigned by the system.

A stream S is a sequence of values. Note that streams can contain streams. Thereby, key-value pairs are represented easily, e.g., ((author Kipp) (degree PhD) (major CS). Note too, that while SGML documents themselves are strings, the parsed versions of SGML documents can be represented by a stream that contains streams. [9, 3].

Next we define the set domains for each grammatical symbol. In the following, cross-product $X \times Y$ denotes the set of ordered pairs (x, y), $x \in X$ and $y \in Y$, while co-product $X + Y$ denotes the disjoint union of X and Y, $\{x_1, ..., x_n, y_1, y_2, ..., y_m\} = X + Y, x_i \in X, y_j \in Y$.

$$
\begin{aligned}
\textbf{Char} &: \{0,1\}^k \\
\textbf{String} &: \textbf{Char} \times \textbf{Char} \times ... \times \textbf{Char} \\
\textbf{Value} &: \textbf{String} + \textbf{Identifier} \\
&\quad + \textbf{Location} + \textbf{Stream} \\
\textbf{Stream} &: \textbf{Value} \times \textbf{Value} \times ... \times \textbf{Value}
\end{aligned}
$$

Store

A **Store** is a function from identifiers to values, represented by shorthand notation $v = f[X]$ which denotes the mapping from identifier X to retrieved value v.

Notation $f' = f[X/y]$ represents the **Storage** function. Storage maps a function to a function: we obtain the new store f' from f by modifying f such that the location represented by X gets the value y and all other locations in f remain unchanged.

In the digital library we have the **Collection** for digital objects and the **Store** for interfaces, streams, and strings. Both collection and store are sets of type **Store**.

$$
\begin{aligned}
\textbf{Store} &: \textbf{Identifier} \rightarrow \textbf{Value} \\
\textbf{Storage} &: \textbf{Store} \rightarrow \textbf{Identifier} \rightarrow \textbf{Value} \rightarrow \textbf{Store}
\end{aligned}
$$

Expression Evaluation

The process of evaluation maps (the syntactic representation of) an expression, a store, and a collection to a value. Generic notation $\mathcal{E}[[X]]$ denotes the syntactic evaluation of X under $\mathcal{E}$ with respect to stores s and c. The same is true for $\mathcal{C}$ (for commands) and $\mathcal{D}$ (for the whole digital library) below.

$$
\mathcal{E} : \textbf{Expression} \rightarrow \textbf{Store} \rightarrow \textbf{Collection} \rightarrow \textbf{Value}
$$

Command Execution

The process of command execution maps (the syntactic representation of) a command, a store, and a collection to a store and a collection.

$$
\begin{aligned}
\mathcal{C} : \textbf{Command} &\rightarrow \textbf{Store} \rightarrow \textbf{Collection} \rightarrow \\
&(\ \textbf{Store} \times \textbf{Collection}\ )
\end{aligned}
$$

Digital Libary

Lastly, a digital library is the process of mapping a command and a special state and collection where all values in the store and collection are empty strings to a store and collection.

$$
\begin{aligned}
\mathcal{D} : \textbf{Command} &\rightarrow \textbf{Store} \rightarrow \textbf{Collection} \rightarrow \\
&(\ \textbf{Store} \times \textbf{Collection}\ )
\end{aligned}
$$

Semantic Evaluation

Now that we have the syntax, domains, and mappings declared, we can specify the exact operations for each. For this, we must instantiate store $s \in$ **Store** and collection $c \in$ **Collection**.

Expression Evaluation

Evaluation of a string yields the string itself. Evaluation of an identifier yields the value in the store to which that identifier refers. Evaluation of a location yields the value in the collection to which the location refers.

$$\begin{aligned}\mathcal{E}[[A]]\ s\ c &= A \\ \mathcal{E}[[I]]\ s\ c &= s[I] \\ \mathcal{E}[[L]]\ s\ c &= c[L]\end{aligned}$$

Evaluation of a stream yields the stream itself, with all contained expressions having been resolved into their corresponding values.

$$\mathcal{E}[[Q]]\ s\ c\ =\ q,$$

$$\text{where } q = \begin{cases} () & \text{if } Q = (), \\ (\mathcal{E}[[Q']]) & \text{otherwise, i.e., when } Q = (Q') \end{cases}$$

$$\mathcal{E}[[Q']]\ s\ c\ =\ q',$$

$$\text{where } q' = \begin{cases} \mathcal{E}[[E]] & \text{if } Q' = E, \\ \mathcal{E}[[E]]\ \mathcal{E}[[Q']] & \text{otherwise, i.e., when } Q' = (E\ Q') \end{cases}$$

The **search** expression calls each stored digital library object's search expression and returns a stream of the collection locations of the digital objects whose search returned true.

The semantics of the search function depends on the semantics of the particular type of the stored digital object. For ASCII text, search should simply be a literal string match or a string match with regular expressions, and will likely be provided through a function library available to the digital library. In structured text, the function should operate using a query language for fielded search (e.g., SDQL from DSSSL [10]). If the digital object is an image (e.g., JPEG, EPS), then the query itself might be a comparable image [5], with the function returning true if a match was found. If the digital object is a table, spreadsheet, or declared space, then the functions that interpret those data types should return matches appropriately.

$$\mathcal{E}[[\textbf{search}\ E]]\ s\ c\ =$$

where $q \in Q$ is a stream as follows:
for each (l, v), $v \neq \epsilon$,
  if $\textbf{search}(c[l], e) = \mathit{true}$ then
    l is appended onto the stream q,
    no operation otherwise,
and where $e = \mathcal{E}[[E]]\ s\ c$.

To support querying with ranked results, we must modify the construction of q as follows:

for each $(l, v) \in c, v \neq \epsilon$,
  $z = \textbf{search}(c[l], e)$, with $z \in \textbf{Integer}$
    if $z > 0$, then the stream $(z\ l)$
      is inserted into the stream q,
      preserving the z-sort on q;
      no operation otherwise,
and where $e = \mathcal{E}[[E]]\ s\ c$.

The expression **alt** with two arguments denotes that value obtained from the evaluation of the second expression be 'presented.' (Recall the usage scenario in the introduction, where a browsing device allows a user to interrogate the system and the system to reply.) The second expression is a linguistic expression (word, phrase, icon) provided as a "hint" to "feed the user forward" to the next interface presentation. Should the user accept the hint and request that the results of the first expression be presented, a verbatim copy of the first expression will be sent (by the browsing device) to the expression evaluator for the digital library. Note that this functionality is analogous to document presentation and hypertext linking [16].

$$\mathcal{E}[[\textbf{alt}\ E_1\ E_2]]\ s\ c\ =\ e,$$

$$\text{where } e = \mathcal{E}[[E_2]]\ s\ c.$$

Expressions as Commands

The evaluation of an expression as a command is a null operation on state and collection. In the context of usage, however, the results of the evaluation of the expression are returned to the user.

$$\mathcal{E}[[E]]\ s\ c\ =\ s\ c,$$

$$\text{where } e = \mathcal{E}[[E]]\ s\ c$$

is passed to the browsing device

The **assignment** command assigns the results of evaluation of E to the identifier I. A new store results. The collection is unaffected.

$$\mathcal{C}[[I = E]]\ s\ c\ =\ s[I/e]\ c,$$

$$\text{where } e = \mathcal{E}[[E]]\ s\ c.$$

The **add** command adds the value resulting from the evaluation of E—a digital library object with metadata, represented by a string or stream—to the library's collection. Conventions for the specification of object and metadata are left to a particular implementation of a digital library. Without the add operation, one cannot house a single book.

$$\mathcal{C}[[\textbf{add } E]]\ s\ c \quad = \quad s\ c[l/e],$$

$$\text{for } l \text{ such that } c[l] = \epsilon \text{ and}$$

$$e = \mathcal{E}[[E]]\ s\ c.$$

The **remove** command removes a digital object from the library's collection. Without this operation, items in the collection may never be retired, nor replaced by new versions, for the lifetime of the library. Location L is available as the output of a search.

$$\mathcal{C}[[\textbf{remove } L]]\ s\ c \quad = \quad s\ c[L/\epsilon]$$

The **interface** expression is a stream of values. Unlike regular assignment, evaluation of Q is delayed until I is evaluated.

$$\mathcal{E}[[\textbf{interface } I\ Q]]\ c \quad = \quad s[I/Q]\ c$$

Finally, the sequencing command allows multiple commands to be sent to the library. The state that results from the execution of C_1 is used as the state for the execution of C_2.

$$\mathcal{C}[[C_1 C_2]]\ s\ c \quad = \quad \mathcal{C}[[C_2]]\ s'\ c',$$

$$\text{where } s'\ c' = \mathcal{C}[[C_1]]\ s\ c.$$

Digital Library

The digital library is the execution of its commands on a state and collection which are initially empty.

$$\mathcal{D}[[C]] \quad = \quad \mathcal{C}[[C]]\ s\ c,$$

$$\text{where for all } I, s[I] = \epsilon, \text{ and}$$
$$\text{for all } L, c[L] = \epsilon.$$

Example of Usage

Let us assume a reasonable concrete syntax and a simple fielded search mechanism for the narrated example of a digital library interaction that follows.

The following digital library command adds *Moby Dick* by Melville to the collection (in the interest of space, the full text of the book is represented by the ellipsis). Note how the metadata is a stream of key-value pairs.

```
add(( type, "book"),
     ( title, "Moby Dick"),
     ( author, "Melville"),
     ( body, "Call me Ishmael..."));
```

In the same way, we add the Dickens classic, *A Tale of Two Cities*. Note that both books have the same metadata structure. Standards for the electronic representation of metadata include Machine-Readable Cataloging (MARC) and the Dublin Core.

```
add(( type, "book"),
     ( title, "A Tale of Two Cities"),
     ( author, "Dickens"),
     ( body, "It was the best
         of times..."));
```

The following assignment leaves the string "nkippvt.edu" in the digital library's store using identifier "admin."

```
admin = "nkipp@vt.edu";
```

The first interface definition follows. This construct binds the value "initial" to the stream of expressions. This interface may be invoked by simply stating "initial" to the expression interpreter. The user interface will receive this data (including the resolution of "admin" with the current state) according to the processing model defined above.

```
interface initial (
  "Simple Digital Library",
  alt( titlescreen, "titles"),
  alt( authorscreen, "authors"),
  "Administrator:", admin);
```

Next we declare two more interface objects and add another book.

```
interface titlescreen (
  "Titles",
  alt( search( "title<=K"), "A - K"),
  alt( search( "title>K"), "L - Z")
  "Administrator:", admin);
```

Table 1: Sample User Interaction

User	Digital Library
initial;	
	Simple Digital Library titles § authors § Administrator: nkipp@vt.edu
titlescreen;	
	Titles A - K § L - Z § Administrator: nkipp@vt.edu
search('title<=K');	
	A Tale of Two Cities § Anna Karenina §
0xae2460;	

```
interface authorscreen (
  "Authors",
  alt( search( "author<=K"), "A - K"),
  alt( search( "author>K"), "L - Z"),
  "Administrator:", admin );

add((( type, "book"),
     ( title, "Anna Karenina"),
     ( author, "Tolstoy"),
     ( body, "All happy families
         resemble one another...")));
```

An example invocation of the interface as a dialog occurs in the columns below. Symbol "§" indicates to the user that a hypertext link may be followed (see Table 1).

where 0xae2460 is the location of the collection item whose title is *Anna Karenina*.

Should the librarian wish to remove *Anna Karenina* from the collection, he or she can submit the following command to the digital library.

```
remove( search( "title='Anna Karenina'"));
```

Discussion

The above description makes a simple, useful formalism for the creation and maintenance of digital libraries. We perceive its primary application is for instruction: to teach students how to characterize digital libraries by giving formalisms to show how they treat collection, metadata, search, and presentation. In the following, we show how our digital library definition is consistent with existing digital library definitions, designs, and implementations.

Handles and the Repository

First, we address Kahn's and Wilensky's discussion of digital object, repository, and handles [11]. We use their notion of digital object as our own, though we do not distinguish a handle from any of the other metadata—if handles are required by an implementation, they may be specified in our model as a string within the digital object stream (e.g., (add ((handle, 'vt/kipp9801') (body, 'A Formal Language...') etc.))). Kahn and Wilensky leave all functions of interface to the various applications of the repository (see Table 2).

Table 2: Comparison of Kahn/Wilensky

Kahn/Wilensky	Ours
digital object	stream
key-metadata	stream
metadata	stream
repository	collection
deposit_do	add
access_do(handle)	search
access reference services	resolve identifier
handle	string
resolve	search
insert	add
delete	remove

World Wide Web

The Web is a very large collection of digital objects, some of whose only metadata is their name of their location (URL). Indeed, organization of the Web is a strict process, in that pages are linked together using only the URL of each digital item as a linking device. Digital objects may be added and removed, interface objects may be written and accessed, and searches may be performed. Therefore, while the Web is a digital library, there is little consensus regarding subtleties of its organization and little hope that all materials in the collection have been reindexed an instant before one's search is performed.

A corollary to this is that Alta Vista and other search engines are also digital libraries—ones whose digital objects are merely words and URLs found on the Web by indexing robots. Furthermore, categorization engines like Yahoo are also digital libraries—ones whose digital objects are locations on the Web and whose information objects categorize the contents of those locations.

Stanford Bus

The crux of Stanford's Digital Library Infobus is engineering *how* the operation we call searching is going to take place in a widely distributed system [17]. Heterogeneous interoperation of digital libraries is made possible by the InfoBus. While our model requires only that searching exist, the InfoBus distributes responsibility—for addition, removal, searching, and consistent presentation of materials—to anyone who 'rides the bus.' Furthermore, the "Interpay" operation for payment of royalties is also part of searching. While Stanford separates the functionality, we include negotiation and payment of royalties as dialog contained in the search command, e.g., (search "title = 'Anna Karenina' maxpay = $6.5 mycard = 'MC-4234-...' "). We conclude that while our model exactly specifies what happens in a digital library, it does not attempt to engineer a solution for the "search" function that scales across a wide network.

Table 3: Comparison to IBM Modules

IBM Modules	Our procedure	Our data
presentation	browser	streams
application	browser	streams
document manager	commands/expressions	strings/streams
storage subsystem	commands	store/collection
database	add/remove/search	streams
catalog	add/remove/search	streams
collection	add/remove/search	collection

IBM Modules

Gladney *et al* describe muliple object-oriented modules for encapsulating data and functionality in a digital library context [5]. This detailed work concerns itself with issues ranging from collection storage to presentation management from an implementation perspective. Because our system separates procedures from data and theirs pairs them (following the object-oriented paradigm), the categorization is more complex (see Table 3).

More interesting, perhaps, is how the particular modules might be implemented using our model as a basis. Security servers are perhaps most interesting in this respect. In our model, all security attributes will appear in the stream of metadata that is associated with the collection item upon its addtion to the archive. Because all retrieval of secure data is through the search expression, necessary information (e.g., passwords) must be presented when each search is made. In an implementation, of course, the security manager will facilitate this operation, but the basis is simple: security is exactly the careful combination of metadata and search criteria. The same argument holds for indexing (optimizing a search by building an index) and filtering (manipulating the output stream of a search).

Gladney ends his enumeration of digital library with "agent modules." Please see the comparison between the formal model and "agents" the next section.

Michigan Agents

In the Michigan model (like IBM modules) all responsibilities for digital library creation and maintenance are shared among agents. There are three types of agents: user interface agents, mediator agents, and collection agents. The user interface agents maintain the input and output to user interfaces, the mediator agents handle the translations (like aggregating results lists) between user interface agents and collection agents, and the collection agents storage and retrieval of the various digital objects in the collection. Like the IBM modules, the agents are objects—they have procedures and data. The mapping, however, is clear. User interface agents correspond to our browser device—both exchanges data with the digital library system. The mediator agents implement most of the searching functionality. The collection agents maintain the collection adding, removing, and retrieving digital objects.

Dienst Federated Servers

The Dienst digital library software, on which the Networked Computer Science Technical Report Library (NCSTRL) is based, implements digital library functionality as we have described it. However, the most significant characteristic of Dienst is that it supports federated searching. Each library has a local collection of materials that are accessed by the Dienst protocol as well as the ability to access the collections other Dienst servers across the Internet. In operation, the search expression is distributed to participating libraries and integrated results are presented to the user.

This is achieved in our model as follows, seeing how each Dienst library a library of libraries. The Dienst library collects digital objects that may be proxies for entire libraries. When the search is performed, all digital objects in the collection are consulted, including the library proxy for the remote sites, and results are returned through that proxy.

4*S*: Sets, Streams, Spaces, and Scenarios

Designers of Agents, Modules, and InfoBus have concentrated their efforts on the various types of searching that can occur in a digital library context, nearly avoiding the obvious: that digital libraries are created to serve a community of users [2]. We propose that more effort be made to support users of digital libraries: to concentrate on building useful and usable interfaces that enable digital librarians to organize collections in a variety of ways for a variety of library users in a variety of usage contexts including homes, laboratories, offices, schools, and walk-up public terminals.

From the above **set**-theoretic formalization, one can see how easily the interface **stream** can be delivered as a hypertext using the HTML syntax, where the "alt" construction maps onto `<a href="E1">E2</a>`. Many digital libraries (e.g., D-lib, NCSTRL) use the Web as their primary user interface.

Furthermore, by simply reencoding the interface stream, a team of digital library students at Virginia Tech, working on the Networked Digital Library of Theses and Dissertations, have shown that the Virtual Reality Markup Language (VRML) can be used to deliver virtual rooms full of virtual books that allow the user to have the *feeling* of being in a library, meanwhile taking full advantage of all the directional cues and aesthetic presentation librarians can utilize in a multi-dimensional **space** (See Figure 1) [12].

Finally, because dialog must occur through time, we will use our formalization to construct usage **scenarios** in the spaces we declare to see how users approach the library, what paths they take, and where they tarry, to

see how we may improve our interfaces to benefit our users.

Acknowledgments

Layne Watson guided the early development of the basic formalism. Paul Mather and Franciso Jaen-Martinez also have contributed to the ongoing definition of 4*S*. The National Science Foundation, through grants CDA-9312611 and IRI-9116991, as well as SURA and U.S. Dept. of Education, FIPSE Program (for their support of our thesis and dissertation project), provided support.

REFERENCES

1. N. J. Belkin, P. Kantor, E. A. Fox, and J. A. Shaw. Combining the evidence of multiple query representations for information retrieval. *Information Processing & Management*, 31(3):431–448, May-June 1995.

2. Christine L. Borgman, *et al.* Social aspects of digital libraries, November 1996. http://www-lis.gseis.ucla.edu/DL/.

3. James J. Clark. *SP: An SGML System Conforming to International Standard ISO 8879–Standard Generalized Markup Language*, August 1997. http://www.jclark.com/sp/.

4. Lisbeth Dixon-Krauss. *Vygotsky in the Classroom.* Longman Publishers, White Plains, New York, 1996.

5. Henry M. Gladney, Edward A. Fox, Zahid Ahmed, Ron Ashany, Nicholas J. Belkin, and Maria Zemankova. Digital library: Gross structure and requirements: Report from a March 1994 Workshop. In John L. Schnase, John J. Leggett, Richard K. Furuta, and Ted Metcalfe, editors, *Proceedings of Digital Libraries '94: The First Annual Conf. on the Theory and Practice of Digital Libraries*, pages 101–107, College Station, TX, June 19-21, 1994. Hypermedia Research Laboratory, Dept. of Computer Science, Texas A&M Univ. Electronic proceedings at http://atg1.WUSTL.edu/DL94.

6. Peter Graham. Dl glossary. email, January 1998. Rutgers University Libraries.

7. Michael H. Harris. *History of Libraries in the Western World.* Scarecrow Press, Metuchen, NJ, 4th edition, 1995.

8. Geoffrey Hunter. *Metalogic: An Introduction to the Metatheory of Standard First Order Logic.* University of California Press, Berkeley, 1996.

9. International Organization for Standardization, editor. *ISO 8879: Standard Generalized Markup Language.* ISO, 1986.

10. International Organization for Standardization, editor. *ISO/IEC 10179: Document Style Semantics and Specification Language (DSSSL).* ISO/IEC, 1996.

11. Robert Kahn and Robert Wilensky. A framework for distributed digital object services. Web page, May 1995. http://www.cnri.reston.va.us/k-w.html.

12. Neill A. Kipp. Case study: Digital libraries with a spatial metaphor. In *SGML/XML '97 Conference Proceedings*, pages 631–639, Alexandria, VA, December 1997. Graphic Communications Association.

13. Virginia Klenk. *Understanding Symbolic Logic, 3rd edition.* Prentice Hall, Inc., Englewood Cliffs, NJ, 1994.

14. Michael Lesk. *Practical Digital Libraries: Books Bytes and Bucks.* Morgan Kauffmann Publishers, San Francisco, CA, 1997.

15. Harry R. Lewis and Christos H. Papadimitriou. *Elements of the Theory of Computation.* Prentice Hall, Englewood Cliffs, 1981.

16. Theodore H. Nelson. *Literary Machines.* Mindful Press, Sausalito, 1990.

17. Andreas Paepcke, Steve B. Cousins, Hector Garcia-Molina, Scott W. Hassan, Steven P. Ketchpel, Martin Röscheisen, and Terry Winograd. Using distributed objects for digital library interoperability. *IEEE Computer*, May 1996. http://computer.org/computer/dli/r50061/r50061.htm.

18. Robert W. Sebesta. *Concepts of Programming Languages.* Addison-Wesley, Menlo Park, 1996.

19. R. D. Tennent. *Principles of Programming Languages.* Prentice-Hall International, Englewood Cliffs, 1981.