TOWARDS A DIGITAL LIBRARY THEORY: A FORMAL DIGITAL LIBRARY ONTOLOGY

Marcos André Gonçalves, Layne T. Watson, and Edward A. Fox.
Virginia Polytechnic Institute and State University
Blacksburg VA 24060
{mgoncalv, ltw, fox}@vt.edu

Digital libraries have eluded definitional consensus and lack agreement on common models. This makes comparison of DLs extremely hard, promotes ad-hoc development, and impedes interoperability. In this paper we propose a formal ontology for digital libraries (DLs) that defines the fundamental concepts, relationships, and axiomatic rules that govern the DL domain, therefore providing a frame of reference for the discussion of essential concepts of DL design and construction. The ontology is an axiomatic, formal treatment of DLs, which distinguishes it from other approaches that informally define a number of architectural variants. The process of construction of the ontology was guided by 5S, a formal model for digital libraries. The resulting ontology can be used to classify, compare, and differentiate the features of different DLs. To test its expressibility we have used the ontology to create a taxonomy of DL services and reason about issues of minimality, extensibility, and composability.

1. INTRODUCTION

Research in Digital libraries (DLs) has historically been very pragmatic. While much attention has been paid to design and implement systems and architectures [Witten03, Castelli03, Payette02, Hussein02], create collections and services [NSDL04, CITIDEL04], and improve algorithms and methods [Giles03], very little has been done to understand the underlying fundamental concepts, their relationships, and the axiomatic rules that govern the DL domain, or in other words, to develop a *theory* of DLs. The necessity of such theory has long being advocated, from the origins of the field, illustrated by Licklider's call for a unified Computer Science(CS)/Library and Information Science(LIS) model [Licklider65], to recent workshops on the future of digital libraries [Larsen04]. The absence of such a theory makes comparison of different DLs architectures and systems extremely hard, promotes ad-hoc development, and impedes interoperability. Its existence may enhance our ability to communicate about and identify new research areas [Sompel03].

In [Gonçalves04], we have presented a partial formal conceptualization of digital libraries by formally defining high-level DL concepts such as digital objects, collections, repositories, services, etc. from basic mathematical concepts such as sets, graphs, functions, sequences, and so forth in a bottom-up manner. However, such conceptualization is incomplete to define a DL theory. A theory should make explicit the implicit relationships that exist among the defined formal DL concepts as well as provide a set of rules or axioms that precisely define and constrain the semantics of concepts and relationships in the theory. This type of formal conceptualization has elsewhere been called an *ontology* [Doan03]. Ontologies specify relevant concepts – the types of things and their properties –

and the semantic relationships that exist between those concepts in a particular domain. Formal specifications use a language with a mathematically well-defined syntax and semantics to describe such concepts, properties, and relationships precisely.

In this work, we define a formal, axiomatic ontology for digital libraries (DLs) that can serve as a frame of reference for the discussion of essential concepts of DL design. The process of construction of such an ontology was guided by 5S, a formal model for digital libraries. We use the resulting ontology to provide answers for questions such as: 1) how should DL services be built from the repository, its collections and metadata catalogs, and from the relationships among different societies that participate in the DL?; 2) which are the dependencies and consistency rules that should follow in a DL model?; 3) which are the fundamental and elementary DL services and how can services be built/composed from other DL services?.

This paper is organized as follows. Section 2 summarizes our earlier results by giving a formal definition of DLs based on the 5S model. Section 3 builds on the core definitions to create an axiomatic, formal ontology for digital libraries. Sections 4 illustrates the expressiveness of the ontology by applying it to create a taxonomy of DL services and to reason about issues of minimality, extensibility, and composability. Section 5 includes a brief discussion of other practical applications of the ontology. Section 6 concludes the paper with a glimpse of future work.

2. BACKGROUD: THE 5S MODEL FOR DIGITAL LIBRARIES

According to the 5S formal model a digital library is a 10-tuple (Streams, Structs, Sps, Scs, St^2, Coll, Cat, Rep, Serv, Soc) in which [Gonçalves04]:

a) Streams is a set of streams, which are sequences of arbitrary types (e.g., bits, characters, pixels, frames);
b) Structs is a set of structures, which are tuples, (G, ϕ), where $G= (V, E)$ is a directed graph and ϕ: $(V \cup E) \rightarrow L$ is a labeling function;
c) Sps is a set of spaces each of which can be a measurable, measure, probability, topological, metric, or vector space.
d) $Scs = \{sc_1, sc_2, \ldots, sc_d\}$ is a set of scenarios where each $sc_k = <e_{1k}(\{p_{1k}\}), e_{2k}(\{p_{2k}\}), \ldots, e_{d\_kk}(\{p_{d\_kk}\})>$ is a sequence of events that also can have a number of parameters $\{p_{ik}\}$. Events represent changes in computational states; parameters represent specific locations in a state and respective values.
e) St^2 is a set of functions Ψ: $V \times Streams \rightarrow (N \times N)$ that associate nodes of a structure with a pair of natural numbers (a, b) corresponding to a portion of a stream.
f) $Coll = \{C_1, C_2, \ldots, C_f\}$ is a set of DL collections where each DL collection $C_k = \{do_{1k}, do_{2k}, \ldots, do_{f\_kk}\}$ is a set of digital objects. Each digital object $do_k = (h_k, Stm_{1k}, Stt_{2k}, \Omega_k)$ is a tuple where $Stm_{1k} \subseteq Streams$, $Stt_{2k} \subseteq Structs$, $\Omega_k \subseteq St^2$, and h_k is a unique identifier for the object.
g) $Cat = \{DM_{C\_1}, DM_{C\_2}, \ldots, DM_{C\_f}\}$ is a set of metadata catalogs for Coll where each metadata catalog $DM_{C\_k} = \{(h, mss_{hk})\}$, and $mss_{hk} = \{ms_{hk1}, ms_{hk2}, \ldots, ms_{hkn\_hk}\}$ is a set of descriptive metadata specifications. Each descriptive metadata specification ms_{hki} is a structure with atomic values (e.g., numbers, dates, strings) associated with nodes.
h) A repository $Rep = \{(C_i, DM_{C\_i})\}$ (i=1 to f) is a set of pairs (collection, metadata catalog); it is assumed there exists operations to manipulate them (e.g., get, store, delete).
i) $Serv = \{Se_1, Se_2, \ldots, Se_s\}$ is a set of services where each service $Se_k = \{sc_{1k}, .., sc_{s\_kk}\}$ is described by a set of related scenarios.

j) Soc = (C, R) where C is a set of communities and R is a set of relationships among communities. SM = $\{sm_1, sm_2, \ldots, sm_j\}$, and Ac = $\{ac_1, ac_2, \ldots, ac_r\}$ are two such communities where the former is a set of service managers responsible for running DL services and the latter is a set of actors that use those services. Being basically an electronic entity, a member sm_k of SM distinguishes itself from actors by defining or implementing a set of operations $\{op_{1k}, op_{2k}, \ldots, op_{nk}\} \subset sm_k$. Each operation op_{ik} of sm_k is characterized by a triple $(n_{ik}, sig_{ik}, imp_{ik})$, where n_{ik} is the operation's name, sig_{ik} is the operation's signature (which includes the operation's input parameters and output), and imp_{ik} is the operation's implementation. These operations define the capabilities of a service manager sm_k. For example, SearchManager $\supset$ {match(q:query, C:collection)[1]} indicates that a SearchManager defines an operation "match" with two parameters, a query and a collection.

The above definition emphasizes *syntactic* aspects, i.e., how digital library concepts are composed or built from previously defined concepts. In the next section, we will explore *semantic* relations and rules of the DL domain.

3. DEFINING A DL THEORY THROUGH AN ONTOLOGICAL ANALYSIS OF THE 5S MODEL

The crux of our contribution with the 5S model was, departing from abstractions of many DL architectural settings, recognizing and formally defining the essential participating concepts in the digital library discourse. In this section, we extend those results to define a DL ontology by specifying the fundamental collaborations or relations that exist among the DL participants and the sets of rules (or axioms) which constrain the semantics of concepts and relations in the ontology.

We organize the presentation and development of the ontology according to the 5S model. For each 'S', we list the concepts and the relations in which they take part. We consider first intra-model relations, i.e., the relations that occur only among concepts of the same 'S' model, along with the corresponding axioms or rules. Afterwards, relations defined between concepts belonging to different Ss are defined representing inter-dependencies. It should be noticed in the discussion below that some concepts such as digital objects and indexes are inherently "cross-S" concepts, i.e., they are defined in terms of concepts belonging to more than one 'S'. For presentation purposes, we will include those "cross-S" concepts within the discussion about the 'S' in which they share most of their relationships.

More formally, a domain is a set of objects of the same DL type. A DL type is characterized by a definition as in [Gonçalves04]. An object is of a type X if its properties (e.g., internal components, organization) satisfy the definition of X. Examples of DL types include the basic Ss and derivative types such as collections, digital objects, etc. An ontological concept is a domain. For example, the statement x $\in$ **Digital Object** says that x is a digital object as defined in [Gonçalves04] and therefore describes x by the ontological concept Digital Object. An n-ary relation is a subset of the Cartesian product $C_1 \times C_2 \ldots \times C_n$ of the domains defined by the respective DL concepts. Let $R \subset A \times B$ be a relation. Then $R^{-1} = \{(b, a) | (a, b) \in R\} \subset B \times A$ is called the *inverse relation* of *R*. A predicate is a function from a Cartesian product to the Boolean values ***true*** or ***false***. A predicate

[1] To simplify notation, we will represent a operation $op_x = (n_x, sig_x, imp_x)$ by $n_x(\{p_{xk}\})$ where $\{p_{xk}\}$ is the set of input parameters of op_x. The output parameters and implementation can be added when a more full description of the operation is required.

p(x) built over a relation among concepts is true if x is a member of the relation, false otherwise. We now proceed to define our meaning of a DL ontology.

Def: An *ontology* is a tuple Ω = (Ontol_Concepts, Ontol_Rels) where:

1. Ontol_Concepts is a family of ontological concepts,
2. Ontol_Rels is a family of relations.

Relations in Ontol_Rels may be operationally realized by one or more rules (e.g., first-order logic axioms) which intentionally specify or constrain which elements of a concept can participate in a relation. Ontol_Rules is a family of rules of a particular ontology.

For notational purposes we will use **bold** to designate ontological concepts (or simply, concepts) and *italics* to define the corresponding predicate. We will use the *dot* "." notation to denote components of the definition of concepts, for example "x.h" specifies the handle of a digital object, y.Img specifies the image (or range) of events of scenario y, and z.op specifies the set of operations of Service Manager z. We also may refer to a component of a tuple-oriented concept by its position in the tuple, for example, z(2) specifies the set of descriptive metadata specifications of a member of a catalog. Finally, we will represent a relation $R \subset A \times B$ by $A\ R\ B$. The notation for 3-tuple relations will use similar variants, depending on the semantics of the relation.

Below we proceed to define the relations and rules of our DL ontology. The relations were defined by carefully analyzing all possible pairs of associations among concepts within the same and between Ss, and contextual information necessary to define some of these relations.

3.1 Intra-Model relationships

Streams

- Concepts: {**text**, **image**, **video**, **audio**}
- Relations:
  - contains $\subset$ **video** $\times$ **image** $\cup$ **video** $\times$ **audio**
    Streams define the basic content types over which digital objects are built, the latter being the ultimate carriers of the information in the DL. However some complex types of streams (e.g., video) may themselves be associated with simpler types of streams (e.g., images, audio). This relation indicates that a video contains a image as one of its frames, or contains a specific audio recording.

Structures

- Concepts: {**do**, **ms**, **C**, $\mathbf{DM_C}$, **Rep**}. Key: do = digital object; ms = descriptive metadata specification; mss = set of descriptive metadata specifications; C = collection, DM_C = metadata catalog for collection C, Rep = repository.
- Relations:
  - is_version_of $\subset$ **do** $\times$ **do**
    Different *manifestations* of a digital object are versions, which normally differ structurally or in terms of their content (e.g., format, encoding, etc.). This relation indicates that a digital object is a version of another digital object. Conceptually a digital object *x* is a slightly different version of digital object *y* in terms of their streams or structures. Note also that since handles are used as identifiers of digital objects they should be globally unique, so no two digital objects, version or not, share the same handle.

Rules. 1. Digital object handles are unique. 2. *x is_version_of y* for two digital objects x and y if they differ in the handle component and at least one other component, but share at least one other of their components (e.g., they have the same set of streams, set of structures, or set of structured_streams).

Symbolic rules. 1. $\forall x, y\ (do(x) \wedge do(y) \wedge (\ x.h_x = y.h_y\ ) \Rightarrow x = y)$;

2. $\forall x,y\ (x \text{ is\_version } y \Leftrightarrow do(x) \wedge do(y) \wedge (x.h \neq y.h) \wedge (\ (x.Stt \neq y.Stt) \vee (x.Stm \neq y.Stm) \vee (x.\Omega \neq y.\ \Omega)) \wedge ((x.Stt = y.Stt) \vee (x.Stm = y.Stm) \vee (x.\Omega = y.\ \Omega)))$.

- belongs_to $\subset \mathbf{ms} \times \mathbf{DM_C} \cup \mathbf{mss} \times \mathbf{DM_C}$

  Digital objects can belong to many different collections. Similarly, descriptive metadata specifications can belong to many catalogs. This relation makes the latter relationship explicit.

  Rule. x belongs_to y indicates that a metadata specification *x* is used to define an element of the metadata catalog *y*.

  Symbolic Rule. $\forall x, y\ (x \text{ belongs\_to } y \Leftrightarrow (ms(x) \wedge DM_C(y) \wedge\ (\exists z \in y: x \in z(2))) \vee (mss(x) \wedge DM_C(y) \wedge\ (\exists z \in y: x = z(2)))$

- part_of $\subset \mathbf{C} \times \mathbf{C} \cup \mathbf{DM_C} \times \mathbf{DM_C}$

  Many DL collections and metadata catalogs are built by aggregating smaller subcollections/subcatalogs. One good example is the National Science Digital Library (NSDL) union catalog which is basically an amalgamation of the metadata catalogs of all the participant projects.

  Rule. x part_of y indicates that collection *x* is a subset of collection *y* or metadata catalog *x* is a subset of metadata catalog *y*.

  Symbolic Rule. $\forall x, y\ (x \text{ part\_of } y \Leftrightarrow ((C(x) \wedge C(y) \wedge x \subseteq y) \vee (DM_C(x) \wedge DM_C(y) \wedge\ x \subseteq y)))$

- describes $\subset \mathbf{mss} \times \mathbf{do} \cup \mathbf{DM_C} \times \mathbf{C}$;

  A digital object may potentially have many descriptive metadata specifications, for example, in standard formats (e.g., Dublin Core, MARC) for sharing purposes, or based on more detailed, community-oriented specific formats. Also qualitative properties of metadata catalogs such as completeness and consistency can be defined in terms of this relationship.

  Rules. 1. *x describes y* indicates that a set of descriptive metadata specifications *x*, belonging to some catalog *q* for collection *p*, describes the content of a digital object *y*, which belongs to that collection *p*. The set of metadata specifications *x* can describe only one digital object, therefore the *describes* relation between sets of metadata specifications and digital objects is a *function*.

  Symbolic rules. 1.1. $\forall x, y\ (x \text{ describes } y \wedge\ mss(x)\ \wedge\ do(y) \Rightarrow\ \exists\ p, q, h:\ C(p) \wedge DM_C(q) \wedge ((h, x) \in q)\ \wedge\ (y \in p) \wedge (y(1) = h))$; *1.2.* $\forall x, y, z\ (x \text{ describes } y \wedge\ x \text{ describes } z \wedge mss(x)\ \wedge\ do(y) \wedge\ do(z) \Rightarrow\ y = z\ )$.

  Rules. 2. The relation q *describes* p, $(q, p) \in DM_C \times C$ indicates that a metadata catalog *q* describes a specific collection *p*. A complete catalog has at least one set of metadata specifications for each digital object in the collection it describes. In a consistent catalog, each set of metadata specifications describes (exactly) one digital object in the related collection. In other words, a complete *describes* relationship between a metadata catalog, and a collection defines a surjective partial function, and a consistent relationship defines a total function. Also note that it is very common that different metadata specifications (e.g., a Dublin Core and a MARC version) may describe the same digital object, so in most cases the *describes* function is not injective.

  Symbolic Rules. 2.1 Catalog/Collection Consistency: $\forall x, y, z\ (C(y) \wedge DM_c(x)\ \wedge mss(z) \wedge x \text{ describes } y \wedge z \text{ belongs\_to } x\ \Rightarrow\ \exists\ p \in y:\ z \text{ describes } p)$;

2.2. Catalog/Collection Completeness: $\forall$x, y, z (*C*(y) $\wedge$ DM_c(x) $\wedge$ do(z) $\wedge$ *x describes y* $\wedge$ *z* $\in$ *y* $\Rightarrow \exists$ m: (*mss*(m) $\wedge$ m *belongs*_to x $\wedge$ *m describes* z))

- stores $\subset$ **Rep** $\times$ **C** $\times$ $\mathbf{DM_c}$
  Captures the fact that a pair (collection, metadata catalog) resides in a physical repository.
  Rule. r stores (x,y) indicates that a repository *r* stores a pair with a collection *x* and the metadata catalog *y* which describes *x*.
  Symbolic Rule. $\forall$x, y, z *(*x *stores (*y,z) $\Rightarrow$ *Rep*(x) $\wedge$ *C*(y) $\wedge$ DM_C(z) $\wedge$ *z describes y*)

Spaces

- Concepts: {**Vec**, **Pr**, **Measurable**, **Measure**, **Metric**, **Top**}. Key: Vec= vector space; Pr = probability space; Measurable = measurable space; Measure = measure space; Metric = metric space; Top = topological space.
- Relations
  - is_a $\subset$ **Measure** $\times$ **Measurable** $\cup$ **Pr** $\times$ **Measure** $\cup$ **Metric** $\times$ **Top** $\cup$ **Vec** $\times$ **Top**.
    x *is_a* y indicates that a space *x* has all the properties/constraints/operations associated with the definition of the space *y* and may include additional properties / constraints / operations. The *is_a* relationship is reflexive, transitive, and anti-symmetric, therefore mathematical spaces that participate in this relation define a partial order.

Scenarios

- Concepts: {**Se**, **Sc**, **e**}; Key: Se = service; Sc = scenario; e = event.
- Relations:
  - contains $\subset$ **Sc** $\times$ **e**
    Make explicit the relationship that an event belongs to a sequence of some scenario of use of a DL service.
    Rule. sc_k *contains* e_{k_j} indicates that an event $e_{k\_j} = sc_k(j)$ is a element of the image/range of a scenario sc_k, for some j belonging to the domain $\{1,2, \ldots, d_k\}$ of sc_k. Recall that scenario is a sequence of events, i.e., it is a function from natural numbers to a set of events.
    Symbolic Rule. $\forall$ x, y (*x contains y* $\wedge$ *Sc*(x) $\wedge$ *e*(y) $\Rightarrow \exists$j: (j $\in$ x.Dom $\wedge$ *y* = *x*(j)))
  - precedes $\subset$ **e** $\times$ **e** $\times$ **Sc**; happens_before $\subset$ **e** $\times$ **e** $\times$ **Sc**
    A scenario of use represents a temporal sequence of events that a user (or another service manager) engages in while interacting with a DL service. The temporal ordering of events is captured by these relations.
    Rule 1. x $precedes_z$ y indicates that an event *x* occurs immediately before *y* in the context of scenario *z.* x $happens\_before_z$ y indicates that both *x* and *y* are elements of sequence *z*, and x happens some time before y, i.e., the sequence value of x is smaller than the sequence value of y.
    Symbolic Rule 1. $\forall$ x, y, z (x $precedes_z$ y $\wedge$ *e*(x) $\wedge$ *e*(y) $\wedge$ *Sc*(z) $\Rightarrow \exists$ i, j: (z *contains* x $\wedge$ z *contains* y $\wedge$ x = z(i) $\wedge$ y=z(j) $\wedge$ i + 1 = j))
    Symbolic Rule 2. $\forall$ x, y, z (x $happens\_before_z$ y $\wedge$ *e*(x) $\wedge$ *e*(y) $\wedge$ *Sc*(z) $\Rightarrow \exists$ i, j: (z *contains* x $\wedge$ z *contains* y $\wedge$ x = z(i) $\wedge$ y=z(j) $\wedge$ i < j))
  - includes $\subset$ **Se** $\times$ **Se** $\cup$ **Sc** $\times$ **Sc**; extends $\subset$ **Se** $\times$ **Se** $\cup$ **Sc** $\times$ **Sc**
    *S*ervices exposed by a DL can be classified either as elementary or composite. Elementary services provide the basic infrastructure for the DL. Examples include collecting, indexing, rating, and linking. Composite services can be composed of other services (elementary or composed) by reusing or extending them. For example, searching and browsing services use indexing and linking services, a relevance feedback service extends the capabilities of a basic searching service, and a lesson plan building service may use already existing searching, browsing, and binding services to find and organize relevant resources. The

problem of composability of services has gained considerable attention recently, mainly in the Web Services community [Benatallah03, Curbera02]. However, DL services are restricted to certain specific types with constrained inputs and outputs, therefore making the problem more manageable and amenable to domain specific techniques. Since DL services are described by correlated, generally slightly variant scenarios of use, similar notions can be applied to those scenarios. For example, consider scenario sc_1= <search(q,C), results({(do_i,w_i)})> for a search service where *q* represents a query, *C* a DL collection, *do* a digital object, and *w* a weight. The scenario sc_2 = <search(q,C), results({(do_i,w_i)}), relevant_docs{do_j}, expanded_query(eq,{do_j}), search(eq,C), results{(do_k,w_k)}> is an extension of sc_1 representing a relevance feedback search.

Rule 1. Let sc_1= <e_1,e_2,…,e_n> be a scenario. A scenario sc_2 = <e_{2x},…,e_{2y}> *includes* scenario sc_1 if it contains all events of sc_1 in the same order they appear, i.e., if event e_i precedes event e_j in sc_1, the same relationship holds in scenario sc_2, or, in other words, sc_2 *includes* sc_1 only if sc_1 is a *consecutive subsequence* of sc_2.

Symbolic Rule 1. $\forall$x, y (x *includes* y $\wedge$ *Sc*(x) $\wedge$ *Sc*(y) $\Rightarrow$ ($\forall$z: *e*(z) $\wedge$ y contains z $\Rightarrow$ x contains z) $\wedge$ ($\forall$p, q: *e*(p) $\wedge$ *e*(q) $\wedge$ p $precedes_y$ q $\Rightarrow$ p $precedes_x$ q))

Rule 2. A service Se_1 includes service Se_2 if it includes all its scenarios, i.e., if $Se_2 \subseteq Se_1$.

Symbolic Rule 2. $\forall$x, y (x *includes* y $\wedge$ *Se*(x) $\wedge$ *Se*(y) $\Rightarrow$ y $\subseteq$ x).

Rule 3. Let sc_1= <e_1,e_2,…e_n> be a scenario. A scenario sc_2 = <e_{2x},…,e_{2y}> *extends* scenario sc_1 if it contains all events of sc_1 in the same relative order they appear, i.e., if event e_i happens before event e_j in sc_1, the same relationship holds in scenario sc_2, or, in other words, sc_2 *extends* sc_1 only if sc_1 is a *subsequence* of sc_2.

Symbolic Rule 3. $\forall$x, y (x extends y $\wedge$ *Sc*(x) $\wedge$ *Sc*(y) $\Rightarrow$ ($\forall$z: *e*(z) $\wedge$ y contains z $\Rightarrow$ x contains z) $\wedge$ ($\forall$p, q: *e*(p) $\wedge$ *e*(q) $\wedge$ p $happens\_before_y$ q $\Rightarrow$ p $happens\_before_x$ q))

Rule 4. A service Se_2 *extends* service Se_1 if Se_2 includes all of Se_1's scenarios, and Se_2 has new scenarios, i.e., there exist scenarios in Se_2 which are not elements of Se_1, or there exist scenarios of Se_2 which extend scenarios of Se_1.

Symbolic Rule 4. $\forall$x, y (x *extends* y $\wedge$ *Se*(x) $\wedge$ *Se*(y) $\Rightarrow$ y $\subseteq$ x $\wedge$ (x$\neq$ y $\vee$ $\exists$p, q: *Sc*(p) $\wedge$ *Sc*(q) $\wedge$ p $\in$ x $\wedge$ q $\in$ y $\wedge$ p extends q))

Societies

- Concepts: {**SM**, **Ac**, **op**}; Key: SM = service Manager; Ac = actor; op = operation.
- Relations
  - redefines $\subset$ **op** $\times$ **op**

    A common reason to redefine or override an operation is to provide more specific functionality for a service manager which inherits an operation from another service manager (see below).

    Rule. A redefined operation has the same name, and often (but not necessarily) the same signature, but a different implementation.

    Symbolic Rule. $\forall$x, y (x *redefines* y $\wedge$ *op*(x) $\wedge$ *op*(y) $\Leftrightarrow$ x.n = y.n $\wedge$ x.imp $\neq$ y. imp)
  - includes $\subset$ **SM** $\times$ **SM**; inherits_from $\subset$ **SM** $\times$ **SM**

    Aggregation and generalization are two special types of relationships between service managers that foster reusability and extensibility. Aggregation, captured in the *includes* relation, models a "whole/part" relationship in which one manager as a whole has other managers as parts, or, in other words, if service manager x includes service manager y, it implies that y is required in order to use service manager x. Generalization, captured by the *inherits_from* relation, means that a manager has all the capabilities defined by another manager, potentially has additional ones, and can redefine others (polymorphism). For

example, LessonPlanBuilding *includes* Binding Manager indicates that a service manager LessonPlanBuilding includes operations of a Binding Manager. Similarly, RelevanceFeedbackSearch Manager *inherits_from* Search Manager indicates that a RelevanceFeedbackSearch Manager has the same capabilities as the Search Manager as well as additional ones (e.g., for query expansion).

Rule 1. x includes y indicates that a service manager *x* has all operations defined in service manager *y* plus others not defined in y.

Symbolic Rule 1 . $\forall$x, y (x *includes* y $\wedge$ *SM*(x) $\wedge$ *SM*(y) $\Rightarrow$ y.op $\subseteq$ x.op $\wedge$ y.op $\neq$ x.op)

Rule 2. x *inherits_from* y indicates that a service manager *x* has all operations from the service manager *y* and defines additional operations, or x redefines some operations of y.

Symbolic Rule 2. $\forall$x, y (x *inherits_from* y $\wedge$ *SM*(x) $\wedge$ *SM*(y) $\Rightarrow$ (y.op $\subseteq$ x.op $\wedge$ y.op $\neq$ x.op) $\vee$ ($\forall$z $\in$ y.op - x.op: $\exists$w $\in$ x.op: w *redefines* z))

- invokes: **op** $\times$ **op**

It is generally useful to specify dependencies between operations when discussing issues of extensibility and reusability. For example, search_similar(do) *invokes* match(q:query, C:collection) indicates that a search_similar operation invokes a match operation, defined in a Service Manager *x* or in another manager that *x* inherits from or includes.

Rule. 1. *f invokes g* indicates that operation *f* may invoke operation *g*, namely, that within the body of operation *f* there is an expression whose evaluation invokes *g* (*g* is a subfunction of *f*). The operation *f* defined in a service manager *x* may only invoke an operation *g*, if *g* also is defined in *x* or in another manager that *x* includes or inherits from.

Symbolic Rule. 1. $\forall$f, g (f *invokes* g $\wedge$ *op*(f) $\wedge$ *op*(g) $\wedge$ ($\exists$p: SM(p) $\wedge$ f $\in$ p $\wedge$ g $\in$ p) $\Leftrightarrow$ g is a subfunction of f $\Leftrightarrow$ $\exists$ functions r, s: g = r $\circ$ f $\circ$ s

- association: **Ac** $\times$ **Label** $\times$ **Ac**

A generic relationship between actors without a pre-defined semantics, this one captures generic societal relationships between communities of actors. For example, the relation (Professor, "teaches", Learner) is self-explanatory.

3.2 Inter-Model Relations

In this section, we identify several relations that cross the borders of Ss. Our emphasis here is on the relationships between the dynamic aspects of the DL, characterized by societies and scenarios, and the more "static" aspects of the DL, characterized by concepts in the other Ss. We also further explore other relationships among the three static Ss.

Scenarios and Societies

- Relations:
  - executes $\subset$ **e** $\times$ **<op>**

    The changes of computational states which are triggered by events in a scenario are computationally realized by invoking operations defined on service managers. Let <op> be the set of finite sequences from op. e_k *executes* $<op>_j$ indicates that the list of operations $<op>_j = <op_{1j}, op_{2j}, \ldots, op_{n\_jj}>$ is executed as the result of the occurrence of event e_k. Also if P_k is the set of event parameters of e_k and P_j is the union of all parameters of all operations in $<op_j>$, $P_j \subseteq P_k$. For example, search(q,C) *executes* match(q,C) states that an event *search* executes an operation *match* (probably defined in a Searching Manager) between a query *q* and the set of digital objects in the collection *C*.
  - recipient $\subset$ {**SM** $\cup$ **Ac**} $\times$ **e**

    In a scenario it is normally useful to identify the societal members that receive events for the purpose of checking consistency, security, etc. For example, the following two

relationships specify recipients of events in a simple searching scenario: Search Manager *recipient* search(q,C); Researcher *recipient* results({(do_i,w_i)}).

Rule. recipient $\subset \{\mathbf{SM} \cup \mathbf{Ac}\} \times \mathbf{e}$ indicates that a specific service manager or actor is the receiver of an event in a scenario. Any actor can be the receiver of any event. If the event has an *execute* relationship with some operation, the receiver must be a Service manager which should have this operation.

Symbolic Rules. $\forall$x, y, z (x *recipient* y $\wedge$ y *executes* z $\wedge$ *SM* (x) $\wedge$ *e*(y) $\Rightarrow \forall$w $\in$ z.Img: w $\in$ x.op).

- participates_in $\subset \{\mathbf{SM} \cup \mathbf{Ac}\} \times \mathbf{Sc}$

This relation makes explicit the societal entities interacting in a scenario.

Rule. Indicates that a service manager or actor *x* participates in a specific scenario *y* of a DL service by being a recipient of an event *z* of scenario *y*.

Symbolic Rule. $\forall$ x, y (x *participates_in* y $\wedge$ (*SM*(x) $\vee$ *Ac*(x)) $\wedge$ *Sc*(y) $\Rightarrow$ $\exists$z: *e*(z) $\wedge$ y *contains* z $\wedge$ x *recipient* z))

For Service Managers, a consequence of the defined relations is that only operations defined in the participating managers should be associated with events of the scenarios in the service. This gives rise to the following consistency rule between a scenario and a society model.

Scenarios-Society Consistency Rule. A scenario *x* is consistent with regards to a set of service managers Y if each operation executed by each event in the scenario is defined in some service manager y $\in$ Y.

Symbolic Rule. $\forall$x, y, z, w (*Sc*(x) $\wedge$ *e*(y) $\wedge$ *op*(z) $\wedge$ x *contains* y $\wedge$ y *executes* *w* $\wedge$ z $\in$ w.Img $\Rightarrow \exists$p: *SM*(p) $\wedge$ p *partipates_in* x $\wedge$ z $\in$ p.op).

- uses $\subset \mathbf{Ac} \times \mathbf{Se}$

In many real DL settings it is useful to specify that only specific kinds of Actors may be allowed to use certain services. For example, while a researcher should be allowed to use all information seeking services, services such as "lesson plan building" and "dissertation submission approval" should be used only by teachers and archivists, respectively.

Rule. Indicates that an Actor is allowed to use a specific service by participating in some of the services' scenarios.

Rule. $\forall$ x, y (x *uses* y $\wedge$ *Se*(y) $\wedge$ *Ac*(x) $\Rightarrow$ $\exists$z: *Sc*(z) $\wedge$ *SM*(w) $\wedge$ z $\in$ y $\wedge$ x *participates_in* z)

- runs $\subset \mathbf{SM} \times \mathbf{Se}$

Rule. Service Manager *x* runs service *y* if all operations executed in all scenarios of y are defined on *x* or in managers that *x* includes or inherits from.

Symbolic Rule. $\forall$x, y (x runs y $\wedge$ *SM*(x) $\wedge$ *Se*(y) $\Rightarrow \forall$ z, p, q, r: (*Sc*(z) $\wedge$ *e*(p) $\wedge$ *op*(r) $\wedge$ z $\in$ y $\wedge$ z contains p $\wedge$ p *executes* q $\wedge$ r $\in$ q.Img $\Rightarrow$ r $\in$ x.op)

Structures, Streams and Spaces

- Relations:
  - $I_C \subset H \times \theta \times \{Vec \cup Pr \cup Metric\}$

    Let C $\in$ Coll be a collection, H be the set of all handles of digital objects in C, $\theta \subset \cup$do(4), do $\in$ C, be a set of all triples (node, stream, interval) associated to digital objects in the collection, where interval is a pair of natural numbers (a,b) corresponding to a portion of the stream (or a substream). An index I_C is a function that maps specific substreams associated with nodes of specific digital object structures to elements of a vector, probability, or metric space. Normally, the elements of these spaces are built by extracting *features* (e.g., text terms, histograms) from the respective substream. In the case of a probability space, the

elements of $H \times \theta$ are mapped to a finite set with a discrete measure assigning positive probabilities to elements of that set.

Symbolic Rule. $\forall x\ (x \in I_{C\_i} \Rightarrow \exists y, z: do(y) \in C_i \wedge x(1)=y(1) \wedge z = x(2) \wedge z \in y(4))$.

$\forall C\ (Coll(C) \wedge (h, s, v) \in I_C \wedge (h, s, v') \in I_C \Rightarrow v = v')$.

Scenarios and (Streams, Structures, Spaces)

- Relations:
  - employs $\subset$ $\mathbf{Se} \times \mathbf{S^3}$; produces : $\mathbf{Se} \times \mathbf{S^3}$

Let $\mathbf{S^3}$ = **Streams** $\cup$ **Strutures** $\cup$ **Spaces** be the union of all concepts of the respective Ss. DL services manipulate, transform, and return instances of the concept types defined in S^3. For example, the notion of distance (as defined by a metric space) or probability (as defined by a probability space) are essential to services which need to compute a similarity measure between objects in the DL or between a patron's intrinsically vague information need and objects in the DL. Examples of services that normally employ spaces to compute these measures include searching, filtering, recommending, visualizing, classifying, and clustering. Also, services exist that transform DL objects (digital objects, metadata specifications, structures, streams) into different types of spaces for many purposes. Examples include services such as indexing, which transforms structured streams into elements of a vector or probability space, rendering or visualizing, which normally takes collections and transforms into a 2D/3D-metric space, or customizing, that normally transforms a space (e.g., a user interface (UI) or a distance function) into another personalized space (e.g., a customized UI or a personalized distance function [Fan]).

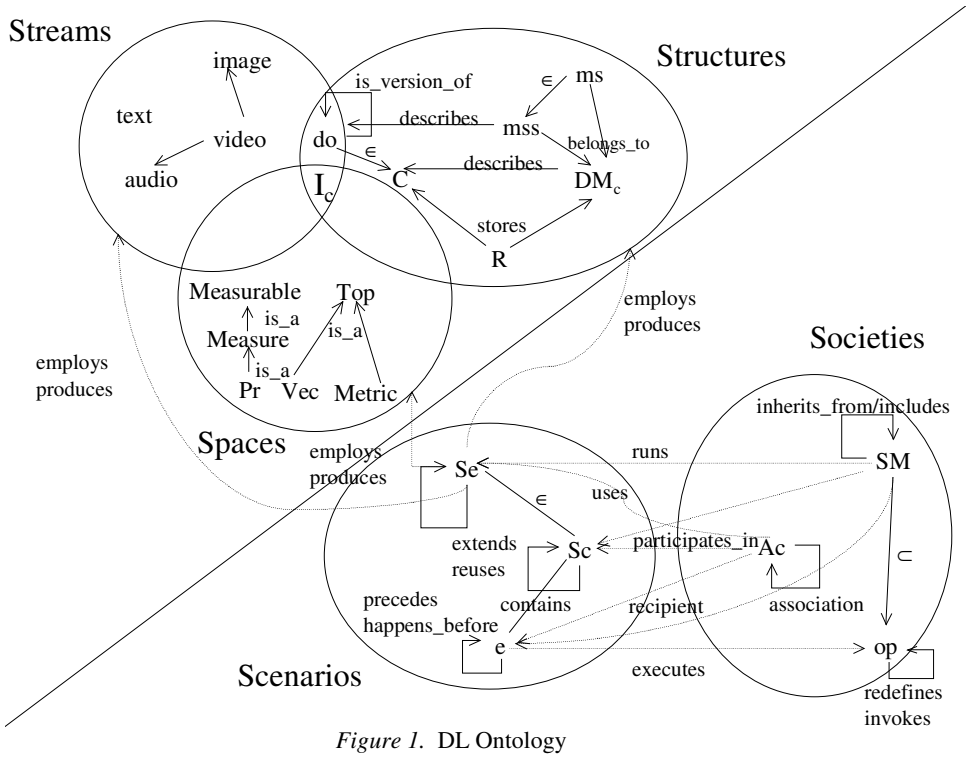

Figure 1. DL Ontology

Due to the complexity and number of possible instances of this relation, we will postpone the discussion to the next section, where we will further characterize the relationships between services and the other "static" Ss by making explicit *employed* inputs and *produced* outputs of events in these services.

The resulting ontology is graphically depicted as shown in Figure 1. Each S model is represented as a circle containing the respective concepts. Normal lines represent inter-model relations while dotted lines correspond to inter-model relationships. Arrows linked to a whole indicate that the relationship can exist among all concepts in a S.

4. A TAXONOMY OF DL SERVICES

Our objective in this section is to further explore some of the most important types of relations in the DL ontology, namely the "employs" and "produces" between Services and the other static "Ss" and the "extends" and "includes" relations among services. More specifically, in this section we want to answer questions such as: 1) Which DL elements are employed or produced by the different DL services? 2) Which are the fundamental DL services? 2) Which kinds of service composition are possible or valid? 3) Which DL services are elementary or composite? These relationships can give insights into how DL services can be built from other DL components such as repositories and societal interactions, as well as be composed with other services by extension or reuse.

Table 1 shows a set of activities or services derived from an expanded list in a DL taxonomy presented in [Gonçalves04]. In the table each service is characterized by parameters (input, output) of the initial and final events of the scenarios that compose those services. All other previous definitions and keys apply here. Those definitions are complemented with the following ones.

Def. 1. A field *fi* is a label associated with a node of a structural or descriptive metadata specification.

Def. 2. A query *q* is the representation of user interest or information need. The exact format of a query is left unspecified here since it is system-dependent.

Def. 3. An annotation ann_{ik} is a descriptive metadata specification that exists only in reference to a digital object do_i.

Def. 4. Hyptxt is an hypertext (see formal definition in [Gonçalves04]); anchor is a node of a hypertext.

Def. 5. A personal binder bi_{u_i} is a subset of some collection $C_k \in$ Coll for an actor *u* of Soc(1).

Def. 6. A log_entry is a descriptive metadata specification about an event of a scenario.

Def. 7. $tfr \subset S^3 \times$ Spaces is a function that transforms any element of a concept in S^3 into a space. Transformers = $\{tfr_1, tfr_2, \ldots, tfr_n\}$ is a set of such functions.

Def. 8. Let $\{do_i\} = \{do_{i1}, do_{i2}, \ldots, do_{in}\}$ be a set of digital objects and $Ct = \{c_1, c_2, \ldots, c_m\}$ be a set of labels for categories. A classifier $class_{Ct}$: $\{do_i\} \rightarrow 2^{Ct}$ is a function that maps a digital object to a set of categories.

Def. 9. A cluster $clu_k = \{do_{1k}, do_{2k}, \ldots, do_{nk}\}$ is a subset of a set of digital objects.

Table 2 shows an organized taxonomy of DL services featured in Table 1, derived from a deep analysis of the entries in that table. The key aspects of defining such a taxonomy were: 1) to separate services dealing with basic concepts such as collections and catalogs from those dealing with higher level societal requirements; and 2) to define the responsibilities and interrelationships among those services and how they collaborate. In this taxonomy, we define a **fundamental** service (denoted by **bold**) as either: 1) one that helps to create elements of basic concepts belonging to our minimal definition of a DL, such as digital objects, metadata specifications, collections, and catalogs; 2) one that belongs to the minimal set of DL services (e.g., Searching and Browsing) proposed in [Gonçalves]; or 3) one that supports the former services in terms of extension or reuse. Similarly a <u>composite</u> (denoted by <u>underlining</u>) DL service is one that takes input from some other service; otherwise the service is called elementary.

Table 1. DL services, and their inputs, outputs

Service	**User input**	**Other Service Input**	**Output**
Acquiring	$\{do_i\}$	C_i	C_j
Annotating	do_i, ann_{i_k}	$(h_i, \{ann_{i\_m}\})$	$(h_i, \{ann_{i\_(m+k)}\})$
Binding	$\{do_i\}$, b_{u_k}	$\{do_j\}$	$b_{u\_(k+i)}$
Browsing	Anchor	Hyptxt	$\{do_i\}$ [2]
Cataloging	do_i, ms_{i_k}	(h_i, mss_{i_m})	$(h_i, mss_{i\_(m+k)})$
Classifying	$\{do_i\}$	$class_{Ct}$, Ct	$\{(do_i, \{c_{k\_i}\})\}$
Clustering	$\{do_i\}$	none	$\{clu_{k\_i}\}$
Conserving	C_i	none	C_k
Copying/Replicating	C_i	none	C_k
Crawling (focused)	C_i	none	C_k
Customizing (interface)	ac_i , trf_k	sp_j	sp_{j_k}
Digitizing	None	none	do_i
Disseminating	$\{h_x, \ldots, h_y\}$	none	$\{do_x,...,do_y\}$
Evaluating	$\{do_i\}$	none	$\{(do_i, w_i)\}$
Expanding (query)	$\{do_i\}$	I_{C_i}, q_i	q_j
Extracting (structure)	stm_i	none	Ψ_{ik}
Filtering	q, $\{do_i\}$ $\{do_i\}$, c_k	I_{C_i} $class_{Ct}$, Ct	$\{do_k\}$ $\{do_k\}$
Harvesting (metadata)	$\{h_i\}$	none	$\{(h_i, mss_{i\_m})\}$
Indexing	C_i	none	I_{C_i}
Linking	C_i	none	$Hyptxt_{ik}$
Logging	none	$e_i(\{p_i\})$	log_entry_j
Measuring	C_i	sp_j	$\{(do_i, w_i)\}$
Rating	do_i ,ac_i	none	$\{(do_i,ac_i,r_k)\}$
Recommending	ac_i, C_k	$\{(do_i,ac_i,r_i)\}$ $\{(do_i, w_i)\}$	$\{do_j\}$
Requesting	h_i	none	do_i
Reviewing (peer)	$\{do_i\}$	none	$\{(do_i, w_i)\}$
Searching	q, C_i	I_{C_i}	$\{do_k\}$
Submitting	do_k,C_i do_i, ms_{ik}, DM_{C_j} op_k, sm_i	none none none	C_j DM_{Ct} sm_j
Surveying	$\{do_i\}$	none	none
Training (classifier)	$\{(do_i, \{c_k\}_i)\}$	none	$class_{Ct}$
Translating (format)	do_i	none	do_j
Translating (language)	do_i	none	do_j
Visualizing	$\{do_i\}$	tfr_k	sp_{ik}

Table 2. A taxonomy of DL services/activities.

Infrastructure Services			**Information Satisfaction Services**
Repository-Building		*Add_ Value*	
Creational	Preservational		

[2] The definition of a browsing service in [Gonçalves04] includes a number of different outputs for browsing events over a hypertext, including internal structures of digital objects and their structured streams. For the sake of simplicity in this discussion we only will consider browsing services whose events' output include only a collection of digital objects.

Acquiring **Authoring** **Cataloging** Crawling (focused) **Describing** Digitizing Harvesting Purchasing **Submitting**	Conserving Converting Copying/Replicating Emulating Renewing Translating (format)	Annotating Classifying Clustering Customizing Evaluating Extracting **Indexing** **Linking** Logging Measuring Rating Reviewing (peer) Surveying Training (classifier) Translating (language/format)	Binding **Browsing** Disseminating Expanding (query) Filtering Recommending **Requesting** **Searching** Visualizing

Another important aspect of the taxonomy which helps to establish connections between these two types of services in terms of reuse or extension relations is the realization that the output of infrastructure services is normally the input of some of the information satisfaction services. Many examples are illustrated in Figure 3 which focuses on **fundamental** services. Acquiring, authoring, cataloguing, describing, and submitting are fundamental infrastructure services which build catalogs and collections. An indexing service takes a collection and a catalog and produces an index used by both searching and browsing services. Linking services work together with indexes to produce a hypertext used for a browsing service to allow criteria-based, ordered, and hierarchical navigation of possibly large collections.

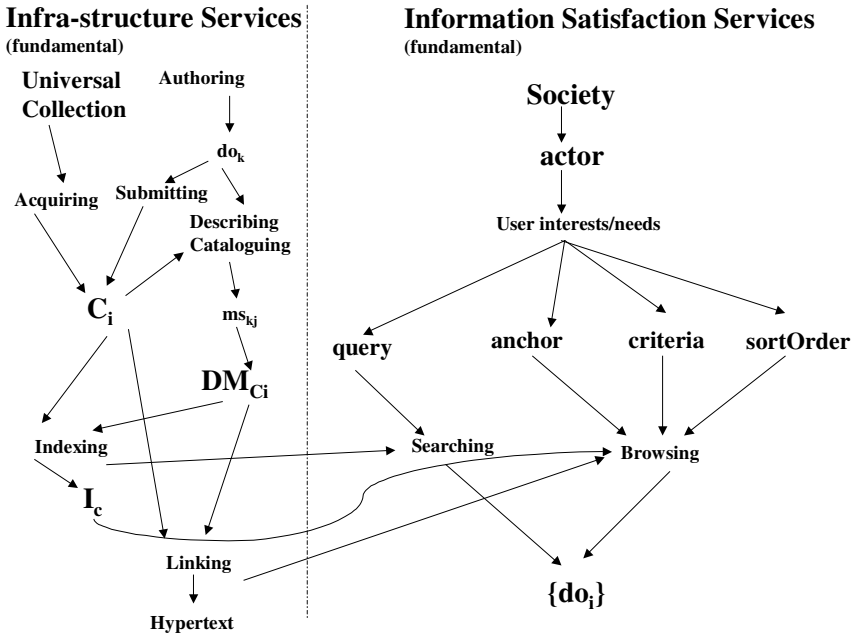

Figure 3. Instantiations of the "Services Definition" model showing inputs and outputs of several examples of infrastructure and information satisfaction DL services.

Figure 4 depicts reuse relations between fundamental and composite information services and between the latter and some non-fundamental, add-value services. Common to the composite

information satisfaction services depicted in the figure is the fact that all of them take a set of digital objects or a collection as input. Recommending takes a user representation (e.g., a expression of interest), and either the output of a Rating or Reviewing service, and produces a subset of the original set of digital objects. Filtering takes a user profile, or a classifier produced by a Training service, and also outputs a subset of a set of digital objects. Similarly, binding takes the output produced by searching or browsing services and returns a subset of it. Visualizing produces a space out of a digital object set/collection while Expanding a query takes the original query submitted to a Searching service and a subset of the response set (i.e., relevant and/or non-relevant documents) and returns a modified query.

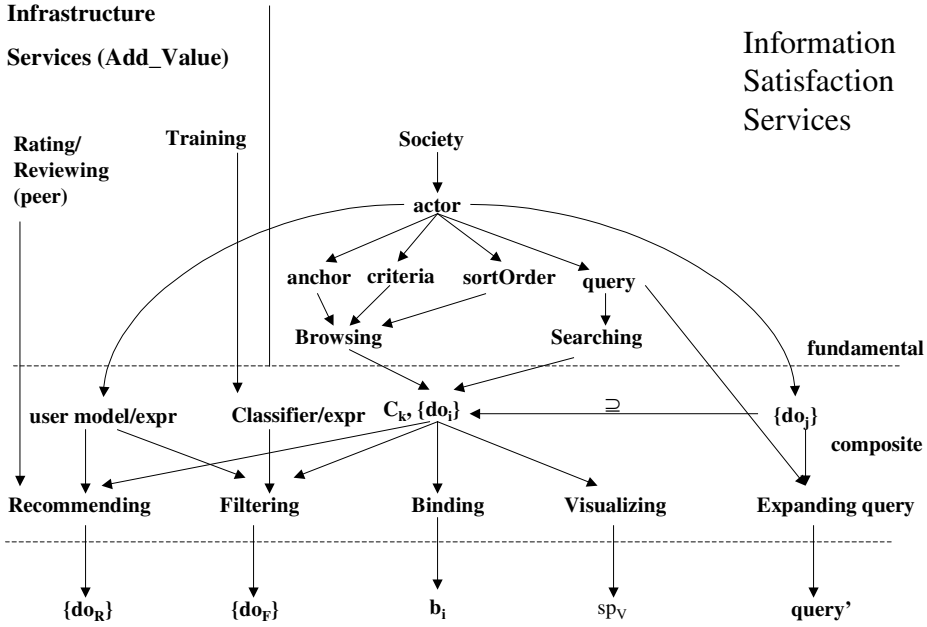

Figure 4. Examples of Compositions of Services.

Many other possible compositions are possible by analyzing the entries in Table 1. Since services such as Recommending, Filtering, Binding, Expanding query, etc., produce a set of digital objects, these sets can be further indexed for searching/browsing purposes. A Relevance Feedback service extends a Searching service and reuses a Expanding query service. An Ontology-Based navigation service may reuse Linking and Classification services while a lesson plan building service may reuse searching, browsing, binding, and describing services. An advanced searching service may reuse an Extracting structure and extend a Searching service to provide support for fielded queries.

5. Practical Applications: Brief Discussion

Space prevents us of elaborating further on the practical implications and use of the proposed ontology. Therefore we only briefly mention some previous applications and ongoing work with the ontology. Previous work include: 1) reengineering a digital library specification language [Gonçalves02, Rohit03]; and 2) developing an XML-based log standard for digital libraries [Gonçalves02b]. Ongoing work include: 1) developing a model of quality in digital libraries; and 2) contrasting disparate DL architectures by comparing what results from expressing them according to the formal ontology model.

6. Conclusions and Future Work

We have presented a digital library formal ontology which complements the syntactical definitions of DLs with semantic relationships and governing axiomatic rules, therefore producing a core theory for the field of digital libraries. A taxonomy of digital library services based on the ontology was presented and used to reason about issues of extensibility/reusability in DLs. Current and future work, besides that described in section 5, include: 1) expanding the services taxonomy to include pre- and post-conditions for service composition; and 2) creating and proving lemmas and theorems about the DL concepts and relationships defined in the ontology.

REFERENCES

[Benatallah03] B. Benatallah, M.Dumas, Quan Z. Sheng , The SELF-SERV Environment for Web Services Composition", IEEE Internet Computing , IEEE Society, Jan/Feb issue, pp. 40-48, 2003.

[Castelli03] D. Castelli, P. Pagano: A System for Building Expandable Digital Libraries. Proc. of JCDL'03. Houston, Texas, 2004, pp. 335-345.

[Curbera02] F. Curbera, M. Duftler, R. Khalaf, W. Nagy, N. Mukhi, S. Weerawarana: Unraveling the Web Services Web: An Introduction to SOAP, WSDL, and UDDI. IEEE Internet Computing 6(2): 86-93 (2002)

[CITIDEL04] The Computing and Information Technology Interactive Digital Educational Library. http://www.citidel.org. (as of May, 2004)

[Doan03] A. Doan, J. Madhavan, R. Dhamankar, P. Domingos, A. Y. Halevy: Learning to match ontologies on the Semantic Web. VLDB J. 12(4): 303-319 (2003)

[Gonçalves02] M. A. Gonçalves and E. A. Fox. 5SL - A Language for Declarative Specification and Generation of Digital Libraries. Proc. of JCDL'02, pages 263-272, July 2002. Portland, Oregon, USA.

[Gonçalves02c] M. A. Gonçalves, M. Luo, R. Shen, M. F. Ali, E. A. Fox: An XML Log Standard and Tool for Digital Library Logging Analysis, Proc. of ECDL'02, pp. 129-143, Rome, Italy, Sept. 16-18, 2002

[Gonçalves04] M. A. Gonçalves, L. T. Watson, N. Kipp, E. A. Fox. Streams, Structures, Spaces, Scenarios, and Societies (5S): A Formal Model for Digital Libraries. ACM TOIS. 22(2): 270-312.

[Hussein02] H. Suleman. Open Digital Libraries. Doctoral Dissertation. Dept. of Computer Science, Virginia Tech, 2002. http://scholar.lib.vt.edu/theses/available/etd-11222002-155624/

[Kelpaure03] R. Kelapure. Scenario-Based Generation of Digital Library Services. Master Thesis. Dept. of Computer Science, Virginia Tech, 2003. http://scholar.lib.vt.edu/theses/available/etd-06182003-055012/

[Larsen03] R. L. Larsen. Knowledge Lost in Information – Report of the NSF Workshop on Research Drections fro Digital Libraries. June 15-17, 2003, Chatham, MA, 2003

[Licklider65] J. C. R. Licklider. *Libraries of the Future*. MIT Press, Cambridge, Mass.

[NSDL04] The National Science Digital Library. http://www.nsdl.org. (as of May, 2004).

[Payette02] S. Payette, T. Staples: The Mellon Fedora Project. Proc. of ECDL 2002: 406-421

[Sompel03] H. Van de Sompel. Roadblocks. In NSF Workshop on Research Directions for Digital Libraries. June 15-17, 2003, Chatham, MA, 2003

[Witten03] I. H. Witten, D. Bainbridge: How to Build a Digital Library Morgan Kaufmann, 2003