Meta-Metadata: A Metadata Semantics Language for Collection Representation Applications

Andruid Kerne, Yin Qu, Andrew Webb, Sashikanth Damaraju, Nic Lupfer, Abhinav Mathur
Interface Ecology Lab
Department of Computer Science and Engineering
Texas A&M University, College Station, TX 77843, USA
{andruid, yin, andrew, damaraju, nic, abhinav}@ecologylab.net

ABSTRACT

Collecting, organizing, and thinking about diverse information resources is the keystone of meaningful digital information experiences, from research to education to leisure. Metadata semantics are crucial for organizing collections, yet their structural diversity exacerbates problems of obtaining and manipulating them, strewing end users and application developers amidst the shadows of a proverbial tower of Babel. We introduce *meta-metadata*, a language and software architecture addressing a metadata semantics lifecycle: (1) data structures for representation of metadata in programs; (2) metadata extraction from information resources; (3) semantic actions that connect metadata to collection representation applications; and (4) rules for presentation to users. The language enables power users to author metadata semantics wrappers that generalize template-based information sources. The architecture supports development of independent collection representation applications that reuse wrappers. The initial meta-metadata repository of information source wrappers includes Google, Flickr, Yahoo, IMDb, Wikipedia, and the ACM Portal. Case studies validate the approach.

Categories and Subject Descriptors
M.4. [**Knowledge Modeling**].

General Terms
Design, Human Factors, Algorithms, Languages, Reliability.

Keywords
metadata, digital libraries, collections, berrypicking, languages

1. INTRODUCTION

Collecting, organizing, and thinking about diverse information resources is the keystone of meaningful digital information experiences, from research to education to leisure. Formally, by an *information resource*, we mean a document, addressed by a Uniform Resource Locator (URL), conjoined with metadata, providing users with an object to peruse supplemented with data for contextualization and organization. We operationalize this crucial conjunction as *metadata semantics* to define an information resource's data structures and supported operations, spanning extraction, use, and presentation. Metadata semantics differ for different kinds of resources. For example, in the ARTstor collection [1], the document for the painting "City Phantasy" by Mark Rothko is a JPEG image; metadata includes fields such as creator, material, and source. In the ACM Digital Library, the document for the scholarly article "Faceted Metadata for Image Search and Browsing" [27] is a PDF; metadata includes fields such as authors, keywords, abstract, references, and citations. In both cases, an end user may wish to find related work, but the procedures will differ. In the case of the "Phantasy", the creator, material, and source fields could be traversed as indices to discover alternatives, while for the "Faceted" article, reference and citation sets are provided directly by the DL.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists.
Conference'04, Month 1–2, 2004, City, State, Country.
Copyright 2004 ACM 1-58113-000-0/00/0004…$5.00.

Metadata semantics are crucial for organizing collections, yet their structural diversity exacerbates the problems of obtaining and manipulating them, strewing end users and application developers amidst the shadows of a proverbial tower of Babel. In response, some efforts work to standardize metadata itself [9]. The present research alternatively takes the heterogeneity of metadata semantics as a given, unifying another level. Semantic web researchers address ambitious problems, like how to support reasoning through ontologies [23]; we instead find sufficient complexity in developing software infrastructure to help developers build collection representation applications supporting information sources as diverse as ARTStor, the ACM DL, Google, Wikipedia, and Flickr – without custom code. We define *information source* as a *type* of information resource, generalized by template-driven publishing by a particular web site and repository.

As metadata is data that describes data, we define *meta-metadata*, a formal language for authoring enhanced wrappers, each specifying the metadata semantics for an information source. We develop a meta-metadata software architecture addressing a metadata semantics lifecycle for tasks involving information collections: (1) data structures for representation of metadata inside programs; (2) extraction from information resources; (3) *semantic actions* that use control structures such as loops to invoke bridge functions to pass metadata to particular methods in collection representation applications; and (4) rules for presentation to users. We present an open source suite of tools and libraries to support the meta-metadata language and architecture for the development of collection representation applications [13]. Meta-metadata information source wrappers shepherd metadata through the lifecycle, maximizing its value. We develop use cases for meta-metadata: Rake, a multi-touch information kiosk, combinFormation, an information discovery and collection visualization application, and a Wikipedia concept parser.

Most current approaches to programming metadata semantics are cumbersome, requiring custom application code to collect and represent metadata from each heterogeneous information source. As far as we are aware, no other general tools integrate specification of data structures, extraction of metadata from heterogeneous sources into instances, and presentation to users.

The meta-metadata scripting language for metadata semantics wrappers promotes code reuse through extensibility, separation of concerns, and application independence. New meta-metadata information source wrappers can extend the definitions of prior ones. Meta-metadata wrapper declarations that define information sources are cleanly separated from application source code, so power user authors do not need access. Likewise, the support libraries for manipulating meta-metadata and the metadata that it is used to generate are application independent, and distributed as open source [13]. Wrappers can be shared across applications.

Meta-metadata semantic actions include call sites for *bridge functions*, through which a meta-metadata script invokes application-specific routines. Different applications implement the same bridge function differently. Examples of bridge functions include parsing a document, which can be used in crawlers, and forming a surrogate [14] to represent a document to users.

This paper begins by surveying prior work. Then, we develop the meta-metadata architecture, which is subdivided into compile-time and run-time components. We follow with a presentation of the meta-metadata language. Through these two sections, initial examples are developed. We develop use cases, and finish with a discussion of implications, addressing the key mechanisms of meta-metadata, and the scope of collection representation applications that it can support.

2. PRIOR WORK

The prior work includes precedents that best resemble the present research and ingredients from which this work is made. Previous research has focused on particular aspects of metadata handling for collection presentation applications, developing HTML presentations. Exhibit [11] provides a general means for publishing metadata, but does not use strong typing or address information extraction. Instead of providing semantic actions, Exhibit assumes HTML is the medium of publication. MarMite, a web browser plug-in, extracts information from web pages, and enables authoring of custom presentations, again using HTML [25]. Dontcheva et al. present a system for collecting, viewing and sharing information from the web [8]. The researchers authored wrappers with XPath-like expressions for information extraction, and used the extracted information to present collections to the user with DHTML. Zotero is a browser plug-in that supports development of wrappers for scholarly article metadata extraction [4]. We appreciate that these systems address the extraction and presentation of metadata, but need more than HTML-based publishing. Further, most of the above shoehorn source metadata into a single target form. Meta-metadata embraces metadata heterogeneity, and so can support diverse application contexts. It is essential to build a framework that supports the reuse of wrappers and extracted metadata across applications, independent of the presentation technology. Freebase [3] takes an alternative approach, using a shallow ontology without strong types for metadata, and authoring of knowledge, rather than wrappers. They do not integrate presentation rules with knowledge representation. It will be interesting to compare the results of these approaches, over time.

Systems that automate metadata extraction from information sources are either domain specific; such as Cui et al. who develop

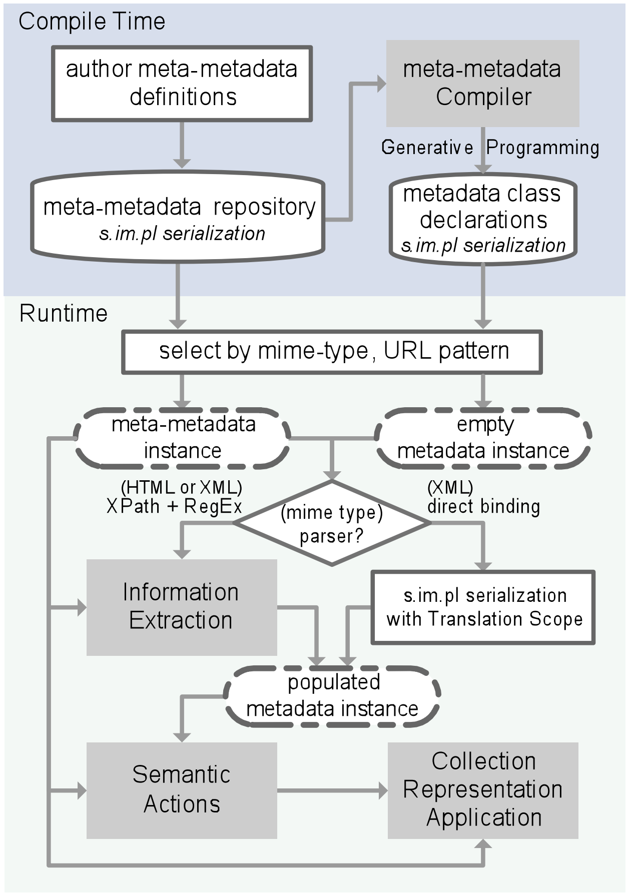


Figure 1: Meta-metadata architecture.

machine learning techniques for semantic markup for biodiversity digital libraries [6], or designed for a particular context, such as [5] and [10]. Such research systems can benefit from using meta-metadata as a representation for their output. This will enable application developers to easily use the products of such research.

Collection visualization applications such as spatial hypertext tools can benefit from meta-metadata. VKB focuses on spatial presentation of structured information to the user, allowing direct manipulations on user generated content [20]. TinderBox enables the user to add notes while also adding semantic markup to describe relationships in the form of maps and charts [2]. By integrating with the meta-metadata architecture, applications like these can benefit from the growing collection of meta-metadata wrappers to expand the set of information resources they can operate on, while unifying the underlying metadata semantics.

The underlying ingredient of meta-metadata is S.IM.PL (Support for Information Mapping in Programming Languages) serialization, a generalized form of `ecologylab.xml` [16] object-oriented XML binding. This multi-platform framework connects programming language objects with serialized representations. The core of the framework is a succinct annotation meta-language, embedded in source code, that specifies which fields are serialized and how. A contribution of the framework is the *translation scope*, which succinctly encodes a set of bindings. One translation scope is used for unmarshalling meta-metadata, while another unmarshalls metadata. The translation scope is a foundation of meta-metadata because it

```
<meta_metadata name="search" extends="document" comment="For processing search engine result pages">
    <collection name="search_results" collection_child_type="search_result" no_wrap="true" >
        <scalar name="heading" scalar_type="String" />
        <scalar name="snippet" scalar_type="String" comment="Context from the search engine"/>
        <scalar name="link" scalar_type="ParsedURL" />
      </collection>
 </meta_metadata>
```

Figure 2: Reusable meta-metadata wrapper declaration for search engine results.

```
/**
 * For processing search engine results
**/
public class Search extends Document
{
    @simpl_collection("search_result")
    @simpl_nowrap
    ArrayList<SearchResult>    searchResults;
}
public class SearchResult extends Metadata
{
    @simpl_scalar MetadataString            heading;
    /**
     * Context from the search engine
    **/
    @simpl_scalar MetadataString            snippet;
    @simpl_scalar MetadataParsedURL         link;
}
```

Figure 3: Metadata subclass declarations generated by the compiler for the search meta-metadata declared in Figure 2.

automates binding XML that represents a type of metadata with a matching metadata subclass. The present implementation is based in Java, with the ability to cross-compile equivalent objects and serialization to Objective C and C# source, facilitating cross-platform distributed programming, including iPhone clients.

3. ARCHITECTURE

The meta-metadata architecture connects sets of compile time and run time modules (Figure 1) to address the lifecycle of metadata utilization in collection representation applications by specifying strongly typed data structures, extraction rules, semantic actions, and presentation rules. Meta-metadata definitions are used first at compile-time by the meta-metadata compiler to generate strongly typed metadata subclass declarations in Java, which in turn are compiled by the Java compiler. S.IM.PL serialization also enables these class definitions to be cross-compiled to C# and Objective C. The resulting classes remain bound to their corresponding meta-metadata. The runtime lifecycle begins with selection of meta-metadata that matches an information resource's URL, and then proceeds through information extraction, which populates the appropriate metadata subclass instance, semantic actions that connect the metadata to the collection representation application, and then presentation.

3.1 Compile-Time

The compile-time phase consists of two stages: authoring and compilation. Power users author meta-metadata wrappers for particular information sources. In the present implementation, these are stored as XML files and installed on the user's machine. The current meta-metadata repository includes declarations for search engines, such as Google, Bing, and Yahoo, and for collections including Flickr, the ACM Portal, CiteSeer, Wikipedia, and IMDb. Information sources are distinctly specified with mime type and URL patterns, with `text/html` as the default mime type. These patterns are matched at runtime to retrieve information source specific meta-metadata.

Figure 2 shows the beginning of a simple example. We declare data structures to support typical search engines. Using the inheritance capabilities of the Metadata Definition Language (Section 4.2), we extend the basic metadata type `document` to define a new type, `search`. The search type consists of a collection of `search_results`, each of which consists of a `heading`, a `snippet`, and a `link`.

The meta-metadata compiler takes meta-metadata XML declaration as input. Where new data types are declared, it generates strongly typed metadata object class definitions in Java (Figure 3), with S.IM.PL annotations that direct serialization. It is invoked as a standalone utility or through Eclipse Ant scripts. This compilation of authored meta-metadata definitions (`search` and `search_result` in Figure 2) to produce `Metadata` subclass declarations (`Search` and `SearchResult` in Figure 3) is a necessary precursor for information extraction. Primitive metadata field classes, such as `MetadataString`, add functionality such as term vectors to the basic scalar types, such as `String`. The compiler generates a translation scope for all generated metadata classes, which serves as the basis for binding Java metadata subclasses to XML elements for direct binding information extraction (Section 0), and while loading saved collections. The compiler uses comments from the source meta-metadata XML to generate Java Doc comments for each output class to enhance readability in the implementation. Omitted from Figure 3 to save space are getter and setter methods that are also

```
<search query="metadata" location="http://www.google.com/search?q=metadata">
    <search_result heading="DCMI Home: Dublin Core Metadata Initiative (DCMI)"
        snippet="Organization dedicated to promoting widespread adoption of interoperable metadata standards"
        link="http://dublincore.org/" />
    ...
</search>
```

Figure 4: Partial instance of results of a Google search for 'metadata' uses Metadata sub-classes of Fig. 3; automatically populated using extraction rules declared in Figure 5, which in turn reuse `search` and `search_results` types declared in Figure 2.

automatically generated by the meta-metadata compiler.

3.2 Runtime

At runtime, the collection representation application uses meta-metadata and metadata to represent, operate on, and present information resources (Figure 1). The process begins with selection of appropriate meta-metadata from the repository, identification of the metadata subclass associated with the meta-metadata, and instantiation of a matching empty metadata instance, a strongly typed generalized data structure that can have nesting. When a match is found through meta-metadata selection (Section 3.2.1), the appropriate parser is invoked to populate the metadata. For HTML and XML documents, extraction rules, specified as XPath and regular expressions, are used by the parser to extract information from a document, populating the metadata instance. For XML, the metadata instance can alternatively be populated by a "direct binding" parser, using a translation scope to look-up the appropriate metadata and meta-metadata using the XML document's root element tag as key. With either parser, next, semantic actions are carried out. The semantic actions language includes bridge functions that pass populated metadata instances and intermediate results to methods in the collection representation application, variable declaration statements, and basic control structures, such as loops and conditionals. The bridge functions are terminals of this part of the runtime cycle; they pass metadata instances to appropriate application functions, given the structure of the particular information source, as specified by the meta-metadata author. Meta-metadata can be used subsequently by the application to efficiently iterate over the fields of a metadata instance or access a field by name.

When the application wants to present the populated metadata instance to the user, rules specified in the meta-metadata wrapper, for the information source, are used to guide presentation. They enable hiding fields, ordering them, and formatting them. This reduces the *metadata noise effect*. It is easy to bombard users with many fields of data that are not meaningful to them. Part of curating an information source is to choose which fields to emphasize in the presentation to users, and which to hide.

In this section, we detail the runtime mechanisms of meta-metadata selection, and the associated algorithms, because their generality and efficiency are crucial to the meta-metadata architecture's practical usability. The subsequent section develops further explanation of how data structures are declared, information extraction performed, and semantic actions specified.

3.2.1 Meta-Metadata Selectors

Meta-metadata selection must be efficient, because this lookup must be performed for every information resource an applications encounters. Selectors are two-level, beginning with mime type, and then URL matching. Mime type lookup is easily implemented with a hash table. URL selection is more complex. Our experience of information sources in the wild has led us to develop three mechanisms: `url_stripped`, `url_path_tree`, and `url_regex`. The meta-metadata author should use the one with minimum computational complexity that will work for the information source at hand. The choice is determined by structure of the URL a website uses to publish an information source. Factors include how arguments are represented as path elements, and whether server farms dispatch requests to multiple hosts.

The simplest selector, `url_stripped`, removes formal arguments (query and hash) from the URL, and uses this as a hash key. This works for sources such as search engines, in which queries are passed as such. The computational complexity is O(1).

As a result of published URL formats, some information sources require more general selectors. A Flickr image page URL format is `http://www.flickr.com/photos/AUTHOR_ID/IMAGE_ID`. The query is not in the URL arguments, but in the path. Stripping the query (here the last 2 subdirectories) from this URL yields `http://www.flickr.com/photos`. However, another Flickr information source URL form, for all photos by an author, is `http://www.flickr.com/photos/AUTHOR/`. Stripping the arguments from the path yields the same result, so this is not an unambiguous selector. Thus, we define the `url_path_tree` selector, using subdirectory wild cards to define the URL key for meta-metadata. With this mechanism, the selector key for Flickr image result page is `http://www.flickr.com/photos/*/*/`, and for Flickr author page, `http://www.flickr.com/photos/*/`. We use '*' as a wild card that matches one or more characters. To represent selectors like these, the runtime infrastructure uses an efficient path tree data structure, reminiscent of radix sort, which separates URL patterns into path components, representing each as a nested hash map of subsequent components, or a terminal. The root phrase is the domain. If no more components exist with the same domain, the corresponding value for this key will be the meta-metadata object itself. If there are multiple path components with the same domain, the corresponding value for this key will be a nested hash table. This 2nd level hash map will contain as keys the next component of the URL pattern. The computational complexity of path tree's nested hash tables is O(n), where n is the number of path components needed to uniquely represent the pattern. Typically n ranges from 2-4 in practice.

A more powerful meta-metadata selector, `url_regex`, uses regular expressions to increase generality, at the cost of higher computational complexity. Some URLs, like the source of a Flickr image, can be served from multiple hosts, requiring matching URL patterns such as `http://farm3.static.flickr.com/2020/2118178242_27fb91853a_m.jpg`. A regular expression selector is `http://farm[0-9].static.flickr.com`. Since regular expressions are expensive to evaluate, for each domain we maintain a hash table of lists of compiled regular expressions. The computational complexity for matching a string to a regular expression, once the expression has been compiled into a discrete finite automata (and this compilation need be performed only once) is O(m), where m is the string length. Computational complexity is O(l * m), where l is the length of the list of patterns that must be matched for a domain. This selector is slower, but as long as the number of `url_regex` wrappers authored per domain is not large, performance is fine.

4. META-METADATA LANGUAGE

We present fundamental components of the meta-metadata language. Further details and examples are online [13]. We begin with the structure of inheritance and types that underlies meta-metadata declaration.

Object-oriented treatment of data structures, including inheritance enables us to build reusable types, such as `search` and `scholarly_article`, to represent species of metadata that can be generalized across sources, in consideration of situated tasks and activities that users need to perform. The Metadata Definition Language (MDL) is used to specify data structures, corresponding to types of information sources. These structures, whether generalizable, or specific to a particular source, are populated at runtime (Figure 1) with data from particular documents to form metadata instances, based on specifications authored by curator / power users using the information extraction components of the meta-metadata language. The formation of instances is followed by semantic actions, which consist of runtime script used to process each metadata instance and bridge to the application. Finally, presentation rules give the application guidelines on how to meaningfully present metadata to the user.

4.1 Inheritance and Types

An inheritance system promotes reuse of metadata classes through meta-metadata declarations. All meta-metadata element declarations either extend previous meta-metadata declarations, adding fields, or reuse existing data structures without adding fields. Whenever a meta-metadata declaration adds new fields, the meta-metadata compiler must be invoked to generate a corresponding `Metadata` subclass. The `extends` attribute of the `meta_metadata` element is used to specify the previously declared `Metadata` subclass that will be extended in the new subclass's declaration, with `Metadata` itself as the default base class. The `name` attribute specifies, with automatic camel case conversion, this newly generated subclass's name. Going back to Figures 2 and 3, we see how the meta-metadata element declared with `name="search"` and `extends="document"` results in generation of a new `Search` class, from `Document`, while the `SearchResult` class inherits directly from `Metadata`.

4.2 Metadata Definition Language (MDL)

The MDL is the component of the meta-metadata language used for specifying internal data structures for representing types of metadata. Metadata definition language (MDL) allows power users to author structured and strongly typed declarations for heterogeneous information sources. Each information source is declared with an initial `meta_metadata` element with any number of nested meta-metadata field children (see example, Figure 2). The name attribute of every `meta_metadata` element declaration must be unique. Each child meta-metadata field is either a scalar field, a composite field, or a collection field. Scalar fields declare a `scalar_type` attribute. Composite fields declare a `type` attribute, reusing a previously declared type, or an `extends` attribute, subclassing a previously defined type. Collection fields set the `child_type` attribute which either specifies a previously defined type for collection elements, or a subsequent inline definition.

We present BNF to describe the key productions of the grammar for type and inheritance declarations. SN refers to a symbolic name, which must begin with a letter.

```
meta_metadata ::=
    '<meta_metadata ' name type extends '>'
    meta_metadata_field *
    selector
    semantic_action *
    '</meta_metadata>'

meta_metadata_field ::=
    scalar | composite | collection

scalar ::=
    '<scalar ' name scalar_type '/>'

composite ::=
    '<composite ' name type extends '>'
    meta_metadata_field *
    '</composite>'

collection ::=
    '<collection ' name child_type '>'
    meta_metadata_field *
    '</collection>'

name ::= 'name="' SN '"'

extends ::= 'extends="' SN '"' | ''

type ::= 'type="' SN '"' | ''

child_type ::= 'child_type="' SN '"'

scalar_type ::= 'scalar_type="' s_type '"'

s_type ::=
    'int' | 'boolean' | 'long | 'float' | 'double'
    | 'String' | 'ParsedURL' | 'Color' | 'Date'…
```

4.3 Information Extraction

While MDL enables specification of optimized metadata data structures for heterogeneous information sources, information extraction components of the meta-metadata language enable us to translate HTML and XML documents as information resources into instances of these data structures. Figure 5 presents a simple example. The `search` type has previously been declared (Figure 2). We reuse the data structure definition, while specifying how to perform information extraction for a specific information source, Google web search.

The example begins by defining a new `meta_metadata` element, with a unique name. Setting the `type` attribute to the previously defined `search` meta-metadata specifies data structure reuse, adding no new data fields, directing the meta-metadata compiler to use the previously declared `Metadata` subclass instead of generating a new one. The simplest meta-metadata selector (Section 3.2.1), `url_stripped`, is specified, with the argument-less URL for Google search.

The `parser` attribute specifies the library class that will be used to perform translation from an information resource to entities and operations in the collection representation application. Currently supported parsers include `xpath`, for extraction from HTML or XML by XPath and regular expressions, and `direct` for S.IM.PL Serialization XML binding. Another parser processes PDF documents. Future parsers will handle other types of documents, such as images, and various formats of audio and video. For the default document meta-metadata, which is used when no source specific selector is matched, the present implementation uses a parser for HTML that implements Koh and Kerne's DOM-based algorithm, translating a document into a set of contextualized image and text surrogates [17]. Developers can build alternative default parsers by implementing an interface.

4.3.1 XPath

The XPath parser supports combining XPath and regular expressions for metadata extraction from an information resource.

```
<meta_metadata name="google_search" type="search" parser="xpath">
        <selector url_stripped="http://www.google.com/search"/>
        <collection name="search_results" xpath="//div[@id='res']/div/ol/li[@*]" >
                <scalar name="heading" xpath="./h3/a" />
                <scalar name="snippet" xpath="./div[1]" />
                <scalar name="link" xpath="./h3/a/@href"/>
        </collection >
        <semantic_actions>
                <get_field name="search_results"/>
                <for_each collection="search_results" as="result">
                        <get_field name="link" object="result"/>
                        <parse_document now="true">
                                <arg value="link" name="container_link"/>
                        </parse_document>
                </for_each>
        </semantic_actions>
</meta_metadata>
```

Figure 5: XPath information extraction and semantic action declarations for Google search engine results.

XPath is a standard for specifying the path in the DOM tree to a specific node [24]. XPath expressions can be specified in a rote way, with every node starting from a document's root element, or, more robustly, using DOM features such as an ID node, which may be more likely to remain intact when small changes are made to a document. Line 2 of Figure 5 takes the latter approach, using XPath to find the `div` node with `@id='res'`, and then specifying the rest of the expression relative to there. The expression selects the first `div` nested inside, then the `ol` element in inside that, and then all the `li` elements nested in there. Note that for XPath expressions declared in nested meta_metadata field declarations, this intermediate result functions as the basis for subsequent simpler relative XPath expressions. Thus, for `heading`, `snippet`, and `link`, we author simple expressions that do not repeat the whole ugly XPath.

In some cases, an XPath expression is not sufficiently specific for extracting the value for a metadata field. Content presenters may put multiple values in a single table cell, or they may prepend or append meaningless strings in proximity of meaningful ones. On these occasions, further extraction can be specified using the `regex` attribute of the `filter` sub-element. All matches will be replaced by the value of `replace` attribute. The default is the empty string, which causes matched text to be eliminated.

4.3.2 Direct Binding for XML-Based Services

Some information sources are available through XML-based services. In such cases, we can use MDL to define metadata XML that matches the structure the service provides, and then use S.IM.PL Serialization to directly bind the XML the service returns in response to a query to the generated annotated classes, forming metadata objects. Figure 6 shows the MDL declarations for a subset (to save space here) of the XML returned by Yahoo Search Web Services [26]. Here, we see use of the `tag` attribute, which is used to override automatic camel case conversion of the element name, so that the XML tags can be set to directly correspond to the service's XML, in which all element tag names are capitalized. No further authoring is needed to perform information extraction with direct binding, though in the repository, semantic actions are provided for the different versions of the service that correspond to web, image, and news search, all using the `yahoo_result_set` type [13].

4.4 Semantic Action Scripting

While all other components of the meta-metadata language are declarative, semantic actions are imperative, like many scripting languages. Semantic actions are declared as part of the meta-metadata template for an information source, in order to enable power users to procedurally specify how the collection representation application will operate on metadata objects. Semantic action statements are specified sequentially as XML elements, nested in a `semantic_actions` XML element, which follows MDL and information extraction statements in a `meta-metadata` element declaration. There are three kinds of semantic actions: variable declarations, control flow statements, and bridge functions.

4.4.1 Variable Declarations

Variables are declared in `get_field`, `def_var`, and loop statements. They are used to pass values as arguments from metadata objects to bridge functions, and as intermediate values in computation, whose presence simplifies subsequent code.

The `get_field` statement assigns scalar, composite, or collection metadata field values to a variable. The name attribute refers both to the metadata field to be accessed, and to the name

```
<meta_metadata name="yahoo_result_set" extends="search" parser="direct" tag="ResultSet" comment="Yahoo Web Service">
        <collection name="results" child_tag="Result" child_type="result" no_wrap="true">
                <scalar name="title" tag="Title" scalar_type="String"/>
                <scalar name="summary" tag="Summary" scalar_type="String"/>
                <scalar name="url" tag="Url" scalar_type="ParsedURL"/>
                <composite name="thumbnails" tag="Thumbnail" collection="ArrayList" child_type="thumbnail">
                        <scalar name="thumb_url" tag="Url" scalar_type="ParsedURL"/>
                        <scalar name="width" tag="Width" scalar_type="Int" />
                        <scalar name="height" tag="Height" scalar_type="Int" />
                </composite>
        </meta_metadata_field>
</meta_metadata>
```

Figure 6: MDL declaration for Yahoo Web Service matches their XML. Information extraction by direct binding is automatic.

```
<def_vars>
        <def_var name="photoTD" xpath="//td[@id='photoswftd]" type="node"/>
        <def_var name="photoTR" xpath="./tr" context_node="photoTD" type="node_list"/>
</def_vars>
```

Figure 7: `def_var` variable declaration statements, taken from a Flickr example.

that will be subsequently used to refer to it. We see as an example the first semantic action declared in Figure 5, which assigns the `search_results` metadata, a collection, to a variable. The next `get_field` statement specifies a context object, that is, a previously defined variable. A name is bound to a field within this object. In the example, a reference to the `link` field within the `result` variable is defined, then passed to the `parse_document` bridge function (Section 4.4.3).

The `def_var` statement enables binding the DOM node result of an XPath expression to a temporary variable. Figure 7 shows two examples. In the first, a single DOM node is bound to the variable `photoTD`. The `xpath` attribute is an absolute one, operating on the whole of the source document. In the next declaration, of `photoTR`, the XPath operates relative to the `context_node` of the previous declaration. Here, the type is a set, a `node_list`.

4.4.2 Control Flow Statements

Control flow statements specify the order of execution of semantic actions. These structures include loops and conditional statement. We considered using XSL [22] as the semantic action language, as it enables specification of control structures in XML. But XSL can only be used for transformation of XML documents. It cannot be used to invoke bridge functions which can act as call sites for operations in the collection representation action. To provide consistency, the control flow structures in the meta-metadata language are similar to those in XSL.

Conditionals include `if` and `choose`. Loops are defined with the `for_each` statement. The attributes are `collection`, defining the source of elements to loop over, and `as`, defining the symbolic name for referencing the collection element value inside the loop. The example of Figure 5 loops through the results of a Google search, referring to each as `result`. It calls the `parse_document` bridge function with the `link` field of each `result`.

4.4.3 Bridge Function Statements

A crucial family of semantic action script statements specifies calls to bridge functions. These semantic actions are call sites for methods inside the collection representation application, enabling metadata to flow from the meta-metadata template into a specific application function. Their terminals are defined in the meta-metadata library source code using constructs such as Java's `interface` structure, so that different applications can define them in different ways.

The following bridge functions have been defined in the context of the use cases described in Section 5. The generality or specificity of their calling structure will be the subject of further research. It is possible that different classes of applications will need different bridge functions, not just to provide different implementations of these functions, which will impact the extent to which semantic action elements of wrappers can be reused across applications. One bridge function, for crawlers, is invoked to handle a hyperlink, perhaps parsing the destination document: `parse_document`. The now argument can help a focused crawler [7] decide the priority of links. Other bridge functions include `handle_image` and `handle_text`. These can be used by applications to directly create interactive elements and add them to collection visualizations. Before and after semantic actions are invoked automatically for application-specific initialization and clean-up.

4.5 Presentation Rules

An important part of curating an information source is specifying aspects of how it will be presented and visualized by interactive applications. Metadata can be very long, while human cognition is limited. The curation role is to help the application help the user focus attention on what is expected to be significant. Presentation rules tell the collection representation application how particular fields in a metadata object should be shown to users, if at all. For example, fields can be hidden in the present implementation by setting the `hide` attribute in a meta-metadata field declaration. This is important, because some fields may be used internally, but not be directly meaningful to the user. Complementary with hiding fields, is specifying the `navigates_to` attribute. This can be used by the application, for example, to afford hypertextual navigation to a document's location by clicking the name of a field like `title`. Other presentation rules specify layers and the use of text styles, with characteristics such as bold and font size. In the right API environment, these could be specified with full CSS [21]. A full set of examples can be found in the meta-metadata repository distributed with the library, and the tutorials [13]. The authors believe that as we and others have more time to use the meta-metadata language for the foundation of collection visualization applications, our vocabulary of presentation rules will grow, with the applications, in complexity and abstraction.

5. USE CASES

We develop three real world use cases of the meta-metadata language and architecture: the Rake multitouch information kiosk, the combinFormation creativity support tool, and a Wikipedia concept parser. It should be noted that except in the last case, the use case applications are information source independent. When developers from one team enhanced the repository of meta-metadata wrappers, the other application also benefited.

5.1 Rake Multitouch Kiosk

A team of four computer science undergraduate students elected to use meta-metadata for a class project. They developed Rake, an information kiosk that connects multi-touch interaction with collecting and filtering information. The application enables users to query information sources, such as Urban Spoon, Flickr, Wikipedia, IMDb, and Yahoo. The search results are presented visually as "information elements," using images with metadata. Users can filter, sort and organize the elements using multi-touch interactions (See Figure 8).

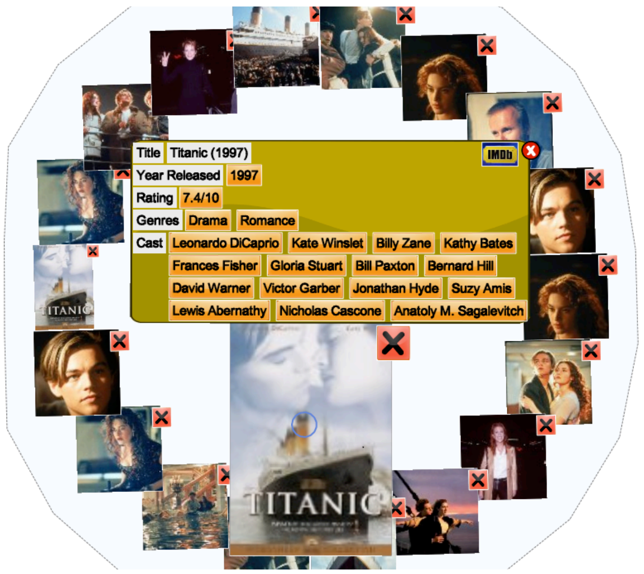

Title Titanic (1997)
Year Released 1997
Rating 7.4/10
Genres Drama Romance
Cast Leonardo DiCaprio Kate Winslet Billy Zane Kathy Bates Frances Fisher Gloria Stuart Bill Paxton Bernard Hill David Warner Victor Garber Jonathan Hyde Suzy Amis Lewis Abernathy Nicholas Cascone Anatoly M. Sagalevitch

Figure 8: The Rake multitouch information kiosk.

The students reported that they chose to use the meta-metadata framework because meaningful interaction with the information elements required detailed metadata. When the user activates a metadata field value in an information element, fluid interaction attracts matching elements.

For the project the students utilized the meta-metadata repository already developed by the combinFormation project. They also authored new meta-metadata wrappers. The new wrapper code was merged into the repository.

On the application side of the framework the students defined an application-specific Metadata subclass. They implemented the bridge function `handle_image` translate each metadata image object into an instance of the subclass, `InformationElement`, which performs display and interaction.

The students said that overall, using the meta-metadata framework expanded project. They reported that first, they did not have to implement functionality to represent documents for each information source. This reduced the work required to support diverse information sources, allowing them to focus on their primary goal, creating interactions. Second, they reported appreciating the ability to generalize their application across information sources without custom code.

5.2 combinFormation

The meta-metadata language and architecture were recently integrated into the creativity support tool, combinFormation [12][14]. combinFormation connects searching, browsing, organizing, modeling, and visualizing information. combinFormation uses the integrative visual representation of *information composition* to represent collections, instead of lists or grids of separate elements. The composition is formed using image and text surrogates, derived from clippings from documents, to represent important ideas from the documents. Representing collections as information compositions of image and text surrogates has been shown to promote creativity in laboratory [15] and field [14] studies. In prior versions of combinFormation, the metadata for surrogates and documents was comparatively lacking in detail. The metadata plays a key role in the user experience of the information composition.

We used combinFormation integrated with meta-metadata to author an information composition about metadata presentation and user experience using some of the information resources cited in this paper. We made particular use of meta-metadata wrapper declarations for `acm_portal`, which in turn, extend a reusable `scholarly_article` meta-metadata base type. The resulting composition is seen in Figure 9. The composition has been composed to concisely convey important issues and ideas related to this paper, in a form that provokes thinking. The user is holding her cursor over a text surrogate from the classic faceted metadata paper [27] mentioned in the introduction. The program displays in-context metadata details on demand.

5.3 Wikipedia Concept Parser

A graduate student not previously involved in the development of meta-metadata used it as the basis for an implementation of the Wikipedia concept labeling algorithm developed by Mihalcea and Mihalcea and Csomai [18], and refined by Milne and Witten [19]. The product of the concept parser is a database of structures enabling labeling text with an ontology of concepts derived from the relationship semantics in Wikipedia.

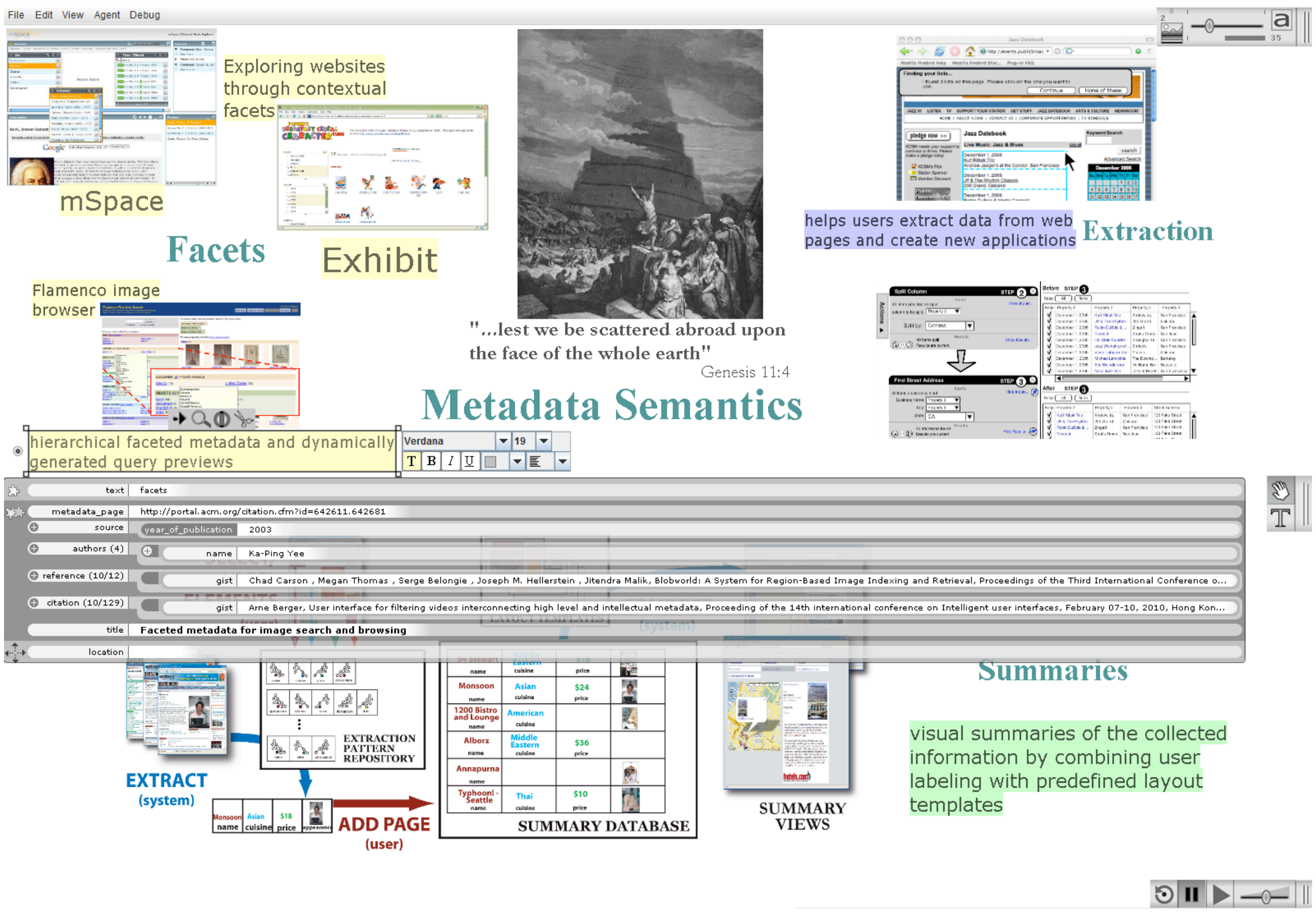
The student reused the meta-metadata type `wikipedia_page` in the repository to enable information extraction of titles, images, paragraphs, semantic links and categories. In his application, he re-defined the link handling semantic action, `parse_document` as the call site into his own semantic link processing procedure, which filters link information, and stores it in an external database. Before and after semantic action hooks were defined to customize initialization and termination.

According to the student, meta-metadata relieved him from writing code addressing information extraction, concurrency, network connections, and error handling. Reusable semantic actions allowed him to work at a high level of abstraction, with flexibility for customizing details.

The meta-metadata library was found to perform well. The parsing was carried out on a workstation with an Intel Q6600 2.4GHz CPU and 4GB RAM. A total of 5,352,582 Wikipedia articles were processed. One hundred concurrent threads were used to saturate the network. The parsing was finished in about 20 hours, with a CPU cost of less than a core and RAM cost of less than 200MB. On average, the meta-metadata library took 0.013 seconds to parse a page, generate a metadata object, and carry out the semantic actions. The products include 6.5GB of link structure records, and 795MB of category records.

6. DISCUSSION

Meta-metadata is designed to support a wide variety of collection representation applications. We have shown a knowledge management application that derives a concept-based ontology, and two personal collection visualization applications, one of which was built by undergraduate students as a course project. The language and architecture are also suitable for supporting applications such as mash-up authoring systems, structured data repositories, semantic web tools, and digital libraries. Meta-metadata provides a unified layer that generalizes and abstracts the diverse metadata semantics of heterogeneous information sources. Application developers are freed from this tower of Babel


File Edit View Agent Debug
Exploring websites through contextual facets
mSpace
Facets
Exhibit
"...lest we be scattered abroad upon the face of the whole earth"
Genesis 11:4
helps users extract data from web pages and create new applications
Extraction
Flamenco image browser
Metadata Semantics
hierarchical faceted metadata and dynamically generated query previews
Verdana
19
EXTRACTION PATTERN REPOSITORY
EXTRACT (system)
ADD PAGE (user)
SUMMARY DATABASE
SUMMARY VIEWS
Summaries
visual summaries of the collected information by combining user labeling with predefined layout templates

text	facets
metadata_page	http://portal.acm.org/citation.cfm?id=642611.642681
source	year_of_publication 2003
authors (4)	name Ka-Ping Yee
reference (10/12)	gist Chad Carson , Megan Thomas , Serge Belongie , Joseph M. Hellerstein , Jitendra Malik, Blobworld: A System for Region-Based Image Indexing and Retrieval, Proceedings of the Third International Conference o...
citation (10/129)	gist Arne Berger, User interface for filtering videos interconnecting high level and intellectual metadata, Proceeding of the 14th international conference on Intelligent user interfaces, February 07-10, 2010, Hong Kon...
title	Faceted metadata for image search and browsing
location	

Figure 9: Information composition about metadata semantics and user experience authored with the combinFormation creativity support tool, integrated with the meta-metadata architecture. The program derives visual surrogates by clipping images and text passages from information resources, and augments them with metadata. Using papers found in the references section (below), this composition has been composed to concisely convey important issues and ideas related to this paper, in a form that provokes thinking. The user is holding her cursor over a text surrogate from the classic faceted metadata paper [27]. The program displays in-context metadata details on demand.

of details. Conversely, inasmuch as the functionality of the application itself is general, a power user can extend it, without accessing source code, to utilize a new information source by authoring a new meta-metadata wrapper. Application developers and power users can share meta-metadata wrapper code, leveraging each others' ongoing efforts. Research projects that use pattern recognition techniques to learn wrappers can use meta-metadata as a representation for their output, enabling this output to be used by diverse application developers.

As collection representation application developers, for us meta-metadata is a holy grail that we have sought for years. On the one hand, the language and architecture let us invest in authoring wrappers for information sources and perform complex operations through a unified structure of information semantics without pinning this work to a specific application. At the same time, we can develop applications of varying complexity that use and reuse the wrappers and particular metadata instances that we collect with them. Meta-metadata enables us to draw our line in the sand, with the information source metadata semantics wrappers repository and information resources collection on one side, and application code on the other. Our Meta-Metadata Guide web site [13] includes a tutorial application that uses less than 20 lines of new application code to collect weather data across Texas. It would be simple to change this to collect beach or skiing weather, and publish this information to the web.

Underlying the meta-metadata language and architecture is a practical and philosophical position. The world is a diverse, heterogeneous poly-lingual place of rich experiences whose formal specification requires many dimensions. Thus, our metadata must also be poly-lingual and multidimensional. Diversity of metadata dialects makes life rich, yet complicated. Meta-metadata seeks a role inspired by Adrienne Rich's *Dream of a Common Language*, and Startrek's universal translator, but without collapsing the world of metadata languages into a dry Esperanto or hegemonic lingua franca.

This paper reports on first steps in the ambitious long term project of developing cyberinfrastructure embracing metadata diversity, and seeking to define a basis of unity on a meta- level. Translation and mapping among dialects is difficult, yet important for meaning-making. The translation scope mechanism is foundational in its ability to enable us to unmarshal poly-lingual metadata to into related classes of objects. The extensibility of meta-metadata enables us to create a hierarchy of metadata subclasses, and operate on them using the polymorphism capabilities of programming languages. The structure of the evolving hierarchy itself, as defined through the inheritance mechanisms of meta-metadata repository entries, defines relationships between dialects. Semantic actions script enables the specification of dialect-specific operations, while providing the means for defining these operations using unified and platform-independent semantics. These constructs provide the foundation for continuing development of meta-metadata cyberinfrastructure.

The development of meta-metadata is a long term investment in cyberinfrastructure. The language and architecture as currently defined are useful enough to support multiple applications, and simple enough to support development by undergraduate students. Meta-metadata wrappers are reusable across applications, except perhaps for the semantic action bridge functions. While we provide a mechanism for different applications to specify different implementations, it is not clear that this level of generality is sufficient. As meta-metadata is integrated into more applications we will learn more about how to best specify this language component.

To make it easier for power users to author and share meta-metadata, future research must develop new tools for authoring and services for distribution. Authoring tools will need to address all phases of the metadata life cycle, not just information extraction. Services will be based on a centralized or distributed repository. They will support dynamic compilation of metadata subclasses and dynamic class loading in applications. A service-based repository will help support a social network of authors to share their meta-metadata definitions. At the same time, the plan to develop a service raises community software issues, such as how versioning will work, and how, as with software development repositories, to enable users to share sometimes, and at other times be mutually insulated.

7. CONCLUSION

We have developed an extensible architecture and framework for managing the metadata lifecycle in collection representation applications. The unity of metadata semantics gained through the use of meta-metadata will enable developers of collection representation tools to focus on the next level of human-centered research issues, such as search, data mining, machine learning, digital libraries, information visualization, and supporting creativity and information discovery. First steps have been taken in the long term cyberinfrastructure development project of providing a foundation for integration of diverse, heterogeneous, poly-lingual metadata.

8. REFERENCES

[1] ArtStor, http://www.artstor.org.

[2] Bernstein, M., Collage, composites, construction. *Proc Hypertext 2003*, 122-123.

[3] Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J., Freebase: a collaboratively created graph database for structuring human knowledge, *Proc SIGMOD 2008*, 1247-50.

[4] Center for History and New Media at George Mason, Zotero, http://www.zotero.org

[5] Cortez, E., Silva, A., Gonçalves, M., Mesquita, F., Moura, E., FLUX-CIM: flexible unsupervised extraction of citation metadata, *Proc JCDL 2007*. 215-224.

[6] Cui, H., Unsupervised semantic markup of literature for biodiversity digital libraries, *Proc JCDL 2008*. 25-28.

[7] Diligenti, M., Coetzee, F., Lawrence, S., Giles, C. L., and Gori, M. Focused Crawling Using Context Graphs, *Proc ACM VLDB 2000*, 527-534.

[8] Dontcheva, M., Drucker. S., Wade, G., Salesin, D., Cohen, M., Summarizing personal web browsing sessions. *Proc UIST 2006*, 115-224.

[9] Dublin Core Metadata Initiative, DCMI Specifications, http://dublincore.org/specifications/

[10] Hetzner, E., A simple method for citation metadata extraction using hidden markov models, *Proc JCDL 2008*.

[11] Huynh, D.F., Karger, D.R., Miller, R.C. Exhibit: lightweight structured data publishing, Proc WWW 2007, 737-746.

[12] Interface Ecology Lab (2005-10), combinFormation, http://ecologylab.net/combinFormation/

[13] Interface Ecology Lab (2010), Meta-Metadata resources and tutorial, http://ecologylab.net/metametadata

[14] Kerne, A., Koh, E., Smith, S. M., Webb, A., Dworaczyk, B., combinFormation: Mixed-Initiative Composition of Image and Text Surrogates Promotes Information Discovery , *ACM Trans Information Systems (TOIS)*, 27(1), Dec. 2008, 5:1-45,

[15] Kerne, A., Smith, S.M., Koh, E., Choi, H., Graeber, R., An Experimental Method for Measuring the Emergence of New Ideas in Information Discovery, *Intl Journal of Human-Computer Interaction (IJHCI)*, 24 (5) July 2008, 460-477.

[16] Kerne A., Toups Z., Dworaczyk B., Khandelwal M., A concise XML binding framework facilitates practical object-oriented document engineering, *Proc DocEng 2008*, 62-65.

[17] Koh, E., Kerne, A., Deriving image-text document surrogates to optimize cognition, *Proc DocEng 2009*, 84-93.

[18] Mihalcea, R., Csomai, A., Wikify!: linking documents to encyclopedic knowledge, Proc CIKM 2007.

[19] Milne, D., Witten, I.H. 2008. Learning to link with wikipedia, Proc CIKM 2008, 509-518.

[20] Shipman, F., Hsieh, H., Airhart, R., Maloor, P. and Moore, J.M., The Visual Knowledge Builder: A Second Generation Spatial Hypertext, *Proc. ACM Hypertext 2001*. 113-122.

[21] W3C, Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification, http://www.w3.org/TR/CSS21/.

[22] W3C, Extensible Stylesheet Language (XSL) Version 1.1, http://www.w3.org/TR/xsl/.

[23] W3C, Web Ontology Language, http://w3.org/2004/OWL/

[24] W3C, XML Path Language, http://www.w3.org/TR/xpath/

[25] Wong, J., Hong, J. Making mashups with marmite: towards end-user programming for the web, *Proc CHI 2007*.

[26] Yahoo, Web Documentation for Yahoo! Search, http://www.yahooapis.com/search/web/V1/webSearch.html

[27] Yee, K., Swearingen, K., Li, K., Hearst, M. Faceted metadata for image search and browsing. *Proc CHI 2003*.