A Hybrid Declarative/Procedural Metadata Mapping Language Based on Python

Greg Janée[1] and James Frew[2]

[1]Alexandria Digital Library Project
Institute for Computational Earth System Science
University of California, Santa Barbara
Santa Barbara, CA 93106-3060
gjanee@alexandria.ucsb.edu
[2]Donald Bren School of Environmental Science and Management
University of California, Santa Barbara
Santa Barbara, CA 93106-5131
frew@bren.ucsb.edu

Abstract. The Alexandria Digital Library (ADL) project has been working on automating the processes of building ADL collections and gathering the collection statistics on which ADL's discovery system is based. As part of this effort, we have created a language and supporting programmatic framework for expressing mappings from XML metadata schemas to the required ADL metadata views. This language, based on the Python scripting language, is largely declarative in nature, corresponding to the fact that mappings can be largely—though not entirely—specified by crosswalk-type specifications. At the same time, the language allows mappings to be specified procedurally, which we argue is necessary to deal effectively with the realities of poor quality, highly variable, and incomplete metadata. An additional key feature of the language is the ability to derive new mappings from existing mappings, thereby making it easy to adapt generic mappings to the idiosyncrasies of particular metadata providers. We evaluate this language on three metadata standards (ADN, FGDC, and MARC) and three corresponding collections of metadata. We also note limitations, future research directions, and generalizations of this work.

1 Introduction

The Alexandria Digital Library (ADL) project[1] has been working to develop a lightweight, federated digital library architecture for heterogeneous georeferenced information. *Federated* means that distributed, autonomous libraries can work cooperatively to provide global discovery of and access to the union of their content. *Heterogeneous* means that a library can contain multiple types of information, including remotely-sensed imagery, textual documents, executable models, and multimedia instructional materials. *Georeferenced* means that, whenever possible, each

[1] http://www.alexandria.ucsb.edu/

item in a library is associated with one or more regions on the Earth's surface; this information is used by ADL to provide a spatial search capability. *Lightweight* means that the burden of setting up and running ADL is small enough that groups and systems that would not ordinarily be thought of as spatial data providers (traditional library catalogs, for example), nor capable of being spatial data providers (small digital library implementations lacking spatial engines, for example), can participate in a spatial system. ADL is not itself a source of spatial data; rather, it is a system that provides a spatial orientation to heterogeneous data sources.

ADL has historically focused on creating libraries and collections via mediation techniques that map library provider services to ADL's services, and that map native collection- and item-level metadata to ADL's common metadata model. The most developed of these techniques uses a schema-mapping language and configurable query translation software to enable virtually any relational database containing metadata to be viewed as an ADL collection. Less developed techniques include prototype gateways to the Z39.50 [1] and SDLIP [22] protocols, and mediators for ad hoc situations.

This service-level mediation approach has proven effective at bringing heterogeneous information sources into the ADL fold, but we have found that it can place an intimidating burden on library providers. To fully support ADL, library providers must run multiple services (both online services that provide search functionality and offline services that gather the statistics for ADL's federated collection-level discovery service), and these services can require complex configuration and substantial operational support. To offer a lighter-weight means of ADL participation, and to take advantage of the widespread availability of metadata via OAI-PMH [18], ADL has become interested in forming collections by directly ingesting both harvested and manually-submitted metadata. Such collection-building hinges on the ability to automatically map item-level metadata to the ADL metadata views (defined in the next section), a task that forms the subject of this paper.

Many other projects have taken a similar approach to collection-building, typically by mapping metadata to Dublin Core[2] and providing services over that common representation. The work we describe in this paper is necessarily specific to the ADL metadata views, but as will be seen, the ADL metadata views are more challenging to map to than Dublin Core, and thus the solution presented here has broad applicability.

2 Problem Statement

The ADL architecture [14] defines three item-level metadata formats that provide to clients a uniform view of ADL content and form the basis for the ADL services. Because ADL continues to store and make available the original source (or "native") metadata, and because the lineage of the mappings from native metadata to the ADL formats is explicitly represented, we refer to the ADL formats as *views*. To give some context to ADL's metadata mapping problem we briefly describe these views.

[2] http://www.dublincore.org/documents/dcmi-terms/

The *bucket view*[3] [13] aggregates and maps native item metadata into a few strongly-typed, high-level search indices, or "buckets." Bucket types, which govern the allowable syntactic representations and search operators, are extensible, as are the bucket definitions themselves, allowing collections to define collection-specific buckets. To support federation-wide interoperability, ADL defines 9 standard buckets that roughly correspond to Dublin Core elements, the key distinction being that ADL buckets are much more structured [7]. For example, the Dublin Core `coverage.spatial` qualified element suggests some encodings but ultimately may be populated by any free text, whereas the ADL spatial bucket type requires specific encodings of a handful of accepted geographic shapes.

The *access view* [16] describes the item's accessibility via zero or more "access points." Each access point corresponds to a single, independent representation of the item; different types of access points reflect fundamentally different modes of accessing content and capture the different kinds of information needed by clients to successfully use the access point. For example, a *download access point* simply returns a representation of the item, while a *service access point* allows the item to be accessed via a programmatic web service. Access points can be grouped into hierarchies to capture both alternative representations and decompositions of the item.

The *browse view* describes available browse-level representations of the item (image thumbnails, for example).

Our goal is to be able to programmatically map native metadata to these metadata views. Because most metadata mappings are declarative (for the most part, mappings are as trivial as "metadata field *A* maps to bucket *B*"), we want to be able to express mapping rules using a correspondingly concise and declarative language. However, not *all* mappings are declarative. In our experience, some mappings require recourse to a procedural language in order to express syntactic transformations (geographic coordinate and date conversions, for example) and to deal with the pervasive problems of metadata incorrectness, inconsistency, nonuniformity, and incompleteness. Fortunately, our experience has shown that such metadata problems are often relatively uniform within the context of a single provider, an observation echoed by Stvilia, et al [27], which calls for a language that can easily accommodate provider idiosyncrasies. Finally, validation of mappings is critical.

3 Related Work

Many other digital library projects have encountered the problems of assimilating distributed, heterogeneous metadata into a common framework; recent examples include NSDL [2] and OAIster [11]. Hillmann, et al, have been investigating more flexible ways of aggregating metadata from multiple sources [12]. These efforts and others have created many project-specific, ad hoc software solutions, typically implemented in Perl or XSLT, as opposed to a generic, reusable infrastructure.

Crosswalks—tables that map relationships and equivalencies between metadata schemes [29]—have been created between all the major metadata standards and are

[3] View definitions can be found at http://www.alexandria.ucsb.edu/middleware/dtds/.

being gathered into repositories [9]. Sathish, et al, have created a visual tool for creating crosswalks [24]. From our perspective, there are three principal limitations of crosswalks. First, the representation of metadata relationships, usually being limited to equivalence, is unrealistically simple. Second, crosswalks provide no *formal* means of describing complex transformations or handling idiosyncrasies. And third, crosswalks are not directly executable. Per Godby [9], "the record structure for the crosswalk object is a separate issue from the implementation details of metadata translation." Our work unifies these two aspects of metadata mapping.

Given that we are mapping metadata from one XML-encoded form to another, XSLT [3] suggests itself, and indeed, it is the tool most commonly employed for this purpose. However, we find XSLT deficient for two reasons:

1. The language is too low-level and verbose; it forces the mapping developer to work at the level of XML elements as opposed to the level of metadata fields.
2. The language is computationally hamstrung. Although it is Turing-complete in theory [17], it is notoriously difficult to express computation beyond simple template matching of XML structures. Coding the kinds of simple transformations that ADL must perform (e.g., the conversion of a geographic coordinate in degrees-minutes-seconds notation to decimal degrees) ranges from extremely difficult to practically impossible.

Several groups have attempted to overcome XSLT's deficiencies, both for general reasons and for the specific purpose of metadata mapping [10], by integrating it into a procedural language such as Java. However, this approach creates a large division between the declarative and procedural aspects of the mappings. Also, Java is generally too rigid and low-level a language for the purposes of metadata transformation; a higher-level scripting language is more appropriate.

Other groups have looked at creating new languages that operate over XML documents. The work of Manghi, et al, in developing a hybrid declarative/procedural language over XML is intriguing [19]. But operating on XML elements is fundamentally the wrong level of abstraction for our problem.

4 A Language for Mapping Metadata

We have developed a high-level language and supporting programmatic framework that allows metadata to be mapped at the level of source metadata fields and destination buckets and access points. Using the "piggyback" design pattern described by Spinellis [26], in which one language is implemented on top and in terms of another, our language is based on Python[4], a popular, open-source scripting language. A complete description of the mapping language is outside the scope of this paper[5]. In the remainder of this section we highlight the language's key features and characteristics; in the following section, where we evaluate the language, we provide a couple illustrative examples.

[4] http://www.python.org/

[5] A language tutorial can be found at http://www.alexandria.ucsb.edu/mm/tutorial.html.

A mapping from a native metadata format to the ADL metadata views is described by an executable Python script having the form

```
from ADL_mapper import *
input()
...declarations...
output()
```

The `import` statement loads the necessary infrastructure into the current Python workspace; the `input` statement reads and parses the native metadata; and the `output` statement performs all processing and outputs the ADL metadata views.

Between the `input` and `output` statements may be placed 24 different kinds of declarations that govern mappings and processing. Since the mappings are executable Python, these "declarations" are Python procedure calls. However, these procedure calls perform no immediate action, but only record the actions to be taken when processing finally occurs at the end of the script, and the calls can generally occur in any order. Thus the calls have all the essential characteristics of declarations, resulting in a language that is simultaneously both declarative and procedural in nature. Declarations can be emplaced in a procedural context (e.g., inside conditional constructs), and can embed and call on procedural code.

The bucket types, buckets, and associated vocabularies and thesauri are not predefined by the language, but can be declared within the language. (Each bucket type also requires a Python module adhering to a simple API to define validation and encoding of the type.) Using Python's module mechanism, these kinds of background declarations can be packaged into standard modules that a mapping simply imports.

There are several ways to specify mappings, but they all share the same general processing model. A query (described below) is executed to form tuples of metadata values which are then passed through a series of mapping-specified filter and converter functions. Filters perform arbitrary processing and may reject a mapping, while converters serve as pattern recognizers (for example, to process dates, one would create a converter function for each supported date format). Tuples that survive any filtering and conversion are passed to a validator. If valid, they are appropriately encoded.

To identify metadata fields in the source metadata document and to form the tuples used by the mapping framework, we have defined a simple query language built on XPath[6] that allows Cartesian products and "outer joins" [28] to be formed from relative and absolute XPath expressions and constant values. This query language is not as powerful as XQuery[7] or DSQL [25], but it has proven sufficient for our purposes and has the twin virtues of being compact and making simple queries simple to express.

Additional language features include the ability to express mapping-specified requirements (which generate errors) and expectations (which generate warnings) to be checked by the mapping framework, which are useful as sanity checks when processing large quantities of metadata. The language also supports "opportunistic" mappings, i.e., mappings that are performed when validation is satisfied but silently ignored when not.

---

[6] http://www.w3.org/TR/xpath

[7] http://www.w3.org/TR/xquery/

As described thus far, the language provides an easy and flexible means of describing mappings declaratively, and inserting procedural code as necessary. But perhaps the most significant language feature is an inheritance-like feature that allows mappings to be derived from other mappings. Derived mappings use Python's import mechanism and take the form

```
from ADL_mapper import *
input()
import parent mapping
...additional declarations...
output()
```

Derived mappings can thus add to parent mappings. In addition, the language provides declarations that allow the derived mapping to augment, undo, or override any parent mapping, requirement, or expectation. This makes it possible to succinctly express small refinements to comprehensive, generic mappings.

5 Evaluation

The mapping language was evaluated by creating mappings for several well-known metadata standards. These mappings were then exercised on collections of real-world metadata. The results of the mappings were visually inspected and, in the case of large collections, analyzed by software written for the purpose.

It should be noted that the mere *ability* to perform a mapping is not a criterion of our evaluation. Mappings can be created by any full-featured programming language, and in simple cases, by less powerful languages such as XSLT. Rather, our criteria are the qualitative considerations that apply to programming language design [23, 8]: writability and readability; brevity; expressiveness; appropriateness of the level of abstraction to the domain; and orthogonality.

In the following subsections we describe three evaluation tests[8].

5.1 ADN/DLESE

In a first test, we created a mapping for the ADN (ADEPT/DLESE/NASA) metadata standard[9], which has been used primarily by DLESE[10], and we exercised the mapping on 5,061 ADN records representing DLESE's entire corpus at the time of the experiment. DLESE exercises considerable review and quality control over its records, resulting in relatively complete and homogeneous metadata, and we had already performed a preliminary analysis of mapping ADN to the ADL buckets [15], so the mapping was easy to create. The mapping specifies 17 simple declarations to map ADN fields to 8 ADL buckets and 2 DLESE-specific buckets. The language fragment shown below maps ADN's notion of temporal coverage to the ADL `dates`

[8] The complete mappings can be viewed at http://www.alexandria.ucsb.edu/mm/.
[9] http://www.dlese.org/Metadata/adn-item/
[10] http://www.dlese.org/

bucket. For each ADN <timeAD> element, a (begin, end) time range tuple formed from the <begin> and <end> sub-elements' date attributes is mapped to the bucket. Some procedural code (a filter function) converts ADN's "Present" time keyword to ADL's numeric representation of the same concept.

```
def convertPresent (v):
    if v[1].lower() == "present":
        return (v[0], "9999")
    else:
        return v

map("adl:dates",
    ["/itemRecord/temporalCoverages/timeInfo/timeAD",
     "begin@date", "end@date"],
    prefilters=convertPresent)
```

For comparison, XSLT code that performs a subset of the same mapping is shown below. The XSLT version performs only a subset because it does not include the conversions, standardizations, duplicate eliminations and other consolidations, and validation that the mapping language implicitly performs.

```
<xsl:template
  match="itemRecord/temporalCoverages/timeInfo/timeAD">
  <temporal-value>
    <range>
      <begin>
        <xsl:value-of select="begin/@date"/>
      </begin>
      <end>
        <xsl:choose>
          <xsl:when
            test="translate(end/@date,
              'ABCDEFGHIJKLMNOPQRSTUVWXYZ',
              'abcdefghijklmnopqrstuvwxyz')='present'">
            <xsl:text/>9999<xsl:text/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="end/@date"/>
          </xsl:otherwise>
        </xsl:choose>
      </end>
    </range>
  </temporal-value>
</xsl:template>
```

5.2 FGDC/Maya

In a second test, we created a mapping for the MesoAmerican Research Center's[11] digital geographic database for the Maya forest region. This is a collection of approximately 60 images, maps, and models having metadata conforming to the ESRI

[11] http://www.marc.ucsb.edu/

profile [5] of the FGDC Content Standard for Digital Geospatial Metadata [6]. Because this collection uses a well-known metadata standard but also exhibits several problems endemic to real-world metadata, we exploited the mapping language's inheritance feature to develop both a generic mapping for the FGDC standard and a derived Maya-specific mapping. The generic FGDC mapping uses 23 declarations to map FGDC fields to the 9 standard ADL buckets. In addition, 10 converter and filter functions are used to convert different date formats and perform other minor transformations.

An interesting aspect of the FGDC mapping is how thesaurus terms are mapped. Mapping thesauri has been explored at length by the NKOS[12] and digital library communities. Our mapping language easily supports mapping terms from a source thesaurus to a destination thesaurus, using boolean combinations and partial equivalences as suggested by Doerr [4] and formalized by SWAD [21]. But in the case of ADL's `types` and `formats` buckets, there is often no comparable source thesaurus to map *from*; instead, a mapping must be inferred from various other pieces of information. For example, the vocabulary associated with the ADL `formats` bucket distinguishes whether an item is available online or offline (or both); if it is offline, whether it is non-digital or digital, and if digital, what type of media it is available on; and if it is online, in what formats it is available. There is no single FGDC vocabulary that captures all this information, so our FGDC mapping is really 5 mappings that look at different aspects of the item's distribution-related metadata. An example is shown below, where the presence of an online linkage is used as an indication that the item is available online. Other mappings look for more specific information.

```
if present("/metadata/idinfo/citation/citeinfo/onlink"):
    mapConstant("adl:formats",
        ("ADL Object Formats", "Online"))
```

In a post-processing step, the mapping framework gathers all such mappings and consolidates the information into a minimal set of terms. For example, if terms "Online" and "Image" both get mapped to the `formats` bucket, the "Online" mapping will be elided as a broader term of "Image." More sophisticated post-processing is possible via programmatic hooks.

As we noted previously, like many real-world collections, the Maya collection appears to adhere to a metadata standard but in fact exhibits a number of deficiencies—omissions, idiosyncrasies, obvious mistakes—that require correction:

1. Item titles are simple, extension-less filenames, e.g., "mex250kr." While possibly appropriate within the context of the collection (perhaps not even then), such cryptic titles do not serve users well in the larger context of a distributed digital library.
2. For some of the items, the FGDC element that describes how an offline, non-digital form of the item can be accessed erroneously contains a licensing agreement. Since the generic FGDC mapping uses this element as evidence that the item is available offline, this error causes the item to appear to be available offline when in fact it is not.
3. The name of the collection's creator, being an implicit piece of collection-level metadata, is not listed among the item-level originators.

[12] Networked Knowledge Organization Systems/Services; http://nkos.slis.kent.edu/.

Using the mapping language, we were able to correct these flaws as follows:

1. Item titles were improved by appending to the FGDC-specified title mapping a filter function that constructs a new title from the given title and several other pieces of information within the metadata.
2. The mapping of the erroneously-populated element was "unmapped" (disabled).
3. An additional mapping to the originators bucket of the collection creator's name (as a collection-wide constant) was added.

Thus by starting with a generic mapping for the FGDC standard, the mapping language's inheritance feature allowed us to create a mapping for the Maya collection with just a few simple declarations.

5.3 MARC/Maps

In a third test, we created a mapping for the MARC 21[13] bibliographic metadata standard (specifically, the MARCXML[14] encoding) and applied the mapping to 47,280 MARC records describing map holdings within UCSB's Davidson Library.

Our mapping is only a prototype; due to MARC's long history, all-inclusive nature (consider that it includes the entire FGDC standard as a subset) and wide range of applications, a complete mapping will take substantial effort to complete. The difficulty here is due simply to the large number of mappings required. With respect to evaluating the mapping language, though, we note that all mappings were particularly trivial to declare given MARCXML's flat structure.

5.4 Performance

The performance of the system is not stellar, but is adequate for our purposes. The time required to perform a mapping is dependent on a number of variables, chief among them the size and complexity of the input XML document and the amount of processing specified by the mapping. To give a rough order of magnitude, mapping a 12KB, 220-element ADN metadata record requires approximately one second on a 1.8GHz dual-processor PowerMac G5. Instrumentation revealed that from two-thirds to three-quarters of the processing time is taken up by Python's XML parser[15] reading and parsing the input document and building a complete DOM tree, and by writing the output XML document. Given that our declarative approach is intrinsically dependent on performing multiple XML queries over this DOM tree, there is little opportunity to significantly increase the performance of the system as presently implemented.

---

[13] http://www.loc.gov/marc/

[14] http://www.loc.gov/standards/marcxml/

[15] We use the PyXML package, available at http://pyxml.sourceforge.net/.

6 Future Work

Our evaluation of the mapping language demonstrated that the language is highly suitable for mapping descriptive metadata to ADL's bucket view. However, prototype mappings to the access and browse views were less successful.

Access-related information is difficult to handle for several reasons. To begin with, such information is poorly supported by metadata standards, if at all. ADN, MARC, and Dublin Core are capable of describing item access by simple URLs only, which is insufficient for the rich kinds of access mechanisms associated with geospatial and scientific data [16]. FGDC provides better support, but in our experience, its access-related fields are too unreliably populated to be of any practical use. METS[16] addresses some of these limitations, but we find there are subtle differences between describing the structure of an item and access to that item.

The underlying issue here, regardless of the native metadata format, is that access-related information is intimately related to both the intrinsic structure of the item and what we might call the item's *instantiation* or *emplacement* in the library, i.e., the item's relationship to the library services that provide access to the item. To effectively map access-related metadata, we believe the mapping language requires additional inputs that capture the context of library servers and services. In addition, validation (link-checking, etc.) of access-related mappings is critical.

7 Conclusion

In automating ADL's collection building and statistics gathering functionality, a need arose for automated mapping of item-level, XML-encoded metadata to the ADL metadata views. This metadata mapping problem can be characterized as a complex specification problem that is largely—but not entirely—declarative in nature, and for which it must be possible to perform arbitrary procedural computations over metadata values. In addition, to operate on real-world metadata any mapping framework must support validation of mappings and the ability to easily adapt mappings to the idiosyncrasies of particular metadata providers.

To answer this need we have developed a general framework for mapping metadata. In this framework, a mapping is implemented as a Python script, and individual mappings are expressed as procedure calls within the script. By delaying all processing to the end of the script, the procedure calls effectively become declarations and the procedures defined by the mapping framework effectively form a language. This framework allows mappings to be specified at a higher level than, say, XSLT, and provides many other features such as implicit value conversions and validation. The ability to derive mappings from more generic mappings facilitates idiosyncratic adaptations.

Our experience with this approach has shown that it is a good match for the problem. Mapping access-related metadata remains a challenge, however, and requires

[16] http://www.loc.gov/standards/mets/

that the mapping framework understand additional concepts such as servers and services, and that it be integrated into a library's ingest workflow.

Our approach has its limitations, of course. Although mappings are easy to express, artifacts of Python syntax intrude significantly; essentially, a mapping developer must also be a Python programmer. This is not a significant concern, for as we have argued in this paper, mappings of real-world metadata often require recourse to a procedural language to handle the various problems encountered, and Python is a better language than most for that purpose. Still, a custom mapping language could make mappings less syntactically stilted [20]. But then, such a language would require a custom parser, a substantial development task. By contrast, our Python-based language is implemented in fewer than 1,000 lines of Python code.

The metadata mapping language and framework software are currently specific to the ADL metadata views, but with only minor modifications both could easily be adapted to another mapping target, especially one requiring translation and validation of relatively structured metadata fields. More broadly, we believe that the hybrid declarative/procedural approach we have taken has applications in many other domains. Any specification problem that is largely—but not entirely—declarative in nature would be a candidate for this approach.

References

1. ANSI Z39.50-1995. *Information Retrieval (Z39.50) Application Service Definition and Protocol Specification*. http://www.loc.gov/z3950/agency/markup/markup.html.
2. William Y. Arms, Naomi Dushay, Dave Fulker, and Carl Lagoze. "A Case Study in Metadata Harvesting: the NSDL." *Library Hi Tech* **21**(2) (June 2003): 228-237. http://dx.doi.org/10.1108/07378830310479866.
3. James Clark (ed.). *XSL Transformations (XSLT)*. Version 1.0. http://www.w3.org/TR/xslt.
4. Martin Doerr. "Semantic Problems of Thesaurus Mapping." *Journal of Digital Information* **1**(8) (March 2001). http://jodi.ecs.soton.ac.uk/Articles/v01/i08/Doerr/.
5. Environmental Systems Research Institute (ESRI), Inc. *ESRI Profile of the Content Standard for Digital Geospatial Metadata*. March 2003. http://www.esri.com/metadata/esriprof80.html.
6. Federal Geographic Data Committee. FGDC-STD-001-1998. *Content Standard for Digital Geospatial Metadata*. June 1998. http://www.fgdc.gov/metadata/contstan.html.
7. James Frew and Greg Janée. *A Comparison of the Dublin Core Metadata Element Set and the Alexandria Digital Library Bucket Framework*. 2003. http://www.alexandria.ucsb.edu/~gjanee/archive/2003/dc-adl.pdf.
8. Carlo Ghezzi and Mehdi Jazayeri. *Programming Language Concepts*. 2nd ed. New York: John Wiley & Sons, 1987.
9. Carol Jean Godby, Jeffrey A. Young, and Eric Childress. "A Repository of Metadata Crosswalks." *D-Lib Magazine* **10**(12) (December 2004).
10. Damien Guillaume and Raymond Plante. "Declarative Metadata Processing with XML and Java." *Astronomical Data Analysis Software and Systems X*. *ASP Conference Series* **238** (2001). http://www.adass.org/adass/proceedings/adass00/O6-03/.
11. Martin Halbert, Joanne Kaczmarek, and Kat Hagedorn. "Findings from the Mellon Metadata Harvesting Initiative." *Proceedings of the Seventh European Conference on Research and Advanced Technology for Digital Libraries (ECDL)* (Trondheim, Norway; August 2003): 58-69.

12. Diane Hillmann, Naomi Dushay, and Jon Phipps. "Improving Metadata Quality: Augmentation and Recombination." *DC-2004: International Conference on Dublin Core and Metadata Applications* (Shanghai, China; October 2004). http://purl.org/metadataresearch/dcconf2004/papers/Paper_21.pdf.
13. Greg Janée, James Frew, Linda L. Hill, and Terence R. Smith. "The ADL Bucket Framework." *Third DELOS Workshop on Interoperability and Mediation in Heterogeneous Digital Libraries* (Darmstadt, Germany; September 2001). http://www.ercim.org/publication/ws-proceedings/DelNoe03/13.pdf.
14. Greg Janée and James Frew. "The ADEPT Digital Library Architecture." *Proceedings of the Second ACM/IEEE-CS Joint Conference on Digital Libraries (JCDL)* (Portland, Oregon; July 2002): 342-35. http://doi.acm.org/10.1145/544220.544306.
15. Greg Janée. *ADN Metadata Mapping*. October 2003. http://www.alexandria.ucsb.edu/~gjanee/archive/2003/adn-mapping.html.
16. Greg Janée, James Frew, and David Valentine. "Content Access Characterization in Digital Libraries." *Proceedings of the 2003 Joint Conference on Digital Libraries (JCDL)* (Houston, Texas; May 2003): 261-262. http://doi.acm.org/10.1145/827140.827185.
17. Stephan Kepser. "A Simple Proof for the Turing-Completeness of XSLT and XQuery." *Extreme Markup Languages 2004*. http://www.mulberrytech.com/Extreme/Proceedings/html/2004/Kepser01/EML2004Kepser01.html.
18. Carl Lagoze and Herbert Van de Sompel (eds.). *The Open Archives Initiative Protocol for Metadata Harvesting*. Version 2.0 (June 14, 2002). http://www.openarchives.org/OAI/openarchivesprotocol.html.
19. Paolo Manghi, Fabio Simeoni, David Lievens, and Richard Connor. "Hybrid Applications over XML: Integrating the Procedural and Declarative Approaches." *Fourth ACM CIKM International Workshop on Web Information and Data Management (WIDM)* (McLean, Virginia; November 2002). http://doi.acm.org/10.1145/584931.584935.
20. David Mertz. "Create declarative mini-languages: Programming as assertion rather than instruction." *Charming Python* (February 27, 2003). http://www.ibm.com/developerworks/library/l-cpdec.html.
21. Alistair Miles and Brian Matthews. *Inter-Thesaurus Mapping*. Retrieved February 22, 2005. http://www.w3.org/2001/sw/Europe/reports/thes/8.4/.
22. Andreas Paepcke, Robert Brandriff, Greg Janée, Ray Larson, Bertram Ludäscher, Sergey Melnik, and Sriram Raghavan. "Search Middleware and the Simple Digital Library Interoperability Protocol." *D-Lib Magazine* **6**(3) (March 2000).
23. Eric Steven Raymond. *The Art of Unix Programming*. Boston: Addison-Wesley, 2004.
24. K. Sathish, K. Maly, M. Zubair, and X. Liu. "RVOT: A Tool For Making Collections OAI-PMH Compliant." *Proceedings, 5th Russian Conference on Digital Libraries (RCDL)* (St. Petersburg, Russia; October 2003). http://RCDL2003.spbu.ru/proceedings/A5.pdf.
25. Arijit Sengupta1 and M. Dalkilic. "DSQL - an SQL for structured documents." *Proceedings, 14th International Conference, CAiSE 2002* (Toronto, Canada; May 2002): 757-760.
26. Diomidis Spinellis. "Notable Design Patterns for Domain-Specific Languages." *Journal of Systems and Software* **56**(1) (February 2001): 91-99. http://www.dmst.aueb.gr/dds/pubs/jrnl/2000-JSS-DSLPatterns/html/dslpat.html.
27. Besiki Stvilia, Les Gasser, Michael B. Twidale, Sarah L. Shreeves, and Tim W. Cole. "Metadata Quality for Federated Collections." *Proceedings of the 9th International Conference on Information Quality (ICIQ)* (Boston, Massachusetts; November 2004): 111-125.
28. Jeffrey D. Ullman and Jennifer Widom. *A First Course in Database Systems*. 2nd ed. Upper Saddle River, New Jersey: Prentice-Hall, 2002.
29. Mary S. Woodley, et al. *DCMI Glossary*. September 15, 2003. http://dublincore.org/documents/usageguide/glossary.shtml.