

META-METADATA: AN INFORMATION SEMANTICS LANGUAGE AND
SOFTWARE ARCHITECTURE FOR COLLECTION VISUALIZATION
APPLICATIONS

A Thesis

by

ABHINAV MATHUR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

December 2009

Major Subject: Computer Science

META-METADATA: AN INFORMATION SEMANTICS LANGUAGE AND
SOFTWARE ARCHITECTURE FOR COLLECTION VISUALIZATION
APPLICATIONS

A Thesis

by

ABHINAV MATHUR

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of
MASTER OF SCIENCE

Approved by:

Chair of Committee,	Andruid Kerne
Committee Members,	Rodney Hill
	Thomas Ioerger
	Jaakko Järvi
Head of Department,	Valerie Taylor

December 2009

Major Subject: Computer Science

ABSTRACT

Meta-Metadata: An Information Semantics Language and Software Architecture for
Collection Visualization Applications. (December 2009)

Abhinav Mathur, B.Tech, Indian Institute of Technology, Guwahati

Chair of Advisory Committee: Dr. Andruid Kerne

Information collection and discovery tasks involve aggregation and manipulation of information resources. An *information resource* is a location from which a human gathers data to contribute to his/her understanding of something significant. Repositories of information resources include the Google search engine, the ACM Digital Library, Wikipedia, Flickr, and IMDB. Information discovery tasks involve having new ideas in contexts of information collecting.

The information one needs to collect is large and diverse and hard to keep track of. The heterogeneity and scale also make difficult writing software to support information collection and discovery tasks. Metadata is a structured means for describing information resources. It forms the basis of digital libraries and search engines.

As metadata is often called, “data about data,” we define *meta-metadata* as a formal means for describing metadata as an XML based language. We consider the lifecycle of metadata in information collection and discovery tasks and develop a meta-metadata architecture which deals with the data structures for representation of metadata inside programs, extraction from information resources, rules for presentation to users,

and logic that defines how an application needs to operate on metadata. *Semantic actions* for an information resource collection are steps taken to generate representative objects, including formation of iconographic image and text surrogates, associated with metadata.

The meta-metadata language serves as a layer of abstraction between information resources, power users, and application developers. A power user can enhance an existing collection visualization application by authoring meta-metadata for a new information resource without modifying the application source code. The architecture provides a set of interfaces for semantic actions which different information discovery and visualization applications can implement according to their own custom requirements. Application developers can modify the implementation of these semantic actions to change the behavior of their application, regardless of the information resource.

We have used our architecture in combination, an information discovery and collection visualization application and validated it through a user study.

DEDICATION

To my family and friends

ACKNOWLEDGMENTS

I am extremely thankful to Andruid Kerne for his continuous encouragement, commitment, and direction. I have gained irreplaceable skills and experiences from his guidance.

I would like to thank my committee members Thomas Ioerger, Jaakko Järvi and Rodney Hill, for their suggestions and feedback during the course of my research.

I express my immense gratefulness to all the members of the Interface Ecology Lab for their help, suggestions, and support. I would especially like to Sashikanth Damaraju, Blake Dworaczyk and Andrew Webb for all the assistance and support provided.

Finally and most importantly, to my parents, I could have never accomplished this without you. Your unconditional love and support have given me strength in the most difficult of times.

TABLE OF CONTENTS

	Page
ABSTRACT	iii
DEDICATION	v
ACKNOWLEDGMENTS.....	vi
LIST OF FIGURES.....	x
1. INTRODUCTION AND PROBLEM STATEMENT	1
2. PRIOR WORK.....	6
2.1 Metadata-based Visualization System	6
2.2 Metadata Building and Extraction Approaches.....	7
2.2.1 Machine Learning and Automated Approach	7
2.2.2 Hybrid Approach: Machine Learning Combined with Manual Approach	8
2.2.3 Programming-based Approaches	9
2.3 combinFormation	10
2.4 Framework for Strongly Typed Metadata.....	10
3. ARCHITECTURE	11
3.1 Compile-time.....	15
3.1.1 Authoring Meta-metadata	16
3.1.2 Compiler: Translate Meta-metadata Declarations to Metadata Definitions	17
3.2 Runtime	20
3.2.1 Efficient URL Pattern Matching.....	21
3.2.2 Parser: Information Extraction Based on Extraction Rules	24
3.2.3 Semantic Action Handler.....	25
4. METADATA DEFINITION LANGUAGE.....	27
4.1 Specifying Strongly Typed Structures for Information Sources	28
4.2 Specifying Heterogeneous Sources by Reusing Existing Definitions	31
4.2.1 Sources with the Exact Same Structure.....	31
4.2.2 Sources That Add New Fields to Existing Defintion.....	32

	Page
5. METADATA EXTRACTION AND VISUALIZATION RULES.....	35
5.1 XPath-based Extraction Rules.....	36
5.2 Direct Binding for XML-based Services	37
5.3 Visualization Rules	38
6. SEMANTIC ACTION SCRIPTING.....	39
6.1 Variable Definition Statements	41
6.2 Control Flow Statements	42
6.2.1 Loop Statements	43
6.2.2 Conditional Statements	44
6.3 Metadata-operation Statements	45
7. USE CASES.....	46
7.1 Google Search	48
7.2 ACM Portal	50
7.3 Yahoo Search	54
7.4 Lines of Code Comparison.....	55
8. USER STUDY	57
8.1 Wikipedia	58
8.1.1 Authoring Meta-metadata for Search Results Page of Wikipedia	58
8.1.2 Authoring Meta-metadata for Article Page of Wikipedia	59
8.1.3 Feedback from Study	60
8.2 Flickr	61
8.2.1 Feedback from Study	62
8.3 IMDB	62
8.3.1 Feedback from Study	63
8.4 Observation	64
9. CONCLUSION AND FUTURE WORK.....	65
9.1 Expressive Meta-metadata Language.....	65
9.2 Minimal Programming Experience Needed.....	66
9.3 Future Work	66
9.3.1 Entity Resolution Problem	66
9.3.2 Changing DOM Problem	67
9.3.3 Meta-metadata Authoring Issues.....	67
9.3.4 Exploring Generality of Semantic Actions	68

	Page
9.3.5 Constraint specification in Metadata Definition Language	68
REFERENCES	70
APPENDIX A: META-METADATA LANGUAGE SYNTAX AND CONSTRUCTS	74
APPENDIX B: BNF FOR META-METADATA LANGUAGE	86
APPENDIX C: ADDING NEW SEARCH ENGINES.....	92
VITA	93

LIST OF FIGURES

	Page
Figure 1 Overview of Meta-Metadata Language	3
Figure 2 Control flow in basic MVC architecture.....	12
Figure 3 Control flow in proposed MVC-based architecture.....	12
Figure 4 Overview of data flow in Meta-Metadata architecture	18
Figure 5 Source definition and generated classes	20
Figure 6 Runtime data flow.....	21
Figure 7 Strongly typed, structured definition of search result set	30
Figure 8 Google search source definition using type attribute.....	32
Figure 9 Adding more fields to an existing definition using extends attribute.	34
Figure 10 XPath-based extraction from an HTML DOM.....	36
Figure 11 Direct binding for XML based web-services.....	37
Figure 12 Visualization rules for Meta-Metadata fields	38
Figure 13 Variable declaration statement.....	41
Figure 14 For loop to iterate over a collection	42
Figure 15 If statement for conditional execution	44
Figure 16 Google search result page	48
Figure 17 Smartphone,iPhone and Palm pre-browsing.....	49
Figure 18 Google search result set	50
Figure 19 Search fields extracted from Google search	51

	Page
Figure 20 ACM portal Meta-Metadata semantic actions	52
Figure 21 Information visualization from ACM portal using combinFormation	53
Figure 22 Yahoo search XML and corresponding Meta-Metadata.....	54
Figure 23 Comparison of lines of Java code vs lines of Meta-Metadata declaration.....	56
Figure 24 Search engine declaration for Wikipedia.....	59

1. INTRODUCTION AND PROBLEM STATEMENT

We introduce *Meta-Metadata*: integrated representations that describe how metadata can be extracted from information resources found in digital repositories and on the Internet, represented internally, acted on by software tools, and presented to users. We define *information resources* as locations that provide data to users for understanding something they experience as significant. An *information source*, in turn, is the template-based generalization of an information resource, which is provided as part of a particular repository or site. Examples of information sources include the Google search engine, the ACM Digital Library, Wikipedia, Flickr, and IMDB. A Google search result page, an article page for the ACM Digital library, an article page for Wikipedia, an image page on Flickr, or a movie page on IMDB are examples of information resources from these information sources.

Collecting, organizing, and thinking about diverse information resources is an essential step in all kinds of research. For example for research articles a user might need to extract the bibliographic information and represent them in a coherent way, including listing articles according to authors and place of publication. Another example includes browsing through an image collection like Flickr and collecting images of interest. Collection management and utilization becomes hard, for end users, as the size of collection increases. Further, heterogeneity of information resources makes it difficult to write software to support information collection and discovery tasks. Koh and Kerne show that people engage in tasks of information collection and visualization inspite of

This thesis follows the style of the *International Journal of Human-Computer Interaction*.

the breakdowns they experience with collection tools (Koh and Kerne, 2006). The study investigated people developing collections for both entertainment purposes, like movie collections, and collections used for academic purposes, like prior work surveys of scholarly articles. In either case, as the collection sizes increases, collections become difficult to manage and utilize. Koh and Kerne recommend assisting end users in associating powerful semantic structures with the documents they collect, because semantic structures will help end users organize, make sense of, and remember the documents and document relationships.

Metadata, data about data, is a structured means for describing information resources. We use metadata as semantic structure in information collection and discovery tasks, and define Meta-Metadata to specify structures for representation of metadata inside collection tools, extraction from information resources, rules for presentation to end users, and logic that defines how a collection tools should operate on metadata.

The *long-term goal* of this research is to build creativity support tools that facilitate collecting and visualizing information in cognitively beneficial forms, and to support a community of developers who want to build such tools. Thus, the *objective of this thesis* is to design and develop a language and architecture that facilitates building a semantic web from World Wide Web through specification of how to bind semantic data from heterogeneous information sources with strongly typed data structures and how to operate on these data structures. We call this XML-language *Meta-Metadata*. We define the *semantic actions* for an information resource as operations performed to generate,

visualize, navigate, and crawl information resource objects. Such actions include creating iconographic image and text surrogates that represent the main ideas of each information resource and provide access to it. This research will build software infrastructure, in the form of XML representations of Meta-Metadata for various information sources, and APIs that facilitate operation on that metadata, independent of the information source. The framework will be extensible, so new information sources can be added without affecting application code. We will integrate computational representation, serialization, extraction rules, and attributes of interactive visualization.

```
<meta_metadata name="scholarly_article" extends="pdf" comment="">
  < DEFINITION OF METADATA, THE SEMANTIC STRUCTURE ASSOCIATED WITH SCHOLARLY ARTICLE />
  < DEFINITION OF SEMANTIC ACTIONS, OPERATIONS TO BE PERFORMED ON SCHOLARLY ARTICLE METADATA/>
</meta_metadata>
```

Figure 1: Overview of Meta-Metadata Language

Figure 1 shows, an overview of information source definition using the Meta-Metadata language. `meta_metadata` tag is used to define an information source. `meta_metadata` tag contains attribute name which specifies the information source. Further there are nested tags which define the metadata for information source and semantic actions to be taken on metadata objects.

The *principal hypothesis* is that an architecture that abstracts the ability to specify the semantics of information collection from information visualization application will facilitate enhancement of existing visualization applications by adding

new information sources. Such architecture will enable one set of developers to concentrate on requirements analysis design, visualization, and use-context involving particular information sources. We call this set of developers as power users. A separate set of developers can focus on more generalized programming for information collection, crawling, processing, and visualization. We call this set of developers as application developers. The architecture is aimed at both these groups of people.

Applications for personal collection visualization provide interfaces to navigate, browse, search and interact within a set of information objects. Architectures of such applications typically include closely tied visualization interaction and manipulation layers, which make them specific to particular information sources. For example, *combinFormation* (Kerne et al. 2008a) is, a creativity support tool that provides users with the ability to search, browse, collect, mix, organize, and think about information. The software previously had custom code for information sources such as Google, Yahoo, and Flickr. Thus, it required developers to be acquainted with the software internals to write more code for new information sources. We use the present *Meta-Metadata* based architecture to re-architect *combinFormation*, thus separating the information collection layer and information visualization layers and enabling power users who are not application developers to add new information sources. At the same time, the *Meta-Metadata* support library is completely separate from *combinFormation*, and so is ready for deployment in other Java applications.

We will develop software architecture for *Meta-Metadata* and its information semantics through these specific aims:

Aim 1: Specify a Metadata Definition Language, to represent different information sources using *Meta-Metadata*, including strongly-typed structure for information fields. A Generative Programming approach will be used for generating strongly typed procedural objects that correspond to metadata from the information sources described using the language. This minimizes custom code writing for each digital collection.

Aim 2: Defining a language for specifying rules for information extraction and presentation, and developing software modules that can be used to extract information from information sources based on extraction rule statements.

Aim 3: Develop an extensible language for semantic actions that specify how collection visualization applications should operate on metadata extracted from a particular source. This will include definition of interfaces for actions to be taken on extracted information and providing programming language control flow structures to reuse these actions.

Aim 4: Validating the architecture through case studies involving *combination*.

Thus we propose to build an abstract layer to provide ways to specify information extraction semantics from various information sources irrespective of the format used to store the information. Actions taken on extracted information, to build semantic connections, will be expressible in a language, which is written in XML.

2. PRIOR WORK

There is a vast range of research done on the ways to visualize the collection and interact with them. The main area in which research has concentrated includes various visualization systems (Bier and Perer, 2005) and ways to extract metadata for these systems (Hetzner, 2008). Various approaches have been proposed to extract and build metadata for information visualization system. These include using machine-based learning techniques, manual entry of metadata, use of hybrid approaches (both machine learning and manual), and programming-based approaches to extract metadata from heterogeneous sources. However, none of these approaches provide an extensible software framework that can be used by developers to build new information visualization application. Our architecture provide software infrastructure that enable a community of developers to build visualization application while providing further scope to automate the definition extraction rules and semantic actions for new information sources.

2.1 Metadata-based Visualization System

In this section, we discuss some of the information visualization systems and the underlying base, metadata, for such systems. Information collection from digital libraries requires effective content access functionalities. There is a need for a coherent way to access, browse, and collect information from diverse information sources. Metadata enables easy discovery of materials and collection browsing. Icon Abacus (Bier and Perer, 2005) is a technique for metadata visualization in 2D space. It uses one axis to display collection according to a selected metadata attribute value in a grid layout, while

other axis can be used to select other possible metadata attributes. For example a collection of documents can be sorted vertically according to date of publication. Each of vertical section can then be sorted alphabetically according to author names. The horizontal axis can be used to display reading status of document, like unread, read, read soon and read latter. Thus it creates a visualization of three attributes: date, reading status and author, in a 2D plane. VITE (Hsieh and Shipman, 2000) is an interface which allows users to create their own visualizations and manipulate structured information. TimePeriodDirectory (Petras et al., 2006) is a metadata infrastructure for the Library Congress Subject Heading (LCSH) that links data with its canonical time period range as well as geographic location. Metadata also tells us about the quality of data. In a scholarly article digital library, the bibliographic information about an article can tell us about its importance. Shiri suggests that metadata can be used to provide a richer *information collection* experience for the user from a digital library (Shiri, 2008).

2.2 Metadata Building and Extraction Approaches

There has been a considerable amount of research on the ways to extract and build metadata from a digital library. Broadly, these approaches can be classified as automatic extraction and manual extraction of metadata.

2.2.1 Machine Learning and Automated Approach

Machine learning approaches focus on automating the task of metadata extraction from information sources. Cui discusses machine-learning techniques for semantic markup of biodiversity digital libraries like eflora.org and algaebase.org (Hetzner, 2008). They use unsupervised learning technique and achieve an accuracy of

99% to 99.5%. Lu et al. use supervised learning techniques to generate metadata for bound volumes of scientific publications (Lu et al., 2008). They have used their approach in the Biodiversity Heritage Library. FLUXCiM (Cortez et al., 2007) is a knowledge-based system, which uses unsupervised learning to extract correct components of citations given in any format. It gives above 94% of accuracy but needs an existing set of sample metadata records from a given area to construct the knowledge base. Hetzner gives an approach to extract citation metadata using Hidden Markov Model (Hetzner et al., 2008). CLiMB (Klavans et al., 2008) uses text mining to get high-quality metadata for images in digital library. TableSeer (Liu et al., 2007) is a search engine for tables that detects tables from documents, extracts metadata, and indexes and ranks the tables. MetaExtract (Yilmazel et al., 2004) is a Natural Language Processing system to automatically assign metadata to the data. Hui et al. uses Support Vector Machine to extract metadata (Hui et al., 2003). These methods can act complementarily to our framework by helping to automatically define new information sources.

2.2.2 Hybrid Approach: Machine Learning Combined with Manual Approach

Hybrid approaches combine the advantages of automatic extraction with human interference, to ensure the accuracy. HarvANA (Hunter et al., 2008) is a hybrid approach for merging the manual metadata with the metadata generated using community tags. This is currently implemented only for *pictureaustralia*. PaperBase (Shiri, 2008) is also a hybrid approach in which the system automatically extracts metadata and populates a web form. The user can then proofread and correct it. Hybrid approaches present a tradeoff between expensive but accurate manual entry and inexpensive but less accurate

automatic extraction. Our approach is unique from both manual and automatic approaches as power users will author the Meta-Metadata and then metadata extraction will be done based on these authored definitions. Further, these definitions can be shared with other people for collaborative tasks.

2.2.3 Programming-based Approaches

Programming-based approaches provide a set of users ability to programmatically specify information extraction mechanism. Instructional Architect (Recker and Palmer, 2006) is an end user tool to access and use NSDL. It is used to find and gather NSDL and web resources to create and share personal collections of information. MarMite (Wong and Hong et al., 2007) is an end user programming tool that enables the creation of mashups. For end user programming, it uses strongly typed data and provides a graphical dialogue box. It is very similar to our approach, but since it requires end user programming, there is a learning curve. Dontcheva et al. present a system for collecting, viewing and sharing information from the web (Dontcheva et al., 2006). It uses extraction rules similar to our system for information extraction and collection. However, it does not define the concept of metadata explicitly. Power users cannot define and specify semantic actions on metadata, like sending the document links for the cited articles to the collecting agent to be crawled latter. Also, the system has strongly tied semantic and visualization layers. Thus, the user cannot use the visualization software of his choice for information collection. Exchange Center (Bainbridge et al., 2006) is a software environment that helps in managing, exploring, and building collections from various repositories. Yaron et al. present an approach for building cross-disciplinary collections among digital libraries

(Yaron et al., 2008) . However, digital library interoperability often requires a custom programming solution. Using our approach power users does not have to write custom code for information collection and visualization from heterogeneous information sources and our system is unique in providing a framework that can be used by other developers to build their own visualization applications.

2.3 combinFormation

combinFormation is a mixed-initiative system which is used for searching, browsing and collecting information in the form of a visual collage consisting of image and text surrogates from web pages and other documents (Kerne et al., 2008). It provides a composition space to build this collage. The composition space functions as a medium of communication between human and agent, to collaboratively engage in the tasks of information discovery. We have applied our Meta-Metadata language and architecture to re-architect *combinFormation* and author new information sources for it.

2.4 Framework for Strongly Typed Metadata

ecologylab.xml (Kerne et al., 2008) is an object-oriented XML binding framework for connecting programming language objects with their serialized XML representation. It involves writing annotated classes with a metalanguage, which can then bind to XML documents to create strongly typed objects. We will use *ecologylab.xml* both for reading Meta-Metadata declarations and generating strongly typed metadata objects. Objects on both levels will be serialized and stored using *ecologylab.xml*.

3. ARCHITECTURE

We developed a software architecture that streamlines the integration of heterogeneous typed metadata into interactive applications. Current approaches to metadata semantics have drawbacks that make it cumbersome to develop general collection visualization applications. Most approaches require custom code to collect metadata from heterogeneous information sources. It is expensive for programmers to manually enter metadata or to write custom code for each information source. This is not feasible for large collections. As far as we are aware, there are no tools that enable the integrated customization of visualization and operations on metadata. We present an architecture that enables developers to change the way metadata is structured, visualized, and operated on in applications without writing custom code for each source. We expect that the flexibility gained through the use of this architecture will enable developers to focus on more important research issues, such as supporting creativity for information discovery (Kerne et al. 2008). Thus, for the purpose of building tools for facilitating the collection and visualization of information in cognitively beneficial forms, architecture is needed that addresses the tasks of information collection, information presentation, and operation through a series of distinct modules.

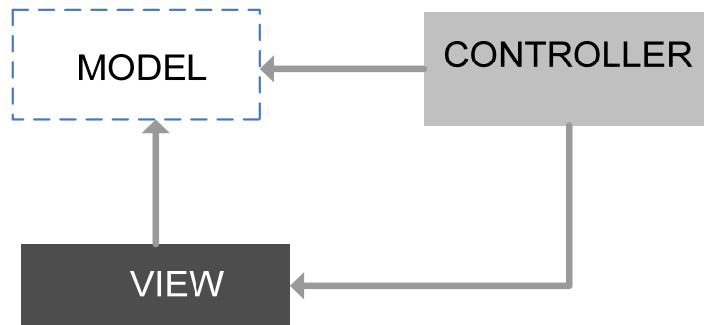


Figure 2: Control flow in basic MVC architecture

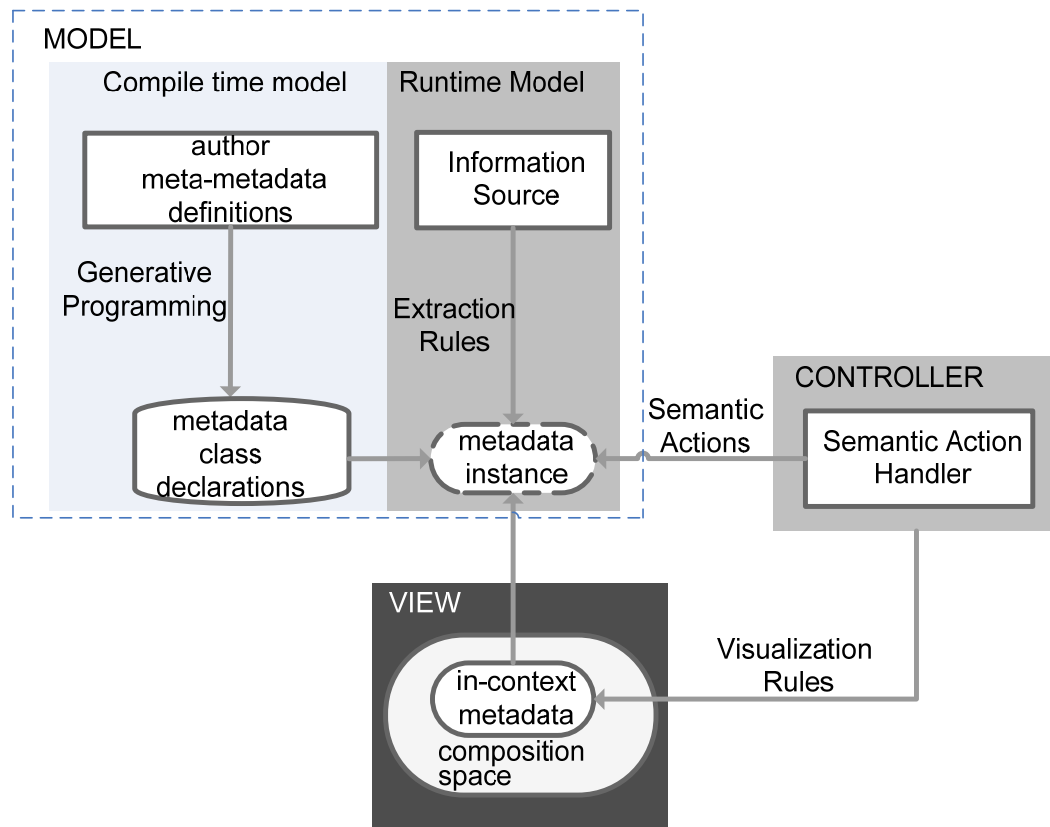


Figure 3: Control flow in proposed MVC-based architecture

In this section, we introduce our architecture while drawing analogies to the Model-View-Controller (MVC) paradigm. The architecture defines modules for information representation, extraction, presentation, and operation. The MVC paradigm defines three distinct components of software: the model is the data being operated on, the view is responsible for the visualization, and the controller describes the control flow of the data and application logic within the software. The MVC paradigm isolates the control logic from the visualization rules, allowing one component to be modified without affecting the other. These three distinct MVC components correspond to the three tasks of information: extraction, visualization, and operation.

Figure 2 shows a traditional MVC model. Figure 3 illustrates how our architecture is based on MVC. The model in our architecture consists of two kinds of components: compile-time and runtime. The compile-time model initially defines the structured and strongly typed definition of information sources. These compile-time specifications define the structure of heterogeneous information sources for data for the application. These structures are then used via generative programming to derive strongly typed classes for the sources

The runtime model consists of instances of these class definitions, which we build during runtime by extracting information from the sources. These instances are then acted upon by the controller. The controller in our architecture defines the semantic actions that operate on the runtime model. These actions are defined as interfaces that applications can then implement according to their custom logic to build data structures for visualizations, while using the runtime model. This abstracts out the controller logic from its

implementation. View in our architecture consists of the information visualization semantics that can be used by applications to drive user interface software. This is analogous to the traditional View of MVC, which refers directly to the user interface layer. Based on the view specified the visualization applications can then build their own custom presentations of metadata. `combinFormation` does this by presenting appropriate in-context metadata with surrogates in a visualization composition space (Kerne et al., 2008).

For this purpose, an abstract layer is defined to describe the structure of source definitions (Model), information about their presentation (View), and logic for actions on them (Controller). It provides a coherent way to extract heterogeneous and strongly typed metadata in a structured format from various information sources. Metadata extraction is independent of the information collection visualization and the information source, so custom code does not have to be written. Application programming by the developer is reduced because we use generative programming to author metadata classes based on Meta-Metadata declarations. Operations on metadata objects are also expressible in this layer.

Our architecture specifies an XML-based *Meta-Metadata* language for defining rules for information collection, extraction, binding visualization, and operations that use metadata. Figure 3 shows a block diagram of the major modules of our architecture. We maintain a repository (Mathur and Kearne, 2009) containing Meta-Metadata definitions, shown in Figure 3. This repository can be shared among applications for reuse in performing their information collection tasks. To use a new information source that is not in the repository, power users author the Meta-Metadata definitions for it with the Meta-

Metadata language. Meta-Metadata definitions specify strongly typed and structured metadata fields, extraction rules, visualization rules, and semantic actions for them. These are used by the compiler module to generate metadata class declarations during compile-time. During runtime Meta-Metadata definitions are translated into Meta-Metadata API objects using the ecologylab.xml framework. These objects are then used by the parser module to derive extraction rules to parse the templated information source and form metadata instances. Visualization applications use Meta-Metadata objects to reference visualization rules, to guide presentation to the user. The Semantic Action handler module acts on metadata objects by obtaining the semantic actions from Meta-Metadata API objects. The semantic action handler exposes a set of interfaces, which are used to define these productions. Different information collection software can then implement these productions for their application-specific logic. These are two phases of execution: compile-time and runtime collectively provide a coherent way to extract information from various digital libraries to build strongly typed and structured representations.

3.1 Compile-time

The compile-time phase consists of two parts. The first part involves manual authoring of strongly typed structured data definitions for various heterogeneous information sources. These definitions are authored by power users using the Meta-Metadata language. They include defining various search engines with their search URLs, definitions for digital libraries like Flickr, ACMPortal, Wikipedia, and IMDB

The second part involves generation of classes from these definitions that can be used in a procedural programming language (currently Java). These definitions are

generated automatically by using our compiler, which can be invoked as a standalone module. Generation of these classes does not require programming knowledge.

3.1.1 Authoring Meta-Metadata

Information discovery tasks require accessing heterogeneous information sources, representing information from these sources with typed metadata, extracting information into appropriate data structures, and acting on extracted information by forming objects such as surrogates, which are then presented to the user.

Information researchers often look for common kinds of data from an information source. For instance, to obtain reviews of restaurants, a user might visit TopTable.com, UrbanSpoon.com, OpenTable.com, or another restaurant review site. All these sites provide information fields such as restaurant menu, rates, bookings and overall ranking. Thus, the common fields of interest from all these sources are the same: menu, rate, overall ranking. Other examples of multifaceted data from heterogeneous sources include obtaining reviews for hotels and for movies. All different sources that provide such information contain common fields of interest such as name, ranking of hotels, and movie actors. To obtain and operate on information from each of these sources, information discovery software requires a definition of these sources in the form of procedural programming classes. But since all these sources have common fields of interest about the information, writing individual classes for each source would be a repetitive programming task. To represent these common metadata fields of interest in a consistent manner for heterogeneous sources, we created a specification for easy-to-read and easy-to-write XML

definitions. These definitions consist of strongly-typed hierarchical structures of metadata fields. They act as the compile-time model for our architecture.

In Figure 4, as a precursor to compile time, the user-authored definitions of information sources are stored in a Meta-Metadata repository. These definitions are authored using metadata definition language that we will develop in Chapter 4. This language provides the ability to specify heterogeneous information sources in strongly typed structured forms. Since information sources are accessed by Universal Resource Locators, each of these definitions includes the assignment of a distinct URL key, which is then used during runtime to uniquely retrieve the matching Meta-Metadata

3.1.2 Compiler: Translate Meta-Metadata Declarations to Metadata Definitions

The compiler module as shown in Figure 4 operates on power user-authored Meta-Metadata declarations. It can be invoked as a standalone utility to generate strongly typed metadata classes in any object-oriented programming language, which is Java in the current implementation. The compiler module then operates on these definitions and uses generative programming to author classes for each of the sources. These classes are annotated with `ecologylab.xml` meta-language. This makes it easy to marshal objects to XML and to unmarshal XML to objects. The compilation of Meta-Metadata-authored definitions to produce metadata classes is a necessary precursor for information extraction.

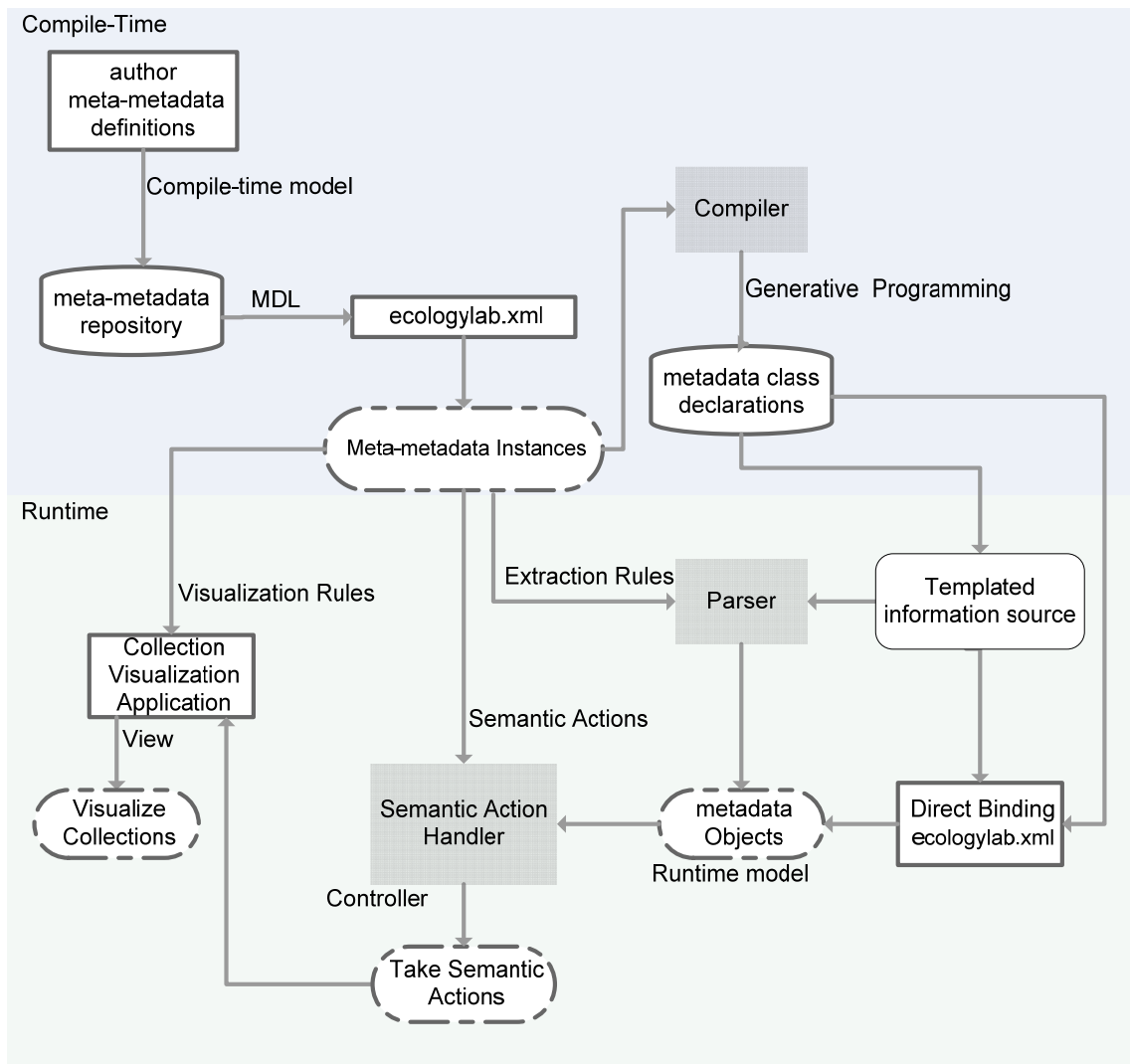


Figure 4: Overview of data flow in Meta-Metadata architecture

The compiler translates authored Meta-Metadata declarations in XML, from a repository, to Meta-Metadata objects using the `ecologylab.xml` framework. In turn, it translates these live objects to output *metadata* class definitions as Java source code. We use the `ecologylab.xml` framework to define both the Meta-Metadata and metadata level object levels, providing us with automatic access to XML data through typed programming language objects that can be again marshaled to XML. The generated metadata classes are also annotated using `ecologylab.xml` meta-language, and thus can also be serialized and saved. The compiler also generates a translation scope for all of the generated metadata classes, which is used to bind Java metadata subclasses to XML elements while loading the saved collections. Our current implementation produces metadata classes in Java. Future versions will also support other languages such as Objective C. The compiler module generates Java-Doc comments for each of these classes to help make them readable. These classes can be used in any structured procedural programming language like Java to obtain instances of strongly typed metadata objects.

Figure 5 shows Meta-Metadata declarations of the search class with scalar and nonscalar fields, and the Java code generated for them. Scalar fields include primitive types like boolean, string, integer, and Parsed URL. Nonscalar types include nested class data types, and also these like ArrayList, which are used for collections. In this figure we generate an ArrayList to represent a set of results, each of which is of type SearchResult. We also generate the SearchResult class, which contain all the metadata fields declared in the Meta-Metadata field of searchResults

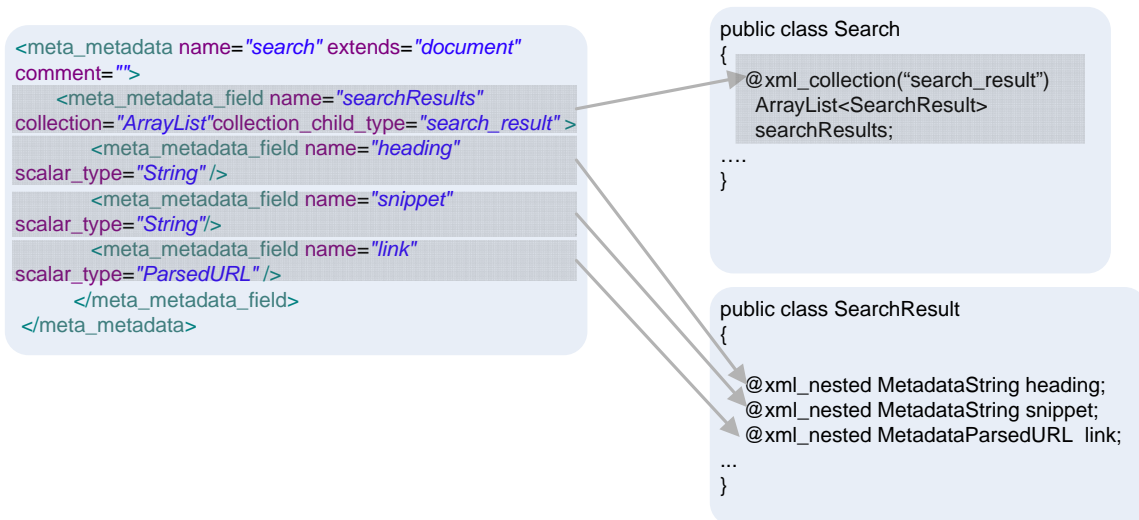


Figure 5: Source definition and generated classes

3.2 Runtime

At runtime, the visualization application uses Meta-Metadatas and metadatas to define, extract, and present structured collections to the user. Figure 6 presents an overview of the runtime control flow. When a URL to an information source is encountered, it is first matched against the URL patterns of various source definitions in the Meta-Metadatas repository, to find the appropriate matching Meta-Metadatas. The parser module then parses the information source based on the extraction rules obtained from selected Meta-Metadatas, extracts information, and populates the metadata objects. These metadata objects are then acted upon by semantic action handler module, according to semantic actions specified in the selected metadata.

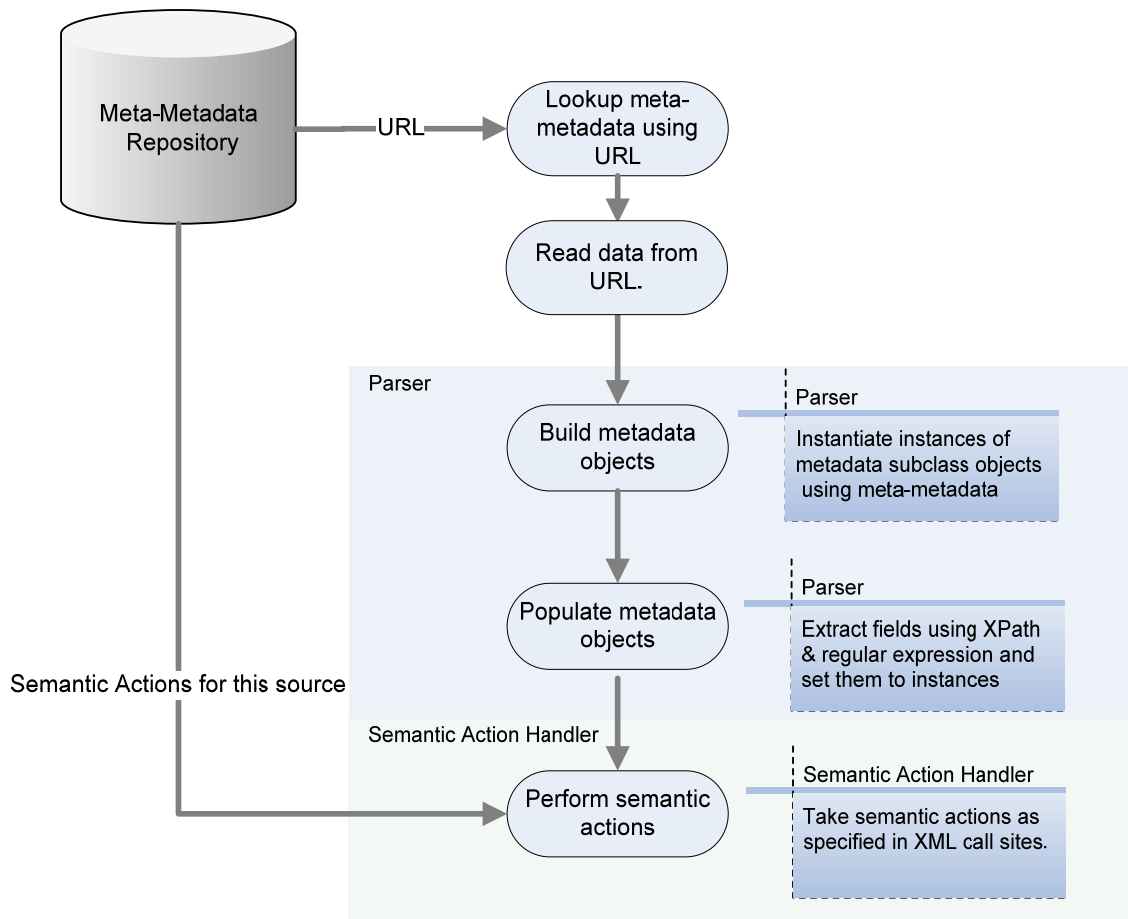


Figure 6: Runtime data flow

3.2.1 Efficient URL Pattern Matching

In Meta-Metadata repository, the Meta-Metadata definitions are indexed by URL pattern. To obtain Meta-Metadata for a particular URL, we match it against the URL key in the Meta-Metadata repository. To account for the diverse structures of URLs used by different template-based websites, multiple URL matching mechanisms are needed. As this matching is performed for each URL encountered, efficient data structures and algorithms are required.

Meta-Metadata URL pattern matching enables the use of three different mechanisms for authoring the URL key. Each of the three different kinds of keys has a different lookup time complexity. One mechanism is based on what we call no query URL, meaning a URL with its query part stripped. For no query URL, we maintain a hash map of Meta-Metadata with the no query URL as the key for this mechanism. The lookup takes $O(1)$ time to find matching Meta-Metadata. For instance, for Google Search URL (<http://www.google.com/search?q=QUERY¹>), we use no query URL (<http://www.google.com/search>) as the key for Google search Meta-Metadata. This is done because all Google searches, irrespective of the search query, will give the same no query URL, and thus it can be used as a key for looking up its Meta-Metadata. Similarly, for Flickr search, the search URL is <http://www.flickr.com/search/?q=QUERY> and so no query URL (<http://www.flickr.com/search/>) can be used as key for Flickr search Meta-Metadata. Since query URL is the fastest mechanism for URL lookup whenever possible no query URL should be used as the key in a Meta-Metadata declaration.

In some cases, no query URL is inadequate as a key for defining Meta-Metadata lookup. For example for a Flickr image result page, the URL pattern is http://www.flickr.com/photos/AUTHOR²/IMAGE_ID³/, and the no query URL is <http://www.flickr.com/photos>. However, for Flickr, there exists another family of pages of interest, those with all the photos by a particular author, which have a URL of the form <http://www.flickr.com/photos/AUTHOR/>. The no query URL for this is also the same:

The Query String

² Author of the image

³ Unique identifier for the image

<http://www.flickr.com/photos>. Thus, in such cases, no query URL cannot be used as a key to uniquely identify the Meta-Metadata. In such cases, we use the URL prefix with wild cards to define the URL key for Meta-Metadata. Thus, the URL key for Flickr image result page is http://www.flickr.com/photos/*/*/, and for Flickr author page, it is http://www.flickr.com/photos/*/. Here we use '*' as a wild card that matches anything between two '/' characters. To store patterns like these, the runtime infrastructure uses an efficient data structure "Prefix Collection," which is a list of nested hash maps, using the fact that the URL patterns can be separated into multiple sections, using the '/' character. The root of the URL pattern is the domain. This section is used as the key in first hash map. If no more patterns exist with the same domain, the corresponding value for this key will be the Meta-Metadata object itself. If there are multiple URL patterns with the same domain, the corresponding value for this key will be another hash map. This inner hash map will contain keys with the next section of the URL pattern, derived by using '/' as a separator. This strategy of nesting the hash maps will keep the sections of the URL pattern that were separated. Normally, the value of this n will be of the range 2 or 3; thus, the lookup time for URL prefix key will be $O(2)$ or $O(3)$, which is also very fast but slower than the no query URL key.

Some URLs, like the source of a Flickr image, can have a number of different patterns such as http://farm3.static.flickr.com/2020/2118178242_27fb91853a_m.jpg or http://farm4.static.flickr.com/3477/3938205388_331febf7a_m.jpg. For these kinds of URL patterns, we cannot use no query URL or URL prefix as a key. For such URLs, we use regular expressions to define the URL key for Meta-Metadata. For the given example,

the URL key is `http://farm[0-9].static.flickr.com`. This regular expression matches all the URLs for the source of a Flickr image. Since regular expressions are expensive to evaluate, we maintain a list of compiled regular expression, and further keep these in a hash map with domain as key, for each domain. When a URL is encountered, and the lookup no query URL and URL prefix fail, we get the list of compiled regular expressions for the URL domain and match the URL against each one of them. Time complexity in this case is $O(\text{list_size} + \text{time for matching})$, so it is the slowest URL lookup mechanism. It should be used only when a source key cannot be defined using no query URL and URL prefix. As long as the number of these patterns per domain is not large, this strategy will be sufficient.

During runtime, the parser module applies extraction rules on the information source to extract the metadata fields as specified by the Meta-Metadata. The extracted information is then bound to the generated classes, generated by Meta-Metadata compiler to produce instances of strongly typed structured metadata objects. This makes metadata extraction independent of the information collection software used.

3.2.2 Parser: Information Extraction Based on Extraction Rules

Information sources present their information using either human-readable Web-based HTML documents or from XML-based Web services. For example, Google Search presents its results Web page. Yahoo Search and Flickr provide semantic XML-based Web services for their search engines. When accessible, semantic web services are the preferred representation, because they are designed for computational data exchange. In either case, we need infrastructure to extract information from sources, bind it to the compile-time

model, and instantiate information objects, which function as the runtime model. For example, for Google Search, we need to parse the HTML page and extract information for each search result. This extracted information then needs to be bound to generated classes using reflection creating the runtime model. For Yahoo or Flickr Search, we need infrastructure to directly bind the semantic XML information to the annotated generated classes, creating the runtime model.

So after the compiler, the second module, the Parser, is used to extract the information from information sources and bind it to instances of the generated classes. The parser extracts information from sources and uses the compile time model to build the runtime model. For Web-based interfaces like Google, it involves downloading the Web page and building an HTML DOM from it. Extraction rules in the forms of XPath and regular expressions are applied to the DOM to extract information. This is then bound to appropriate Java classes generated in the previous step to obtain metadata information objects using reflection. For XML-based Web services, we directly use the ecologylab.xml framework. This framework allows us to bind XML to generated classes, which are annotated with meta-language provided by it. The parser thus instantiates objects using the compile time model and information sources during runtime, thus building the runtime model in the form of metadata instances. This runtime model can then be utilized by visualization software or can be serialized and saved.

3.2.3 Semantic Action Handler

Visualization applications need to act on information objects to create presentations and visualizations and manipulate them. Semantic actions specify the

generic actions that must be performed on information sources in order to use them in visualization applications. For instance, for Google Search metadata objects, we need to connect and download and process the result document for each item in the result set. These result documents can then again be acted upon based on their document types. For image documents, it will involve forming image surrogates and visualize them. This is the case with image collections like Flickr. In other cases such as while processing a document with many useful citation links, it involves creating nested document containers as parts of the citation chain graph and enabling future parsing of these citations by an information collecting agent.

For this purpose, we declared the Semantic Action Handler module. It is the controller of our MVC-based architecture. It provides a set of method declarations as interfaces, which can be invoked through declarations in XML, using the Meta-Metadata language. This XML file is the call site for the operations that need to be performed on metadata objects. Applications can then implement these methods according to their needs. The runtime infrastructure operates on the runtime model using the actions invoked in XML and delegates the implementation of these actions to application-specific handlers. A power user can author the XML specifying the call sites and logic for actions on metadata without changing or having knowledge of the application code. An application developer can focus on implementation of these actions according to needs of task contexts. This enables the separation of application logic from the semantic processing logic.

4. METADATA DEFINITION LANGUAGE

Metadata definition language (MDL) allows power users to author structured and strongly typed declarations for heterogeneous information sources. These source declarations are stored in a Meta-Metadata repository which is then used during compile time to generate metadata classes and during runtime to get the extraction rules, visualization rules and semantic actions for the source.

Many information sources have structural similarity. For instance, all search results have a title, snippet, and result link field. Defining different sources having common fields with same data type makes the source definition task cumbersome. Thus, MDL should provide an ability to reuse the existing source definitions for defining new sources that have the same structure. For instance, we can define the search definition with title and snippet of type String and result document link of type URL. Other search sources like Google and Flickr, which have exactly same structure, reuse this definition by declaring that they are of search type. They do not need to redefine the data types for their fields.

In case of some source definitions, MDL needs the ability to reuse and enhance an existing source definition by adding extra fields to it. For example, we have a source definition for PDF documents. Now, to define a new source for scholarly articles, which are PDF documents with additional fields like conference name, year of publication, references, and citation, we should be able to reuse the definition of a PDF document. Thus, MDL should provide the ability for specifying that a source extends another source by adding new fields to it. This is the same inheritance provided by object-oriented programming languages.

For each information source there is a root `meta_metadata` element with any number of nested `meta_metadata_fields` (see example, Figure 6). Each `meta_metadata` field is either a scalar field, a nested field, a collection field or inherits its type from a previously declared Meta-Metadata definition. The non-scalar fields include nested fields specified using either `is_nested` attribute or collection fields specified using the `collection` attribute. Collection fields have the `collection_child_type` attribute which specifies the type of collection elements. The non-scalar fields can be declared either directly inside the `meta_metadata` element or previously as named and typed `meta_metadata` for reuse.

Four primary primitive subclasses of metadata are supplied: `document`, `media`, `image` and `entity`. The `document` class is the base class for all document types, including web pages, PDF documents and search documents. The `media` class is used for all media elements including image and text elements. The `image` class is the base class for image documents. `Entity` is used with a unique key in the form of a URL to refer to other objects, enabling the representation of citation graphs.

In this chapter, we discuss the MDL that we have developed and how it can be used for structured, strongly typed, and hierarchical representation of heterogeneous information sources that can be reused or extended to define new sources. We illustrate it by developing an example of search definition and scholarly article definition

4.1 Specifying Strongly Typed Structures for Information Sources

Strongly typed structured definition of information sources is used during compile-time to generate strongly typed classes that can be used in any procedural programming

language. This eliminates the need to write a custom code for each heterogeneous information source. This also allows power users to declare new information sources for an information discovery task without knowing the application source code.

These structures consists of scalar fields with primitive data types like string, int, URL, or boolean as well as nested fields that are a collection of these fields with data types like ArrayList.

The definition of the search class is shown next in Figure 7. It shows a collection of search results from a search results document and corresponding XML to describe its structure. This XML defines the search results document as collection of results. The collection attribute is used to describe that the results form an *ArrayList* of items of class *search_result*. The *collection_child_type* attribute is used to denote the class of items in the *searchResults* ArrayList. The *scalar_type* attribute describes the type of fields that constitute the structure of a *search_result*.

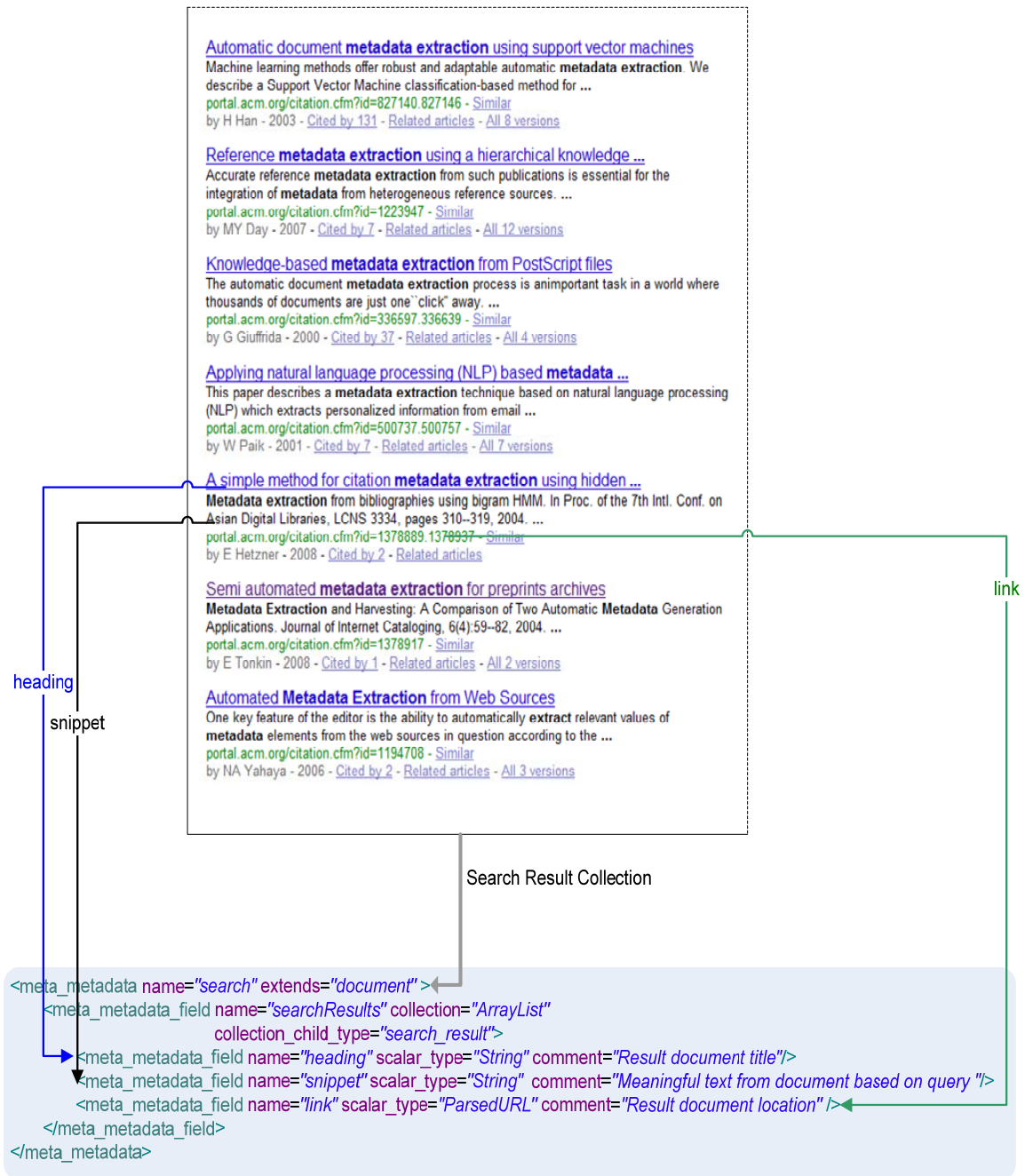


Figure 7: Strongly typed, structured definition of search result set

4.2 Specifying Heterogeneous Sources by Reusing Existing Definitions

Structural similarity among various information sources allows us to reuse existing source definitions to author new definitions. There are two main ways in which we can use structural similarity: sources having exactly same structure but different extraction rules (simple type re-use) and sources that require adding new fields to an existing source definition (inheritance).

4.2.1 Sources with the Exact Same Structure

Some sources, like searches have the same structure, shown in Figure 7 For instance, Google Search, Yahoo Search, and Wikipedia Search have the same fields and structure, but they have different rules to extract information from their HTML pages to populate runtime instances. To define such sources, we use the `type` attribute in the Meta-Metadatas definition to declare the Meta-Metadatas structure, which is exactly the same as the structure of this source. We then author the extraction rules to extract various fields. Because the structure is the same, we do not need to generate any class for this source and so we set the `generate_class` attribute to *false* for this source. During compile time, we generate classes for only one such definition. The objects of this class are instantiated and populated during runtime based on the extraction rules of the specific source, which is obtained from Meta-Metadatas repository using a URL pattern matching (See section 3.2.1).

Figure 8 shows the declaration of Google Search Meta-Metadatas. Since its structure is the same as that of the previous Meta-Metadatas definition of search, we define it to be of search type, by setting `type` attribute to `search` and indicate the compiler to

skip class generation for it by setting the `generate_class` attribute to `false`. For other fields, like heading, snippet, and link, we simply author the extraction rules without specifying the data types. These are inferred from the initial search definition.



Figure 8: Google search source definition using type attribute

During runtime, when we get a URL that matches the `url_base` for Google search metadata, the parser module builds instance of `Search` class and populates it using extraction rules from Google Search metadata.

4.2.2 Sources That Add New Fields to Existing Definition

Some source definition adds additional fields to an existing source definition for their description. For example, we have a definition for a PDF document in the repository that defines fields that can be embedded in a PDF document, such as author, summary,

and key words. Now we want to define another new source, Scholarly article, which is a PDF document with additional fields like references, citations, and year of publication, in addition to the fields that a PDF document has. To define such sources, which add additional fields to an existing definition, we use the `extends` attribute. This attribute works analogous to *extends* key word in object-oriented programming language like Java in providing re-use of data structures from parent classes. Inheritance in MDL however does not provide dynamic dispatching for methods and runtime type determination as in object-oriented programming languages like Java or C++.

Figure 9 shows the source definition of a scholarly article using the `extends` attribute. Note that these are not the full definitions of both the sources. For clarity, some of the fields have been omitted from both definitions.



Figure 9: Adding more fields to an existing definition using extends attribute.

5. METADATA EXTRACTION AND VISUALIZATION RULES

While MDL allows us to specify heterogeneous information sources using Meta-Metadata, we need infrastructure to extract information from them. This will allow us to build metadata objects that can be visualized and acted on using the visualization application and serialized for storage. For this purpose, the Meta-Metadata language has a set of metadata extraction and presentation rules.

Data from an information source can be obtained through either a web page or XML-based web service. If the data is obtained through a web page, the web page has to be parsed to extract the information fields specified in the metadata. In the case of XML-based web services, the XML can be directly bound to generate metadata classes to form metadata objects. Thus, the Meta-Metadata language should be able to specify for given information source the mechanism from which to form metadata objects.

This mechanism is achieved for web pages by parsing the HTML DOM, using extraction rules in the form of XPath and regular expressions. These XPath are written specific to a particular field of a particular information source. Templatization of information sources makes XPath an effective way to extract information. After information extraction, we may want to format the information by applying regular expressions onto it. For XML-based web services, we use the `ecologylab.xml` framework to bind the information XML directly to the metadata class to form metadata objects.

The type of parsing mechanism to be used is defined using the `binding` attribute. This attribute can have either `xpath` or `direct` as values, for XPath-based parsing of HTML DOM or direct binding of XML-based web-services. If binding attribute is not

present, we use the parsing methods developed by Koh, which parses the HTML DOM, and creates image and text surrogates (Koh and Kearne, 2009).

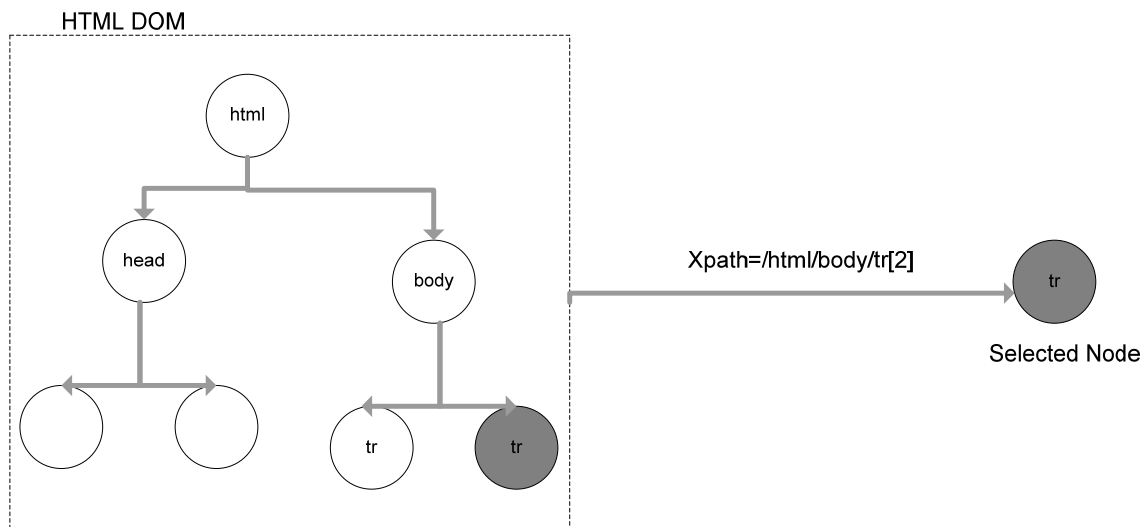


Figure 10: XPath-based extraction from an HTML DOM

Meta-Metadata language also provides a means to specify visual appearance of fields. This information can then be used with visualization tools to display various fields accordingly. These visualization rules enable one to change the visual properties without modifying or knowing the visualization layer.

5.1 XPath-based Extraction Rules

XPath-based extraction rules are used for information sources that provide a web-based interface. In such cases, during runtime, the web page is downloaded and parsed and the HTML DOM is formed. Now extraction rules in the form of XPaths are applied on this DOM to extract metadata fields specified using MDL. This information is set to generate

metadata classes to form metadata objects via reflection. As needed, this information can be modified using regular expressions before setting it to metadata objects. Figure 10 shows how XPath-based extraction rules can be used to select information from an HTML DOM. Here we apply an XPath expression ‘/html/body/tr[2]’ on the HTML DOM shown and select the highlighted node.

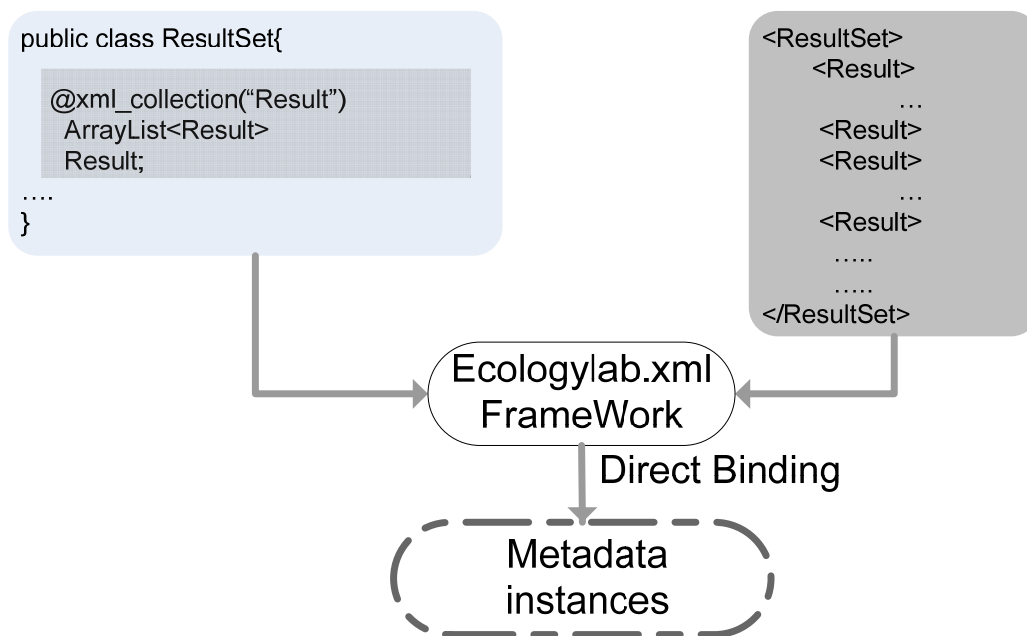


Figure 11: Direct binding for XML based web-services

XPaths for extraction from a Google Search page DOM are given in Figure 7.

5.2 Direct Binding for XML-based Services

Some information sources provide an XML-based service to access them. In such cases, we use ecologylab.xml framework to directly bind the XML to generate annotated

classes and form metadata objects. This is shown in Figure 11 for Yahoo search class. In this figure, we show the `ResultSet` class obtained through generative programming. The XML obtained from yahoo search is also shown. Now `ecologylab.xml` framework is used to obtain an object that contains each of the `Result` elements in an `ArrayList`.

5.3 Visualization Rules

Visualization rules tell the visualization software how a particular field should be shown to users. This includes the style of fonts and whether the field should be shown or not. These rules specified using some special attributes for the `Meta-Metadata` field like `style` and `hide`. The `style` attribute is used to specify the style of fonts to render the text of the field and `hide` attribute is used to specify the visibility of the field. By default, each field is visible.

Figure 12 shows the `style` attribute for title field is `h1` and `tag_link` field should not be visible.

```
<meta_metadata_field name="title" navigates_to="full_text" scalar_type="String" style="h1" />
<meta_metadata_field name="tag_link" scalar_type="ParsedURL" hide="true"/>
```

Figure 12: Visualization rules for Meta-Metadata fields

6. SEMANTIC ACTION SCRIPTING

After metadata objects are created, visualization applications operate on them to build semantic visualizations. We define semantic actions as the set of operations to be performed on an individual metadata object or a set. These operations include forming image and text surrogates that represent the main ideas of the information sources and provide access to it or connecting and downloading the information from the source. Depending on information source, these actions might need to be performed only under some particular conditions.

Current architectures have semantic actions closely tied with the visualization layer. Thus, it involves change and familiarity with the application code to add, remove, or modify any action. To abstract out the logic of operations on metadata objects from the application, the Meta-Metadata language includes a set of semantic actions that can be defined in the XML for each information source. The syntax is similar to XSL. We abstract out the logic for operations on metadata as interfaces. The collection visualization application can implement these interfaces according to its needs. Thus, the control flow structure of actions (conditionals and loops) coupled with the metadata specific actions are separated from the application. A power user can modify the call sites and sequence of these actions to change the application behavior. The application developer has to implement these actions according to the needs of visualization application.

This gives power users the ability to add new functionality and sources to information collection application without modifying source code. For instance, suppose there is an application that crawls and visualizes the citation for the graph of a scholarly

article. This application will have actions to download the scholarly article, get all the citations, and connect to them. Now, suppose, the user need to gather information about the authors too. If the logic for crawling and downloading the citations is in the application, it would require developers to modify the source code and rebuild the application. On the other hand, using our MVC-based architecture, the user can add an extra semantic action in XML to crawl author pages, too. No modification in the application source code is needed, and the user need not know about the application source code. Thus, semantic actions enable us to segregate the controller from the view and model and can change the logic of operation on the model by changing the semantic actions in XML

Semantic actions scripting include three kinds of statements: variable definition statements, control flow statements, and metadata-operation statements. Variable definition statements are declared under `def_vars` tag and are used to declare variables which are either a DOM node or a DOM nodelist. These can be used in the MDL to write concise XPath as well as in control flow and Meta-Metadata operations to operate on document DOM directly.

Control flow statements and Metadata-operation statements are declared under `semantic_actions` tag. Control flow statements include loops and conditional statements which govern the flow of control of operation on metadata objects. Metadata-operation statements operate on metadata objects to either build image and text surrogates from them, download them immediately, or send them to crawler to be downloaded latter.

The architecture provides an ability to store the values returned by actions in variables, which can then be used as arguments for other actions. Thus, they provide some of the same functionality as other procedural programming languages.

```
<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 13: Variable declaration statement

6.1 Variable Definition Statements

Variable definition statements declare variables of type node and node list. Figure 13 shows two sample variable definition.

Each variable is defined using a `def_var` tag. Each tag has following attributes.

- a. `name`: Name of the variable
- b. `xpath`: XPath expression to be applied to get the node or node list
- c. `type`: The kind of variable. It can either be a node representing a DOM node or a `node_list`, representing a DOM Node list.
- d. `context_node`: The node on which the XPath should be applied. If this attribute is missing XPath is applied on the root node of DOM

6.2 Control Flow Statements

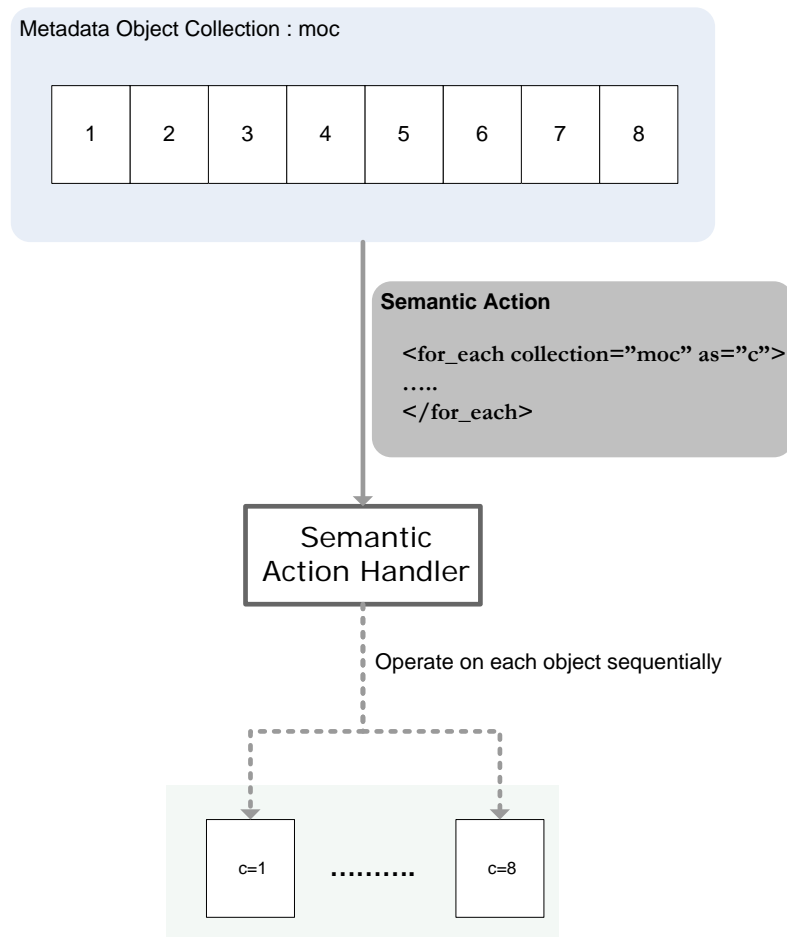


Figure 14: For loop to iterate over a collection

Control flow actions define the control flow structure for the procedural programming language metadata objects. These structures include loops and conditional statement. XSL is a language that provides the ability to specify control structures in XML. But XSL can only be used for transformation of XML documents. It cannot be used

to define interfaces which can act as call sites for operations on procedural programming language objects. For this reason, we have defined control flow structures in Meta-Metadata language that are similar to syntax in XSL but differ in that they can operate on procedural programming language objects.

Each of the control flow statements can have other control flow or metadata-operation statements within them. There are two kinds of control flow actions: *For loop* to iterate over a collection and *If statement* to execute a set of metadata operation statements conditionally.

6.2.1 Loop Statements

Loops are used to iteratively perform a set of actions on a collection of objects. Meta-Metadata language includes a `for_each` semantic action that is syntactically very similar to XSL for-each statement. It is shown in Figure 14.

In the `for_each` semantic action, we iterate over a collection that we refer to as `moc` using the `collection` attribute. We access each item of this collection using the `as` attribute and refer it as `c`. As shown in Figure 13, the semantic action handler then operates on each object iteratively. The set of actions that needs to be performed on this collection are written between the `<for_each></for_each>` tags.

The loop iterates over collections of metadata objects extracted from a webpage. We make the practical assumption that a webpage will only offer a finite collection of metadata, resulting in loops that will always terminate.

6.2.2 Conditional Statements

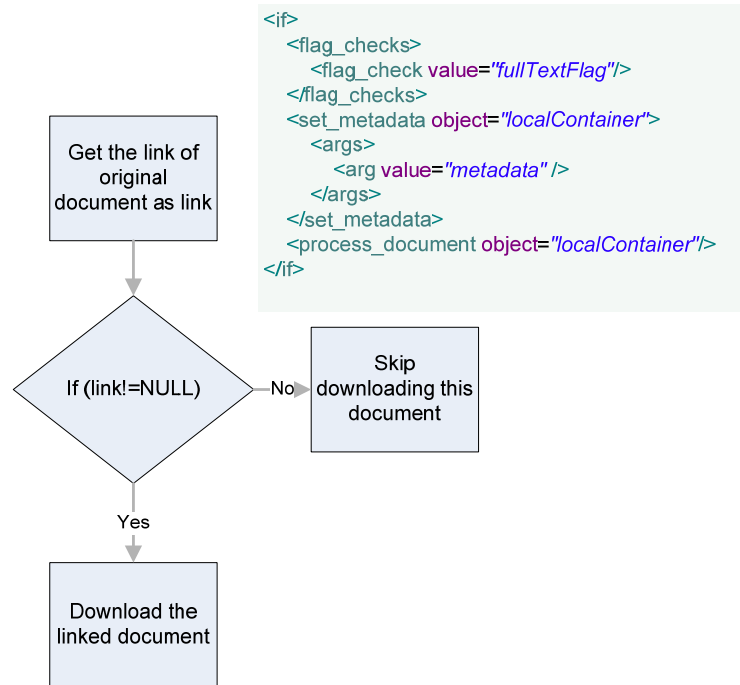


Figure 15: If statement for conditional execution

Conditional statements are used to specify the conditions only in which a set of semantic actions should be performed. Sometimes we need to perform a semantic action only if certain conditions are satisfied. For instance, in ACM Portal digital library, the metadata of an article also contains the link to the actual document. The semantic actions for ACM Portal will thus include getting the link of the actual document and then downloading the linked document. But if for some reason this link is not present, the semantic action for downloading the linked document should not be taken. Thus we need a mechanism to represent this logic in XML. This logic is shown as a flow chart in Figure 15.

XSL has a `<if>` tag to write conditional statements in XML. We have developed a similar `<if>` tag in our language. It has a list of `flag_check` elements nested inside it. Each `flag_check` element has a boolean variable as `value` attribute. The actions inside an *if* statement will be executed only when all the boolean variables are true. In the XML shown in Figure 15, we take `set_metadata` and `process_document` actions only if the flag specified in `flag_check` is true.

6.3 Metadata-operation Statements

Metadata-operation statements specify the operation that visualization application can perform on metadata objects. These actions act as the interfaces that different applications can implement according to their needs. Meta-Metadata language currently provides a set of metadata actions, each of which is described in detail in the Appendix A.

7. USE CASES

We develop use cases to illustrate the use of the Meta-Metadata architecture in practice. All these use cases have been developed by power users, having programming background. Power users write Meta-Metadata definitions in XML using the Meta-Metadata language.

7.1 Google Search

Querying Google gives a web page containing a list of results. This is shown in Figure 16 for a search for iPhone. We use MDL to define the information fields of the Google search result page as shown in Figure 8. After this source definition, during compile-time we call the Compiler module to generate strongly typed annotated metadata classes. Since Google source definition is of type search, we do not generate any separate class for Google. The generated classes for Search are shown in Figure 5.

We use combinFormation as collection visualization application. The user task involves gathering information about latest in the field of smart phone technology and compares two existing smart phones: iPhone and Palm Pre, based on their features.

User issued Google Searches with following queries: 'iPhone', 'Palm Pre' and 'smart phones'. Note that we have shown Google results page only for one query: iPhone.

Using extraction rules defines for Google Search Meta-Metadata. The values are then set in the generated class, to get the Google search metadata object. These metadata objects are then operated on by the Semantic Action Handler module to take action on them. For Google search results, these actions involve getting the link of result document, creating a connection to it, and parsing the linked document page. After browsing through the search collection returned, in the form of image and text surrogates, the user was able to make innovative collage shown in Figure 17.

In the collage of Figure 17 user has shown the latest features and driving factors for smart phones today. All these are shown in middle line which also serves as separator for comparing iPhone and Palm Pre. On the left side user has collected information about iPhone, which includes a picture of it and main features, while on left user has collected similar information about Palm Pre.

Google

Web [+ Show options...](#) [Compare on eBay](#)

[Apple - iPhone - Mobile phone, iPod, and Internet device.](#)
iPhone 3GS is a GSM cell phone that's also an iPod, a video camera, and a mobile Internet device with email and GPS maps.
www.apple.com/iphone/ - [Cached](#) - [Similar](#) - [🗨](#) [📄](#) [🗕](#)


[Buy iPhone](#) [A Guided Tour](#)
[Features](#) [Apps for iPhone](#)
[Downloads](#) [Software Update](#)
[Support](#) [Learn how to use all the ...](#)

[More results from apple.com »](#)

[Apple - iPhone - View all the features of the new iPhone 3GS.](#)
The new **iPhone** 3GS cell phone features faster performance, a video camera, voice control, and GPS maps with compass.
www.apple.com/iphone/iphone-3gs/ - [Cached](#) - [Similar](#) - [🗨](#) [📄](#) [🗕](#)


[iPhone - Wikipedia, the free encyclopedia](#)
The **iPhone** is an Internet and multimedia enabled smartphone designed and marketed by Apple Inc. Because its minimal hardware interface lacks a physical ...
en.wikipedia.org/wiki/IPhone - [Cached](#) - [Similar](#) - [🗨](#) [📄](#) [🗕](#)

News results for iPhone

 [AT&T CEO has ominous words for iPhone users during CTIA](#) - 1 day ago
Could this be why New Yorkers are seeing an average of up to 30% of their **iPhone** calls dropped, according to one report? His long term solution is more ...
[Computerworld](#) - [730 related articles »](#)

[Seattle Post Intelligencer](#) [AT&T Says No Tethering For iPhone. Yet](#) - [InformationWeek](#) - [1168 related articles »](#)
[Apple iPhone Update Fixes Bugs](#) - [InformationWeek](#) - [128 related articles »](#)

Shopping results for iPhone

 [Apple iPhone Smartphone 8 GB - shared - AT&T - GSM](#)
★★★★☆ 1348 reviews - \$450 new, \$320 used - 15 stores

[Apple iPhone Smartphone 4 GB - shared - AT&T - GSM](#)
★★★★☆ 1348 reviews - \$600 new, \$250 used - 5 stores

[Apple iPod touch 16 GB Digital player - 16 GB flash](#)
★★★★☆ 1148 reviews - \$175 new, \$170 used - 25 stores

[Apple iPhone: iPhone 3G reviews, news, photos and videos - CNET.com](#)
Check out CNET's **iPhone** coverage, including up-to-the-minute news about the latest **iPhone** 3G S, in-depth reviews of the different **iPhone** models, ...
www.cnet.com/apple-iphone.html - [Cached](#) - [Similar](#) - [🗨](#) [📄](#) [🗕](#)

[iPhone Dev Center - Apple Developer Connection](#)
Apple's official web development connection for the **iPhone**.
developer.apple.com/iphone/ - [Cached](#) - [Similar](#) - [🗨](#) [📄](#) [🗕](#)

Will It Blend? - iPhone

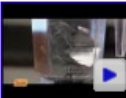
 1 min 37 sec - Jul 10, 2007 - ★★★★★
Everybody knows that the **iPhone** can make phone calls, play movies & music, surf the web, and a lot more. But, Will It Blend? That is the question.
www.youtube.com/watch?v=qq1ckCkm8YI - [Related videos](#) - [🗨](#) [📄](#) [🗕](#)

Figure 16: Google search result page

iPhone and iPod

iPhone 3GS makes your entire iPhone experience more responsive.

Multi-Touch user interface

Latest in Smart Phone Technology. digital compass. portrait keyboard

See your carrier for details.

Cisco IPsec VPN servers,

breakthrough Internet

Smartphone reviews, phone reviews, notebook

to challenge Apple's iPhone.

vertical QWERTY slider,

Palm OS emulator

tweak their Palm Pre's,

Palm Pre community

That compromise: the screen.

Palm created quite a buzz

Still won't touch the Pre though.

iPhone OS SDK

iPhone SDK Programming,

There are thousands of iPhone apps to choose from.

for native iPhone applications.

iPhone Developer Program

iPhone (including jailbroken iPhones)

PwnageTool for iPhone

highly targeted audience

your life running smoothly.

"Seriously fantastic.

Now, I'm never without my smartphone in my pocket.

Figure 17: Smartphone,iPhone and Palm pre-browsing

7.2 ACM Portal

In the ACM Portal use case, we first issue google search on site portal.acm.org.

The image shows a Google search interface. The search bar contains the text "metadata extraction site:portal.acm.org" and a "Search" button. Below the search bar, there are navigation options: "Web", "Show options...", and "Compare on eBay". The search results are displayed in a list format, with each result including a title, a brief description, a URL, and additional information like "Cited by" and "Related articles".

Automatic metadata extraction from museum specimen labels
 Hui Han , C. Lee Giles , Eren Manavoglu , Hongyuan Zha , Zhenyue Zhang , Edward A. Fox,
 Automatic document **metadata extraction** using support vector machines ...
portal.acm.org/citation.cfm?id=1503425 - Similar -
 by PB Heidorn - 2008 - [Cited by 1](#) - [Related articles](#) - [All 8 versions](#)

A Simple Method for Citation Metadata Extraction using Hidden ...
 before for **metadata extraction** from citations [22, 6, 18, 4, Bibliographic **meta-data extraction** using probabilistic finite state transducers. ...
portal.acm.org/ft_gateway.cfm?id=1378937&type=pdf - Similar -
 by E Hetzner - 2008 - [Cited by 4](#) - [Related articles](#) - [All 4 versions](#)

Automatic Extraction of Table Metadata from Digital Documents
 and their **metadata**, an automatic table **metadata extraction** ... matic table-**metadata-extraction** algorithm is designed and tested on PDF. ...
portal.acm.org/ft_gateway.cfm?id=1141835&type=pdf - Similar -
 by Y Liu - 2006 - [Cited by 12](#) - [Related articles](#)

Semi Automated Metadata Extraction for Preprints Archives
 heading of automated **metadata extraction**. At the most ba- ... For various reasons,
metadata extraction sometimes fails entirely. ...
portal.acm.org/ft_gateway.cfm?id=1378917&type=pdf - Similar -
 by E Tonkin - 2008 - [Cited by 1](#) - [Related articles](#) - [All 3 versions](#)

Reference metadata extraction using a hierarchical knowledge ...
 Accurate reference **metadata extraction** from such publications is essential for the integratic
 of **metadata** from heterogeneous reference sources. ...

Figure 18: Google search result set

The results page obtained is shown in Figure 18. The metadata extracted from one of the search result is shown in Figure 19. Figure 20 shows the source definition for ACM Portal. During a Google search for a query, if we download an ACM Portal page, which is the case in the current scenario, the runtime infrastructure uses the source definition of ACM Portal from the source repository to parse and build metadata objects from it and take semantic actions. Semantic actions for ACM portal are not shown in this figure. In the case of combinFormation, these semantic actions are implemented to download the linked PDF document, parse it, form images and text surrogates from it and form citation chains for references and citations. These surrogates provide in-context metadata, which can again be acted on by the user to create more search queries. Visualization of in-context metadata with nested citations using combinFormation is shown in Figure 21.

Semi automated metadata extraction for preprints archives	
Metadata Extraction and Harvesting: A Comparison of Two Automatic Metadata Generation Applications. Journal of Internet Cataloging, 6(4):59–82, 2004. ...	
portal.acm.org/citation.cfm?id=1378917 - Similar	
by E Tonkin - 2008 - Cited by 1 - Related articles - All 2 versions	
String heading	"Semi automated metadata extraction for preprints archives"
String snippet	"Metadata Extraction and Harvesting: A Comparison of Two ... "
ParsedURL link	http://portal.acm.org/citation.cfm?id=1378917

Figure 19: Search fields extracted from Google search

```

<meta_metadata name="acm_portal" binding="xpath" url_base="http://portal.acm.org/citation.cfm" type="scholarly_article"
  generate_class="false" package="ecologylab.semantics.library.scholarlyPublication">
  <meta_metadata_field name="title" xpath="//td[@class=medium-text]/strong/child:text()" navigates_to="metadata_page"/>
  <meta_metadata_field name="location" xpath="//a[@name=FullText]/@href" comment="The Linked PDF"/>
  <meta_metadata_field name="abstract" xpath="//p[@class=abstract]/child:text()"/>

  <meta_metadata_field name="source" is_nested="true" xpath="//td[@class=small-text and @colspan=2]"/>
  <meta_metadata_field name="title" xpath=".//span[0]" comment="Name of conference or journal." navigates_to="location"/>
  <meta_metadata_field name="archive" xpath=".//a[child:text()='archive']/@href" comment="Archive of conference or journal."/>
  <meta_metadata_field name="location" xpath=".//a[child:text()='table of contents']/@href"/>
  <meta_metadata_field name="pages" scalar_type="String" xpath=".//div[@class=small-text]/child:text()[starts-with(., Pages:)]"
    regular_expression="Pages:" replacement_string=""/>
  </meta_metadata_field>

  <meta_metadata_field name="authors" shadows="author" xpath="//div[@class=authors]/tr" collection="ArrayList"
    collection_child_type="author">
    <meta_metadata_field name="name" scalar_type="String" xpath=".//td[1]/a"/>
    <meta_metadata_field name="affiliation" scalar_type="String" xpath=".//td[2]"/>
    <meta_metadata_field name="location" scalar_type="ParsedURL" xpath=".//td[1]/a/@href"/>
  </meta_metadata_field>

  <meta_metadata_field name="references" collection="ArrayList" xpath="//a[@name=references]/..following-sibling::table//
    a[@hrefstarts-with(., citation)]" collection_child_type="scholarly_article">
    <meta_metadata_field name="location" scalar_type="ParsedURL" xpath="./@href"/>
    <meta_metadata_field name="title" scalar_type="String" xpath=".//child:text()" navigates_to="location"/>
  </meta_metadata_field>

  <meta_metadata_field name="citations" collection="ArrayList" xpath="//a[@name=citings]/..following-sibling::table/a[@hrefstarts-
    with(., citation)]" collection_child_type="scholarly_article">
    <meta_metadata_field name="location" scalar_type="ParsedURL" xpath="./@href"/>
    <meta_metadata_field name="title" scalar_type="String" xpath=".//child:text()" navigates_to="location"/>
  </meta_metadata_field>
</meta_metadata>

```

Figure 20: ACM portal Meta-Metadata semantic actions

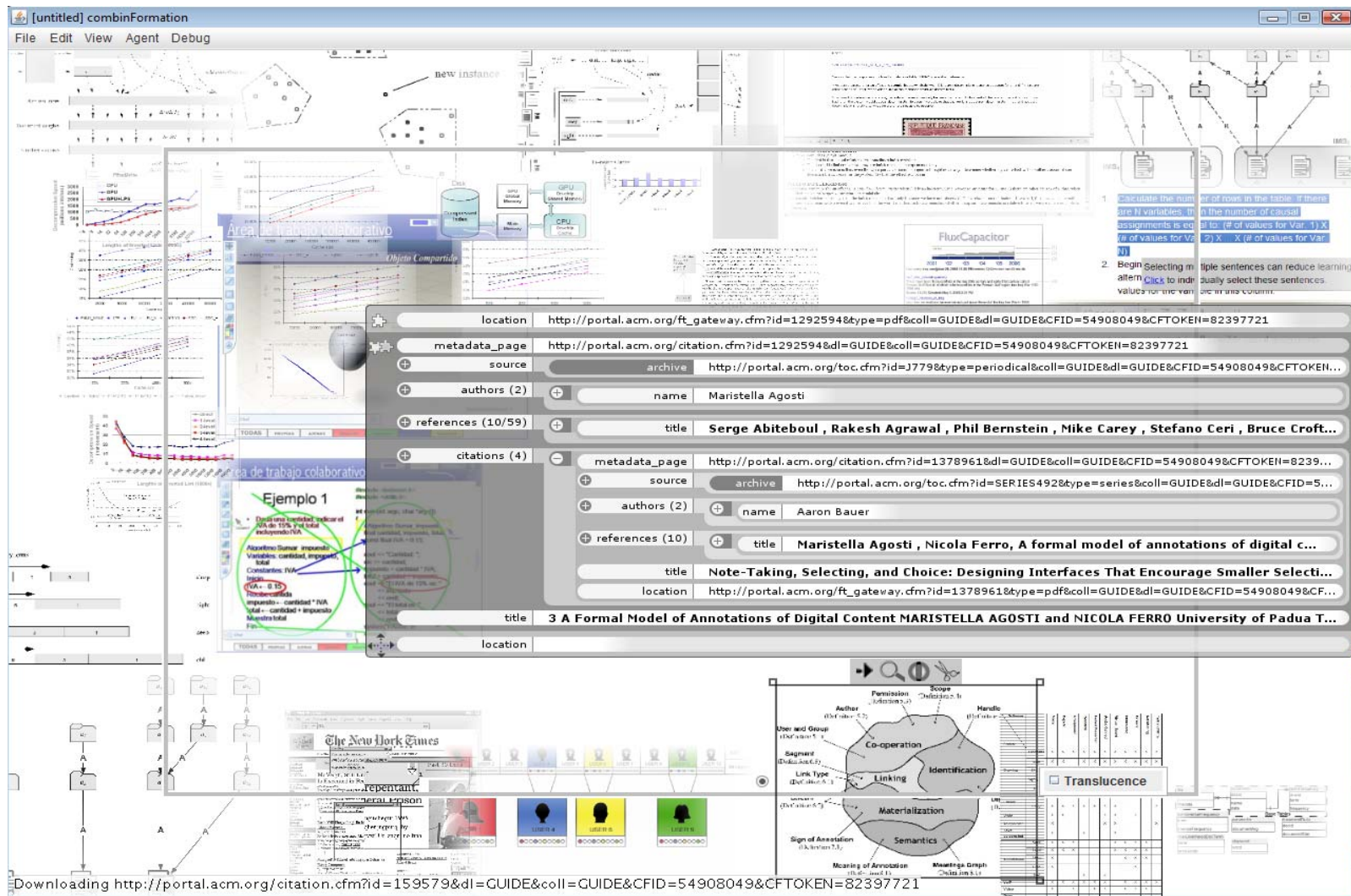


Figure 21: Information visualization from ACM portal using combinFormation

7.3 Yahoo Search

```

<ResultSet>
  <Result>
    <Title>Sunset</Title>
    <Summary>
      Covers the thirteen Western states. A lifestyle magazine covering gardening and outdoor
      living, travel, food, and the home.
    </Summary>
    <Url>http://www.sunset.com/</Url>
    <ClickUrl>http://www.sunset.com/</ClickUrl>
    <DisplayUrl>www.sunset.com/</DisplayUrl>
    <ModificationDate>1253862000</ModificationDate>
    <MimeType>text/html</MimeType><Cache>
  </Result>
  <Result>
    .....
  </Result>
  .....
</ResultSet>

```

```

<meta_metadata name="ResultSet" extends="search" binding="direct"
  url_base="http://api.search.yahoo.com/WebSearchService/V1/webSearch" user_agent_name="firefox_2"
  comment="Yahoo Web Search Class">
  <meta_metadata_field name="Result" collection="ArrayList" collection_child_type="result" >
    <meta_metadata_field name="Title" scalar_type="String"/>
    <meta_metadata_field name="Summary" scalar_type="String"/>
    <meta_metadata_field name="Url" scalar_type="ParsedURL"/>
    <meta_metadata_field name="ClickUrl" scalar_type="ParsedURL"/>
    <meta_metadata_field name="DisplayUrl" scalar_type="ParsedURL"/>
    <meta_metadata_field name="ModificationDate" scalarType="Int"/>
    <meta_metadata_field name="MimeType" scalar_type="String"/>
  </meta_metadata_field>
  <semantic_actions>
    <get_field name="Result"/>
    <for_each collection="Result" as="res">
      <get_field name="Url" object="res"/>
      <create_container_for_search name="resultContainer">
        <args>
          <arg value="Url"/>
        </args>
      </create_container_for_search>
      <process_search>
        <args>
          <arg value="resultContainer"/>
          <arg value="Url"/>
        </args>
      </process_search>
    </for_each>
  </semantic_actions>
</meta_metadata>

```

Figure 22: Yahoo search XML and corresponding Meta-Metadata

Yahoo provides both a web-based interface as well as an XML-based web service for its search results. We have written Meta-Metadata for XML-based web service, as these documents are meant for computation data exchange and are thus easier to process programmatically as compared to HTML-based interfaces. Figure 22 shows a sample XML obtained from Yahoo web service and corresponding Meta-Metadata declarations for it.

As shown, the binding type for the Meta-Metadata is declared as direct. This tells the parser during runtime to bind the XML directly to the generated class. No extraction rules are defined for any of the Meta-Metadata field as ecologylab.xml framework provides the ability to directly bind XML to annotated metadata class and form metadata objects.

7.4 Lines of Code Comparison

In this section, we discuss the lines of code that was needed in Java to add a new information source in `combinFormation` with the lines of XML declaration needed. We have compared all the three preexisting source definitions in `combinFormation` written in Java with the new declarations of these sources in XML. In each case, we have observed that number of lines of XML needed were significantly less than lines of Java code required. Figure 23 gives a bar graph comparing the lines of Java code and lines of XML for each of the source.

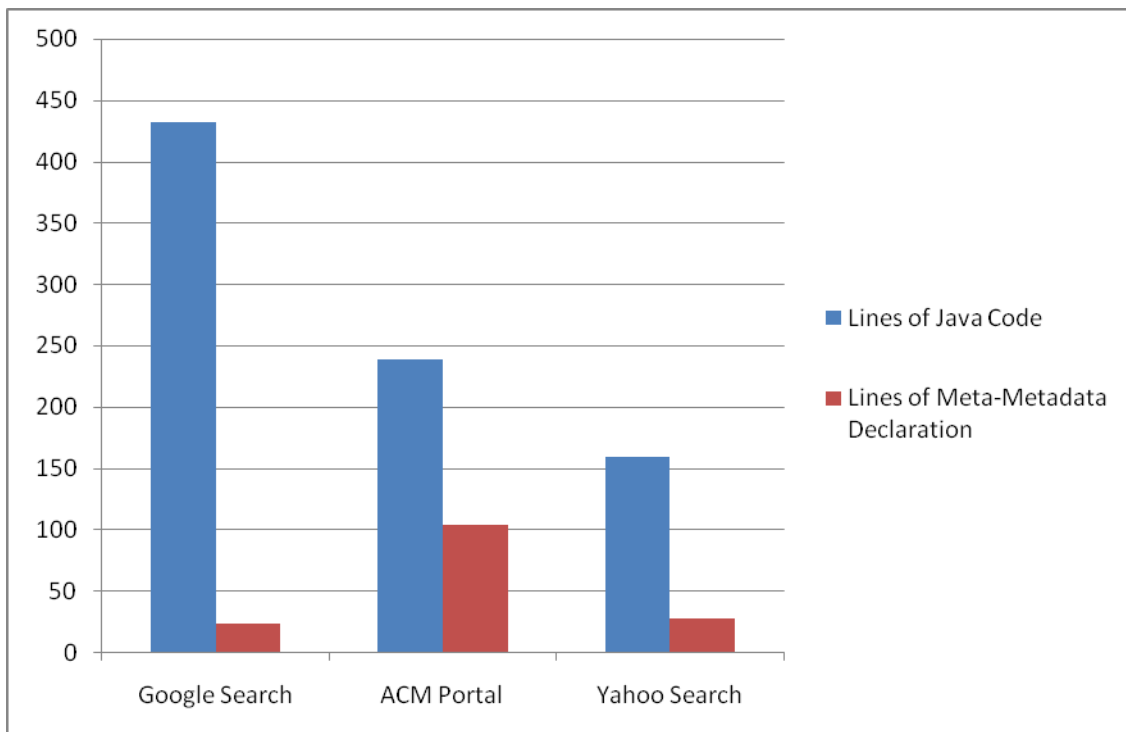


Figure 23: Comparison of lines of Java code vs lines of Meta-Metadatas declaration

8. USER STUDY

To validate the usability of our architecture we conducted a user study, in which participants were power users, who used the Meta-Metadatas to author templates for information sources. We asked three users to use the Meta-Metadatas, each for a different source. The users were all computer science students having programming background. These sources included digital collections of images like Flickr, databases like IMDB for movies, and article collections like Wikipedia. Each of these sources uses a specific web template to display their information. The abundance of digital collections in these sources made them very important for the task of collection visualization.

During the study, the user tasks involved authoring of Meta-Metadatas declarations and extraction rules for structured metadata extraction, to create metadata objects from these diverse templated sources and operations on metadata objects thus created. These operations included iterating through the collection set, connecting to and parsing documents, forming citation chains, and creating and visualizing image and text surrogates.

We gathered qualitative data from this user study. Users were asked to fill out a questionnaire that included questions about their background and experience with authoring of information sources for collection visualization applications. They were also asked about their expertise level with technologies like XSL, XPath, and regular expressions. Finally, they were asked about their experience of writing source definitions using Meta-Metadatas language and number of lines of custom code (if any) that was needed for this. The results of the study are discussed in this section.

8.1 Wikipedia

Wikipedia (http://en.wikipedia.org/wiki/Main_Page) is a free encyclopedia built using Wiki software. It allows collaborative authoring of articles on various topics. It has a collection of about 3,037,664 articles. Each article page consists of text and images describing the topic, as well as links to other articles both within Wikipedia and external documents.

Using the Meta-Metadata, the user performed the following tasks.

- Authored support for the Wikipedia search engine.
- Authored Meta-Metadata for the search results page of Wikipedia.
- Authored Meta-Metadata for each article page of Wikipedia.

Details of each of these tasks and feedback are discussed in following sections.

8.1.1 Authoring Meta-Metadata for Search Results Page of Wikipedia

Wikipedia can be searched for articles using a search engine. The first step in authoring Meta-Metadata for Wikipedia is specifying the search URL that can be used to retrieve the search results from Wikipedia. The user was able to use the search engine definition data structure discussed in Appendix C, to define a new search engine for Wikipedia. The structure for the Wikipedia search URL and its declaration are shown in Figure 24.

After connecting to the search URL, the next task involves parsing the search results page, forming metadata objects from it, and taking semantic actions on them. For this, the user defined Meta-Metadata for the Wikipedia search page. This Meta-Metadata consisted of a collection of extraction rules for search results collection and semantic

actions on each object of this collection. These semantic actions included getting the hyperlink for each result document, creating a container for it, and parsing the result document container. Also, a unique URL key for this Meta-Metadata was defined to uniquely retrieve the Meta-Metadata when a Wikipedia search URL is encountered.

The user was easily able to reuse the predefined search definitions for search and Google Search to write the Meta-Metadata for Wikipedia search.

```

http://en.wikipedia.org/w/index.php?title=Special%3ASearch&search=QUERY&fulltext=Search.
<search_engines>
  <search_engine name="wikipedia"
    url_prefix="http://en.wikipedia.org/w/index.php?title=Special%3ASearch&redirs=1&search="
    numResultString="&limit=" startString="&offset="
    url_suffix="&fulltext=Search&ns0=1" />
</search_engines>

```

Figure 24: Search engine declaration for Wikipedia

8.1.2 Authoring Meta-Metadata for Article Page of Wikipedia

The results documents from Wikipedia search are links to articles on Wikipedia. These articles contain images and links to other Wikipedia articles as a part of their contents. The Meta-Metadata authored by the user extracts a collection of images for each article page, and a collection of links to other articles on Wikipedia that are referred from within the article body. The user had a beginner level in XPath and regular expressions and was able to write extraction rules for these fields.

The user had a beginner level of knowledge in XSL syntax and with help from the documentation of semantic actions was able to choose and script the semantic actions

create image surrogates for images on the page and semantic links for other documents which are referred from the current page.

8.1.3 Feedback from Study

We observed that the user used the built-in XPath utility of the Google chrome browser and the Firefox XPather plugin to author extraction rules. Having a programming background in other languages and a beginner level knowledge of XML- based language like XSL was sufficient to script the control flow logic in semantic actions. The documentation provided for semantic actions was helpful during the scripting of actions for performing the desired tasks. The user did not have write Java code. The entire template definition for Wikipedia was performed using Meta-Metadata.

The user requested the following features to facilitate Meta-Metadata authoring which have been developed as part of the present research.

- Ability to write relative XPath.
- Explicit error messages, in case some extraction rule is invalid, or semantic

action could not be executed

8.1.3.1 Ability to Write Relative XPath

Initial designs of extraction rules enabled writing XPath only on the document root node. This resulted in very large XPath expressions for each field. This is cumbersome for authoring, inefficient computationally, and makes the metaMetadata repository hard to read and maintain. The ability to write relative XPath enables selecting some child nodes of interest and then writing XPath relative to these child nodes. To declare these child nodes of interest, we implemented variable declaration statements. These statements

(discussed in Section 8.1) allow a user to declare a set of commonly use nodes and write XPaths for these nodes. These nodes can then be used inside the extraction rules for various Meta-Metadata fields to write concise XPath expressions.

8.1.3.2 Explicit Error Messages

During runtime, if an error occurs, either during parsing or executing semantic actions, the runtime architecture previously reported Java errors. These errors were not intuitive and it was difficult for power users to debug their Meta-Metadata declarations based on them. We have now added the functionality in the Meta-Metadata language interpreter, to catch these Java exceptions and generate explicit meaningful error messages, which the Meta-Metadata author can use to debug his declarations

8.2 Flickr

Flickr (www.flickr.com) is a well-known image collection site. It has a collection of about 3.6 billion images. It provides the ability to search images by name and by author. Images on Flickr also have tags that connect images based on users supplied keywords, creating a folksonomy (Camreon et al., 2006). For this site, the user performed following tasks.

- Authored search URL for, search by image name and search by author name
- Authored Meta-Metadata for search by image name, results page
- Authored Meta-Metadata for search by author name page
- Authored Meta-Metadata for image description page, which is the result link of both search by image name or search by author name page. This Meta-Metadata consisted of actual image source, title, caption and tags for the image.

8.2.1 Feedback from Study

The initial design of the Meta-Metadata language had only no query URL as a way to define the unique URL key for Meta-Metadata. In case of Flickr, image description page has pattern `http://www.flickr.com/photos/AUTHOR/IMAGE_ID/`; the no query URL is `http://www.flickr.com/photos`. For search by author name we URL pattern `http://www.flickr.com/photos/AUTHOR/`. The no query URL for this is also same `http://www.flickr.com/photos`. Thus, there was a need of a different mechanism to define URL key apart from no query URL. For this purpose, we implemented URL prefix with wild card support. Thus, the URL key for Flickr image result page is `http://www.flickr.com/photos/*/` and for Flickr author search page is `http://www.flickr.com/photos*/`. Here we use ‘*’ as a wild card that matches anything between two ‘/’ characters.

Flickr image can have a number of different URL patterns like `http://farm3.static.flickr.com/2020/2118178242_27fb91853a_m.jpg` or `http://farm4.static.flickr.com/3477/3938205388_331febfb7a_m.jpg`. For these kinds of URL patterns, we cannot use no query URL and URL prefix as a key. Thus, we introduced another mechanism to specify URL key using regular expressions.

All the various ways to specify URL keys, along with their lookup complexity, are discussed in Section 3.2.1.

8.3 IMDB

IMDB (Internet Movie Database) is an online database that provides information about movies and artists. This has two families of pages of interest. The first one being the

movie details page, which includes all the information about a particular movie. The other page of interest is the profile page of artists, which includes information about a particular artist. For this site, the user performed the following tasks:

- Authoring Meta-Metadata for each movie title. This included extraction rules for movie title, poster, rating, directors, writers, genres, plot, tagline, and cast and declaring semantic actions to operate on the metadata object thus created. These semantic actions included creation of image surrogate from the poster, a text surrogate from tag line and connecting to, and parsing the pages for the cast of the movie.
- Authoring Meta-Metadata for each artist. This included extraction rules for artist name, birth details, biography, awards, and list of movie titles and semantic actions to operate on this metadata. These semantic actions included creating of image surrogate for the artist image and nested chaining of all the movie title he was involved with.

8.3.1 Feedback from Study

The user was able to write definitions for both the family of pages using the Meta-Metadata language. The error messages proved useful during the authoring process and helped the user to debug the authoring of definitions. In one such situation, the user added an extra field and did not use the compiler to generate new class. Now when the user tried to access that newly added field, he got an appropriate error message during runtime, which helped the user to determine that compiler needed to be run again.

8.4 Observation

We observed that the users, with some programming background, were able to add new information sources to combinFormation using the Meta-Metadate language. Users were not expert in XPath but were able to use tools like XPather to write extraction XPaths. The control flow statements were very similar to other scripting language and did not required extra effort to learn. The set of metadata operation statements allowed users to do the desired tasks with metadata objects created.

Finally, we also provided two test files to users which helped them to test their XPaths and Regular expression before authoring the Meta-Metadate. This proved very useful as users used it to debug, test, and fix their extraction rules before scripting Meta-Metadate.

9. CONCLUSION AND FUTURE WORK

As a part of this research we have designed and developed software architecture and scripting language that enables transformation of the regular World Wide Web into the semantic web. This is exciting. It allows power users to define the information extraction semantics and script information manipulation logic for collection visualization applications. This can also be used for creating semantic web browsers. We discuss the following features of our Meta-Metadata language

- Expressive ability of Meta-Metadata language, and
- Minimal programming experience needed.

9.1 Expressive Meta-Metadata Language

The Meta-Metadata language that we have developed as a part of this research is expressive to describe information extraction, visualization, and manipulation semantics from diverse heterogeneous sources. During the user study, the power users, having some programming background, were able to describe structure of different information sources using the Meta-Metadata definition module. The extraction rules, in the form of XPath and regular expression, were sufficient to extract various metadata fields from the information source. The semantic action scripting module provided them ability to operate and manipulate the metadata objects. The overall number of lines of XML needed to author a new source was less as compared to lines of code needed to author a new source in Java.

9.2 Minimal Programming Experience Needed

Meta-Metadata language was designed for power users, which we assume have some programming background. During the user study, users who were computer science students, did not have to put any effort in learning the control structure of the semantic action script and were able to author Meta-Metadata by learning from previously authored Meta-Metadata declarations. A beginner level knowledge of XPath, regular expression and XSL, was sufficient to write the extraction rules and semantic actions.

9.3 Future Work

This research lays the foundation for considerable future research. Below is the list of potential research areas to explore:

- Entity Resolution Problem
- Changing DOM problem
- Meta-Metadata authoring issues
- Exploring generality of semantic actions
- Constraint specification in Metadata Definition Language

We explain and discuss these problems in this section:

9.3.1 Entity Resolution Problem

Entities are data structures which contain a nested metadata object, its location and its semantic text. Entity resolution problem arises when there is a citation chain while parsing information source. In such cases, we need to keep a track of instances of metadata objects so as to recycle a metadata object, only when there is no more reference to it, in order to prevent memory leak. Also we need to maintain a data structure to dynamically

update the nested metadata instances of entities, when the document present at its location is downloaded and parsed. This data structure will be used to build citation chains refereeing this entity.

9.3.2 Changing DOM Problem

Metadata for many information sources like Google, Flickr, ACM Portal, IMDB, and Wikipedia are extracted using XPath. These XPath are written by power users for the DOM created by a template of these sources to present their information and are very specific to the particular template. During the course of this research, we found that these information sources sometimes make minor modification in their templates. In such cases, many of the extraction rules will either fail or will not work as expected. We call this problem the changing DOM problem.

The changing DOM problem is not just the characteristic of XPath-based extraction. Even if we have had custom Java classes written to parse these sources, the extraction might fail, if the template is changed. One way to solve this problem could be to use machine learning techniques. In such an approach, we could store a template for an information source also in another repository, along with the Meta-Metadata for it. Now in case a new template is used for this information source, we can use machine learning techniques to find out the changes and modify XPath accordingly.

9.3.3 Meta-Metadata Authoring Issues

Although the Meta-Metadata authoring is reasonable easy, there are some issues that might involve more study. For example, for given information source, what fields need to be extracted and what semantic actions need to be taken are sometimes very

complicated. This opens up research in the areas of usability and human computer interaction to obtain only appropriate metadata fields, create a sufficient number of surrogates (both image and text), and perform only the required level of metadata nesting for a given information source. Also, research needs to be done so as to automate the Meta-Metadata authoring as some times the extraction rules, like regular expression, could be complicated. Sometimes URL key authoring can also be complicated, involving the use of regular expressions. These choices have to be made by power user while authoring Meta-Metadata and may not be trivial.

9.3.4 Exploring Generality of Semantic Actions

The extraction module of Meta-Metadata language builds metadata objects which can be serialized into XML and thus can be used by any information collection and visualization application. The semantic actions are available as abstract methods and have to be implemented according to application. We have implemented these actions for `combinFormation`. We believe that other applications should be able to implement these actions. Some actions like `create_container` or `queue_document_download` involve use of data structures which might be specific to `combinFormation`. The future research in this direction should explore the possibility of implementation of semantic actions in other visualization applications like Visual Knowledge Builder (Shipman et al., 2001).

9.3.5 Constraint Specification in Metadata Definition Language

There is scope for improvement in the Metadata Definition Language. The current implementation of metadata language only provides two kinds of value specification for a

field, a singular scalar field, or a collection. A field can be a scalar field like integer, string and URL. In this case the field can have either 0 or 1 value associated with it. The other declaration provided is a collection field like ArrayList. In this case a field can have either 0 or many values associated with it. For example, consider the case of authoring the metadata definitions for a 'Book' object. We know that any book must have at least one author associated with it, however there is no way for a user writing Meta-Metadata definitions to specify such constraints. We believe that such constraints could improve the languages ability to express not only the structure, but also higher level semantics of a metadata object.

REFERENCES

- Bainbridge, D., Ke, K. and Witten, I. (2006) Document level interoperability for collection creators. *Proceedings of Joint Conference on Digital Libraries*, 105-106.
- Bier, E. A. and Perer, A. (2005) Icon abacus: Positional display of document attributes. *Proceedings of the 5th ACM/IEEE Joint Conference on Digital Libraries*, 289-290.
- Camreon, M., Mor, N., Boyd, D. and Davis, M. (2006) HT06, tagging paper, taxonomy, Flickr, academic article, to read. *Odense, Denmark: Proceedings of the Seventeenth Conference on Hypertext and Hypermedia*, 31-40.
- Cortez, E., Silva, A., Gonçalves, M., Mesquita, F. and Moura, E. (2007) FLUX-CIM: flexible unsupervised extraction of citation metadata. *Proceedings of Joint Conference on Digital Libraries*, 215-224.
- Cui, H. (2008) Unsupervised semantic markup of literature for biodiversity digital libraries. *Proceedings of Joint Conference on Digital Libraries*, 25-28.
- Dontcheva, M., Drucker, S., Wade, G., Salesin, D. and Cohen, M. (2006) Summarizing personal web browsing sessions. *Proceedings of User Interface Software and Technology*, 115-224.
- Hetzner, E. (2008) A simple method for citation metadata extraction using hidden markov models. *Proceedings of Joint Conference on Digital Libraries*, 280-284.

- Hsieh, H-W. and Shipman, F. (2000) VITE: A visual interface supporting the direct manipulation of structured data using two-way mappings. *Proceedings of the 5th International Conference on Intelligent User Interfaces*, 141-148.
- Hui, H., Giles, C., Manavoglu, E., Zha, H., Zhang, Z. and Fox, E. (2003) Automatic document metadata extraction using support vector machines. *Proceedings of Joint Conference on Digital Libraries*, 37-48.
- Hunter, J., Khan, I. and Gerber, A. (2008) Harvana: Harvesting community tags to enrich collection metadata. *Proceedings of Joint Conference on Digital Libraries*, 147-156.
- Kerne, A., Koh, E., Smith, S., Webb, A. and Dworaczyk, B. (2008a) combinFormation: Mixed-initiative composition of image and text surrogates promotes information discovery. *ACM Transactions on Information Systems*, Vol. 4, 1-47.
- Kerne, A., Toups, Z., Dworaczyk, B. and Khandelwal, M. (2008b) A concise XML binding framework facilitates practical object-oriented document engineering. *Proceedings of Document Engineering*, 62-65.
- Klavans, J., Sheffield, C., Lin, J. and Sidhu, T. (2008) Computational linguistics for metadata building. *Proceedings of Joint Conference on Digital Libraries*, 427-427.
- Koh, E. and Kerne, A. (2006) I keep collecting: College students build and utilize collections in spite of breakdowns. *European Conference on Digital Libraries*, 303-314.

- Koh, E. and Kerne, A. (2009) Deriving image-text document surrogates to optimize cognition. *Proceedings of the 9th ACM Symposium on Document Engineering*, 84-83.
- Liu, Y., Bai, K., Mitra, P. and Giles, C. (2007) TableSeer: Automatic table metadata extraction and searching in digital libraries. *Proceedings of Joint Conference on Digital Libraries*, 91-100.
- Lu, X., Kahle, B., Wang, J. and Giles, C. (2008) A metadata generation system for scanned scientific volumes. *Proceedings of Joint Conference on Digital Libraries*, 167-176.
- Mathur, A. and Kerne, A. (2009) *Meta-Metadatas Repository*. Retrieved from <http://ecologylab.net/code/java/cf/config/semantics/metametadatas.zip>. 15 October 2009.
- Petras, V., Larson, R. and Buckland, M. (2006) Time period directories: A metadata infrastructure for placing events in temporal and geographic context. *Proceedings of Joint Conference on Digital Libraries*, 151-160.
- Recker, M. and Palmer, B. (2006) Using resources across educational digital libraries. *Proceedings of Joint Conference on Digital Libraries*, 240-241.
- Shipman, F., Hsieh, H., Airhart, R., Maloor, P. and Moore, J.M. (2001) The visual knowledge builder: A second generation spatial hypertext. *Proceedings of the ACM Conference on Hypertext*, 113-122.
- Shiri, A. (2008) Metadata-enhanced visual interfaces to digital libraries. *Journal of Information Science*, 763-775.

- Wong, J. and Hong, J. (2007) Making mashups with marmite: Towards end-user programming for the web. *Proceedings of Special Interest Group on Computer-Human Interaction*, 1435-1444.
- Yaron, D., Davenport, J., Karabinos, M., Leinhardt, G., Bartolo, L., Portman, J., Lowe, C., Sadoway, D., Carter, W. and Ashe, C. (2008) Cross-disciplinary molecular science education in introductory science courses: An NSDL Matdl collection. *Proceedings of Joint Conference on Digital Libraries*, 70-73.
- Yilmazel, O., Finneran, C. and Liddy, E. (2004) MetaExtract: An NLP system to automatically assign metadata. *Proceedings of Joint Conference on Digital Libraries*, 241-242.

APPENDIX A: META-METADATA LANGUAGE SYNTAX AND CONSTRUCTS

The Meta-Metadata language which we have developed allows power users to add new information sources for collection visualization applications without knowing the details of the application code. In this document we discuss the syntax and constructs of the language.

Adding a new information source involves following steps:

1. Authoring the data structures to represent and store the information.
2. Specifying the way to extract information from the source and build instances of the data structures.
3. Writing the logic for operations on these instances to build visualizations.

Meta-Metadata language provides a Metadata Definition Language which can be used to author data structures for information sources and specify the ways to build instances representing them. For writing logic for operations on these instances it provides a scripting language which can be used to do the semantic scripting. These two modules are discussed below.

1. Metadata Definition Language

Metadata definition language allows us to define new information sources using Meta-Metadata definitions.

When defining a new information source we are either extending an existing information source definition by adding new fields to it or are specifying how the various fields of an existing source can be filled by using extraction rules.

A Meta-Metadata definition is constructed by:

- Defining a unique name for the Meta-Metadata. This is done by using the `name` attribute of `meta_metadata` XML element. Any valid, meaningful and unique string can be used as its value.
- Define the `binding` attribute. This attribute is used to tell the kind of parsing will be used for the information source. There are three different kinds of bindings possible:
 - a) `xpath`: This is used in case we need to parse a HTML DOM and create metadata objects from it. Examples include Google Search, ACM Portal.
 - b) `direct`: This is used when the information source provides an XML containing the information fields. Examples include Yahoo Search.
 - c) If the `binding` attribute is not present we use the basic HTML parser. Example include New York Times
- Define either the `type` or `extends` attribute.
- Define the `generate_class` attribute. The default value of this attribute is `true`, which case we will generate classes for the source definition.
- Define either `url_base` or `url_prefix` attribute. These should be unique for every source definition we author in the repository.
- Define the `user_agent_name` attribute. This is the string a web browser provides to a web server.

- Define the `package` attribute. This attribute tells which Java package the generated classes should be placed into. It is used to organize the generated classes.
- Define the `comment` attribute. This attribute adds meaningful descriptions to the source.
- Define the Meta-Metadata fields which will comprise the metadata for the information source. This is done using `meta_metadata_field` tag nested inside the `meta_metadata` tag.

Defining Meta-Metadata fields

There are three types of Meta-Metadata fields which can be defined using Data Definition Language:

- **Scalar Fields:** These are primitive data types and other single valued data types like `String`, and `ParsedURL`.
- **Nested Fields:** These are the fields which contains a list of scalar fields inside them. These can be used to define new metadata classes within an existing Meta-Metadata definition.
- **Collection Fields:** These are the fields which consist of a set of list of scalar fields. These can be used to define new metadata classes within an existing Meta-Metadata definition.

We discuss below the attributes which are common to scalar fields and collection fields as well as attributes which are specific to each of them.

A) Common Attributes

1. `name`: This attribute gives the name of a Meta-Metadate field. Any meaningful name string can be used as the value of this attribute.
2. `xpath`: This attribute is required for Meta-Metadate with binding as 'XPath'. Any valid XPath expression can be used as the value of this attribute.
3. `context_node`: Node on this the XPath has to be applied. Default value of this node is DocumentRoot for scalar fields and is collection object node for collection fields.
4. `regular_expression`: The regular expression to apply on extracted information.
5. `replacement_string`: The string to be used to replace the match.
6. `comment`: Description about the field.

B) Scalar Fields

1. `scalar_type`: This field defines the data type of Meta-Metadate field.

C) Nested Fields

1. `is_nested` : It is true for nested fields

D) Collection Fields

2. `collection`: Defines the type of collection attribute. Example ArrayList
3. `collection_child_type`: The class name for the collection field.

4. `generate_class`: Defines whether we need to generate a new class for the collection field.

Attribute Name	Purpose	Possible Values	Required/Optional
<code>name</code>	The name of the Meta-Metadata field	Any valid string	Required
<code>scalar_type</code>	The data type of the Meta-Metadata field	String, ParsedURL, Int, Boolean	Required for scalar fields
<code>XPath</code>	The XPath expression to be applied for extraction		Required for fields of Meta-Metadata with <code>binding='XPath'</code>
<code>context_node</code> [Default Values] Document Root for scalar fields. Collection node for collection fields.	Node on which the XPath has to applied	Any valid XPath expression	Optional
<code>collection</code>	The type of collection fields	ArrayList	Required for collection fields
<code>collection_child_type</code>	The class for the collection type.	Any collection class name	Required for collection fields
<code>generate_class</code> [Default Value] true	Tells whether to generate class for a collection field	true, false	Optional
<code>regular_expression</code>	The regular expression which needs to be applied on the value extracted from current field	Any valid regular expression	optional
<code>replacement_string</code>	The string which should be replaced with matching string	Any valid string	Optional

Table above summarizes all the fields, their purpose, default values, possible values and whether there are required or not. Following are the various attributes which can be used while defining a Meta-Metadatas field.

Semantic Action Scripting

There are three kinds of statements which scripting language provides:

1. Variable Definition Statements: These statements are used to declare new variables, which are either a DOM node or a DOM node list. They can then be used in conjunction with Meta-Metadatas field declarations, when forming extraction rules to write concise XPath expressions. They can also be used in control flow and metadata-operation statements to operate on document DOM directly.
2. Control Flow Statements: These statements include loops and conditional statements which govern the flow of control of operation on metadata objects.
3. Metadata-Operation Statements: These statements operate on metadata objects created to either build image and text surrogates from them, download them immediately or send them to crawler to be downloaded latter.

We discuss the syntax and description of each of these statements below:

1. Variable Definition Statements

Variables are declared with the `def_vars` XML tag. There are following attributes possible for this element.

- `name`: This attribute defines the name of the variable. Any meaningful descriptive string can be used as its value.

- `xpath`: This attribute defines the XPath expression to be used to evaluate the value of this variable. Any valid XPath expression can be used as its value.
- `type`: This attribute defines the types of variable. It can be either a single DOM node or a DOM Node list. Depending on this it can have either `node` or `node_set` as its value.
- `node`: This attribute defines the name of the node on which the XPath expression should be applied. If this is not present the XPath is applied on the Document Root.
- `comment`: This attribute gives a meaningful description of the variable and its purpose.

2. Control Flow Statements

Control flow statements are declared inside `semantic_actions` XML element. There are two kinds of control flow statements:

- `for_each`: This statement is used to loop over a collection of metadata objects of same kind. It can have a sequence of both control flow and metadata-operation statements nested inside it. Each of these statements is then executed repeatedly for each instance of metadata object in the collection. It is declared using `for_each` XML tag. It must have two attributes, `collection` and `as`, which gives the name of the collection object and iterator over the collection object. It can also have following optional attributes.
 - i. `start`: Specifies the starting index of the loop.

- ii. `end`: Specifies ending index of the loop.
- iii. `current_index`: Specifies current index of the loop
- iv. `size`: Specifies the size of the collection
- `if`: This statement is used for conditional execution of control flow and metadata-operation statements. It is nested inside a specific control flow or metadata-operation statement. It has a list of `flag_check` elements nested inside it. Each `flag_check` has a mandatory `value` attribute which gives the names of the flag which must be true for the parent metadata-operation or control flow operation to execute.

3. Metadata-operation statements

Metadata-operation statements allow us to operate on metadata objects to download and parse documents and form image and text surrogates from them. Each of them is called by their own specific name tags and arguments. There are following similar attributes and nested fields in each of them.

- a) Possible Attributes: We give below the possible attributes which are possible for each of the metadata-operation statements. However depending on the name of metadata –operation statements all of these attributes might not be needed.
 - `object`: This attribute gives the name of the object on which the operation has to be performed. Any valid object name which has been defined can be used as its value. Default value of this attribute the metadata object created from the current Meta-Metadata definition.

- `name`: This attribute contains the name of the returned object from the action. Any meaningful unique name can be used as its value. Depending upon whether the action returns any value or not, this attribute can be present or absent.

b) Possible Nested Fields: We give below the possible nested fields which are possible for each metadata-operation statements. These fields include the arguments to be passed, and checks to be performed on the returned value and before performing the current action. Depending upon the metadata-operation some of these nested fields might be absent. We discuss all of the possible nested fields below:

- `checks`: This element includes inside it one or more check elements which contain the validations to be performed on the returned value from an action. check element has two mandatory attributes:
 - i. `condition`: This attribute gives the condition to be checked on return value. It can have two possible values: `NOT_NULL` and `METHOD_CHECK`. The first one checks if the returned value is not equal to null and the second one is used for methods returning a boolean value.
 - ii. `name`: It gives the name of the flag to be set as true if the return value satisfies the condition specified using the condition attribute

- `if`: This element gives the conditions only under which the current action must be executed.
- `args`: This element includes inside it one or more `arg` elements which contains the name of the objects which needs to be passed as an argument for the current operation. `arg` element has one mandatory attribute `value` which contains the name of the object which needs to be passed as an argument for the current operation.

We now discuss the specific metadata-operations which are available to the user:

- A) `get_field`: This action is used to get the value of a metadata field from metadata object. It has two mandatory attributes:
- `name`: Name of the field to get
 - `object`: Name of the object whose field we have to get.
- B) `create_container_for_search`: This action creates a container object that is a search result. The container created can then be processed (downloaded) to immediately download and parse the associated search document or sent to the agent for processing later. It takes one argument which is the link(data type `ParsedURL`) of the search result and returns the resulting container object of type `Container`. It implicitly sets the metadata of the resulting container object to the search class object.
- C) `create_container`: This action returns the container object, of type `Container`, associated with a given link. If container already exists for the link, it

is returned else a new container is created and returned. We have to set the metadata for this container explicitly. It takes following arguments:

- Either a link to document or an entity or document.
- Anchor text string. This is optional.
- Boolean specifying the citation significance. This is optional.
- Anchor context string. This is optional.
- Float specifying citation significance value. This is optional.

D) `set_metadata`: This action sets the metadata of a container. It operates on a container object and takes the metadata object to be set as argument. If a container was created using `create_container` action we need to set its metadata explicitly using this action.

E) `process_document`: This action takes a container object as argument and downloads it immediately.

F) `queue_document_download`: This action takes a container object as argument and sends it to the crawler agent to download later. The crawler then puts it into its list of candidates to be downloaded. Based on application specific weighting strategies this container may or may not be downloaded. For combination, we add the container to the candidate container list of `CfInfoCollector`, and then based on the weights and term vector, `CfInfoCollector`, may or may not download it. Notice the difference between this action and `process_document` action where the container is downloaded immediately.

- G) `create_semantic_anchor`: This action creates semantic anchors for the selected `<a>` elements in the container, and adds them to candidate containers for the parent container. These containers can then be downloaded based on the weighting strategy of the visualization application. It takes following arguments:
- Link of the candidate container. This can be obtained from the href attribute of the select `<a>` elements.
 - The text which describes the `<a>` element. This is optional.
 - The text of the paragraph in which `<a>` element is present(context). This is optional.
- H) `create_and_visualize_img_surrogate`: This action creates a image surrogate and sends it to the visualization layer for visualization. It takes the following arguments:
- The URL of the image.
 - Height of the image. This is optional.
 - Width of the image. This is optional.
 - href attribute of the `` element. This is optional.

APPENDIX B: BNF FOR META-METADATA LANGUAGE

```

<meta_metadata>::=      "<meta_metadata name="<SV>
                        ("extends=" | "type=")<SV>
                        ["binding="<SV>]
                        ("url_prefix="|"url_base=")<SV>
                        ("user_agent=")<SV>
                        ["comment="<SV>]
                        ["generate_class="<SV>]
                        ["package="<SV>]
                        ">"
                        [<def_vars>]
                        {meta_metadata_field}
                        [semantic_actions]
                        "</meta_metadata>"

<def_vars>::=          "<def_vars>" {def_var} "</def_vars>"

<def_var>::=           "<def_var name=" <SV>
                        "xpath="<SV>
                        "type="<SV>
                        ["node="<SV>]
                        ">"

<meta_metadata_field>::= "<meta_metadata_field name="<SV>

```

```

[("scalar_type="<SV>|
  "collection="<SV>"collection_child_type="<SV>)]
["xpath="<SV>]
["regular_expression="<SV>]
["replacement_string="<SV>]
["hide="<SV>]
["style="<SV>]
["navigates_to="<SV>]
["shadows="<SV>]
["is_nested="<SV>]
["is_facet="<SV>]
["generate_class="<SV>]
{<meta_metadata_field>}
"</meta_metadata_field>"
<semantic_actions>::=
  "<semantic_actions>"
  {<semantic_action>}
  "</semantic_actions>"
<semantic_action>::=
  <get_field>|
  <create_container_for_search > |
  <process_search >|
  <create_container >|
  <set_metadata >|

```

```

<process_document>|
<create_and_visualize_img_surrogate>|
<queue_document_download>|
<create_semantic_anchor>|
<get_xpath_node>
<get_field>::=      “<get_field name=” <SV>
                    [<if>]
                    [<checks>]
                    ”/>”
<create_container_for_search>::= “<create_container_for_search
                                name=”<SV>
                                ”>”
                                [<if>]
                                [<checks>]
                                <args>
                                ”</create_container_for_search >”
<process_search>::= “<process_search>”
                    [<if>]
                    [<checks>]
                    <args>
                    “</process_search>”
<create_container> ::=      ” <create_container  name=”<SV>

```

```

“return_type=”<SV> ”>”
[<checks>]
[<if>]
<args>
“</create_container >”
<set_metadata>::= “<set_metadata object=”<SV> ”>”
[<checks>]
[<if>]
<args>
”</set_metadata>”
<process_document>::= “<process_document object=”<SV>
[<checks>]
[<if>]
“/>”
<create_and_visualize_img_surrogate> ::=
”<create_and_visualize_img_surrogate>”
[<checks>]
[<if>]
<args>
”</create_and_visualize_img_surrogate>”
<create_semantic_anchor>::= ”<create_semantic_anchor>”
[<checks>]

```

```

    [<if>]
    <args>
    "</create_semantic_anchor >"
<queue_document_download>::= "<queue_document_download>"
    [<checks>]
    [<if>]
    <args>
    "</queue_document_download>"
<get_xpath_node>::= "<get_xpath_node name="<SV>
    "xpath="<SV>
    ["node="<SV>]
    "return_object="<SV>
    [<checks>]
    [<if>]
    "</get_xpath_node>"
<checks>::=
    "<checks>"
    {check}
    "</checks>"
<check>::=
    "<check name="<SV>
    "condition="<SV> "/>"
<if>::=
    "<if>"
    {flag_check}

```


APPENDIX C: ADDING NEW SEARCH ENGINES

```
<search_engines>
  <search_engine name="google"
    url_prefix="http://www.google.com/search?hl=en&amp;q="
    numResultString="&amp;num="
    startString="&amp;start="/>
  ...
</search_engines>
```

Figure C-1: Search URL definition for Google search engine

The ability to add new search engines provides the ability to add new digital libraries and collections like Wikipedia to collection visualization application without having to write any custom code for it. For this reason we have a search engine definition data structure that can be used to define search URLs for different search engines. Search URL definition for Google Search engine is shown in Figure 8. In this figure, name attribute tells the name of the search engine, which is *google* in this case. url_prefix attribute gives the starting string for search URL. This is then appended by the query string, which is followed by numResultString, denoting the number of search results to obtain. It is then followed by startString, which gives the index of first result in the result set. Note that we have escaped & by using & in the xml. This is shown in Figure C-1.

VITA

Name : Abhinav Mathur

Address : Abhinav Mathur
c/o Dr. Andruid Kerne
Department of Computer Science
Texas A&M University
College Station TX 77843-3112

Email Address : abhinav@abhinavmathur.net

Education : B.Tech., Computer Science and Engineering, IIT Guwahati, 2006

M.S., Computer Science, Texas A&M University, 2009