Software Requirements Specification

vyasa

Prepared by Fred Eaker

2006 November

Table of Contents

Revision History

Name	*Date*	*Reason for Change*	*Version*
Fred Eaker feaker@users.sourceforge.net	October 2006	Creation	1.0
Fred Eaker feaker@users.sourceforge.net	November 2006	JSR-170	1.1.0
Fred Eaker feaker@users.sourceforge.net	December 2006	Open-source Java	1.1.1

1. Introduction

1.1 Purpose

This document specifies the software requirements for the **vyasa** digital library application.

1.2 Document Conventions

The **vyasa** digital library application is frequently referred to as "the system."

Metadex refers to the combination of metadata and indexes generated by digital asset analysis (see section 3.2).

1.3 Intended Audience and Reading Suggestions

This document describe the project scope for software developers. Readers should also be familiar with the JSR-170 (Java Content Repository) API specification[1].

1.4 Project Scope

vyasa is a digital library application that incorporates the functions of digital asset and document management systems. Its primary purpose is to facilitate information retrieval and knowledge discovery by providing comprehensive, automatic metadata generation through ontological reasoning and semantic analysis.

Versioning, workflow management, transactions, locking and digital rights management are currently beyond the scope of this project.

2. Overall Description

2.1 Product Perspective

vyasa leverages the features of a variety of asset management functions to create a comprehensive digital library solution. The strength of the project lies in the implementation and cohesion between its individual parts. These parts work together to create a flexible knowledge discovery tool that reveals the latent knowledge contained in a collection of digital assets.

1 http://jcp.org/en/jsr/detail?id=170

2.2 Product Features

The features of the system fall into 4 main divisions: loading, retrieval, search and analysis.

The system loads any type of digital asset to analyze it and automatically generate metadex. Analysis results are confirmed by the user, and once complete, the original digital asset as well as its associated metadex are stored in a repository. Metadex supports search and retrieval through the JSR-170 API (Level 2).

The system conforms to JSR-170 Level 2 but does *not* provide the following optional features: transactions, locking and versioning.

2.3 Operating Environment

The system is developed in the Java programming language. It runs on any hardware and operating system supported by the latest Java Virtual Machines.

Sun Microsystems open-sourced the Java platform in November 2006.[2] Wikipedia currently lists 6 implementations of Java.[3] Developers should identify any possible incompatibilities across these implementations and Apple's implementation for OS X.

2.4 Design and Implementation Constraints

vyasa is an open source project and will be licensed under an Open Source Initiative approved "protective" license[4]. The exact open source license to be used by the system is not known at this time. According to the Free Software Foundation, an application that uses many different open source components under different licenses requires an analysis of those components and their respective licenses.[5] Developers should ensure that the licenses of open source components are compatible[6] to make this analysis easier (see section 5.4.2).

2.5 User Documentation

User documentation is delivered as a PDF manual and in-application contextual help.

2 http://www.sun.com/smi/Press/sunflash/2006-11/sunflash.20061113.1.xml
3 http://en.wikipedia.org/wiki/Java_%28programming_language%29#Java_Implementations
4 As defined by Mark Webbink in his article "Understanding Open Source Software" http://www.nswscl.org.au/journal/51/Mark_H_Webbink.html
5 http://www.gnu.org/licenses/gpl-faq.html#ManyDifferentLicenses
6 http://www.gnu.org/licenses/gpl-faq.html#WhatIsCompatible

3.0 System Features

3.1 Digital Asset Loading (DAL)

3.1.1 Description and Priority

DAL functions handle the process of loading multiple types of digital assets prior to analysis.

Priority Level: Low

3.1.2 Stimulus/Response Sequences

1. User signals the system that they would like to load a digital asset into the repository.
2. System responds by presenting the user with an interface for loading a single digital asset.
3. User selects a local or remote digital asset to be loaded.
4. System informs the user if the digital asset is unreadable.
5. System passes the readable digital asset to the digital asset analysis (DAA) process.

3.1.3 Functional Requirements

DAL-1: Provide the user with an interface to point to a local or remote digital asset.

DAL-2: Verify the existence and readability of the asset. Inform user if digital asset is unreadable.

DAL-3: Save the asset in a temporary storage location and create a programmatic reference to its location.

DAL-4: Pass the asset to the digital asset analysis process.

3.2 Digital Asset Analysis (DAA)

3.2.1 Description and Priority

DAA functions handle the analysis of digital assets and the generation of their metadex.

Priority Level: High

3.1.2 Stimulus/Response Sequence

1. Receive a digital asset that needs analysis.
2. Determine the type of digital asset that is being analyzed (document, image, audio, video).
3. Based on the asset type, perform an analysis to determine: 1) its contents and properties, 2) relationships to assets already in the repository and 3) updates to the metadex of the assets already in the repository reflecting the existence of the new asset.
4. Present the results of analysis to the user for confirmation and provides methods for manually entering metadex information not found during analysis.
5. Send the digital asset and its metadex to the digital asset repository along with ay updates to existing assets to reflect the existence of the new asset.

3.1.3 Functional Requirements

DAA-1: Determine the type of digital asset that is begin analyzed (document, image, audio, or video) using Magic DB.[7]

DAA-2: Based on asset type, perform an analysis of the asset's contents and properties.

Asset Type	*Index*	*Metadata*
document	full-text, semantic	Dublin Core, taxonomy
image		Dublin Core, EXIF, TIFF, folksonomy
audio		Dublin Core, ID3, APE, Vorbis, folksonomy
video		Dublin Core, folksonomy
other	full-text, semantic (if possible)	Dublin Core, folksonomy

DAA-3: Provide the user with an interface to 1) verify the accuracy of the automatically generated metadex and 2) manually enter or upload data into the metadex.

DAS-4: Pass the asset and its metadex to the digital asset repository.

7 http://magicdb.org/

3.3 Digital Asset Repository (DAR)

3.3.1 Description and Priority

DAR provides storage, retrieval and query functions for digital assets and associated metadex.

Priority Level: High

3.3.2 Stimulus/Response Sequence

1. Store, retrieve, update and query digital assets and metadex.

3.3.3 Functional Requirements

DAR-1: Store digital assets and associated metadex in the repository.

DAR-2: Provide methods to search across metadex.

DAR-3: Allow updates to be made to digital assets and their metadex (versioning is beyond scope).

4.0 External Interface Requirements

4.1 User Interfaces

Detailed user interface requirements have been purposefully omitted from this document. A separate user interface specification for the system's default GUI will be created.

The system's user interface is intended to be customizable by any designer or developer that chooses to do so. This flexibility is accomplished by ensuring that content and presentation remain completely separate through the model-view-controller (MVC) design pattern. Product branding will also be supported with the use of custom graphics.

4.2 Hardware Interfaces

All hardware interfaces are handled by the Java Virtual Machine.

4.3 Software Interfaces

Conformance to Level 2 of JSR-170.

4.4 Communications Interfaces

The JSR-API will be the sole interface for with the repository. Providing an additional web service communication layer is currently beyond scope.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The performance of digital asset indexing and searching is measured by response time and precision. Performance in document retrieval is measured by several formulas specific to the science of information retrieval.[8]

The performance of browsing and displaying is measured by response time and is heavily influenced by user perception.

There are currently no benchmarks available for response time or precision. See Appendix C.

5.2 Safety Requirements

There are no safety requirements.

5.3 Security Requirements

Security of the copyright-protected content that may be stored in the system, known as digital rights management, is currently beyond the scope of this project.

5.4 Software Quality Attributes

5.4.1 Understandability

Application code and comments should be written descriptively. The names of classes, methods and variables should be self-descriptive. Methods and classes will be commented to detail their purpose.

5.4.2 Completeness

All external libraries including their respective licenses should be documented.

5.4.3 Conciseness, Consistency and Efficiency

8 http://en.wikipedia.org/wiki/Information_retrieval#Performance_measures

Application code will be reviewed regularly to remove redundancy, reduce complexity and increase efficiency.

5.4.4 Portability

Testing across multiple platforms (Windows, OS X, Linux) and implementations of the Java platform should ensure that code and external libraries are not platform or implementation-dependent.

5.4.5 Maintainability

Application code will be cohesive and have easily recognizable functionality. Classes will be abstract enough to facilitate changes in data structures. Class and function modularity should be implemented to avoid the need for major refactoring.

5.4.6 Reliability

Checked exception handling is highly recommend. Extensive testing is also required.

5.4.7 Security

See section 5.3.

6. Other Requirements

6.1 Internationalization

Digital assets and their metadex, especially text documents, may originate from any location in the world. Internationalized analysis, search and display will be supported through the use of Unicode and through the separation of content and presentation. More research needs to be done on this requirement to ensure that the application is designed appropriately. See Appendix C.

Appendix A: Glossary

DAA—digital asset analysis; a function that involves collection of data about the digital asset (metadata) and, if the asset is a text document, various indexes that identify its contents (full-text, semantic, ontological).

DAL—digital asset loading; a function that facilitates the selection and loading of digital assets.

DAR—digital asset repository; stores digital assets along with their metadex.

digital asset—any form of text and/or media that have been formatted as a binary source.[9]

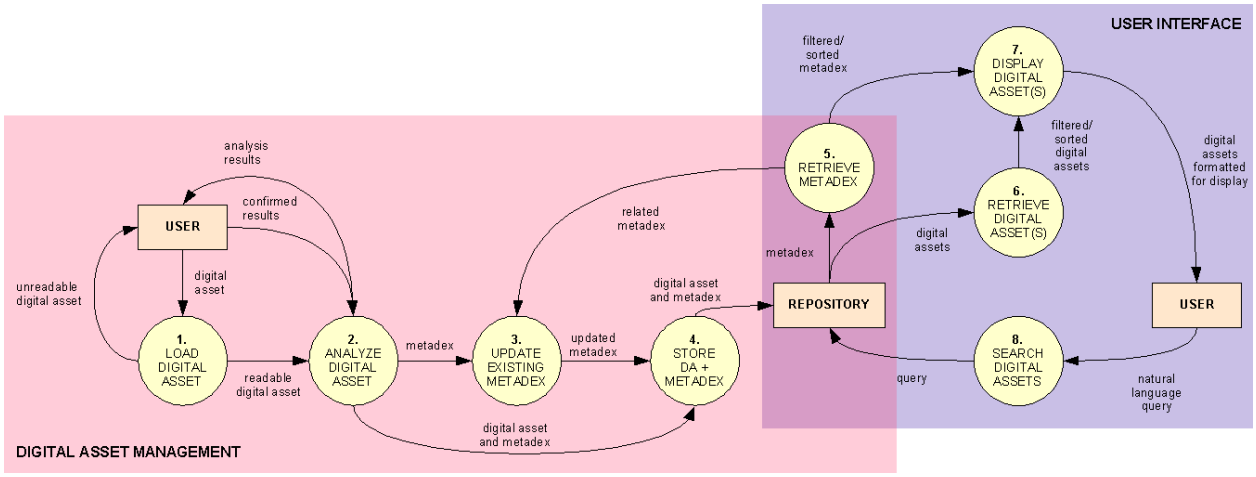
metadex—a combination of metadata and indexes used to identify and search digital assets.

repository—a central place where data is stored and maintained; it can be either remote or local to the user or system.

Appendix B: Analysis Models

Data Flow Diagram (DFD)

The following data flow diagram represents the entire system. The scope of this document is represented by the light red box encompassing processes 1-5. The user interface DFD (represented by a light blue box) is included to give a broader view of the system until a formal user interface specification is created.



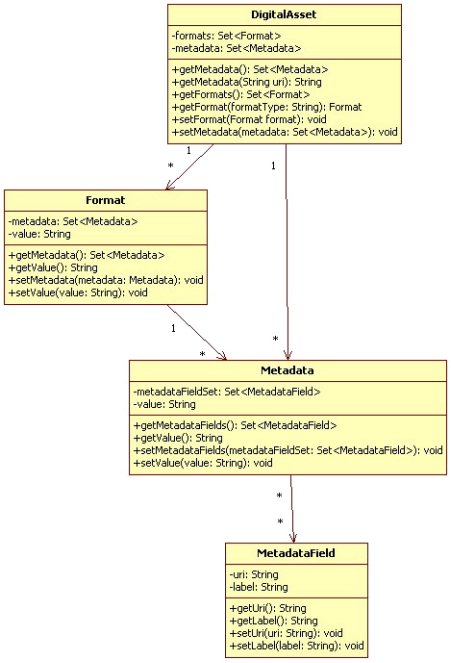
9 http://en.wikipedia.org/wiki/Digital_asset

Class Diagrams

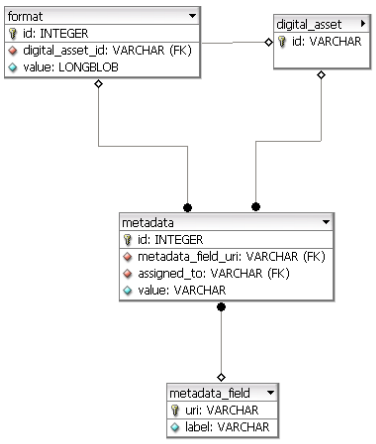
A `DigitalAsset` object is composed of a set of `Format` objects and a set of `Metadata` objects.

The `Metadata` object is composed of a `MetadataField` object and its corresponding value. `MetadataField` objects consist of a URI and label.

The `Format` object represents multiple presentations of a digital asset. For example, a digital asset could be represented as XML or XHTML, WAV or MP3 as well as TIFF or JPG. Formats not only share the metadata of their parent digital asset, but also have their own specific metadata. For example, a digital asset that has `dc:type = Image` could have multiple formats where `dc:format = tiff` (original) and `dc:format = jpeg` (thumbnail).



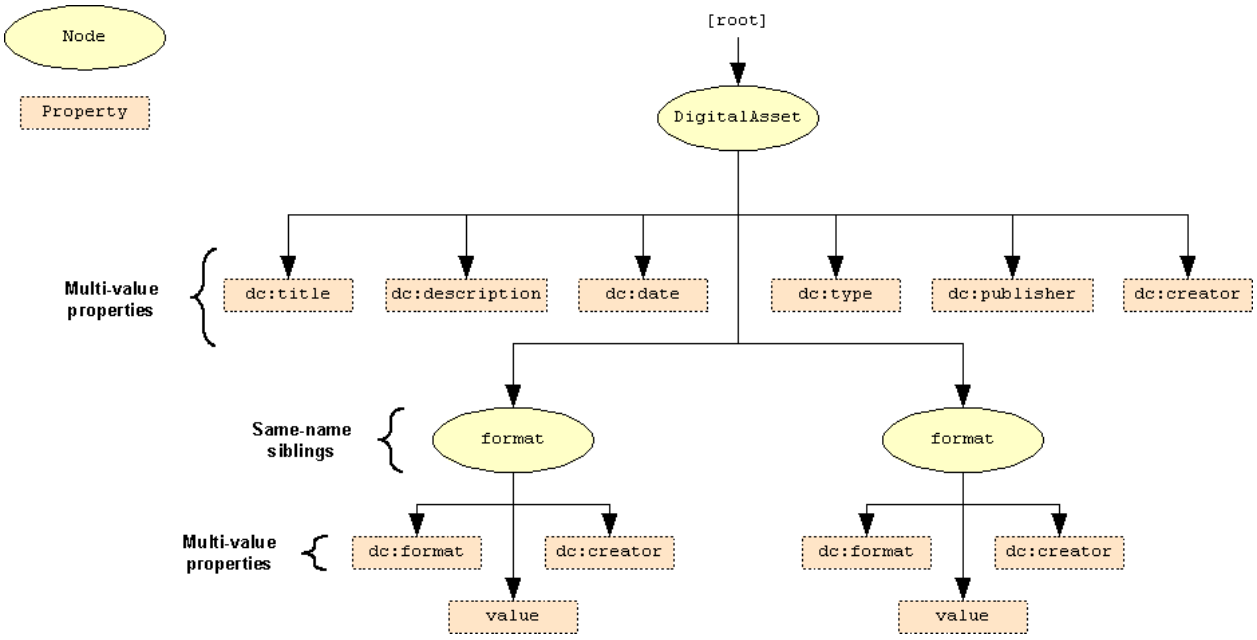
This class model is mapped and implemented into the following relational database model:



These models demonstrates the follow attributes:

- A particular metadata URI and label does not change between digital assets or formats, only the actual metadata *values*.
- A digital asset or format can be assigned multiple values of the same metadata (i.e., multiple authors, languages, etc).

The models described above are represented below in a tree-structure as required by JSR-170:



In this model, digital assets are child nodes of the root node. The metadata of the digital asset is represented as multi-value properties. Formats are child nodes of the digital asset and their metadata are multi-value properties. The value property of a format node is *not* multi-value (format as a same-name sibling handles the possibility of multiple formats).

Appendix C: Issues List

Issue Name	***Description***	***Priority***
Internationalization	Research options for accommodating multiple languages in indexing, searching and user input.	High
Software license	Maintain a list of all external libraries and their respective licenses. Contact the Free Software Foundation and/or the Open Source Initiative to determine what licenses are compatible and what license would work for this project.	Middle
User interface specification	A study of usability and an evaluation of the user interfaces of other document management systems will be useful.	Middle
Java implementation research	Research the various implementations of the Java platform to determine what discrepancies need to be considered.	Middle
Performance requirements	Evaluate other document management systems to determine their level of performance and what can be expected of this project.	Low