

Name Access Control Module

Final Report

Team MetaCat

Matthew
Horoszowski
Ben Greenwood

Rob Busack
Dean Rzonca

Anthony Lyo

Sponsored by University of Rochester River Campus Library
Nathan Sarr, Jennifer Bowen, David Lindahl, and Jeffery Suszczynski

Faculty Mentor

Robert Bubacz

Project Overview

Currently, the University of Rochester uses an aging proprietary system to hold digital information on material in the library collection. This system, while functional, has limited features and a non-intuitive user interface. They would like to extend the functionality of the system with custom-written software.

One module of the new eXtensible Library software will be the Name Access Control Module. Lots of bibliographic data has been manually entered into Marc Bibliographic records, Marc Authority records, and Dublin Core records, but not without errors. The responsibilities of the Name Access Control Module are finding erroneous data in Subject and Author fields of bibliographic records, and identifying the correct authority record that it should be matched to.

The Name Access Control Module will take the existing MARC and Dublin Core metadata from the existing XML files, parse them, create relations between bibliographic records and authority records, and then provide an easy mechanism for other systems to search and interact with this data. This is complicated by the fact that not all the data in the existing database is correct. When creating relations between records, the system will be able to identify and correctly match records that have alternate forms of the authority name. The system will create new authority records if none exists in the system. There are a variety of rules for matching these records, and it must be easy to modify the matching rules in our system. Additionally, the system must allow librarians to view and confirm or reject matches that the system is uncertain about.

This project will cover the NACM, an API for the NACM, and a GUI to demonstrate the functionality of the NACM. Other applications that will eventually make up the rest of the eXtensible Library are not part of this project.

The NACM will be responsible for converting the existing legacy information to a more open system that can be easily accessed by other programs, as well as through the NACM API. As such, it will be the foundation of the eXtensible Library system. It will also provide functionality such as being able to view reports about the data, as well as detecting and attempting to fix incorrect information.

Basic Requirements

The project main purpose was to build out a database that links bibliographic data to authoritative data. First it has to build out its store of that data, and then use those stores to create links. A number of methods were employed to perform the matches, from trying different data to manipulating the data through a set series of transformations.

The first requirement for the project dealt with the importing of data. There were three record types that had to be imported, MARC auth data, Marc bib data, and Dublin Core bib data. All of these would be imported as xml, or as raw data that is then converted to xml. These files would have to be identified and the required information extracted before entries could be made in the database.

The next major requirement came with creating a matching system. The matcher makes use first of the data's authoritative name information, then its alternate name information in order to match books to authors.

To help the match process along, a number of different string transformers were applied to the data, in hopes that one will find a match. These transformers are used with every type of match.

Finally if a match can not be found with any of the methods described, a new authoritative record needs to be created to match the bibliographic data present. This generated authority record would help match two or more books together, even when no preexisting authoritative data exists for them.

The primary output of the program is the database itself, along with the various APIs that can be used to read it. This project will be used in other software applications, after the database and links have been built out.

Constraints

The software was written in Java, and also required MySQL and Hibernate to be installed. It was developed and tested with Windows XP as the expected operating system on which it will run. No minimum hardware requirements were expected beyond what is necessary for the software listed, but of course the slower the hardware, the longer the batch processing can be expected to take. No anticipations of coexisting but conflicting software packages on the target machine were made.

Software Dependencies –

- Java SE 1.5 Runtime Environment
- MySQL 5.0
- Hibernate Core and Hibernate Annotations

Development Process

Iterative and incremental process was used for this project. This process allowed the development team to implement incrementally and learn from the earlier deliverables of the system. It also allowed the customers a chance to give feedback on the deliverables.

The process consisted of initialization step, and iteration step.

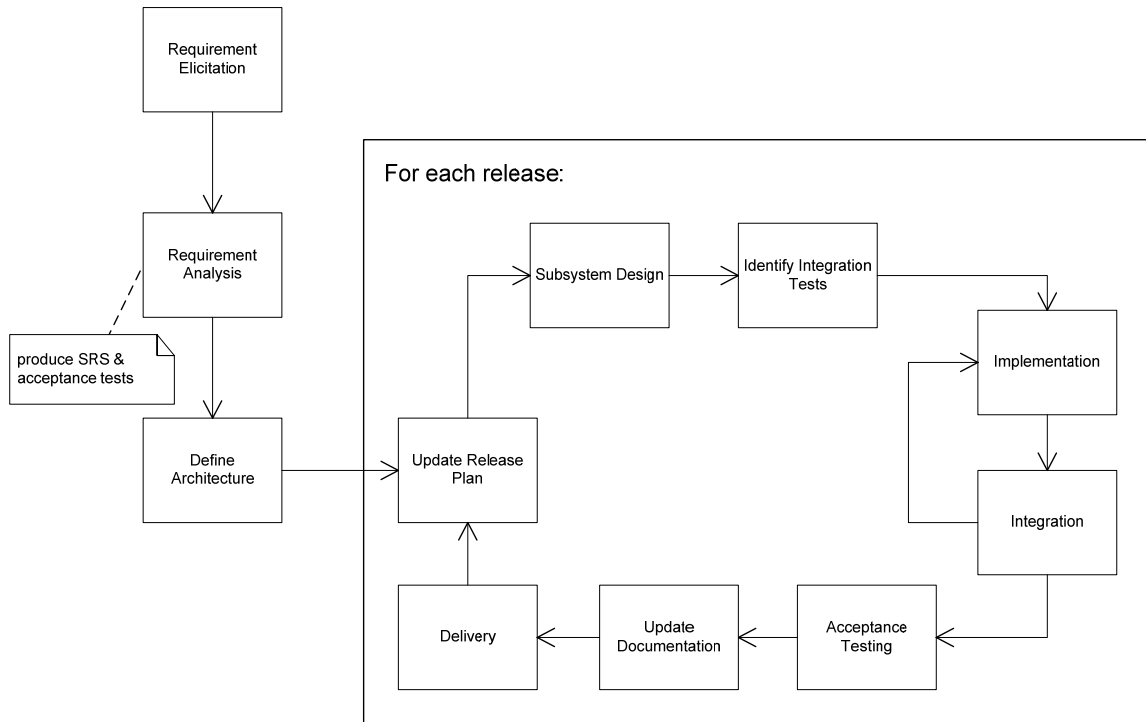


Figure: Iterative and incremental process flow

The initialization step consisted of domain analysis, scope analysis, and planning. During this release, the development team discusses the details of the project with the customers regarding the key aspects of the problem. The team then created an initial project plan, a draft requirements and specifications (SRS). Using SRS, an initial architecture for the system was created so that the team could start working on implementation.

The initialization step set the motion for the first iteration of the project. Each project's iteration involves updating release plan, subsystem design, integration test plan, implementation, integration, acceptance testing, documentation revision, and, finally, deliverables for review. At the end of iteration, the customers got a chance to review the deliverables and give feedback to the team. The next iteration took account of the customers' feedback and makes adjustments accordingly.

Project Schedule: Planned and Actual

The project schedule was divided into four different releases, with the first three releases each containing one of the three different matching process the team would develop, and the fourth containing smaller features and improvements that were discovered during the creation of the first three releases.

The incremental nature of the team's development process allowed for a constant revision of the project schedule. Each major release saw shifting in the priorities and time tables of the release following it.

With the updating of the schedule, the team never fell to far from its expected goals. In the end the final schedule had been modified, but on average developed similar to the initial schedule. For every task that was over estimated another had been underestimated, with the constant schedule revisions ensuring that no development time was lost as a result.

System Design

The NACM is split up into a set of different components. The GUI sits as one component, reaching down and touching the API components. These components are the Importer and Match-Driver system, as well as the DAO system.

Importer

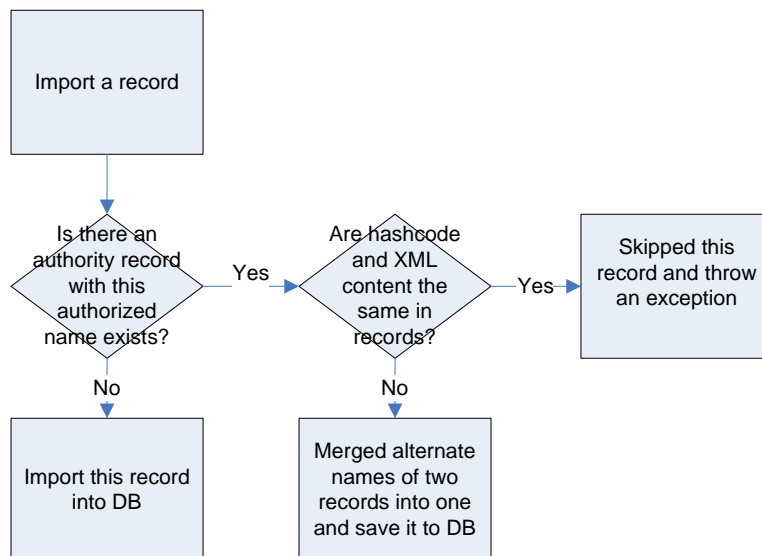


Figure 1 - Process for importing an authority record

Importing a New Authority Record

Pseudocode for importing an authority record

```

Function import(AuthorityRecord authNew) {
    if there is an authority record with authNew.authorizedName exists {
        if( hashcodes & the xml file content of two records are not the same) {
            merge two records into one
        } else {
            Skip authNew
        }
    } else {
        Import authNew
    }
}
  
```

Merging Two Authority Records

When two authority records are merged, we combine alternate forms of names and save the new xml file as the original xml in the database.

Importing a Bibliographic Records

```
Function import(BibliographicRecord bibNew) {  
    if( XMLhashcode for bibNew already exists in the database) {  
        Skip bibNew  
    } else {  
        Import bibNew  
    }  
}
```

Class Diagram

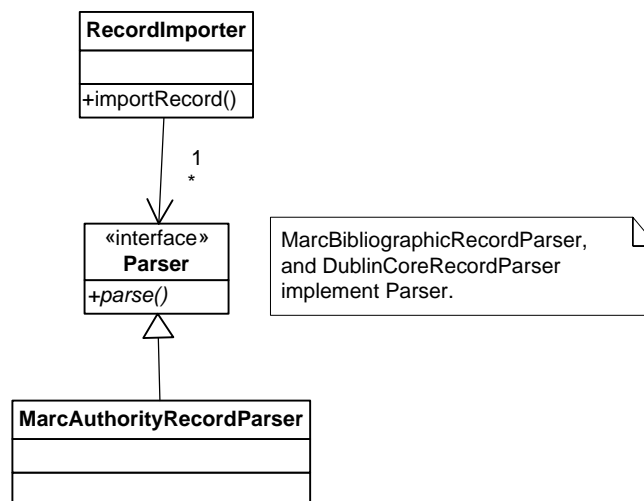


Figure 2 – Class diagrams for importer module

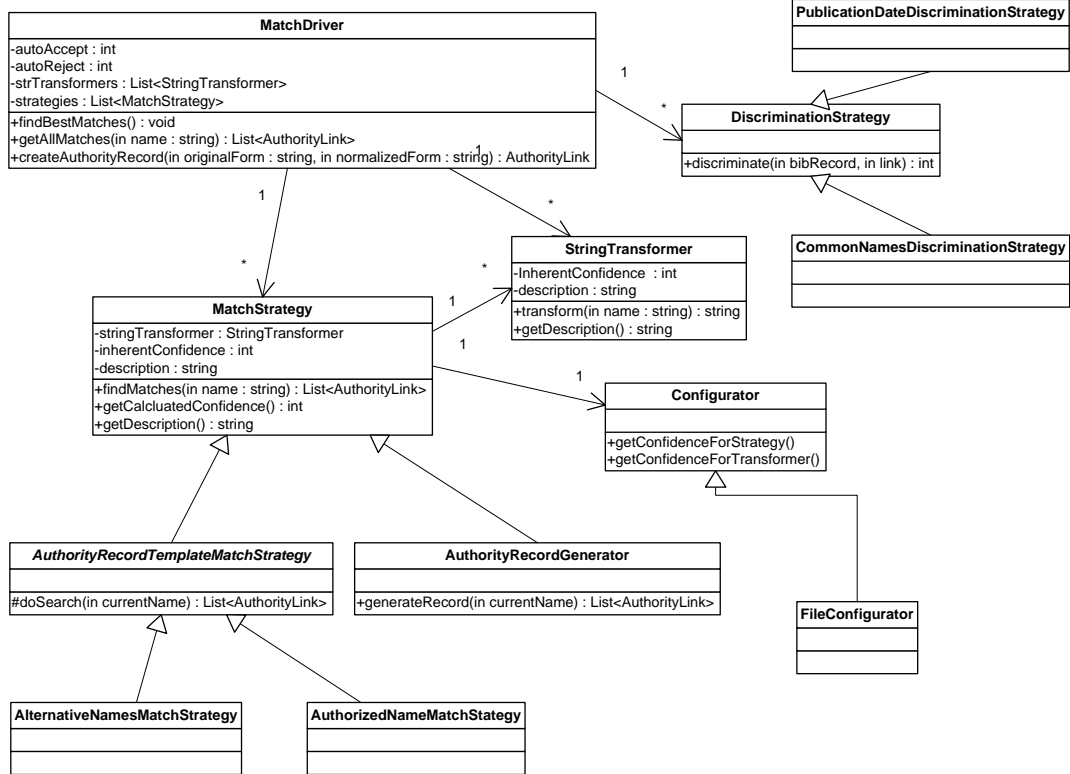
Parser

This is an interface for Parser. MarcAuthorityRecordParser, MarcBibliographicRecordParser and DublinCoreRecordParser implement Parser.

RecordImporter

This class uses multiple parsers to parse appropriate record formats. It checks an input file and identifies what type of record it is. Then it calls appropriate parser to parse the record.

Match-Driver



Descriptions

MatchDriver

This class is responsible for starting the matching process. It contains a list of string transformers and matching strategies to use for matching. MatchDriver goes through several steps to perform matching, and it is described below.

At start up, before matching has been requested, the MatchDriver creates a list of MatchStrategy-StringTransformer pairs. A properties file (see `java.util.Properties`) is used to list the `autoAcceptThreshold`, the `autoRejectThreshold` and the confidence levels of MatchStrategies and StringTransformers. All those values are integers between 0 and 100, representing percentages.

MatchDriver then creates a list of pairs, using all available MatchStrategies, paired with all available StringTransformers, so there will be $N * M$ pairs. The MatchDriver checks the confidence level of the strategy/transformer pair, calculated by multiplying the transformer confidence by the strategy confidence, and discards any pairs whose confidence is less than the auto reject threshold. The list of MatchStrategy objects is then sorted in order of decreasing confidence.

The MatchDriver then repeatedly grabs a small set of unmatched bib records. For each record, the MatchDriver will iterate through the sorted pairs until it finds a match, or the pair list is exhausted. If a match is found, the link is run through discriminators, which can modify the overall confidence of the link based on factors such as how common a name is and the publication date verses author birth date. After that, the link between the records is stored. If no record is found, an auth record is created using the bib record's information, and a link between the generated record and the bib record is stored.

In MatchDriver

```
void findBestMatches() {
    while(dao.hasUnmatchedRecords)
        records = dao.fetchUnmatched( batchSize )
    for each record {
        for each name {
            foreach MatchStrategy // Strategy-Transformer pairs
            {
                links = strategy.findMatches(name)
                if ( links.length >= 0 ){
                    break
                }
            }
            if ( links.length > 0 ){
                applyDiscriminators(link)
                storeLink(links)
            } else {
                links = generateRecord(name)
                storeLink(links)
            }
        }
    }
}
```

```

        } // next name in record
        dao.fastImport(bib record)
    } // next record
}

List<Link> getAllMatches(Name) {
    foreach MatchStrategy // Strategy-Transform pairs
        matches.add( strategy.findMatches(name) )
    } // next strategy

    return matches
}

```

In AuthorityRecordTemplateMatchStrategy

/* All links returned by this should have the same confidence. If we ever find a situation where they are not all the same confidence, this should sort the list so the highest confidence is first. */

```

List<Link> findMatches (name) {
    searchString = transformer.transform(name)
    linksFound = doSearch (searchString) // this is what will be overridden
in subclasses

    for each linksFound
        link.confidence = (inherentConfidence *
                           transformer.inherentConfidence) / number of links

    return linksFound
}

int getCalculatedConfidence() {
    return this.inherentConfidence * myTransformer.inherentConfidence;
}

// This method is abstract in AuthorityRecordTemplateMatchStrategy; implemented
in subclasses
List<Link> doSearch(searchString) {
    return dao.findBy[whatever](searchString)
}

```

MatchStrategy

This is an abstract class. It defines a method named findMatches() which find matches and creates links between records. It also has methods for calculating its overall confidence when given a transformer to use. Currently there are 2 instance-able strategies; AuthorizedNameMatchStrategy and AlternateNameMatchStrategy.

StringTransformer

This class performs different operations on strings. The various transformers perform actions on names such as swapping first and last names, abbreviating names, removing titles, and various other manipulations.

AuthorityRecordTemplateMatchStrategy

This is an abstract class. It uses a template method, `doSearch()`, to compare authorized name and alternate names.

AuthorizedNameMatchStrategy

This class is a `MatchStrategy` to compare authorized names.

AlternativeNamesMatchStrategy

This class is a `MatchStrategy` to compare alternate names.

DiscriminationStrategy

This is an abstract class. It defines a method that takes a link and can modify the link confidence.

CommonNameDiscriminationStrategy

Decreases the confidence of a link if the surname is very common.

PublicationDateDiscriminationStrategy

Decreases the confidence of a link if the publication is before the author's birth date.

Thresholds

There are two thresholds in the configuration file.

Auto-accept threshold

When a match results in a confidence level greater than this threshold, we create a link between the bibliographic record and the authority record. We mark the link status as confirmed. All the matches that are above this confidence level will not be shown to the users to confirm or reject.

Auto-reject threshold

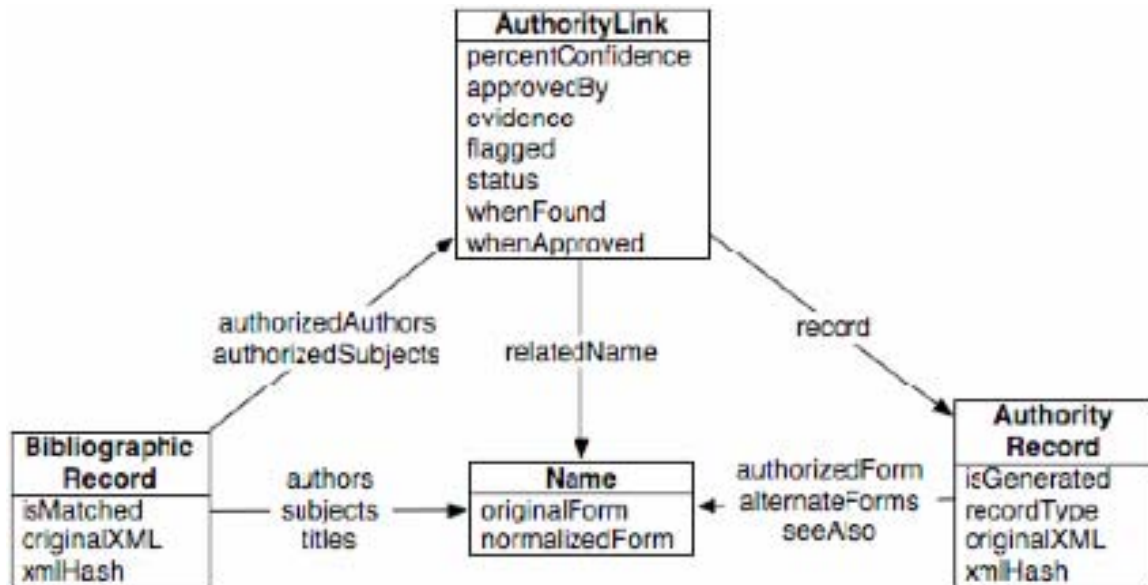
When a match results in a confidence level lower than this threshold, we ignore the result. But if the confidence level is between

those two thresholds, we create a link between the bibliographic record and the authority record. We mark this link as to be reviewed by the user (either to reject or to confirm).

Assumptions

- No two authority records have the same authorized name.
- Library of Congress id is unique for every MARC authority record.

DAO and Database



Descriptions

BibliographicRecord

Represents a bibliographic record. Contains a set of authors, subjects and titles. Also contains a set of AuthorizedAuthors and AuthorizedSubjects, which can be used to retrieve authority records for these items. A flag indicates whether or not matching has been performed. The original XML from which file was imported is saved, along with a hash code to facilitate fast comparisons for duplicate checking.

AuthorityRecord

Represents an authority record. Contains a single authorized form of the name, along with a set of alternate forms and see-also references. The recordType indicates whether it is a MARC, Dublin Core or XC generated record. Like bibliographic records, the original XML along with a hash code is also stored.

AuthorityLink

Links a BibliographicRecord to an AuthorityRecord. Information about the match is also stored here. A related name is used to link this back to a single author or subject in a BibliographicRecord. This should reference a Name directly from the parent BibliographicRecord.

Name

A single name of an author, subject or title. Contains the original form of the name, exactly as it was typed, along with a normalized form.

Constraints

Comparisons between domain objects only use scalar fields, so collections are not examined when equals() is called. This improves performance by allowing all collections to remain lazily-loaded until they are explicitly requested.

Data Persistence

Class Diagram

QuickTime™ and a
TIFF (LZW) decompressor
are needed to see this picture.

Descriptions

DAOFactory

Creates DAO objects that are bound to a common database connection. All DAOs with a common session work within the same transaction, and are isolated from DAOs on other connections, allowing for thread safety that can be managed by the client. Objects loaded by a DAO created by one DAOFactory cannot be saved by DAOs created by another factory, nor should they be combined with objects from other factories.

BibliographicRecordDAO

Responsible for creating updating, loading and removing bibliographic records. A number of finder methods are provided to facilitate searching for records. All operations are cascaded to associated Names and AuthorityLinks, so deleting a BibliographicRecord will also delete associated AuthorityLinks. Only

updating is cascaded to AuthorityRecords, so deleting a BibliographicRecord will not delete an AuthorityRecord.

AuthorityRecordDAO

Responsible for creating, updating, loading and removing authority records. A number of finder methods are provided to facilitate searching for records. All operations are cascaded to associated Names, so deleting an AuthorityRecord will also delete any associated Names. For this reason, Name instances should not be shared among AuthorityRecords and other AuthorityRecords, or AuthorityLinks or BibliographicRecords.

AuthorityLinkDAO

Responsible for updating and loading associations between BibliographicRecords and AuthorityRecords. A single finder method is provided which accepts a Filter. Results are filtered, sorted and paginated according to the Filter. Only updated operations are cascaded to Names and AuthorityRecords, since the lifecycle of the related Name is shared with the lifecycle of its parent bibliographic record, and AuthorityRecords have their own lifecycles.

Filter

Contains parameters for retrieving data to be displayed to the user. Boundaries and sorting conditions can be set individually and passed as a single parameter to a DAO for a specific search.

Features

Using Hibernate for data persistence offers many useful features. All collections of associated objects in the domain model are lazily loaded, so they will only be retrieved when they are specifically requested. Database queries are all cached automatically, and objects are only saved if they have actually been modified. Operations automatically cascade among objects, making a really simple interface for modifying data. This also adds a few constraints, listed below.

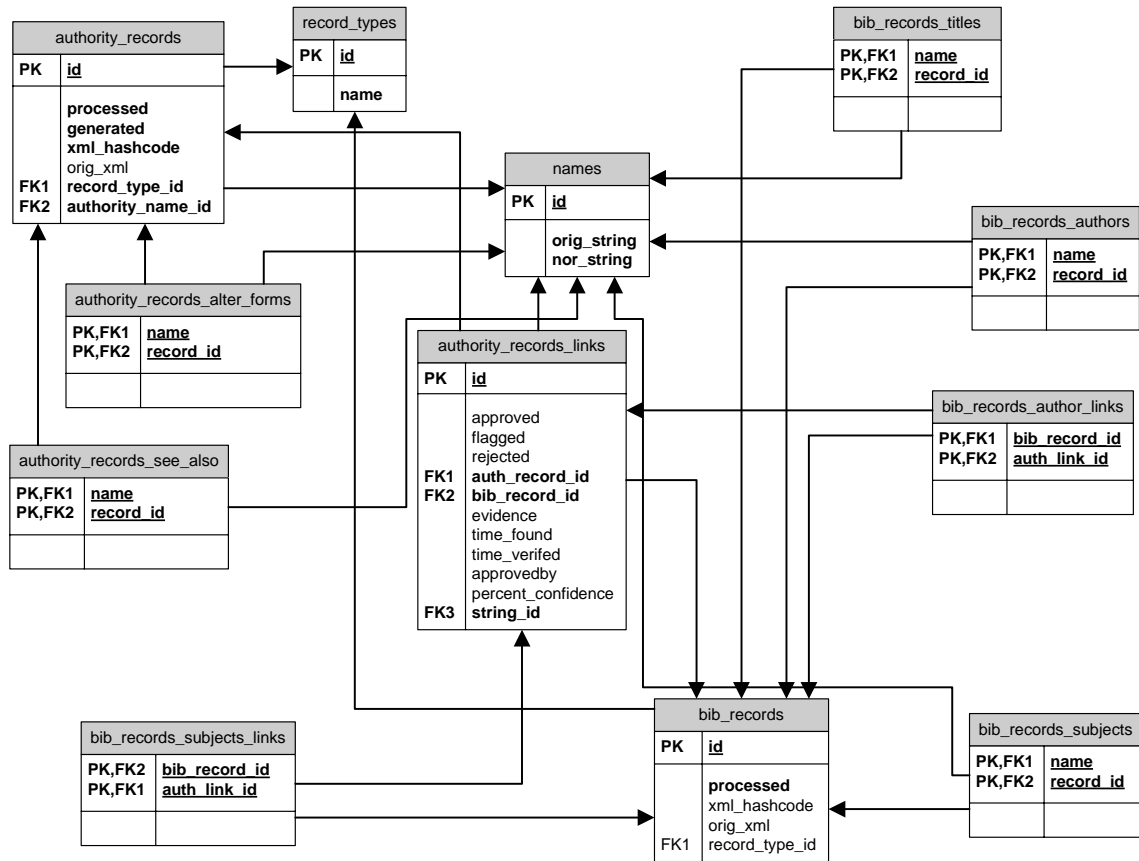
Constraints and Guidelines

All domain objects are uniquely identified by their ID field. This field is automatically managed by the DAOs. Two domain objects having the same ID are, for all practical purposes, the same object, and they will represent the same row in the database. This is an important consideration when sharing associated objects (such as Names) between higher-level objects like bibliographic records or authority records. BibliographicRecords and AuthorityRecords manage the lifecycles of all directly associated objects, so deleting a bibliographic record will always delete all of the associated Names, whether or not they are being used by another bibliographic record (which is likely to cause an error).

The BibliographicRecordDAO and AuthorityRecordDAO both have fastInsert methods as well as store methods. When doing batch processing, always use fastInsert, as this bypasses any caching between the DAO and database. On large batch compare operations, the session can still become cluttered and cause the program to run out of heap space and crash. To avoid this, periodically close the session using the close() method on the DAOFactory being used, and create a new DAOFactory to work within a new session.

A further word of caution: findAll() methods will return all records in a table, regardless of the memory needed. fetch methods are provided instead of findAll methods for production use.

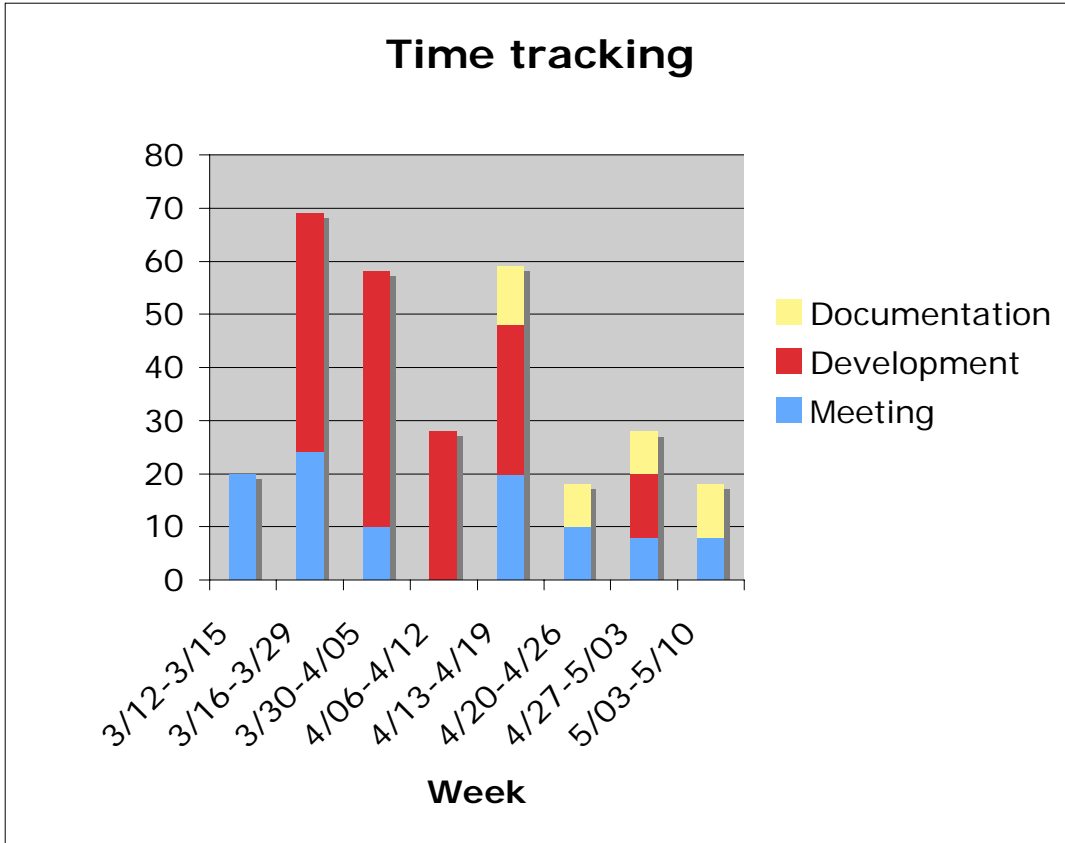
MySQL Data Model



Process and Product Metrics

Effort by type of activity

Record the amount of time the team spends on different activities such as requirements, design, implementation, documentation, and others. It shows how much time the team is spending on each activity, and attempts to improve efficiency if possible.



Test metrics

Analyze code coverage, # of test cases per requirement (use case, user story etc), % of tests passed successfully.

Defects by types

Record the number of defects found in different releases of the project such as requirements, design, implementation, documentation, and others.

Issue status	Count	Percentage
Open	0	(0%)
Taken	1	(1%)
On hold	6	(7%)
Rejected	3	(3%)
Completed	78	(89%)
Total	88	

Issues by sections	Open	Taken	On hold	Rejected	Completed	Total
Client GUI	0	1	2	2	32	37
Importer/Exporter	0	0	2	0	26	28
Matching Algorithm	0	0	3	1	25	29
DAO	0	0	1	0	14	15
MySQL DB	0	0	0	0	5	5
Process	0	0	2	0	3	5
Documentation	0	0	1	0	4	5
Other	0	0	0	0	3	3
Public API	0	0	0	0	1	1

Product State at Time of Delivery

The product at delivery is largely as was originally expected. All three of the main matching phases are implemented, and many of the smaller features are completed as well. The program makes use of all the data provided, performs authority name, alternate name, and generated matching, as well as making use of string transformers to help the matching process along.

The prototype GUI is working well, with only a few usability issues left (such as the manual matching requiring a wide screen to be user friendly). Future development will work to incorporate the existing product in other software projects. New functionality could be added to support new types of records, new string transformers, and new methods for searching and getting back data from the database.

Project Reflection

The team's process was very well suited for this project, and it is largely due to this that we feel the project went well. By using an incremental and iterative process we were able to deal with loose initial requirements by constantly updating them. At the start of the project the domain of the project was not well known by any member of the team,

and all domain knowledge had to be acquired from a source that had never had to think about quantifying their process down into something a computer could repeat. A lot of back and forth communication with the sponsor helped to build out the teams knowledge of how data matching was being done manually by library staff. As the team's knowledge grew, the requirements where refined and the project progressed towards its goal.

It would have been helpful for the project to have visited the U or R library staff earlier in the development process. More usability testing with the entire staff would have helped the team understand the problem sooner. Also the early usability testing would have show what areas of the project needed the most work sooner, allowing more to have been done.

In the end the team learned a lot about software development from this project. The very loose nature of the initial requirements taught the team how to collect better requirements from a source that is not quite sure what it wants. While there were some lesson learned of a technical nature, mostly dealing with improving the performance of the product, most of the team's lessons where of a social nature. The team learned to work better together with itself as well as the client to produce a product that all could be happy with in the end.

Name Access Control Module

User Guide

Requirements:

Java Runtime Environment 6
MySQL 5

Installation

The Name Access Control Module (NACM) is provided as a Java jar file located within the provided zip archive. Extract the zip file to the installation location of your choice.

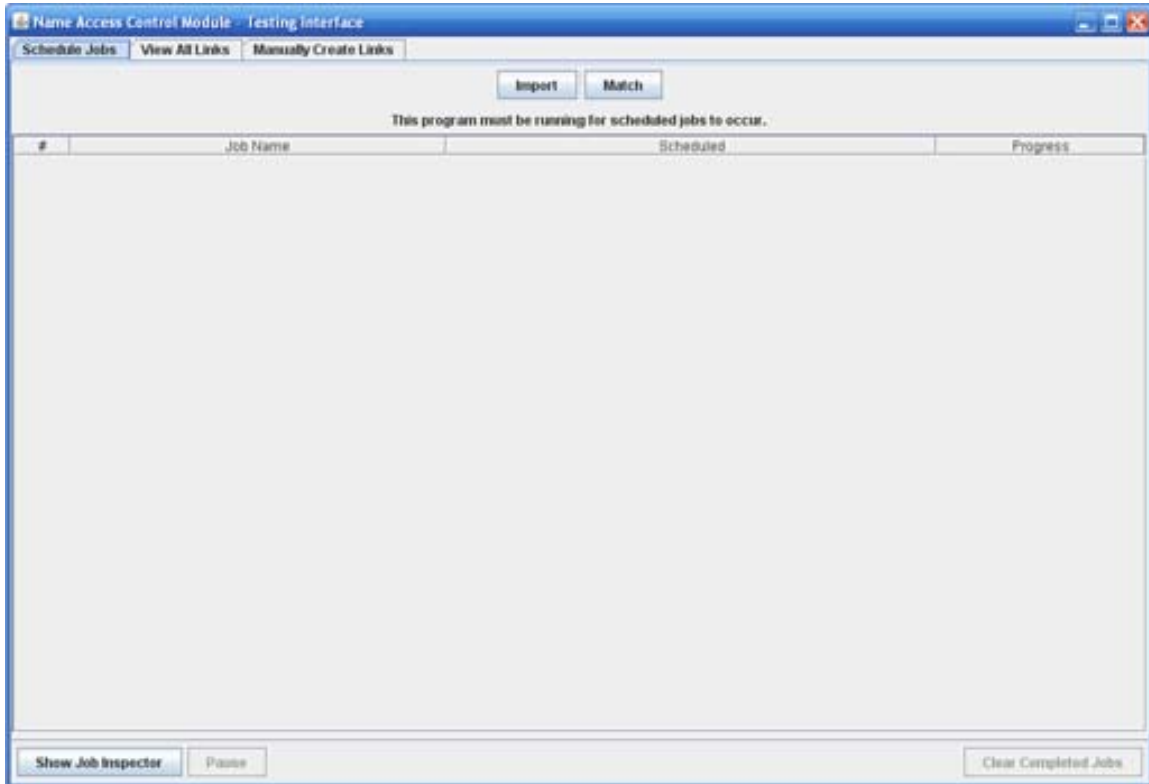
The local MySQL database must be configured before you can run the NACM. A new database user must be created with the login name “nacm” and the password “nacm”. Then a new database must be created named “nacm”. The user must have full permission to this database. Finally the provided sql create script must be run in order to fill out the nacm database schema.

Running

The NACM can be run by executing the extracted jar file via the java runtime engine. This can be done from command line by typing “java -jar nacm.jar”. Once run the NACM graphical user interface will load, showing three selectable tabs. Each tab will bring up a different panel of the NACM. The Schedule Jobs panel is brought up by default.

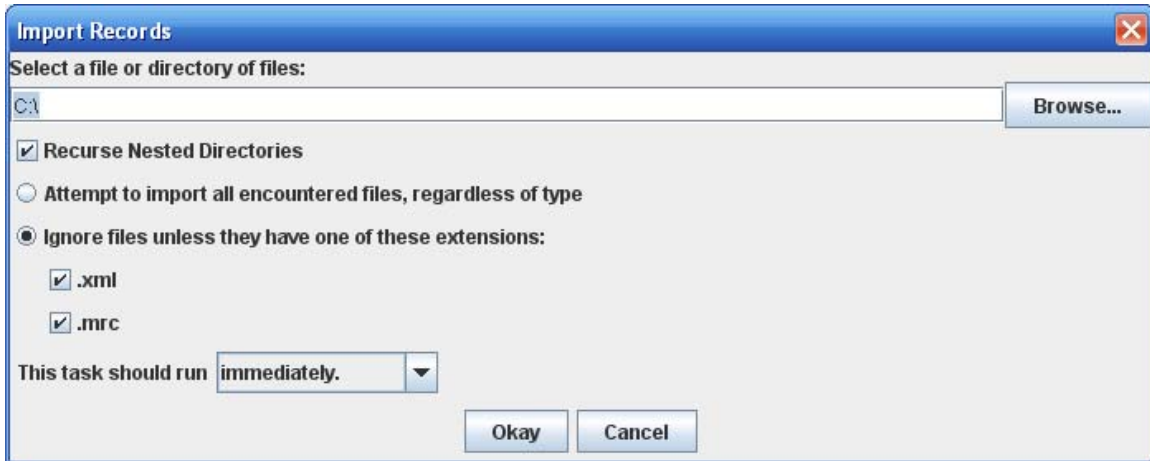
Schedule Jobs Panel

The Schedule Jobs panel is the main area of the program to initiating record importing and record matching. The two buttons at the top are used to initiate these processes.

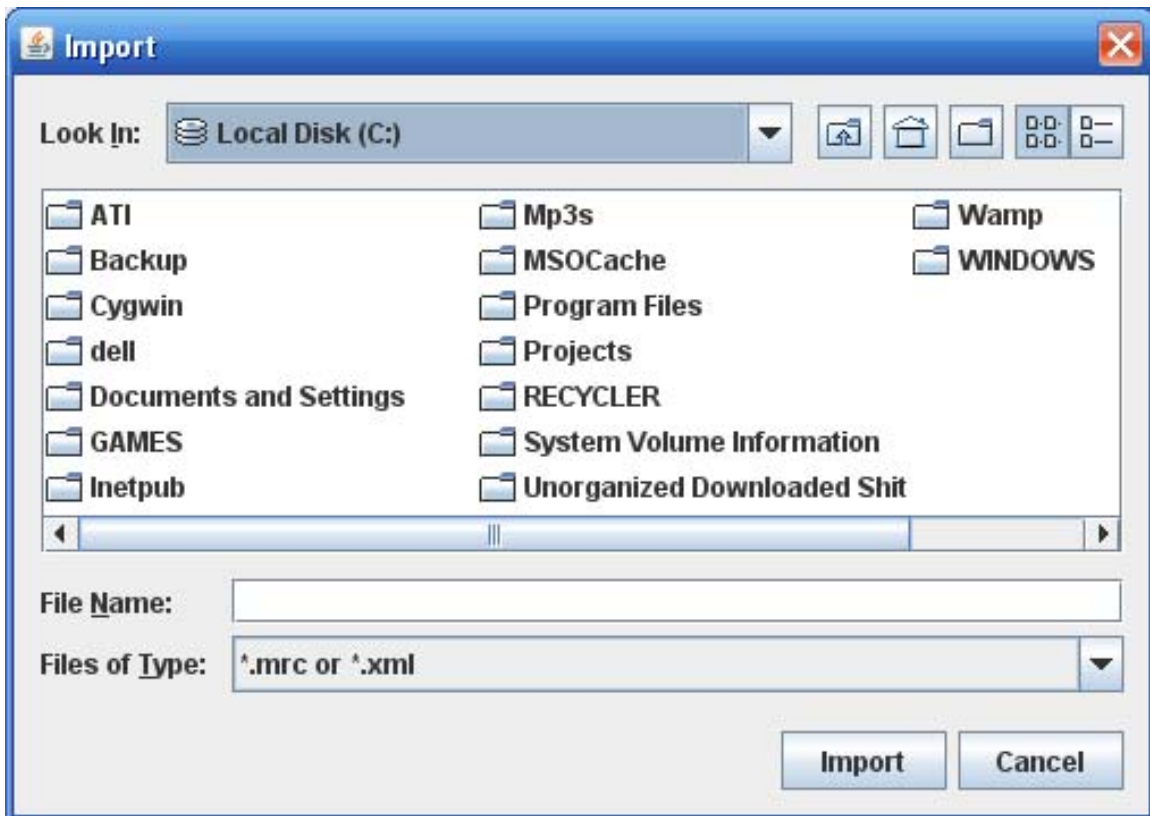


Importing

To start a importing job click on the Import button. This will bring up a new dialog.



Importing can be done by either selecting a single file to import, or by selected a directory of files to import all at once. To do either click on the Browse... button and it will bring up a file or folder select dialog.



To select a single file for import, simply find the file you are looking for and select it. Then click import.

To select a entire directory for import, find the directory you want to import and select it. Then click import.

If you are importing a directory, and want all sub directories to be imported, ensure that the “Recurse Nested Directories” option is check. If you do not want sub directories imported, make sure it is unchecked.

If you have marc or Dublin core files that do not end with either .xml or .mrc and want them imported select “Attempt to import all encountered files, regardless of type”. If all of the files to import have the .xml or .mrc extension, select “Ignore files unless they have one of these extensions” and then select the types of extensions you want imported.

If you want the task to run as soon as you finish configuring it, select this task should run “immediately”. If you want the task to run at a specified date and time select “at this date and time” and then specify the date and time. Finally if you want the task to run after a different task, select “after this job completes” and then the job you want it to run after.

Click Okay if you are happy with your selection, or Cancel if you want to abort the import. If Okay was click a new line will be added to the Scheduled Jobs panel showing the job. If start immediately was selected the import will have been started.

Matching

To start a matching job click on the Match button. This will bring up a new dialog.



If you want to record a trace of what is done during the matching process, make sure the “Store a trace of each record’s matching process” option is checked.

If you want the task to run as soon as you finish configuring it, select this task should run “immediately”. If you want the task to run at a specified date and time select “at this date and time” and then specify the date and time. Finally if you want the task to run after a different task, select “after this job completes” and then the job you want it to run after.

Click Okay if you are happy with your selection, or Cancel if you want to abort the import. If Okay was click a new line will be added to the Scheduled Jobs panel showing the job. If start immediately was selected the match will have been started.

Examining Jobs

Once a job has been created and started, you can view more information about it. If you select the job then click on show job inspector, you can view this information.

The screenshot shows the 'Name Access Control Module - Testing Interface' window. At the top, there are tabs for 'Schedule Jobs', 'View All Links', and 'Manually Create Links'. Below these are 'Import' and 'Match' buttons. A message states: 'This program must be running for scheduled jobs to occur.' Below this is a table with columns: '#', 'Job Name', 'Scheduled', and 'Progress'. The table contains one row: '#1', 'Import all records in auths directory.', 'Running now: 460 of 5,172 succeeded, 121 skipped', and a progress bar. Below the table is the 'Job Inspector' section. On the left, it displays: 'Name: #1. Import all records in auths directory.', 'Scheduled to run: Running now', 'Started at: May 15, 2007 11:23 AM', 'Time Elapsed: 1m22s', 'Est. Time Remaining: 13m38s', 'Completed at:', and 'Steps: 460 of 5,172 succeeded, 121 skipped'. On the right, it says 'A total of 121 steps were skipped for these reasons:' followed by a table with columns '#', 'Reason', and 'Reason'. The table has one row: '#121', 'NoAuthorizedNameFoundException', and 'Reason'. Below this is a scrollable list of error messages: 'msplit1003.xml: No authorized name found', 'msplit1008.xml: No authorized name found', 'msplit1009.xml: No authorized name found', 'msplit1015.xml: No authorized name found', 'msplit1016.xml: No authorized name found', 'msplit1017.xml: No authorized name found', 'msplit1022.xml: No authorized name found', 'msplit103.xml: No authorized name found', 'msplit1030.xml: No authorized name found', and 'msplit1031.xml: No authorized name found'. At the bottom, there are buttons for 'Hide Job Inspector', 'Pause', and 'Clear Completed Jobs'.

The section in the lower left shows information about the task, such as total run time and estimated time to completion. The section on the right is used to report errors from the task. The top part gives a list of types of errors found, as well as the number of error raised. When a error is selected you can get more detailed information about the last 100 occurrences of this error.

View All Link Panel

Enter limitations about which links should be shown. The table will automatically refresh.

Show only flagged

Show: unverified accepted rejected

Confidence range: Date range:

Min: 0% Min: Dec 31, 1969 7:00 PM

Max: 100% Max: May 15, 2007 11:32 AM

Hide Filter Options Refresh Show subfields in name columns

These filter settings match 7228 out of 7228 existing links, divided over 8 pages.

You are viewing page 1 of 8, with 1000 links per page. < Previous Page Next Page >

Flagged	Marked As	Authoritative Name	Bibliographic Name	Confidence	Reason for Match	Date Found	Marked By
	approved	a:Smith, Henry Nash. b:...	a:Smith, Henry Nash, b:...	85%	Authorized Match	May 15, 2007 11:42 AM	System
	approved	a:Smith, Henry Nash. b:...	a:Smith, Henry Nash, b:...	85%	Authorized Match	May 15, 2007 11:36 AM	System
	approved	a:Hill, Martyn. b: c: d: q:...	a:Hill, Martyn. b: c: d: q:...	85%	Authorized Match	May 15, 2007 11:40 AM	System
	approved	a:Smith, Henry Nash. b:...	a:Smith, Henry Nash. b:...	85%	Authorized Match	May 15, 2007 11:40 AM	System
	approved	a:Smith, Gregg b: c: d: q:...	a:Smith, Gregg. b: c: d: q:...	85%	Authorized Match	May 15, 2007 11:37 AM	System
	approved	a:Smith, Henry Nash. b:...	a:Smith, Henry Nash, b:...	85%	Authorized Match	May 15, 2007 11:42 AM	System
	approved	a:Smith, Harriet Elinor. b:...	a:Smith, Harriet Elinor. b:...	85%	Authorized Match	May 15, 2007 11:36 AM	System
	approved	a:Smith, Charles Martin...	a:Smith, Charles Martin. ...	85%	Authorized Match	May 15, 2007 11:39 AM	System
	approved	a:Hill, Thomas, b: c: clari...	a:Hill, Thomas, b: c: clari...	85%	Authorized Match	May 15, 2007 11:36 AM	System
	approved	a:Hill, Martyn. b: c: d: q:...	a:Hill, Martyn. b: c: d: q:...	85%	Authorized Match	May 15, 2007 11:39 AM	System
	approved	a:Smith, Charles Martin...	a:Smith, Charles Martin. ...	85%	Authorized Match	May 15, 2007 11:39 AM	System
	approved	a:Smith, Lawrence Leig...	a:Smith, Lawrence Leig...	85%	Authorized Match	May 15, 2007 11:38 AM	System
	approved	a:Thomas, Ralph L. b: c:...	a:Thomas, Ralph L. b: c:...	87%	Authorized Match	May 15, 2007 11:37 AM	System
	approved	a:Thomas, Augusta Rea...	a:Thomas, Augusta Rea...	87%	Authorized Match	May 15, 2007 11:37 AM	System
	approved	a:Johnson, Jeffrey b: c: d:...	a:Johnson, Jeffrey. b: c:...	87%	Authorized Match	May 15, 2007 11:42 AM	System
	approved	a:Johnson, Anna. b: c: d:...	a:Johnson, Anna. b: c: d:...	87%	Authorized Match	May 15, 2007 11:37 AM	System
	approved	a:Miller, Jonny Lee b: c:...	a:Miller, Jonny Lee. b: c:...	87%	Authorized Match	May 15, 2007 11:39 AM	System
	approved	a:Johnson, Claudia L. b:...	a:Johnson, Claudia L. b:...	87%	Authorized Match	May 15, 2007 11:39 AM	System
	approved	a:Miller, Mitch b: c: d: q:...	a:Miller, Mitch. b: c: d: q:...	87%	Authorized Match	May 15, 2007 11:39 AM	System
	approved	a:Thomas, Augusta Rea...	a:Thomas, Augusta Rea...	87%	Authorized Match	May 15, 2007 11:37 AM	System
	approved	a:Thomas, Milton. b: c: d:...	a:Thomas, Milton. b: c: d:...	87%	Authorized Match	May 15, 2007 11:37 AM	System
	annorved	a:Johnson, Claudia L. b:...	a:Johnson, Claudia L. b:...	87%	Authorized Match	May 15, 2007 11:37 AM	System

0 links selected for editing.

Flag Unflag

Mark selection as unverified Your name: unspecified user

This section shows a list of links between bibliographic and authoritative records in the system. The top section gives many options for filtering the results, while the list in the middle is the listing of links.

If show only flagged is click the only links that will be displayed are the ones that have previously been marked “flagged”. These flags are there for the user’s convenience, and have no actual effect on the links themselves.

The show option: unverified, accepted, and rejected filter on the current status of the links. Unverified links still need to be processed, accepted links have been recorded as being correct, while rejected links are links that the system generated but have since been marked as incorrect.

The confidence range allows you to set a minimum and maximum confidence value to show. This allows you to only look at a certain set of confidence level, allowing for a focused look at links that need review before being approved.

The date range filter allows you to filter on when the match occurred, letting you work only on links that took place on a certain day and time.

The hide/show filter button toggles the filter controls, giving more room to the links as needed.

The refresh button applies a filter, processing then returning the result into the links window.

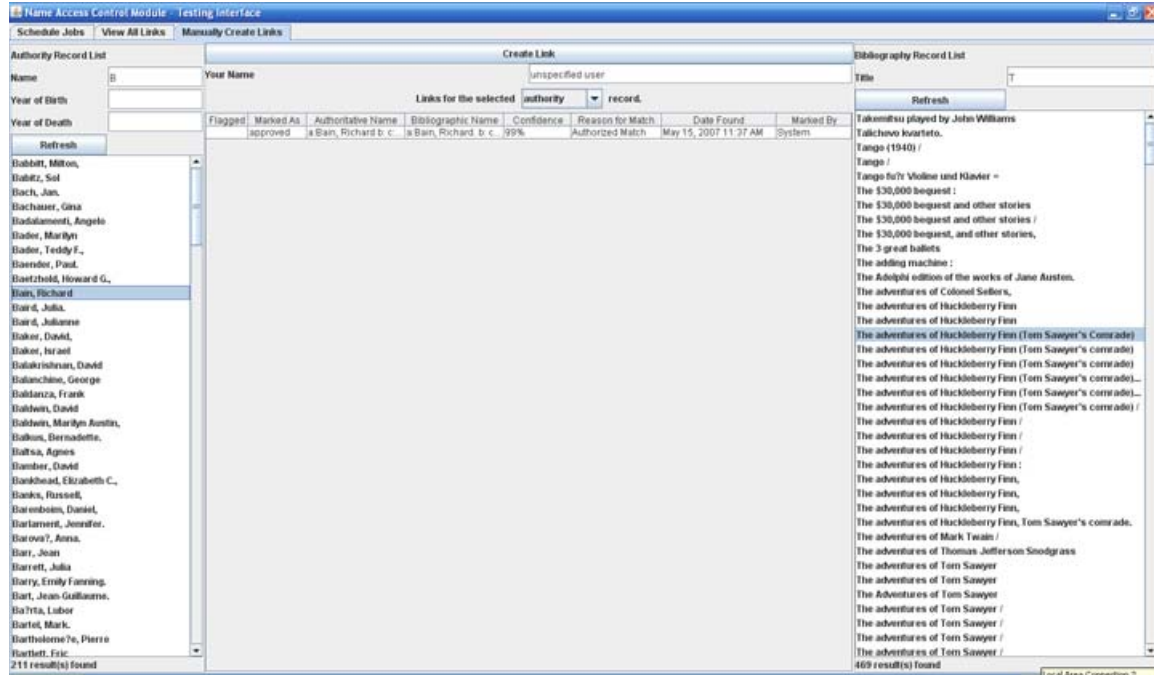
The show subfields in name columns toggles whether or not marc subfields are displayed, allowing for simple or detailed display of information about names.

The page controls allow for viewing results that can't fit in a single window. You can specify what page you want to jump to, or use the next and previous page buttons to jump one page at a time.

The center of the panel holds the list of links. You can sort the results of your filter by clicking on each columns heading. This will sort according to that column. Repeated clicking will change the sorting from ascending to descending order and back again.

The bottom of the panel holds controls for marking links. You must have links highlighted for any of these controls to work. The flag and unflag buttons are used to mark the selected links as flagged or to remove that marking. The mark selection as option allows you to set the status of links to either accepted, rejected, or unverified. If you put down your name next to this control, it will be recorded when you make your change.

Manually Created Links Panel



The manually created links panel allows both the viewing of records in the system and the forcing of links between those records. The left section displays authoritative data, the right bibliographic data. The center holds link information, in the same format as of the Show All Links panel.

On the authoritative panel filtering can be done by specifying any of the fields there and clicking refresh. The bibliographic filtering is done the same way but on the right. You can select records from either side to see links for those records.

In the center you can mark which of the two sides you want to view links for, authoritative or bibliographic. If you want to create a link between two sides just select both records and click Create Link.

You can mark links as accepted, rejected, and unverified in this panel as well. Just put in your name, select your record, and then select the labeling. Any records selected will have the label associated with them.