2014

CS 6604 SPRING 2014 TERM PROJECT

CINET Registry Project Report

MENTOR: DR. KEITH BISSET
TEAM: ADITYA AGASHE, HARSHAL HAYATNAGARKAR, SARANG JOSHI

NETWORK DYNAMICS AND SIMULATION SCIENCE LABORATORY, VIRGINIA BIOINFORMATICS INSTITUTE, VIRGINIA TECH | Blacksburg, VA, USA

Contents

Acknowledgements

We take this opportunity to express our profound gratitude and deep regards to our guide Dr. Edward Fox for his exemplary guidance, monitoring and constant encouragement throughout the course of this project. The blessing, help and guidance given by him time to time helped us in developing CINET Registry successfully.

We also take this opportunity to express a deep sense of gratitude to Dr. Keith Bisset from NDSSL for his cordial support, valuable information and guidance, which helped us in completing this task through various stages.

We are also grateful to Dr. Madhav Marathe for providing us with this wonderful opportunity to work with NDSSL. We thank him for providing us with excellent suggestions for improvements, encouragement and guidance through the course of this project.

We are obliged to staff members of NDSSL, for the valuable support provided by them for accessing the NDSSL infrastructure. We are grateful for their cooperation during the period of our project.

This work is based upon work sponsored by the National Science Foundation (NSF) under Grant No. OCI-1032677, SDCI NMI New: From Desktops to Clouds - A Middleware for Next Generation Network Science.

1. Introduction

Cyber-infrastructure for Network Science (CINET) is a computational and analytic framework for the network science researcher and education. The cyber-infrastructure (CI) part of CINET is responsible for coordinating the interactions between user interface, digital library, resource managers, data broker, and execution broker components. CINET uses HPC resources to service experiment execution requests. CINET provides many realistic graphs for analysis. Galib, NetworkX, and SNAP are the computational engines that provide the capability to analyze different properties of the graphs. CINET hosts the Granite system and graph dynamical systems calculator (GDSC) system as public use applications. Datasets used by CINET are currently cataloged in relational database and a new proposal is to migrate a new digital object based catalog 'Registry'.

2. Motivation

Current implementation captures domain independent metadata for the networks. If we want to capture domain specific metadata in current implementation it requires changing the ER model by adding new database tables. For example for an airline network domain specific metadata such as disease information, transportation mode and so on requires creation of a new database table per kind. However this approach does not scale well as the number of tables keeps increasing. An approach to solve this problem could be a digital object-based model and repository. For this project, the repository implementation has been already chosen as ‘Fedora Commons’ and based upon the choice, following solution architecture is proposed and implemented.

Another motivation to migrate from relational database to digital object based repository lies in the limitation of relational model to capture relationships. In relational model relationships between entities is captured using foreign keys and primary keys. In relational model it is difficult to model any semantics about relationships explicitly. Whereas digital object repository model closely follows object oriented paradigm, this helps us modelling inheritance and containment relationships in more intuitive manner.

Digital objects and digital object repository these two concepts are borrowed from the field of digital libraries which represent a body of knowledge for services around these concepts and ensures a holistic approach. Services can be create, update, delete, read of digital objects, preservation of digital objects and so on.

3. Problem Statement and Scope of project

This project aims to develop registry component based on digital object repository and Fedora Commons has been selected as implementation technology. First step in project is to model digital object schema from existing relational schema.

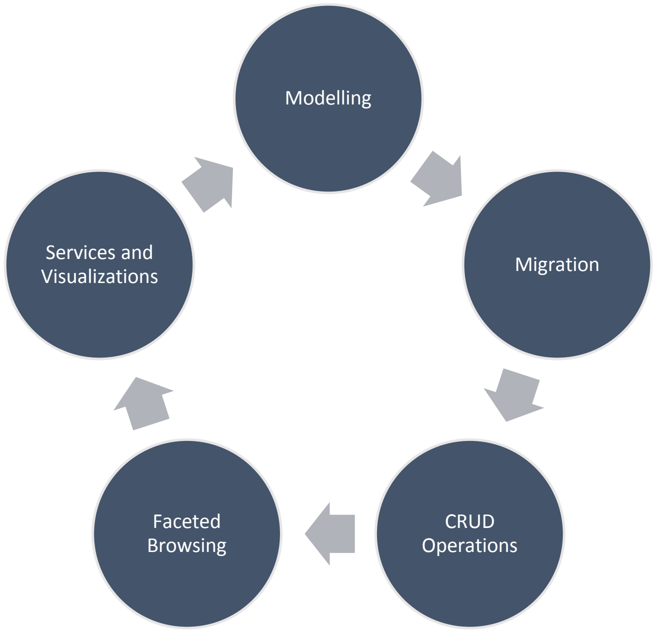


Figure 1 - Workflow

Following activities have been identified for this project and each of the activity is applicable to individual domain entity and aggregate entity.

Modeling: Current relational schema is used as a reference for modeling the digital objects. The relationships among tables are explicitly captured with semantics as Complex Digital Objects. For detailing domain-specific metadata, the model employs taxonomy and keywords as two possible mechanisms. Instance-specific metadata and pluggable-metadata for digital objects are explored. Finally, the realized model is evaluated vis-à-vis the relational model.

Migration: Migration activity requires digital object classes defined as part of modeling process. The activity then exports the data from the database into CSV format for all tables. A collection of migration routines is written which read each entry in these flat files, create appropriate digital objects for that entry, copy values into created instances and finally set relationships with other digital objects depending upon the integrity constraints of the model.

CRUD Operations: Once the relational catalog data is migrated into digital objects, the system provides RESTful web-services and HTML-based user interface (UI) for performing create, update, read, delete, search and browse operations on these objects. ***The UI takes care of domain-specific metadata, object associations and other aspects of model.***

Faceted Browsing: Faceted browsing is provided as a way of browsing through the collection of digital objects of different types and also integrates textual search features. If supported by selected technology, hierarchical facets shall be provided.

Services and Visualizations: A set of application-specific richer services is provided on top of repository and CRUD services. Incentivization and Memoization services are implemented .In addition Utilization service would also be implemented.

4. Overall Architecture

Being proof-of-concept, project our aim was to develop it using Rapid Development tools or environments or tools. For this reason we choose the Ruby on Rails as programming language for this project. Ruby on Rails follows model view controller (MVC) pattern. Ruby on Rails facilitates rapid development with rich features such as Scaffolding. Rails scaffolding quickly to generates the simple model, view and controllers. Scaffolding greatly reduces the developer's efforts.

Model-View-Controller: It is a software architecture pattern for maintaining multiple views of the same data. It divides a given software application into three interconnected components - Models for maintaining data, Views for displaying all or a portion of the data and controller for handling events that affect the model or the view. A view can be any output or representation of information. The third part, controller accepts input and converts it to commands for the model or view. Because of the separation, multiple views and controllers can interface with the same model.

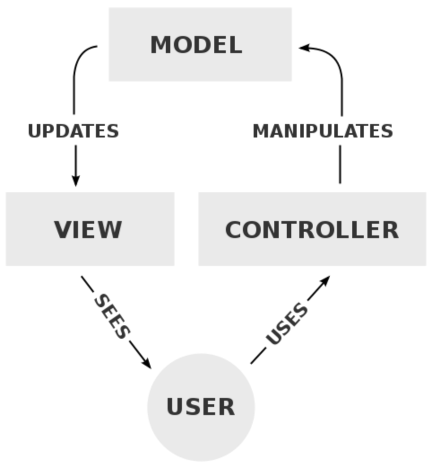


Figure 2 - Model-View-Controller interactions

Ruby on Rails: Is open source web application framework which runs via Ruby programming language. Ruby on rails makes use of well-known software engineering patterns and principles such as active record pattern, convention over configuration (CoC), don't repeat yourself (DRY) and model-view-controller (MVC). In a default configuration, *model* in the Ruby on Rails framework maps to database table, and a Ruby file. A *controller* is a component of Rails that responds to external requests from the web server to the application by determining which view file to render. A view in the default configuration of Rails is an '.erb' file, which is compiled to HTML at run-time. Ruby on Rails includes tools that make common development tasks easier "out of the box", such as *scaffolding* that can automatically construct some of the models and views needed for a basic website.

Ruby on Rails is intended to emphasize 'Convention over Configuration' (CoC), and the 'Do not Repeat Yourself' (DRY) principles. 'Convention over Configuration' means a developer only needs to specify unconventional aspects of the application. For example, if there is a class Sale in the model, the corresponding table in the database is called sales by default. It is only if one deviates from this convention, such as calling the table "products sold" that the developer needs to write code regarding these names. Generally, Ruby on Rails conventions lead to less code and less repetition. “Don't Repeat Yourself" means that information is located in a single, unambiguous place. For example, using the ActiveRecord module of Rails, the developer does not need to specify database column names in class definitions. Instead, Ruby on Rails can retrieve this information from the database based on the class name.

Project Hydra [1] is an extension to Ruby on Rails framework that integrates Fedora Commons into Rails. The technology stack of Hydra is discussed below.

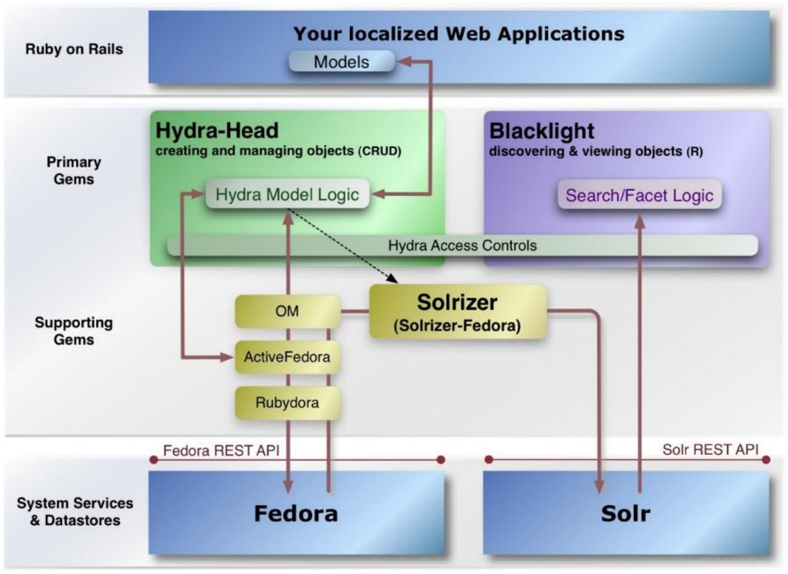


Figure 3 - Project Hydra Stack

- ***Fedora Commons*** provides a robust, durable repository layer for persisting and managing digital objects. It describes metadata objects in the form of XML and inter-object references as RDF links and a core repository service which exposed as web-based services with well-defined APIs. In addition, an array of supporting services are available including search, messaging, administrative clients, and so on. More information about Fedora-commons is available at [2] and [3]. Fedora's disseminator features allow us to place an abstraction layer between it and our Hydra heads, shielding an institution's applications from any future changes to the repository structure.
- ***Solr*** indexes and gives fast access to information about the institution's resources. Solr can be used as a lingua franca: content from any source that can generate a Solr index (perhaps an OPAC, or repository metadata records with different schema) can potentially be brought into a Hydra discovery environment.

- ***HydraHead*** is a Ruby gem that works with ***ActiveFedora*** to provide create, update and delete actions against objects in the repository, as well as to support various content management actions (e.g., upload file, edit metadata, change permissions).
- **ActiveFedora** implements ActiveRecord pattern, similar that of Rails but for Fedora.
- ***Solarizer*** takes care of updating the Solr index, whenever Fedora objects are created, updated or deleted.
- ***Blacklight*** is a Ruby gem for faceted searching, browsing and tailored views of objects on top of Solr and Rails.

Scaffolding: In this project we used the Rails scaffolding to generate code for models, controllers and views we need in order to perform CRUD operations in digital objects.

Sencha ExtJs Desktop: CINET Registry uses the **S**encha ExtJs [4] example web-based desktop application which has look and feel similar to an operating system desktop environment. The desktop is a metaphor to organize various user activities and services such as Incentivization as native desktop applications. The web application looks and feels like a native desktop application.

5. Design

5.1. Model

'Fedora Commons' model can be interfaced using 'ActiveFedora' component of Hydra stack. 'ActiveFedora' implements 'ActiveRecord [XXX]' pattern for accessing and manipulating persisted domain objects. In order to achieve this functionality, digital objects are represented as model classes extending 'ActiveFedora::Base' class. For metadata stream, 'ActiveFedora::OpinionatedMetadata' should be specialized and referred in the model class. Together these two collections of classes help in organizing associations among complex digital objects. Typically 'is-part-of' relationship is captured in these model classes. As of today, 'has many' relationship can be captured but 'has one' relationship is not supported. Thus, one may use former one to realize the latter. As per Rails convention, one model class is designed per entity and placed in directory 'app/models'. Typically, metadata stream classes are placed in directory 'app/models/datastreams' directory.

For current implementation, we have taken Granite relational database schema as the baseline to model digital objects. The Granite schema is shown in Figure 4 – Granite database schema. The core entities of this schema are: base_network, base_measure, analysis and analysis_content.

- **base_network:** It represents master collection of networks and captures details about type, name, description, data file and identity of contributor as metadata. It is referred by analysis_content.
- **base_measure:** A measure is a function or algorithm which runs on a network to evaluate property of graph. The entity represents master collection of algorithms and captures details about type, name, description, platform and identity of contributor as metadata. It is referred by analysis_content. E.g. centrality of the graph is measure.

- **analysis:** It represents analysis submitted by the user. It is referred by ‘analysis_content’.
- **analysis_content:** An analysis is split into multiple analysis jobs and is modeled as ‘analysis_content’ and has one network and one measure. The entity refers to ‘base_network’ and ‘base_measure’ instances through foreign key relationships.

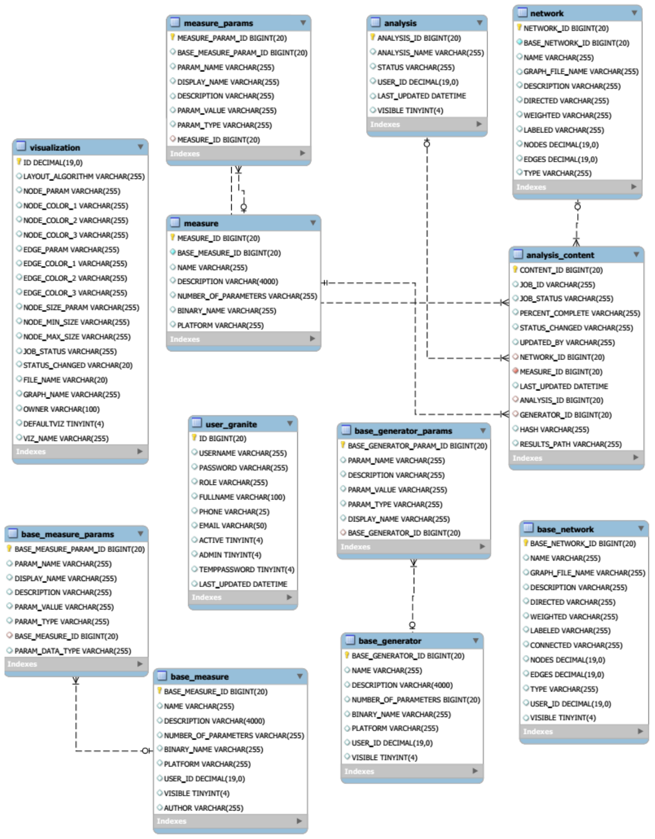


Figure 4 – Granite database schema

In addition, the schema has ‘base_measure_param’, ‘measure’, ‘param’ and ‘user’ entities. For details, please consult Granite schema document.

Current list of digital object classes are: Network, Measure, Param, MeasureInstance, ParamInstance, Analysis, AnalysisJob, User and Resource.

- **Network:** This complex digital object is equivalent of ‘base_network’ Granite schema entity, which represents the network datasets in CINET. It refers to digital objects of ‘Network’ and ‘User’ objects. The class is extended by domain-specific specializations such as ‘TransportationNetwork’, ‘BiologicalNetwork’ and so on, by referring to ‘type’ attribute that is received from database schema. Network class contains generic and domain-independent metadata for networks whereas specializations help in capturing domain-specific metadata for each type, which is non-trivial in case of relational model. In addition, attribute ‘keywords’ is a way to an arbitrary list of domain-specific keywords, useful in searching and browsing of networks. In current implementation, however, the network taxonomy is not fully functional and results of search operation on base class does not include objects of derived class. This could be limitation as one has to explicitly know all derived classes in order to for example browse and search combined set of objects.

- **Measure:** Equivalent of ‘base_measure’ entity. Refers to parameters abstracted by ‘Param’ class instances. Referred by ‘MeasureInstance’ class and its cross relationship to indicate instantiation. Refers to ‘User’ class to indicate contribution.
- **Analysis:** Equivalent of ‘analysis’ entity. Refers to collection of ‘AnalysisJob’ instances,
- **AnalysisJob:** Equivalent of ‘analysis_content’. The class is enhanced with more attributes, ‘resource’, ‘execution time’ and ‘count’, representing the instance of ‘Resource’ class on which analysis job was executed, amount of time taken for execution of the job and number of times the very analysis job was requested. These enhancements are used in services.
- **Resource:** This is new class to represent various computing resources. Currently, it describes type of resources, CPU cores, memory and storage. The analysis jobs executed on a particular resource could be referred from an instance of Resource. Possible types of resources are – CLUSTER, GRID and CLOUD. In future, a taxonomy could be defined so that resource-specific information such as virtualization could be placed.
- **User:** Equivalent of ‘user’ entity. However, it also converges with ‘contributor’ concept and a taxonomy could be a future enhancement.

Please refer to Figure 5 - Model UML class diagram which renders classes and their taxonomy.

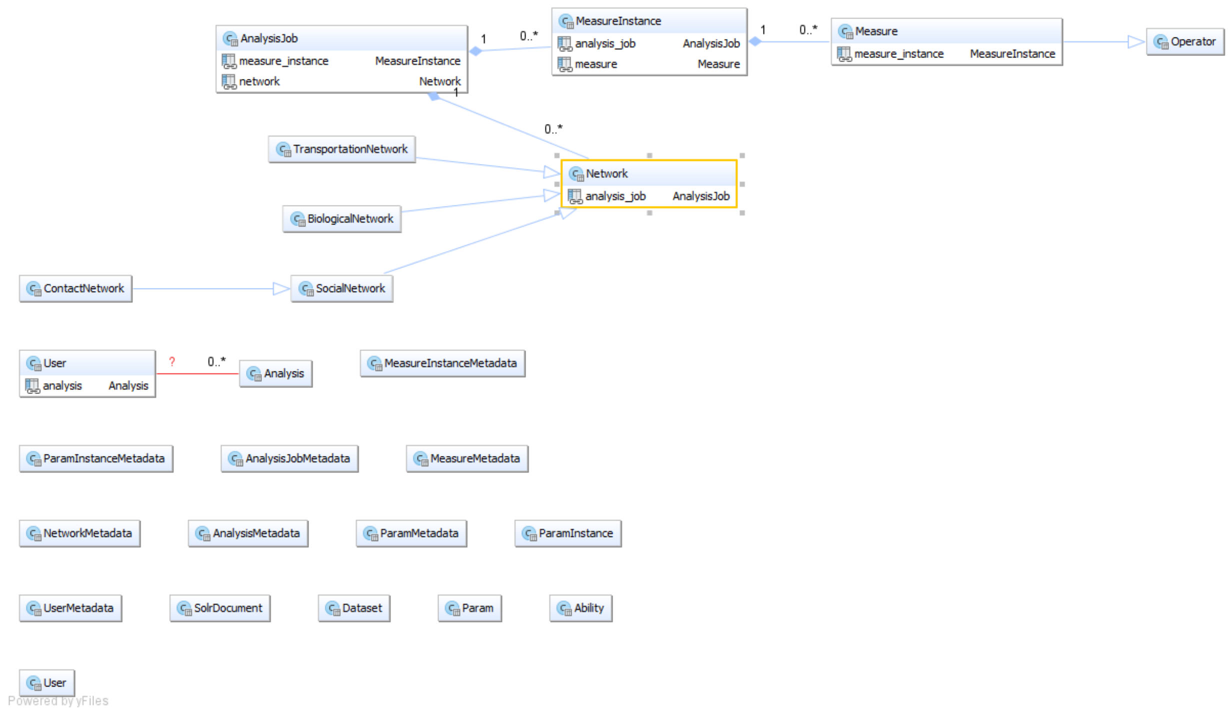


Figure 5 - Model UML class diagram

5.2. Services

REST is an acronym standing for Representational State Transfer and to describe an architecture style of networked systems. In the last few years REST has emerged in the last few years alone as a predominant Web service design model. We provide RESTful web services on top of the technology stack. Below is a short description for each of the services -

Incentivization: Incentivization service highlights user contributions. It thus gives a list of top-k users in terms of contributions. For example, it provides data about how many networks and measures are contributed or analyses performed by a user. These contributions are visualized using charts and table grids.

Memoization: Running an analysis is expensive in terms of time and resources. Memoization service helps to avoid repetition of analyses by returning existing data when network, measure and parameters match. It uniquely identifies an analysis and thus helps in saving resources.

Utilization: Utilization service provides an overall summary of how network datasets, measures and computation resources have been utilized. Thus it provides the list of top-k Networks, Measures.

6. Using CINET Registry

Figure 6 Project Blacklight Faceted Browser Launch Screen shows launchscreen for CINET registry faceted browser.

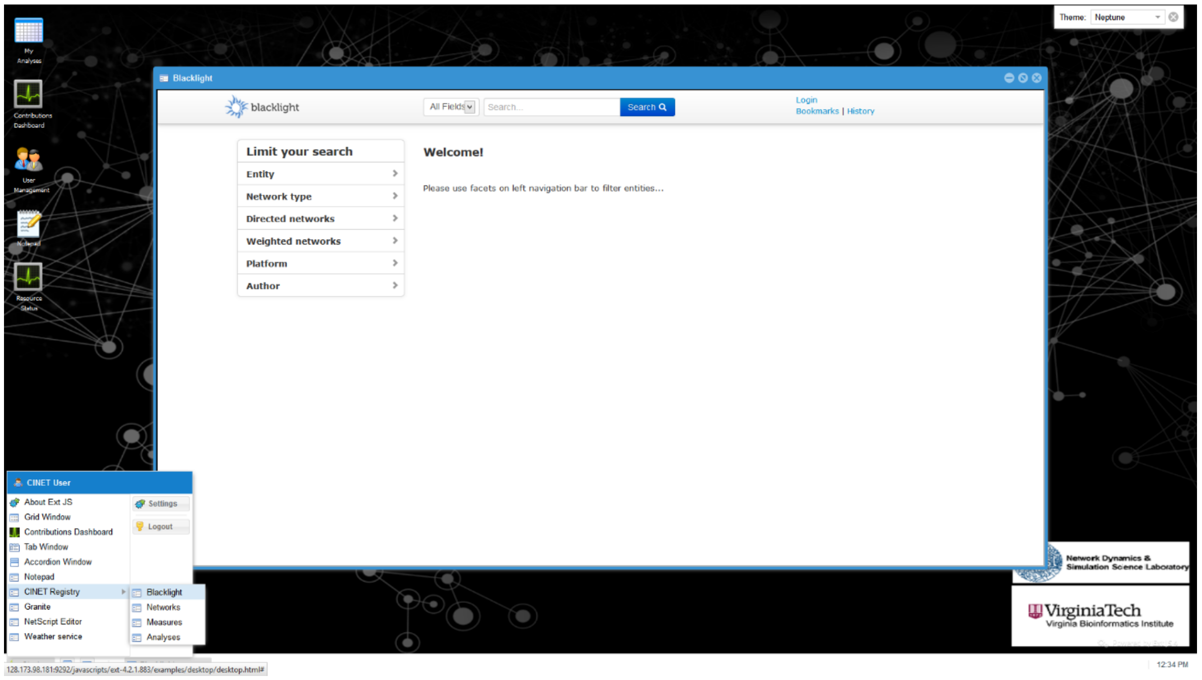
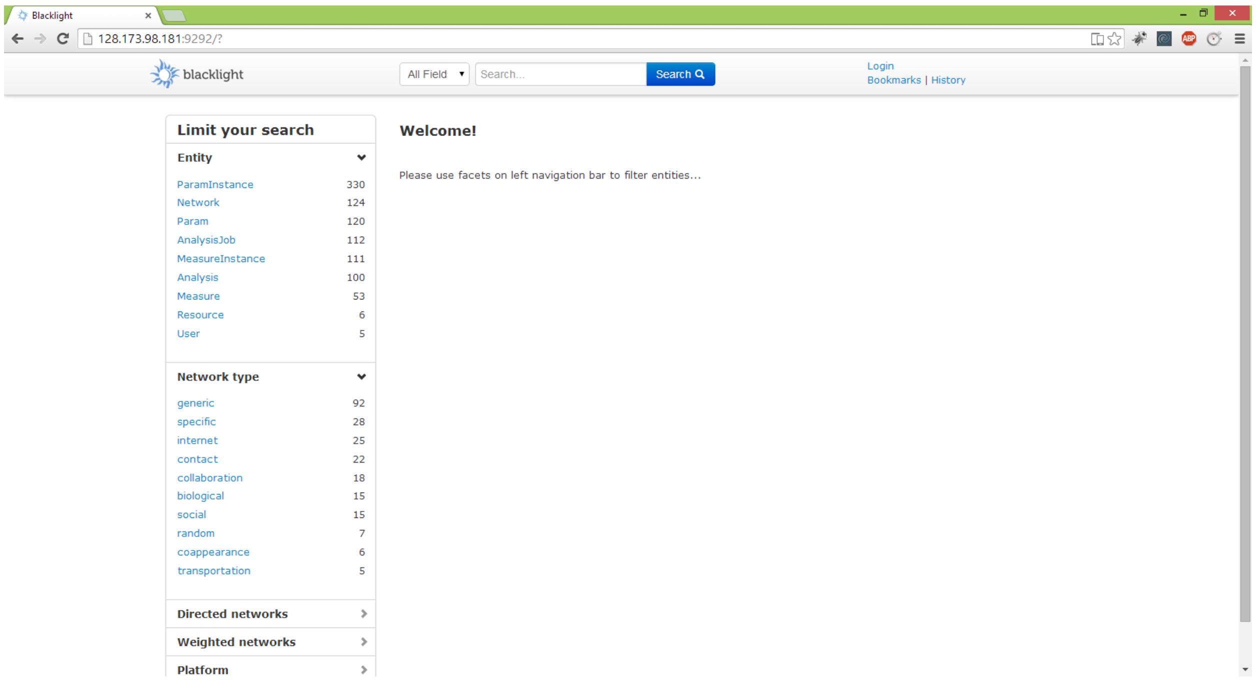


Figure 6 Project Blacklight Faceted Browser Launch Screen

Figure 7 Project Blacklight - Expanded facets shows faceted browsing of CINET registry, Fig [XXXX] shows all facets available for filtering in CINET registry.



Limit your search

Entity

ParamInstance	330
Network	124
Param	120
AnalysisJob	112
MeasureInstance	111
Analysis	100
Measure	53
Resource	6
User	5

Network type

generic	92
specific	28
internet	25
contact	22
collaboration	18
biological	15
social	15
random	7
coappearance	6
transportation	5

Directed networks

Weighted networks

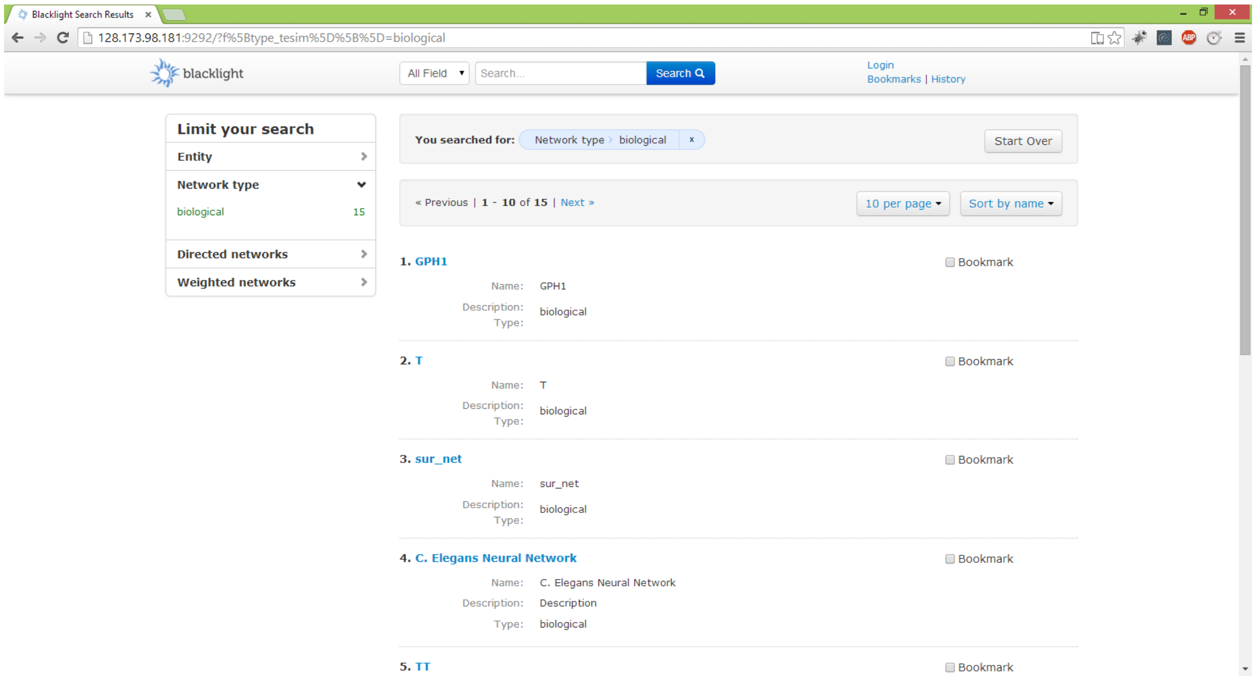
Platform

Welcome!

Please use facets on left navigation bar to filter entities...

Figure 7 Project Blacklight - Expanded facets

Figure 8 Filtered Networks based upon facet selection shows biological networks in CINET registry. Biological networks can be filtered using facets. To show biological networks select "*Network*" from Entity and then select "*biological*" network from Network Type.



1. GPH1 — Bookmark
Name: GPH1
Description:
Type: biological

2. T — Bookmark
Name: T
Description:
Type: biological

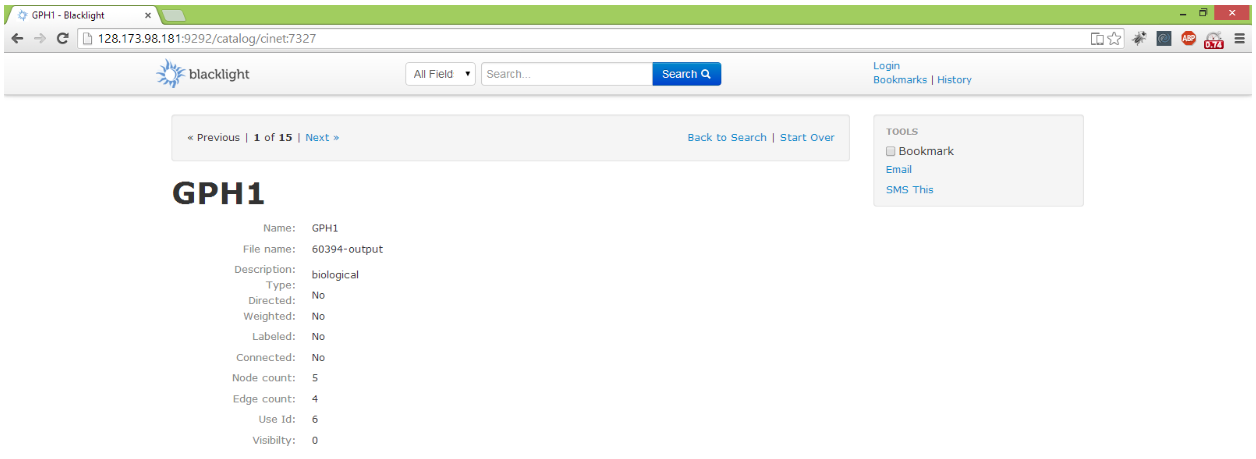
3. sur_net — Bookmark
Name: sur_net
Description:
Type: biological

4. C. Elegans Neural Network — Bookmark
Name: C. Elegans Neural Network
Description: Description
Type: biological

5. TT — Bookmark

Figure 8 Filtered Networks based upon facet selection

Figure 9 Network Digital object with metadata shows network digital object, it list complete metadata of selected Network entity.

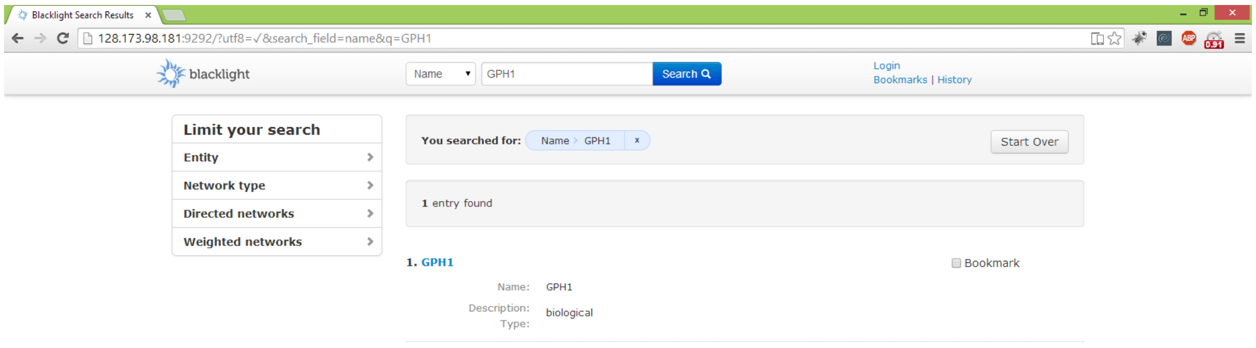


GPH1

Name:	GPH1
File name:	60394-output
Description:	
Type:	biological
Directed:	No
Weighted:	No
Labeled:	No
Connected:	No
Node count:	5
Edge count:	4
Use Id:	6
Visibilty:	0

Figure 9 Network Digital object with metadata

Figure 10 Keyword Search shows search result based on keywords entered by user in search box



You searched for: Name › GPH1 ×

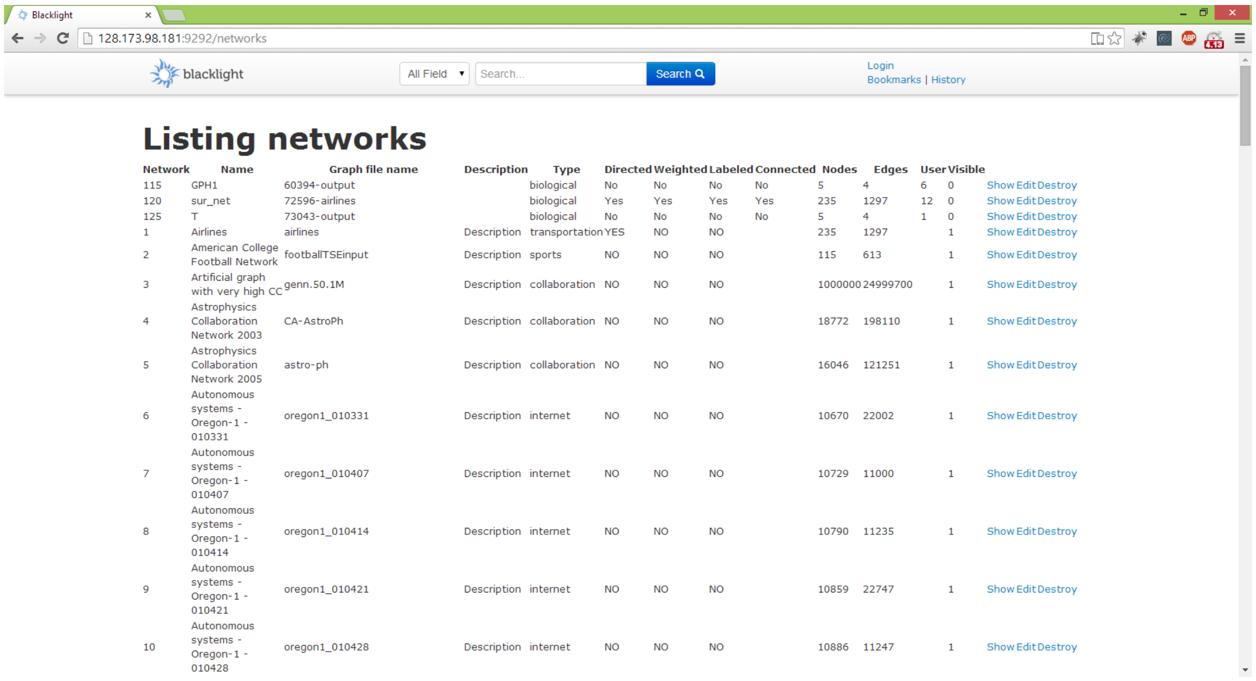
1 entry found

1. GPH1

Name: GPH1
Description: biological
Type:

Figure 10 Keyword Search

Figure 11 List of Network entities with CRUD operations shows the list of all network entities available in CINET registry. Each network entity is shown as row in this list. At the end of each row “Show”, “Edit”, “Destroy” links are available which will perform Read, Update and Delete operations on digital objects. Similarly other entities can also listed like in Figure 11. To navigate here go to route ‘/networks’ in browser’s address bar.

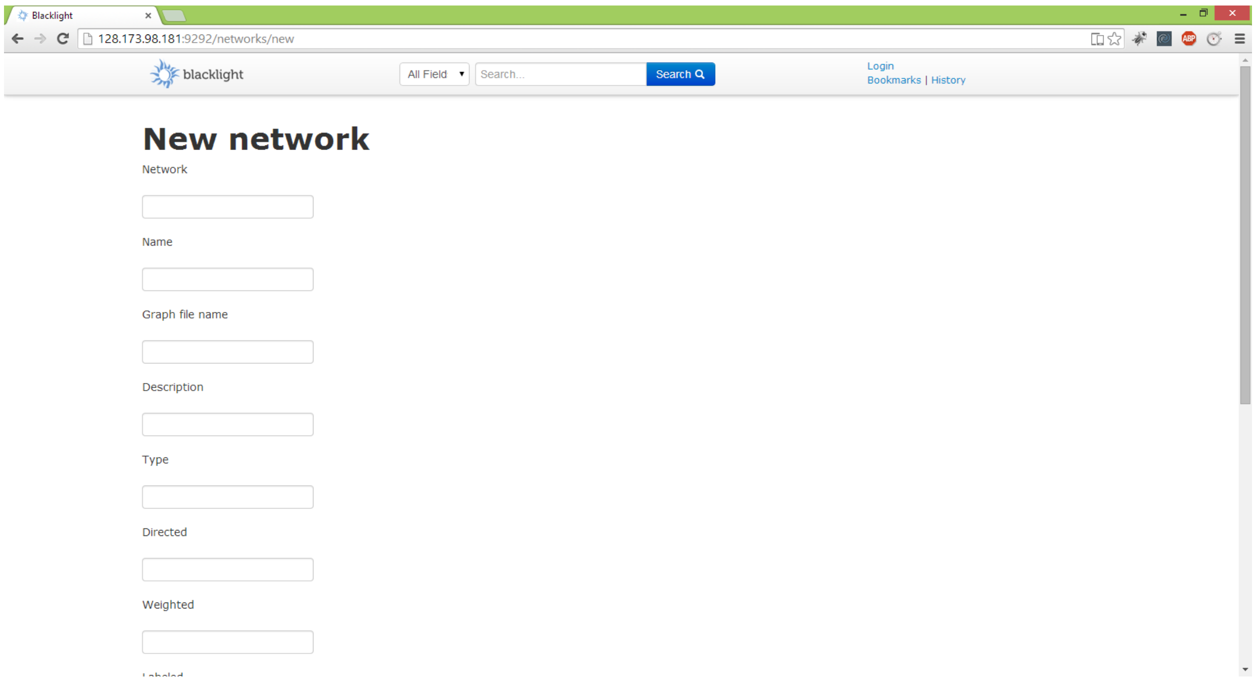


Listing networks

Network	Name	Graph file name	Description	Type	Directed	Weighted	Labeled	Connected	Nodes	Edges	User	Visible	
115	GPH1	60394-output		biological	No	No	No	No	5	4	6	0	Show Edit Destroy
120	sur_net	72596-airlines		biological	Yes	Yes	Yes	Yes	235	1297	12	0	Show Edit Destroy
125	T	73043-output		biological	No	No	No	No	5	4	1	0	Show Edit Destroy
1	Airlines	airlines	Description	transportation	YES	NO	NO		235	1297		1	Show Edit Destroy
2	American College Football Network	footballTSEinput	Description	sports	NO	NO	NO		115	613		1	Show Edit Destroy
3	Artificial graph with very high CC	genn.50.1M	Description	collaboration	NO	NO	NO		1000000	24999700		1	Show Edit Destroy
4	Astrophysics Collaboration Network 2003	CA-AstroPh	Description	collaboration	NO	NO	NO		18772	198110		1	Show Edit Destroy
5	Astrophysics Collaboration Network 2005	astro-ph	Description	collaboration	NO	NO	NO		16046	121251		1	Show Edit Destroy
6	Autonomous systems - Oregon-1 - 010331	oregon1_010331	Description	internet	NO	NO	NO		10670	22002		1	Show Edit Destroy
7	Autonomous systems - Oregon-1 - 010407	oregon1_010407	Description	internet	NO	NO	NO		10729	11000		1	Show Edit Destroy
8	Autonomous systems - Oregon-1 - 010414	oregon1_010414	Description	internet	NO	NO	NO		10790	11235		1	Show Edit Destroy
9	Autonomous systems - Oregon-1 - 010421	oregon1_010421	Description	internet	NO	NO	NO		10859	22747		1	Show Edit Destroy
10	Autonomous systems - Oregon-1 - 010428	oregon1_010428	Description	internet	NO	NO	NO		10886	11247		1	Show Edit Destroy

Figure 11 List of Network entities with CRUD operations

Figure 12 Add New Network shows form to create new network digital object. To navigate to this screen use route '/networks/new'.



New network

Network

Name

Graph file name

Description

Type

Directed

Weighted

Figure 12 Add New Network

7. Future Work

1. **Performance Improvement for Database:** Current implementation of registry uses Derby as backend database for Fedora Commons. Derby insert operation are slow. Due to slower insert operations of Derby database, creating large number of digital objects takes time. Hence to make this system scalable database need to be changed to more scalable options.
2. **Instance-specific metadata:** Taxonomy is the hierarchy of abstractions. It helps to capture domain specific aspects of a particular entity. When taxonomy tree is shallow but broad instance-specific metadata makes more sense.
3. **Disseminators for graph format conversion:** CINET supports multiple graph libraries. Each graph library have its own format for storing graph data. To convert from one format to another format methods need to be written. Disseminators can be used for this purpose. Disseminators are methods on top of digital objects.
4. **Authentication and Authorization:** Current implementation of registry doesn't consider authentication and authorization.
5. **Integration with CINET:** This registry is developed is developed in isolation. Registry can be integrated with CINET.

References

[1] "Project Hydra," [Online]. Available: http://projecthydra.org/.

[2] "Fedora Commons Home Page," [Online]. Available: https://wiki.duraspace.org/display/FF/Fedora+Repository+Home.

[3] "Fedora Home Page," [Online]. Available: http://www.fedora-commons.org/.

[4] "Sencha ExtJS," [Online]. Available: http://www.sencha.com.