

# **Open Ontology Repository: Architecture and Interfaces**

Ken Baclawski  
Northeastern University

# Outline

- ◆ Requirements
- ◆ Architecture
- ◆ Interfaces
- ◆ Data Model
- ◆ Future Work

# Requirements

- ◆ Goals
- ◆ Nonfunctional requirements
- ◆ Use case descriptions
- ◆ Wiki page: [http://ontolog.cim3.net/cgi-bin/wiki.pl?OpenOntologyRepository\\_Requirement](http://ontolog.cim3.net/cgi-bin/wiki.pl?OpenOntologyRepository_Requirement)

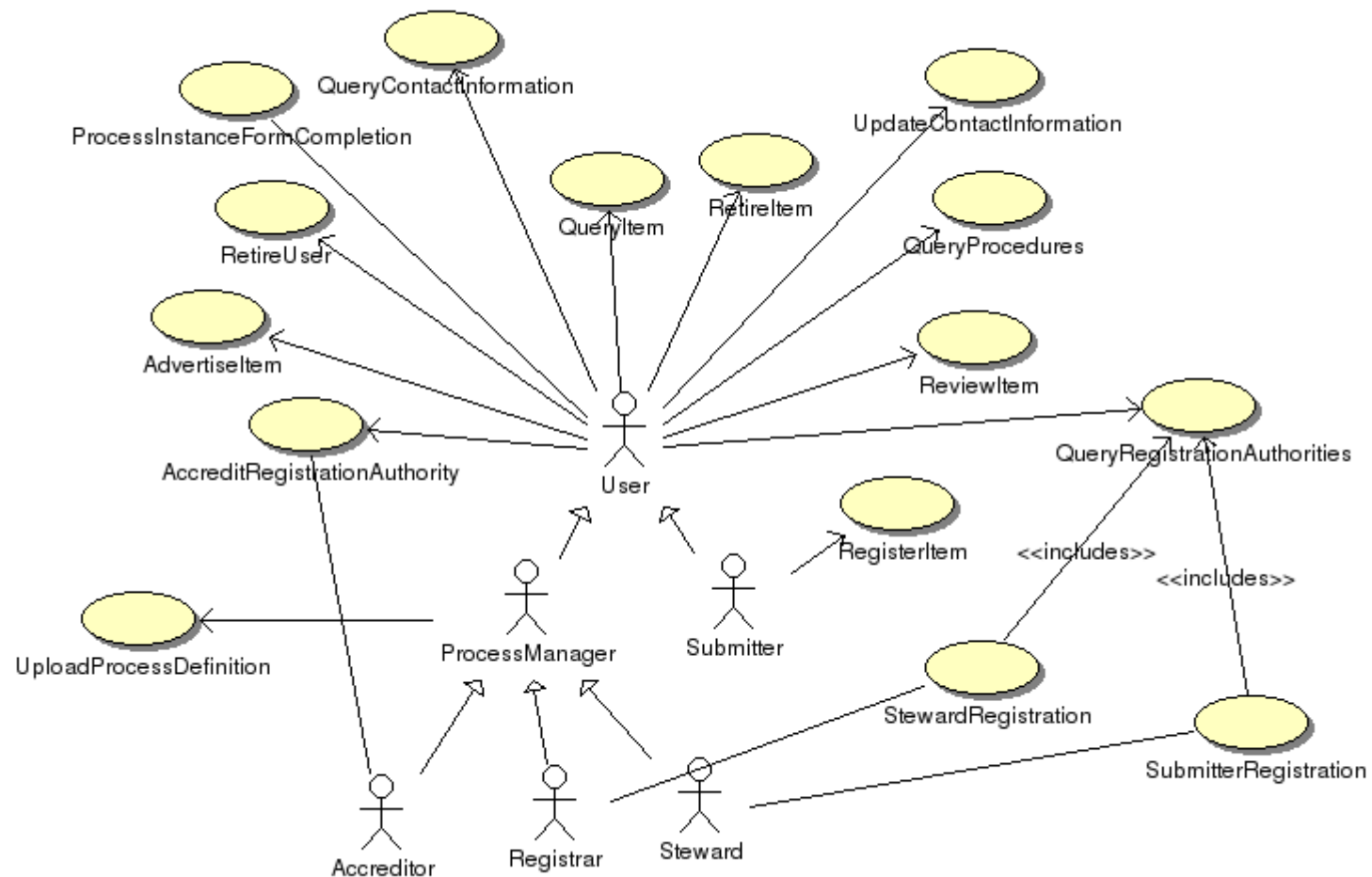
# OOB Goals

- ◆ A well-maintained persistent store (with high availability and performance) where ontological work can be stored, shared and accessed consistently;
- ◆ Mechanisms for registering and “governing” ontologies, with provenance and versioning, made available (logically) in one place so that they can be browsed, discovered, queried, analyzed, validated and reused;
- ◆ Services across disparate ontological artifacts supporting cross-domain interoperability, mapping, application and inferencing; and
- ◆ Registration of semantic services to support peer OOBs

# Nonfunctional Requirements

- ◆ The repository architecture shall be scalable.
- ◆ The repository shall be distributed.
- ◆ The specification of the repository shall be sufficiently detailed and platform independent to allow multiple implementations.
- ◆ The repository shall be capable of supporting ontologies in languages that have reasoners [supporting inferencing].
- ◆ The repository architecture shall support distributed repositories.
- ◆ The repository architecture shall not require a hierarchical structure.

# Use Cases



# Architecture

- ◆ Goals
- ◆ Modularity Targets
- ◆ Proposed Architecture

# Architecture Goals

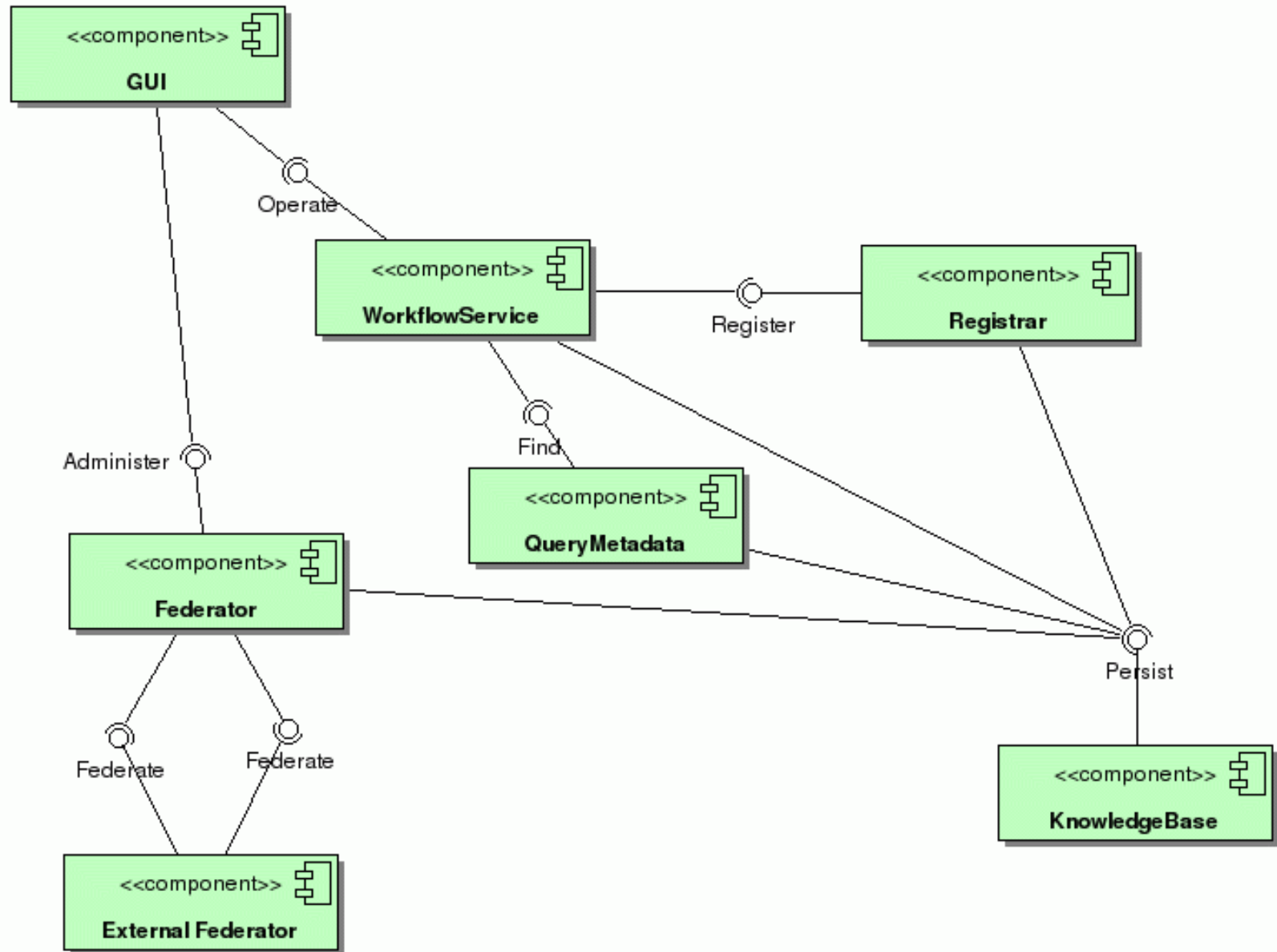
- ◆ OOR requires an open and well documented architecture to
  - Allow multiple communities and organizations to participate in the OOR
  - Produce standard OOR functionalities and behaviors.
- ◆ OOR Architectural Principles
  - **Decoupling of responsibilities** – To support multiple knowledge representations/languages
  - **Implementation/Platform independence** – To support acceptance, multiple instances, and evolution
  - **Ontologically driven** – To allow for evolution of the OOR and reduce overall development costs



# Modularity Targets

- ◆ Registry functions
- ◆ Repository functions
- ◆ KR languages
- ◆ Gatekeeping policies
- ◆ Intellectual Property Rights policies
- ◆ Federation mechanisms
- ◆ Value-added services

# Proposed Architecture



# Interfaces

## ◆ WADL (REST)

- Uses URL formatting of parameters
- Parameters are strings of various kinds: path, query, form, matrix, header, cookie

## ◆ WSDL (SOAP)

- Uses XML format for parameters and return values
- Maps operations to methods
- Maps XML parameters to objects

# WADL/REST

- ◆ BioPortal core was refactored to use JAX-RS
- ◆ URL mapping specified by annotations
- ◆ WADL generated from the JAX-RS resource classes
- ◆ Resource methods call the WSDL/SOAP methods.
- ◆ Refactored OOR core runs in Tomcat.

# WSDL/SOAP

- ◆ Derived from the BioPortal Service classes
- ◆ WSDL generated using JWS
- ◆ There are 126 methods:
  - Ontology Registration (6)
  - Find Ontologies (25)
  - Search and Navigation within one ontology (18)
  - Differences between ontologies (5)
  - Evaluations and Metrics (16)
  - Notification and Subscriptions (8)
  - Generation of RDF (5)
  - Ontology Development (22)
  - Administration (21)

# WSDL/SOAP

- ◆ WSDL and SOAP SEI available at [OOD Interface](#)
- ◆ Examples:

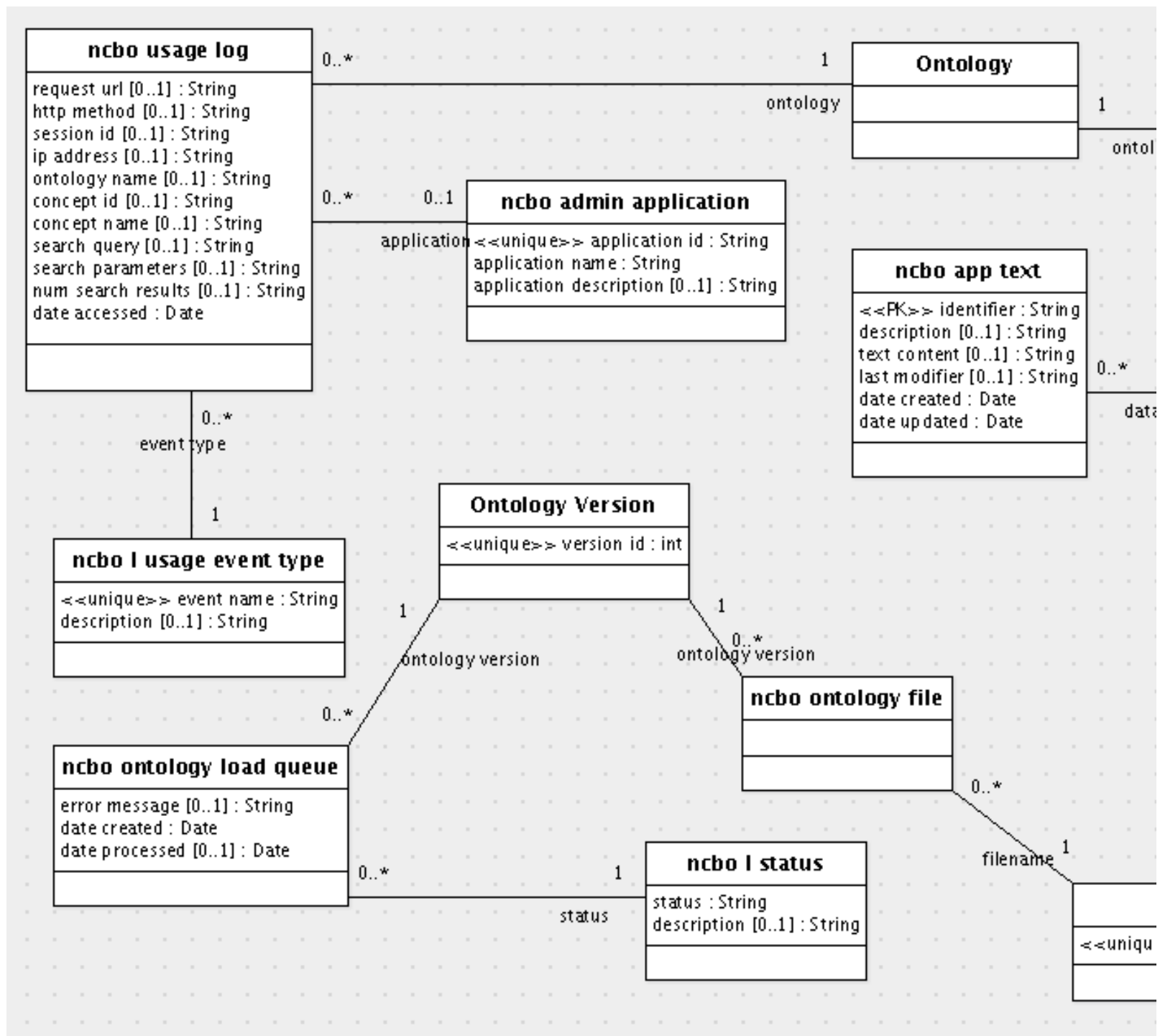
```
public List<OntologyBean>  
    findLatestActiveOntologyViewVersions() throws  
        Exception;
```

```
public Page<SearchBean> executeQuery1(String expr,  
    boolean includeProperties, boolean isExactMatch,  
    Integer pageSize, Integer pageNum, Integer  
    maxNumHits) throws Exception;
```

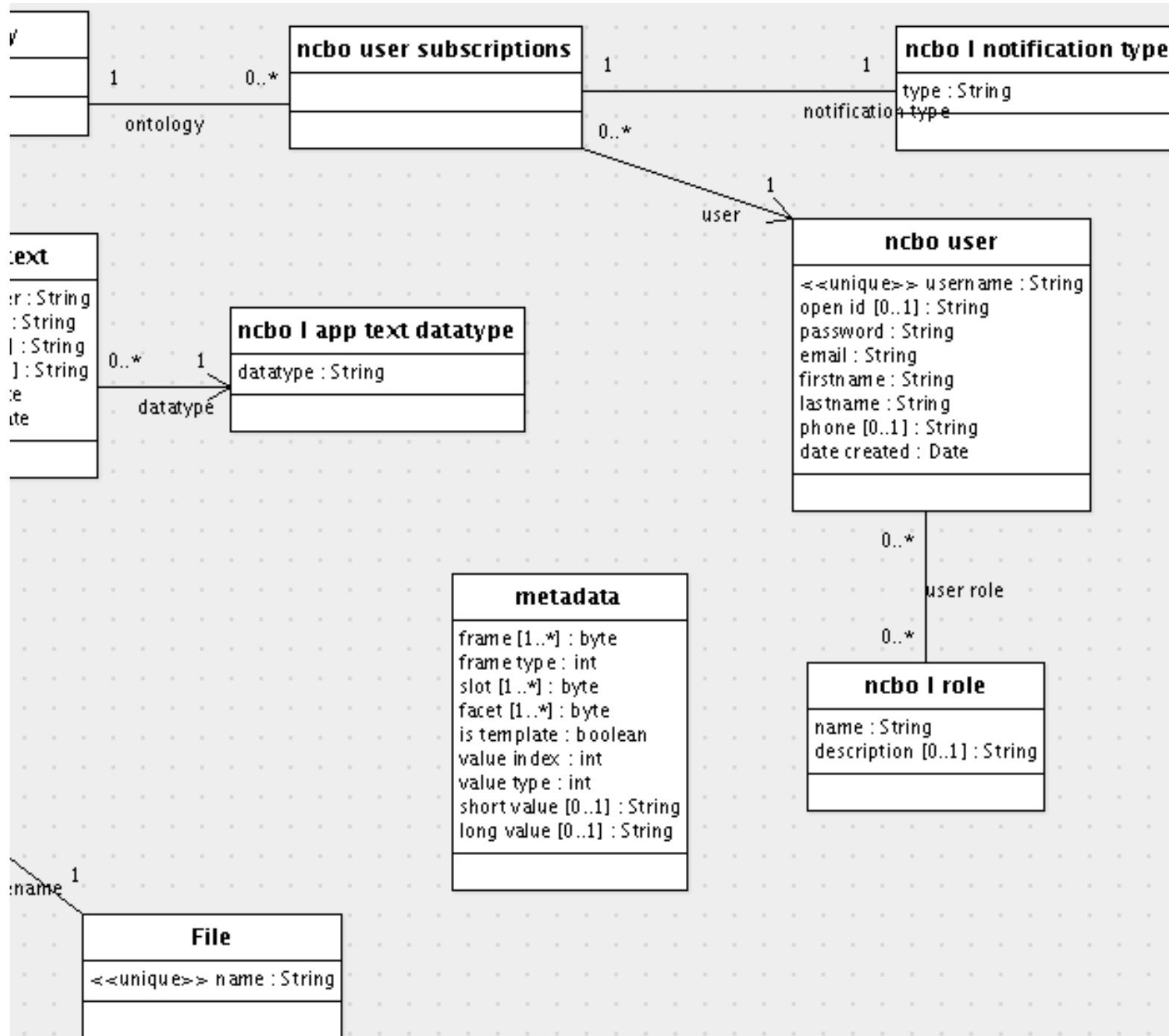
```
public Page<SearchBean> executeQuery2(String expr,  
    boolean includeProperties, boolean isExactMatch,  
    Integer maxNumHits) throws Exception;
```

# Data Model

- ◆ Data stored in MySQL
- ◆ UML class diagram shown on next two slides.







# Suggestions for Future Work

- ◆ Refactor database component
- ◆ Split core into two components
- ◆ Integrate the gatekeeper
- ◆ Develop and integrate the federator