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

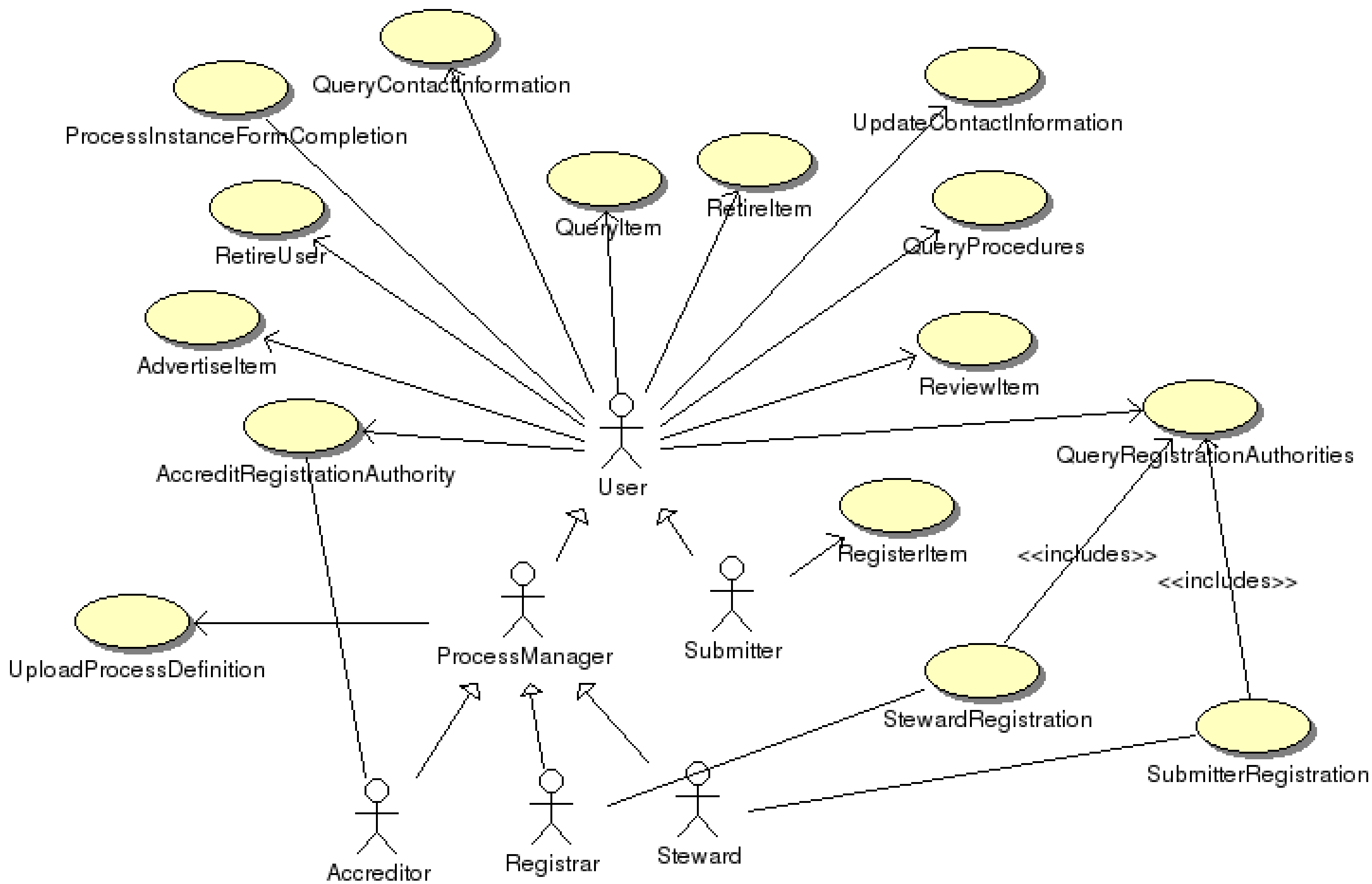
OOR Goals

- 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 OORs

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



QueryContactInformation
ProcessInstanceFormCompletion
UpdateContactInformation
QueryItem
RetireItem
RetireUser
QueryProcedures
AdvertiseItem
ReviewItem
AccreditRegistrationAuthority
User
QueryRegistrationAuthorities
RegisterItem
<<includes>>
<<includes>>
UploadProcessDefinition
ProcessManager
Submitter
StewardRegistration
SubmitterRegistration
Accreditor
Registrar
Steward

Architecture

- Goals
- Modularity Targets
- Proposed Architecture

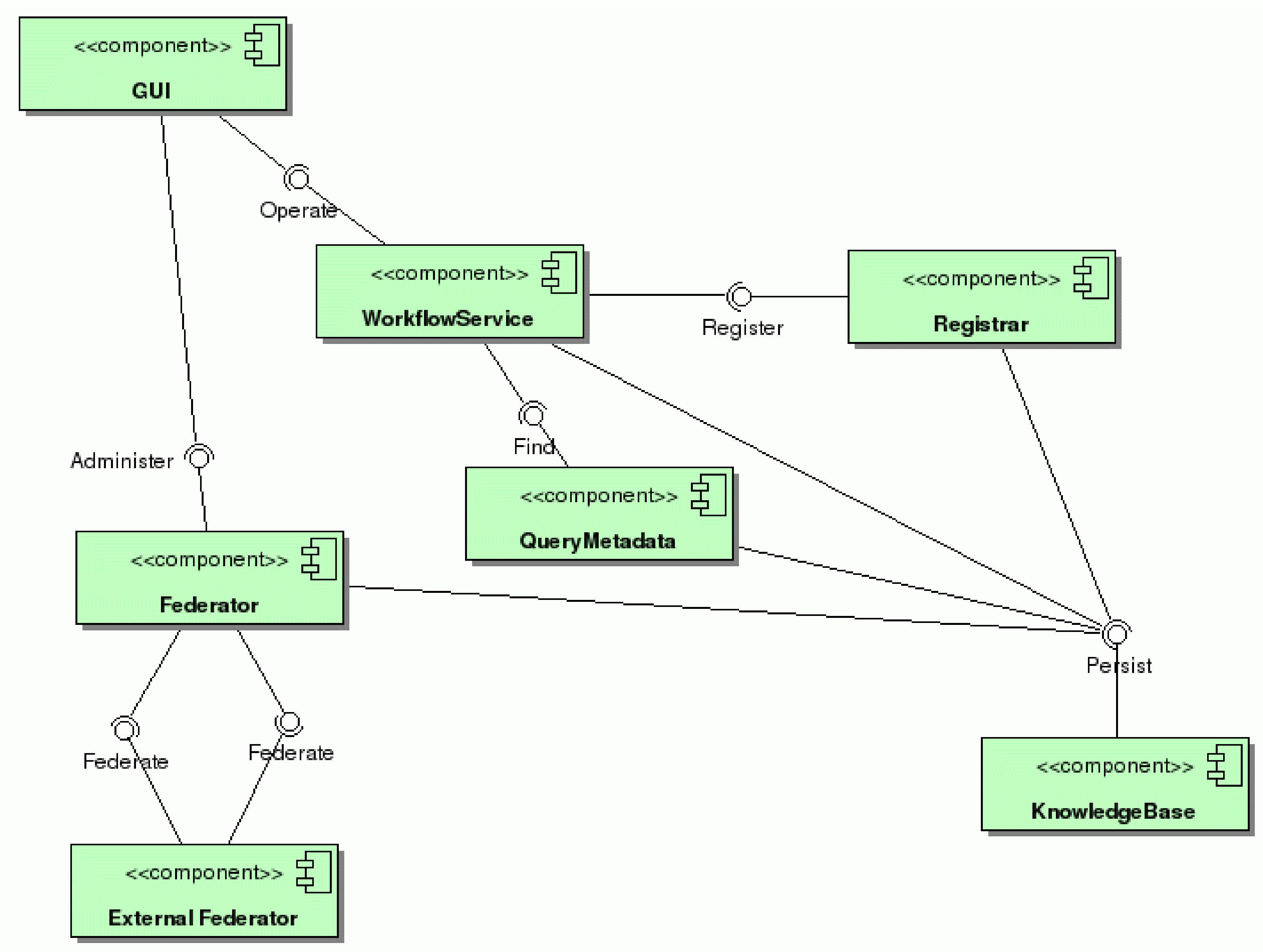
Architecture Goals

- 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)

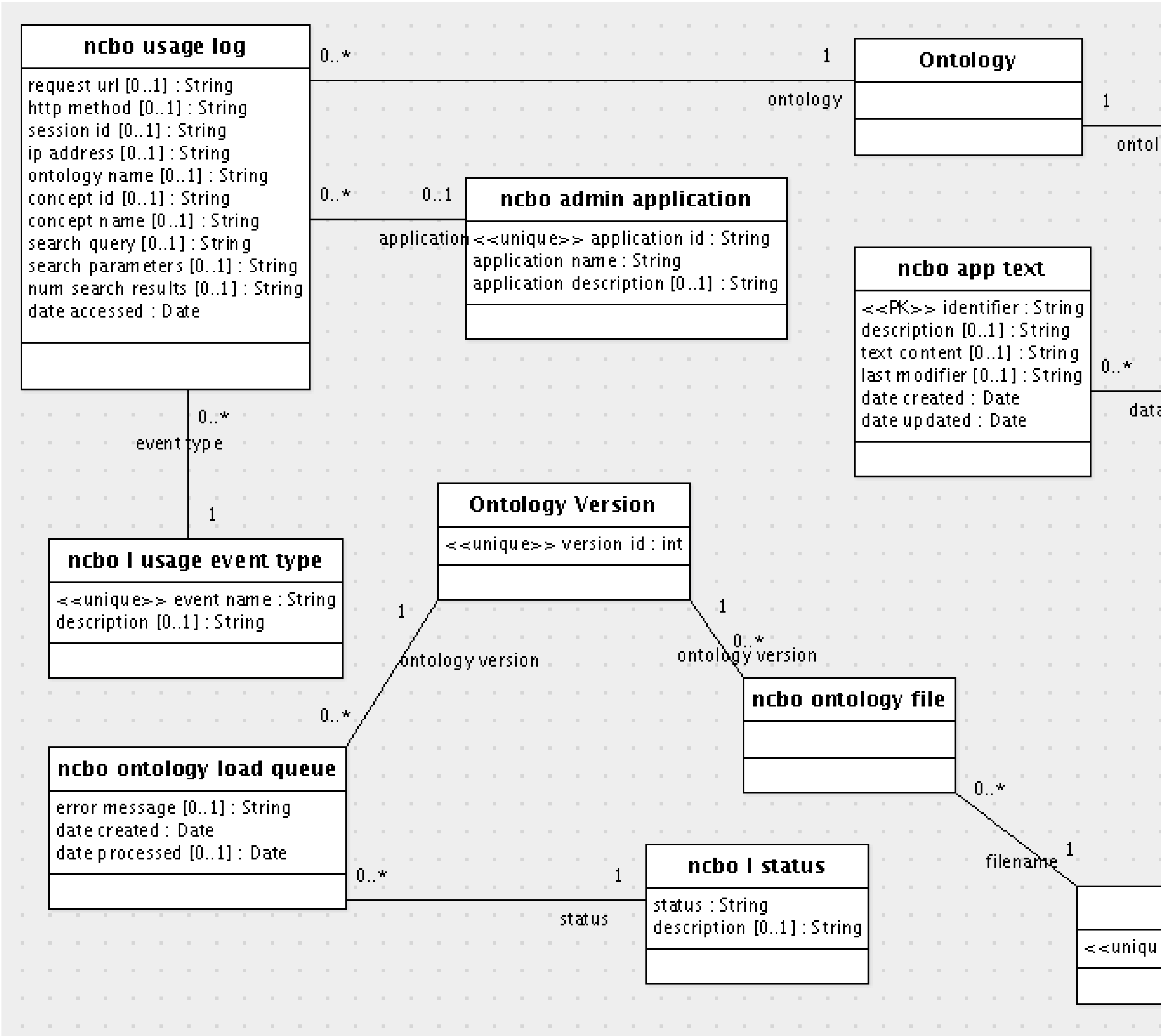
WSDL/SOAP

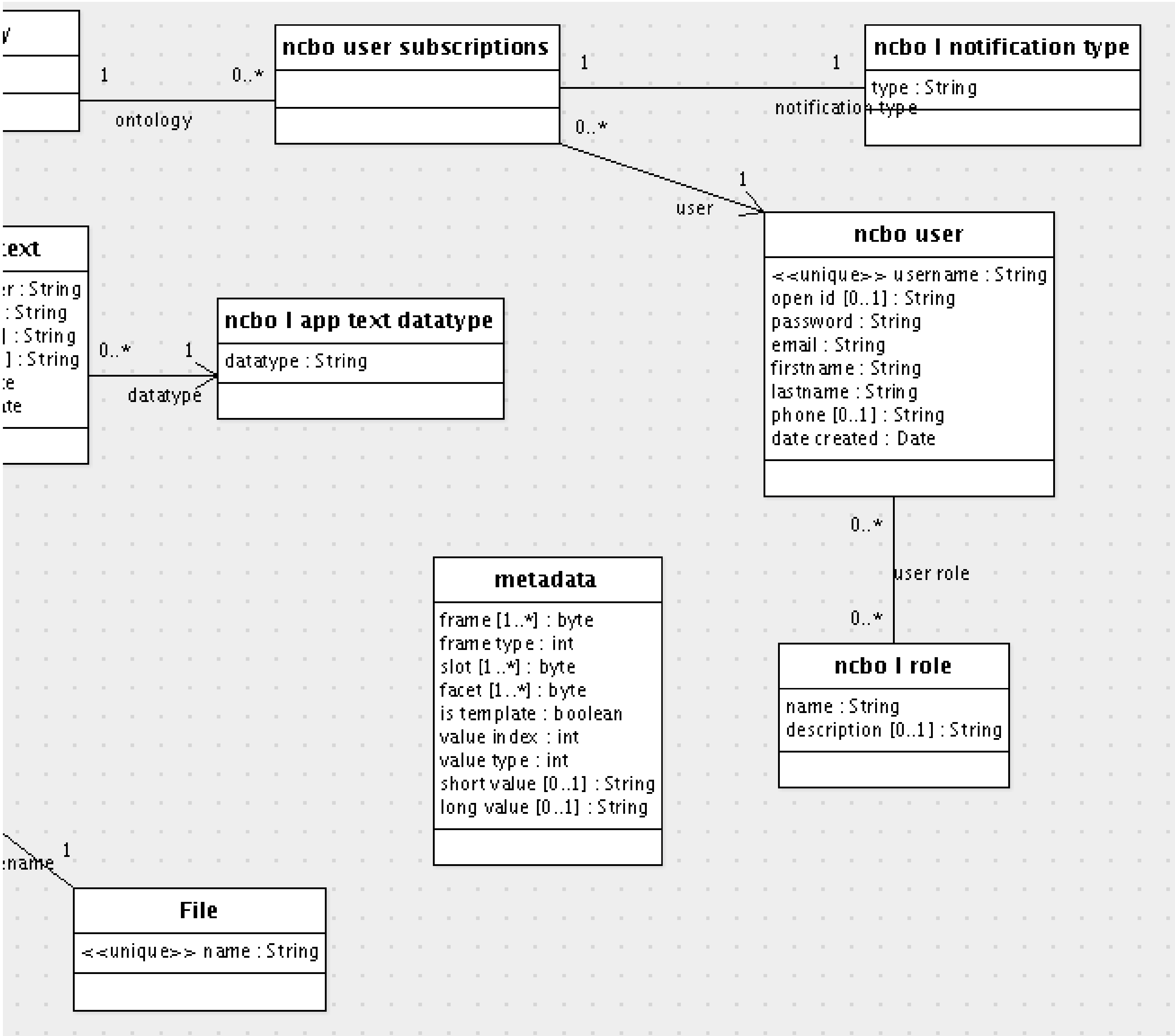
- WSDL and SOAP SEI available at OOR Interface
- Examples:

```
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.


ncbo usage log
request url [0..1] : String
http method [0..1] : String
session id [0..1] : String
ip address [0..1] : String
ontology name [0..1] : String
concept id [0..1] : String
concept name [0..1] : String
search query [0..1] : String
search parameters [0..1] : String
num search results [0..1] : String
date accessed : Date
0..*
1
ontology
Ontology
1
ontol
0..*
0..1
application
ncbo admin application
<<unique>> application id : String
application name : String
application description [0..1] : String
ncbo app text
<<PK>> identifier : String
description [0..1] : String
text content [0..1] : String
last modifier [0..1] : String
date created : Date
date updated : Date
0..*
data
0..*
event type
1
ncbo l usage event type
<<unique>> event name : String
description [0..1] : String
Ontology Version
<<unique>> version id : int
1
ontology version
0..*
1
0..*
ontology version
ncbo ontology file
0..*
filename
1
ncbo ontology load queue
error message [0..1] : String
date created : Date
date processed [0..1] : Date
0..*
1
status
ncbo l status
status : String
description [0..1] : String
<<uniqu


ncbo user subscriptions
ncbo l notification type
type : String
notification type
ontology
1
0..*
user
ncbo user
<<unique>> username : String
open id [0..1] : String
password : String
email : String
firstname : String
lastname : String
phone [0..1] : String
date created : Date
ncbo l app text datatype
datatype : String
datatype
user role
ncbo l role
name : String
description [0..1] : String
metadata
frame [1..*] : byte
frame type : int
slot [1..*] : byte
facet [1..*] : byte
is template : boolean
value index : int
value type : int
short value [0..1] : String
long value [0..1] : String
File
<<unique>> name : String

Suggestions for Future Work

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