caNanoLab – cancer Nanotechnology Laboratory

Software Design Description

Release 1.5

Document Change Record

Version Number	Date	Description
1.0	09/06/06	Initial document
1.1	02/15/07	Initial document
1.2	08/23/07	Initial document
1.2.1	10/22/07	Updated database ER diagram
1.4	07/11/08	Updated for release 1.4
1.5	10/23/09	Updated for release 1.5

TABLE OF CONTENTS

1. INTRODUCTION

The purpose of this document is to describe the design for the cancer Nanotechnology Laboratory (caNanoLab) Release 1.5.

1.1 PROJECT BACKGROUND

caNanoLab is a collaborative effort between the NCI Center for Biomedical Informatics and Information Technology (NCICBIIT), the Nanotechnology Characterization Laboratory (NCL), and Cancer Centers of Nanotechnology Excellence (CCNEs). The project goal is to develop a data portal that allows researchers to share information on nanoparticles including the composition of the particle, the functions (e.g. therapeutic, targeting, diagnostic imaging) of the particle, the characterizations of the particle from physico-chemical (e.g. size, molecular weight, surface) and in vitro (e.g. cytotoxicity, blood contact) nanoparticle assays, and the protocols of these characterization. The data portal is designed to facilitate information sharing in the biomedical nanotechnology research community to expedite and validate the use of nanotechnology in biomedicine under the spirit of the NCI's cancer Biomedical Informatics Grid (caBIG™) program.

The caNanoLab 1.0 release expanded upon the caLab .50 release that focused on capturing the NCL workflow and workflow artifacts. Release 0.5 features included:

- ***Sample Management*** – Accessioning samples (nanoparticles), creating aliquots, searching samples and aliquots
- ***Execute Workflow*** – Creating assay runs, assigning inputs (aliquots and input files) and outputs (result files) to assay runs, searching workflow components for assay results
- ***File Upload*** – Uploading assay result files
- ***Basic Security*** – User authentication via the NCI's Common Security Module (CSM)

The design of release 1.0 features involved extension of the caLab .50 web portal and the generic laboratory object model (lab-OM) with nanoparticle annotations and role-based security. In release 1.0, nanoparticle annotations are represented in a nanoparticle object model (nano-OM) defining nanoparticle compositions, functions, and physical and in vitro characterizations. Release 1.0 features included:

- ***Annotate Nanoparticles*** – Annotating nanoparticles with characterizations resulting from physical and in vitro nanoparticle assays, searching and retrieving nanoparticle characterizations in a secure fashion.
- ***Advanced Security*** – User authorization via the Common Security Module (CSM), User management via the User Provisioning Tool (UPT)

The caNanoLab 1.1 release focused on searching remote caNanoLab grid services for publicly available nanoparticle data. This release requires grid-enabling of the nano-OM and the generation of a caBIG grid (caGrid™) data service with custom operations that allow caNanoLab data to be shared in a federated approach. Release 1.1 features include:

- ***Auto discovery of grid service*** – Automatically discovering caNanoLab grid services registered with a caGrid index server

- ***Remote Nanoparticle Search*** – Searching publicly available nanoparticles from remote caNanoLab grid data services connected to the caGrid

- ***Remote Nanoparticle Composition Data Search*** – Searching publicly available nanoparticle composition data from remote caNanoLab data services connected to the caGrid

- ***Remote Report Search*** – Searching publicly available nanoparticle reports from remote caNanoLab data services connected to the caGrid

The caNanoLab release 1.2 focused on protocol management and end-user usability enhancements that allow for more efficient data entry for characterization annotation. The caNanoLab 1.2.1 release focused on MySQL database support. Release 1.2 features include:

- **Protocol Submission** – Submitting protocols with protocol type, name and version, and uploading associated protocol files

- **Protocol Search** – Searching protocols and protocol files based on protocol types, names and titles

- **Usability Enhancements** – Adding/Removing characterization files and derived data, associating previously submitted protocols to characterizations, deleting characterizations, copying characterizations from a particle to other particles from the same source, and enhancing data entry for function annotation

The caNanoLab release 1.3 focused on additional end-user usability enhancements. Release 1.3 features include:

- **Public Browse** – Browsing publicly available protocols, nanoparticles and reports

- **Detail Views and Summary Views –** Displaying detail information (protocols, instrument configuration, characterization files and derived data) of a single characterization, as well as displaying summary information of all characterizations of a certain type

- **Print and Export of Detail Views and Summary Views** – Printing and exporting of detail views and summary views

The caNanoLab release 1.4 focused on extensions to the 1.0 nano-OM to better capture nanoparticle composition and functionalization, as well enhancements to glossary terms. Release 1.4 features include:

- **Enhanced Nanoparticle Sample Submission and Composition Submission –** New structure for nanoparticle sample, composition and functionalization, as well as new metadata constraints for composition and physical characterizations

- **Seamless Local/Remote Search** – Seamlessly browsing publicly available data (protocols, nanoparticles and reports) stored both locally and across production caNanoLab grid services in a seamless fashion

- **Product Upgrades** – Upgrading the portal and the grid service to the caCORE SDK 4.0 and caGrid 1.2

The caNanoLab release 1.5 focuses on extensions to the 1.0 nano-OM to capture additional curated characterization assays and any uncurated custom assays, to capture additional source metadata and curated techniques and instruments and data conditions associated with characterization assays. Release 1.5 features include:

- **Additional Curated Assays and Custom Assays** – Submitting relaxivitiy as a physico-chemical characterization and transfection as an in vitro characterization, as well as submitting uncurated custom characterization types, characterizations and assays.

- **Enhanced Source Metadata** – Submitting investigators, organizations and manufacturers associated with a nanoparticle.

- **Data Conditions** – Submitting data conditions (temperature, time, media solvent, etc.) associated with characterization data.

- **Curated Techniques and Instruments –** Submitting techniques and instruments associated with characterizations through a curated list.

- **Advanced Search** – Submitting range queries through a dynamic query builder using nanoparticle point of contact, composition and characterization information.

- **Enhanced Summary Views –** Viewing summaries of nanoparticle composition, characterizations, and publications in a single page with print and export capabilities.

- **Cross-references to PubChem** – Submitting PubChem data source name and data source ID for chemical names associated with nanomaterial entities or functionalizing entities.

- **Administrative Functions** – Submitting site preferences such as site name and site logo.

- **Product Upgrades** – Upgrading the web application to use CSM 4.1 and the grid service to caGrid 1.3

1.2 IDENTIFICATION

The system identification information to which this document applies is as follows:

- ***Title***: cancer Nanotechnology Laboratory
- ***Abbreviation:*** caNanoLab
- ***Version:*** 1.5
- ***Release Number:*** 1.5

1.3 REQUIREMENTS

The following is a list of high-level functional and technical requirements for caNanoLab 1.5:

Req. #	Requirement
1.0	**The system shall provide advanced security that provides support for role based authorization**
1.1	The system shall leverage the NCICB Common Security Module (CSM) version 4.1 and the User Provisioning Toolkit (UPT)
1.1.1	The system shall allow administrators to configure groups and group privileges (read only, read-write)
1.1.2	The system shall provide support for user management
1.2	The system shall upgrade to CSM 4.1
2.0	**The system shall support submission and retrieval of nanoparticle annotations**
2.1	The system shall allow users to annotate and search nanoparticles based on the composition specific to each nanoparticle type
2.2	The system shall allow users to annotate and search nanoparticles based on characterizations, functions, keywords and summary/descriptions associated with characterizations
2.3	The system shall allow users to search nanoparticles and its annotations through advanced searches that include range queries
2.3	The system shall provide support for submission of physico-chemical characterizations
2.4	The system shall provide support for submission of in vitro characterizations
2.5	The system shall provide support for submission of in vivo characterizations (not yet with curated data)
2.6	The system shall allow user to create custom characterization types, characterizations and assay types

2.5	The system shall allow authorized users to submit assay results (e.g. distribution graphs) and associate assay results with annotations (e.g. size distribution graph, z-avg. size, PDI)
2.6	The system shall allow authorized users to submit techniques and instruments associated with the characterizations
2.7	The system shall allow authorized users to set visibility of submitted samples and their assay results
2.8	The system shall allow users to upload and retrieve sample publications.
2.9	The system shall allow authorized users to set visibility of uploaded publications.
2.10	The system shall allow users to associate publications with multiple nanoparticles
2.11	The system shall allow users to submit and search protocols
2.12	The system shall allow authorized users to set visibility of submitted protocols
2.13	The system shall allow authorized users to copy characterizations from one nanoparticle to other nanoparticles from the same nanoparticle source
2.14	The system shall allow authorized users to delete characterizations
2.15	The system shall allow user to specify both base composition of a nanoparticle and the functionalization of a nanoparticle.
2.16	The system shall allow user to specify the chemical associations between different nanoparticle entities in case of a complex nanoparticle, or the chemical association between a nanoparticle entity and a functionalizing entity.
2.17	The system shall allow user to upload files for nanoparticle composition.
2.18	The system shall allow user to specify multiple functions for a nanoparticle.
2.19	The system shall allow user to enter custom characterization types, characterizations, and assays.
2.20	The system shall allow user to copy compositions from one nanoparticle to other nanoparticles from the same nanoparticle source
2.21	The system shall allow authorized users to delete compositions
2.22	The system shall allow authorized users to remove publication associations from nanoparticles.
2.23	The system shall link chemical names entered in composition to PubChem
3.0	**The system shall be engineered for caBIG compatibility**

3.1	The system shall leverage the caCORE SDK and associated components
3.2	Concepts for supported objects/attributes shall be registered in the EVS
3.3	The object model shall be loaded into the caDSR
3.4	The system shall be upgraded to make use of caCORE SDK 4.0
4.0	**The system shall grid-enable the caNanoLab object model and register** **The grid service in the grid index server**
4.1	The system shall auto-discover remote caNanoLab data services
4.2	The system shall search public protocols, samples and publications against remote caNanoLab grid services
4.3	The system shall search public nanoparticle composition data against remote caNanoLab grid services
4.4	The system shall allow users to seamlessly search locally stored protocols, samples and publications as well as the publicly available protocols, samples and publications stored in remote caNanoLab grid services.
4.5	The system shall be upgraded to caGrid 1.3
4.6	The system shall search public composition data, physic-chemical characterization data and in vitro characterization against remote caNanoLab grid services
5.0	**The system shall provide definitions of terms and easy access to these definitions.**
6.0	**The system shall provide administrative functions for user to upload a logo and set a site name**

Table 1-1: Requirements Matrix

These requirements are further expanded in the *caNanoLab 1.0, 1.1, 1.2, 1.4 and 1.5 Use Case Specification* documents.

1.4 SYSTEM OVERVIEW

The primary goals of the caNanoLab Release 1.5 effort are to:

- Provide for the efficient storage and retrieval of nanoparticle information to expedite and validate the use of nanotechnology in medicine
- Develop and support an initial standard for representing nanoparticle information to facilitate data sharing in a semantically interoperable fashion in the spirit of caBIG™

- Implement a system that facilitates data sharing in the nanotechnology community in a secure fashion

1.4.1 System Development

caNanoLab release 1.5 is being designed and developed following the Rational Unified Process (RUP) in an iterative fashion. As such, it is intended that this document will be updated frequently during the elaboration phase.

1.4.2 Operations and Maintenance

The reference implementation of caNanoLab 1.5 is being deployed and maintained at the Advanced Biomedical Computing Center (ABCC) in NCI Frederick for use at the NCL.

1.4.3 Key Stakeholders

Key stakeholders associated with the effort are defined as follows:

- Nanotechnology Characterization Laboratory (NCL) – End users of the caNanoLab system
- Cancer Centers of Nanotech Excellence (CCNEs) – Consumers of caNanoLab data and end users of the caNanoLab systems installed at the centers
- Advanced Biomedical Computing Center (ABCC) – Provides caNanoLab operations and maintenance support
- National Cancer Institute Center for Bioinformatics and Information Technology (NCICBIIT) – Provides development and metadata support

1.5 DOCUMENT OVERVIEW

This document provides an overview of caNanoLab system design. It is intended to describe and capture the caNanoLab design decisions and is written as a reference for both the NCI customer and the NCI contractors/developers involved with the evolution of caNanoLab. This is a living document and will be updated to reflect the evolution of development, as well as future enhancements and upgrades.

The main sections of this document are listed and described below.

- ***Section 1. Introduction:*** provides a purpose statement, project background, requirements matrix, system overview, and document overview.

- ***Section 2. System Architecture:*** describes an overview architecture design of the J2EE system and the various environments that will be used throughout the development lifecycle.

- ***Section 3. Components:*** describes design details of the system components: domain model, data model, package structure, user interface, business services, file repository structure, and user authentication and authorization.

- ***Section 4. Grid-Enabled Architecture*:** describes an overview of the grid-enabling architecture and design details of the caNanoLab grid data service.

1.6 REFERENCE DOCUMENTS

Document Name	Location
caNanoLab 1.5 Installation Guide	http://gforge.nci.nih.gov/frs/download.php/7483/caNanoLab_1.5_Installation_Guide.pdf
CSM 4.1 Programmer's Guide	http://gforge.nci.nih.gov/docman/view.php/12/15147/caCORE_CSM_v41_ProgrammersGuide.pdf
caCORE SDK 4.0 Developer's Guide	https://gforge.nci.nih.gov/docman/view.php/148/8650/caCORE%20SDK%204.0%20Developer's%20Guide_101007.pdf
caGrid 1.3 Documentation	http://wiki.cagrid.org/display/introduce13/Documentation

Table 1-2: Reference Documents

2. SYSTEM ARCHITECTURE

2.1 SYSTEM ARCHITECTURE OVERVIEW

The caNanoLab system is designed as a Java 2 Enterprise Edition (J2EE) n-tier system that includes five loosely coupled yet tightly cohesive functional tiers: a client tier, a presentation tier, a business tier, a persistence tier and an EIS tier. The following architecture diagram (Figure 2-1) depicts the overall caNanoLab system architecturem. An overview of each tier is provided in the sections immediately following the diagram. Note that the caNanoLab grid data service architecture is described separately in section 4, not as a part of the caNanoLab system architecture.

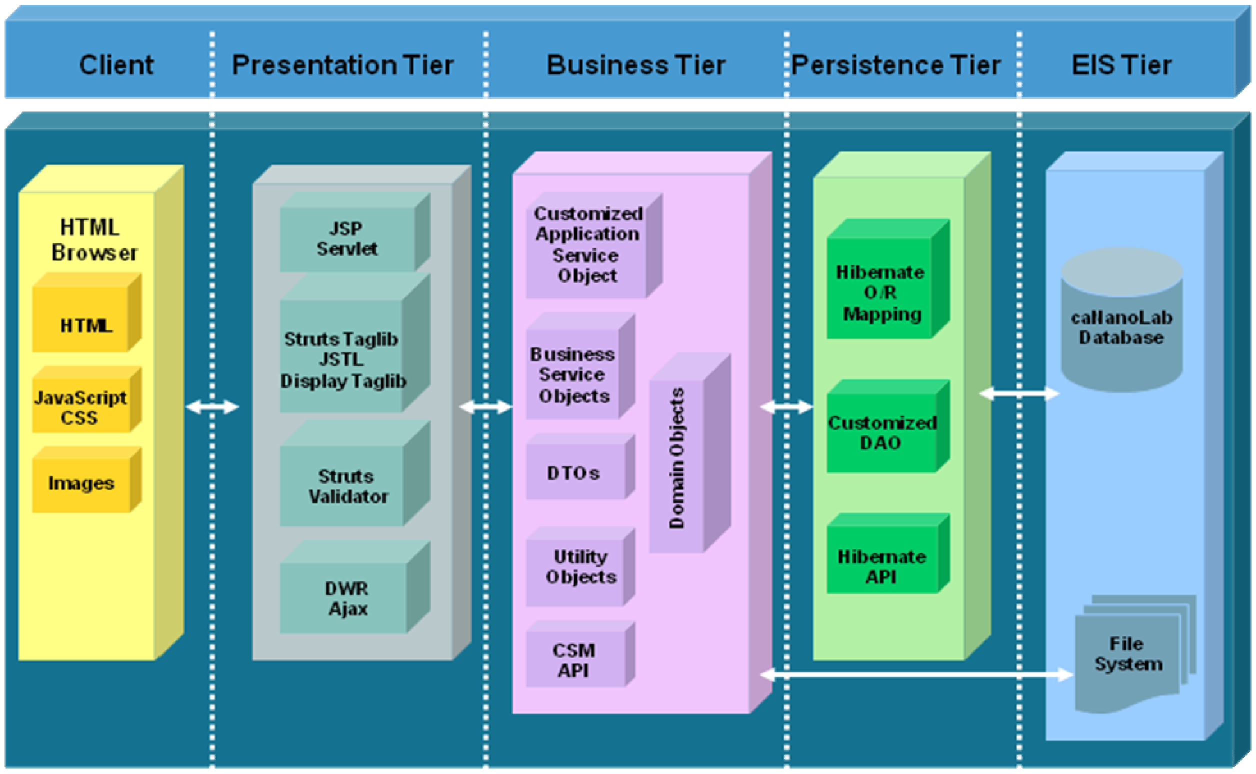


Figure 2-1: System Architecture

2.1.1 Client Tier

The caNanoLab client tier, like all other J2EE web based systems, consists of web browsers (e.g. Internet Explorer, FireFox, etc.) serving HTMLs, JavaScripts, Cascading Style Sheets (CSS) and images.

2.1.2 Presentation Tier

The caNanoLab presentation tier is implemented using the Struts Servlet/JSP Framework following the Model-View-Controller pattern. The Struts ActionServlet intercepts every request from the client tier and populates a Struts ActionForm with HTTP request parameters defined from user inputs. These request parameters are validated through the Struts Validator Framework. Once validated, the ActionServlet initiates an appropriate Struts Action to process the request and pass the response back to a JSP page for rendering. DWR Ajax framework, Struts tag library, display tag library and JavaServer Pages Standard Tag Library (JSTL) are used to help dynamically render HTML pages displayed in the client browsers.

Tiles Framework is used to separate layout from content such that the same layout components are reused in different JSP pages with different contents, and developers only need to maintain one set of HTML layout codes.

2.1.3 Business Tier

The business tier is responsible for implementing the business logic in the application and bridges the gap between the presentation tier and the persistence tier (described in the next section). The caNanoLab business tier consists of a customized application service object extending the application service object provided the caCORE SDK, business service objects, data transfer objects (DTO), utilities objects, and domain objects (Plain Old Java Objects generated by the caCORE SDK), of which the metadata have been registered with the Cancer Center Data Standards Repository (caDSR).

When a HTTP request is passed to the Struts action for processing, an appropriate business service object is invoked by the action class to carry out the associated business logic. The business service objects call the application service to communicate with the persistence tier to retrieve/store domain objects from/to the underlying data sources. The retrieved domain objects are transformed into DTOs if necessary before they are passed back to the presentation tier for viewing. The business objects call utility objects if necessary to assist with the data processing and data transformation. The business objects also works with the utility objects to retrieve/store data from/to files on the file system. In addition, the business service objects calls the Common Security Module (CSM) API for user authentication and authorization.

2.1.4 Persistence Tier

The persistence tier is responsible for executing queries to the underlying data sources and fetching query results. The caNanoLab persistence tier is implemented using the Hibernate object-relational mapping (ORM) framework. Domain objects are mapped to database tables through the Hibernate ORM files generated by the caCORE SDK. A customized data access object (DAO) extending the DAO provided by the caCORE SDK is implemented to provide create, read, update and delete (CRUD) functions. The DAO in turn uses Hibernate APIs to generate and execute SQL queries, and is used by the application service object in the business tier to communicate with the underlying databases.

2.1.5 Enterprise Information System (EIS) Tier

The EIS tier is the backend tier that is responsible for providing and storing data. The caNanoLab EIS tier consists of a caNanoLab database and a file system. The caNanoLab database contains database objects for storing the caNanoLab application data, and also contains database objects for storing CSM authentication and authorization data. The file system stores composition files, characterization files, reports and protocols uploaded in the application.

2.2 SYSTEM DEPLOYMENT

The caNanoLab system is designed to be deployed in any Servlet container that conforms to Java Servlet specification 2.4 and JSP specification of 2.0 with an underlying server hosting the MySQL database. For both NCICBIIT and NCL deployments, JBoss Java 2 Enterprise Edition (J2EE) application server integrated with Apache web server is used to host the application. At the NCICBIIT, the caNanoLab system is deployed in a four-tier hardware environment comprising development, QA, staging and production. At the NCL, the system is deployed in a two-tier hardware environment comprising of staging and production.

3. COMPONENTS

3.1 DOMAIN MODEL

The caNanoLab domain model is designed following the caBIG design principles that include UML modeling, open APIs, controlled vocabularies and caDSR registered metadata. The caNanoLab domain model has been semantically annotated with concepts from the Enterprise Vocabulary System (EVS). The data elements describing the caNanoLab domain model has been registered into the caDSR. The caNanoLab domain model (nano-OM) focuses on developing a standard representation of nanoparticles and their compositions and functionalizations, as well as results of characterization assays applied to the nanoparticles. The nano-OM has been modeled after NCL's nanoparticle assay cascade and the Nanoparticle Ontology (NPO) from the Washington University. The annotation for each class and attributes are maintained in EVS, and these concepts form the basis of a standard ontological hierarchy of terminology applicable to the use of nanotechnology in medicine.

As shown in the package hierachary diagram in Figure 3-1, the nano-OM is composed of the following seven high-level packages:

- `particle` – Defines classes describing a nanoparticle and the high-level information directly associated with a nanoparticle, such as its composition, its functions and its characterizations.
- `nanomaterial` – Defines classes describing different types of materials that are used in a base nanoparticle construct.
- `agentmaterial` – Defines classes describing different types of functionalizing materials that can be used to associate with the base nanoparticle construct to functionalize the nanoparticle.
- `linkage` – Defines classes describing different kinds of associations between different types of materials within a nanoparticle composition.
- `Characterization` – Defines classes describing physico-chemical, in vitro and in vivo characterization assays applied to a nanoparticle in three different sub-packages: `characterization.physical`, `characterization.invitro`, `characterization.invivo`, as well as a class `OtherCharacterization` capturing any other uncurated assays.
- `common` – Defines classes that are commonly shared in the model, for example, `File`, `Keyword`, `Finding`, etc.
- `function` – Defines classes describing different types of functions a nanoparticle could have.

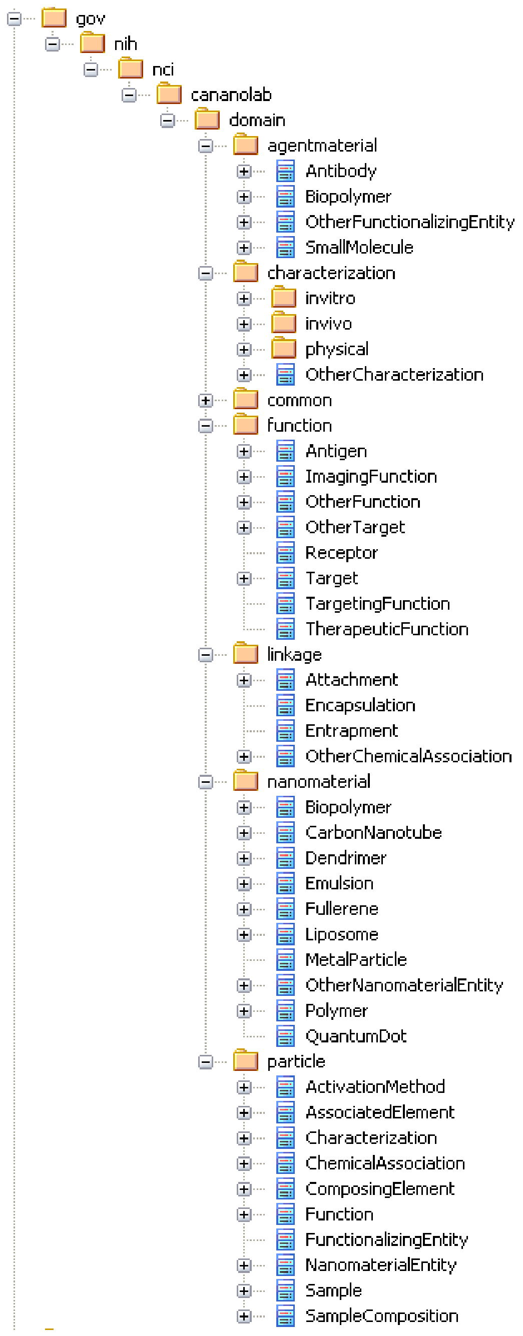


Figure 3-1: nano-OM Package Hierarchy

In release 1.5, we've renamed the original `NanoparticleSample` class to `Sample` class to broaden the system scope to capture non-nanomaterial samples (such as drugs) that are used in a nanoparticle study along with nanoparticles. The subsections that follow describe nano-OM classes and attributes in more details from three overview perspectives: sample, composition and characterization.

3.1.1 Sample Overview

As shown Figure 3-2, at the highest level, a Sample has a primary PointOfContact and an optional collection of other PointOfContact, an optional SampleComposition, an optional collection of Characterization, and an optional collection of keywords and an optional collection of publications.

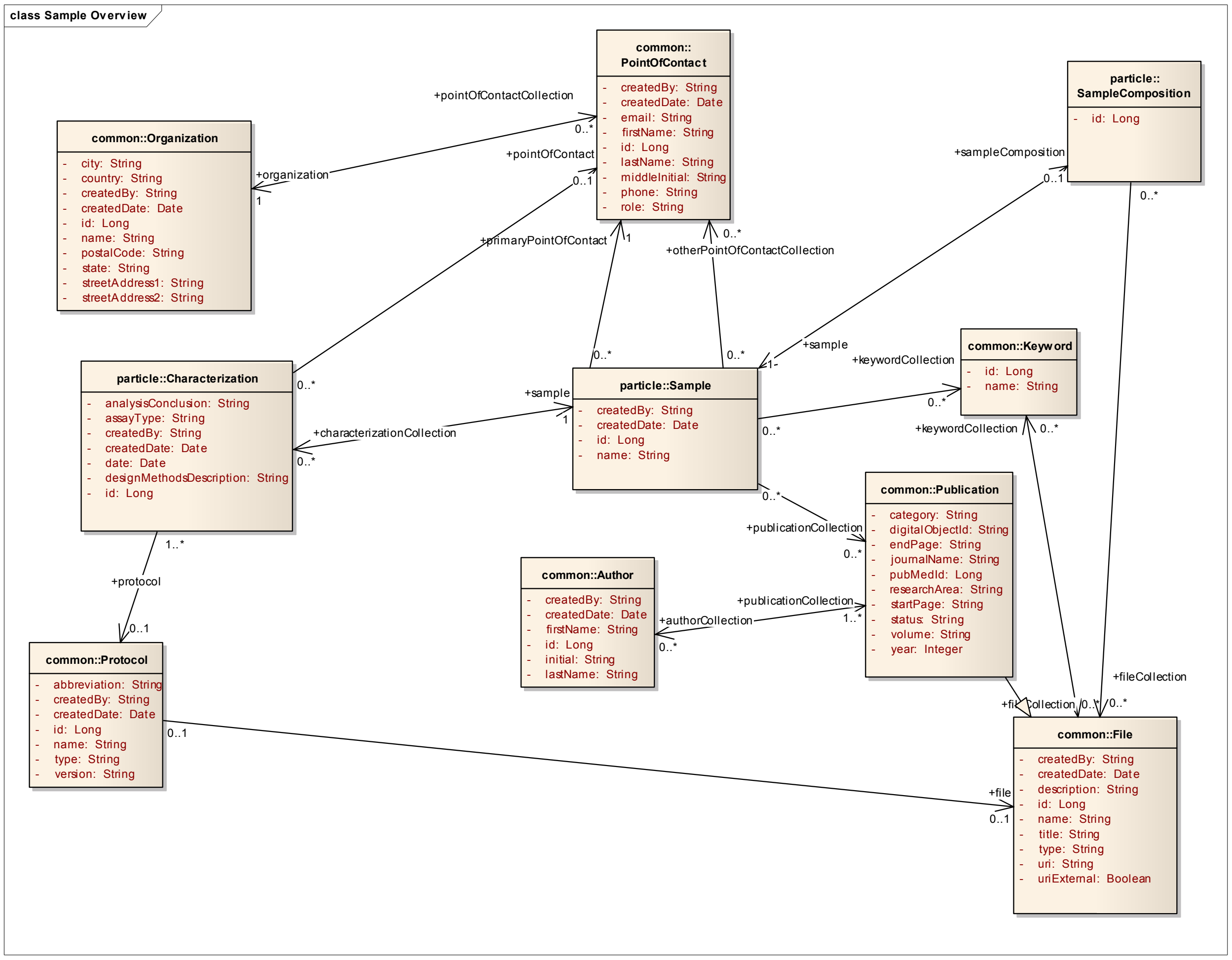


Figure 3-2: Nanoparticle Sample Overview (nano-OM)

3.1.2 Composition Overview

As shown in Figure 3-3, a SampleComposition of a Sample has a collection of NanomaterialEntity objects, an optional collection of FunctionalizingEntity objects, an optional collection of ChemicalAssociation objects (Encapsulation, Attachment, Entrapment, Other) and an optional collection of File objects. Each NanomaterialEntity has an optional collection of ComposingElement objects, and each ComposingElement object has an optional collection of Function objects as inherent functions of the sample. Each FunctionalizingEntity has a collection of Function objects as the functions of the sample resulted from sample functionalization. Each ChemicalAssociation object has an optional collection of File objects.

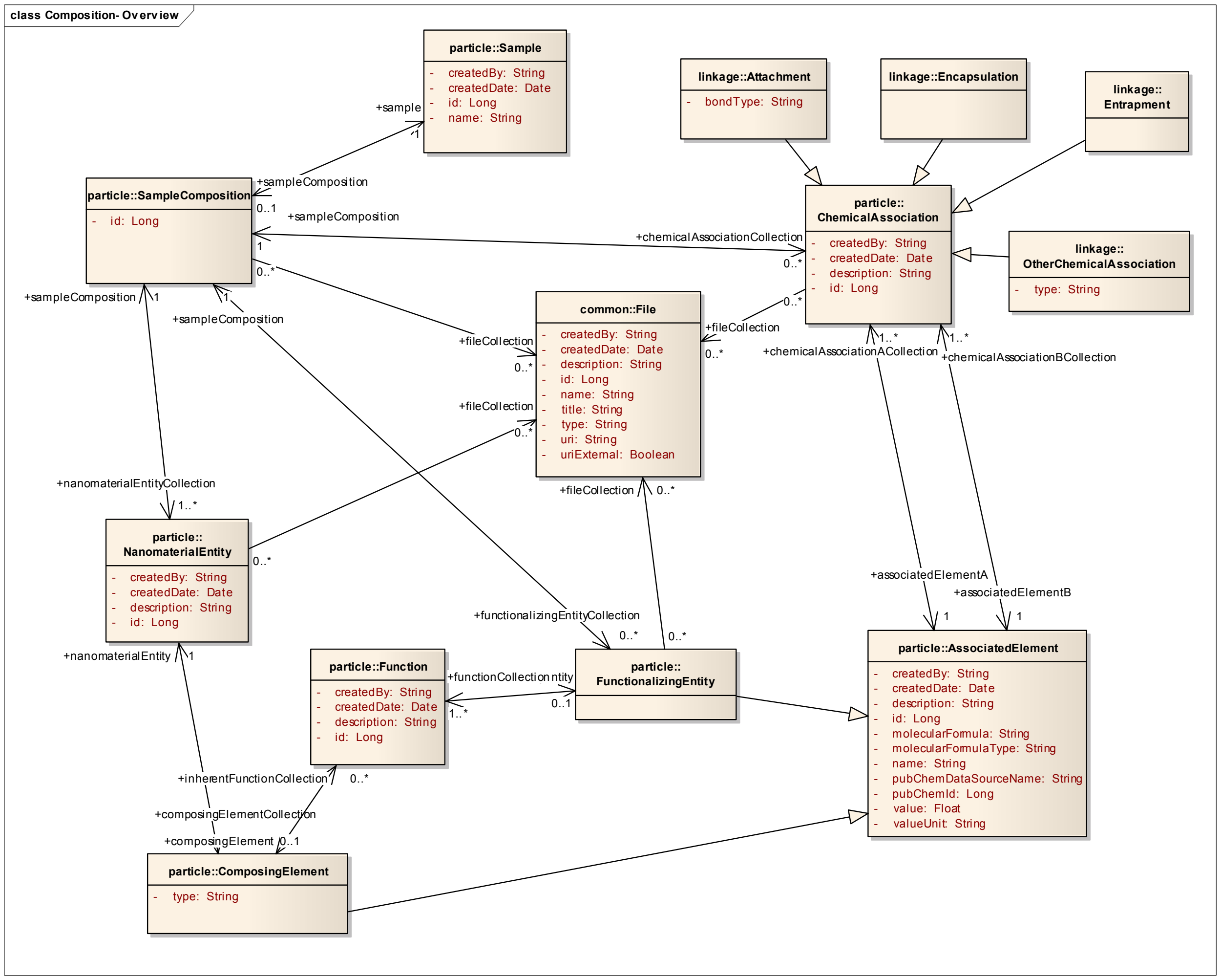


Figure 3-3: Composition Overview (nano-OM)

3.1.2.1 *Composition – Nanomaterial Entity*

As shown in Figure 3-4, a NanomaterialEntity describes the base constructs of a Nanoparticle Sample. In the nano-OM, we describe the following types of NanomaterialEntity: Biopolymer, CarbonNanotube, Dendrimer, Emulsion, Fullerene, MetalParticle, Polymer, QuantumDot, and OtherNanomaterialEntity for any uncategorized nanomaterial entities. Each NanomaterialEntity has an optional collection of ComposingElement that describes the individual constituents of a NanomaterialEntity base construct. The inherence functions of a nanoparticle sample are described as a collection of Function objects associated with the ComposingElement object. In the nano-OM, we describe three types of Function objects: ImagingFunction, TargetingFunction and TheraupeticFunction, with others categorized as OtherFunction.

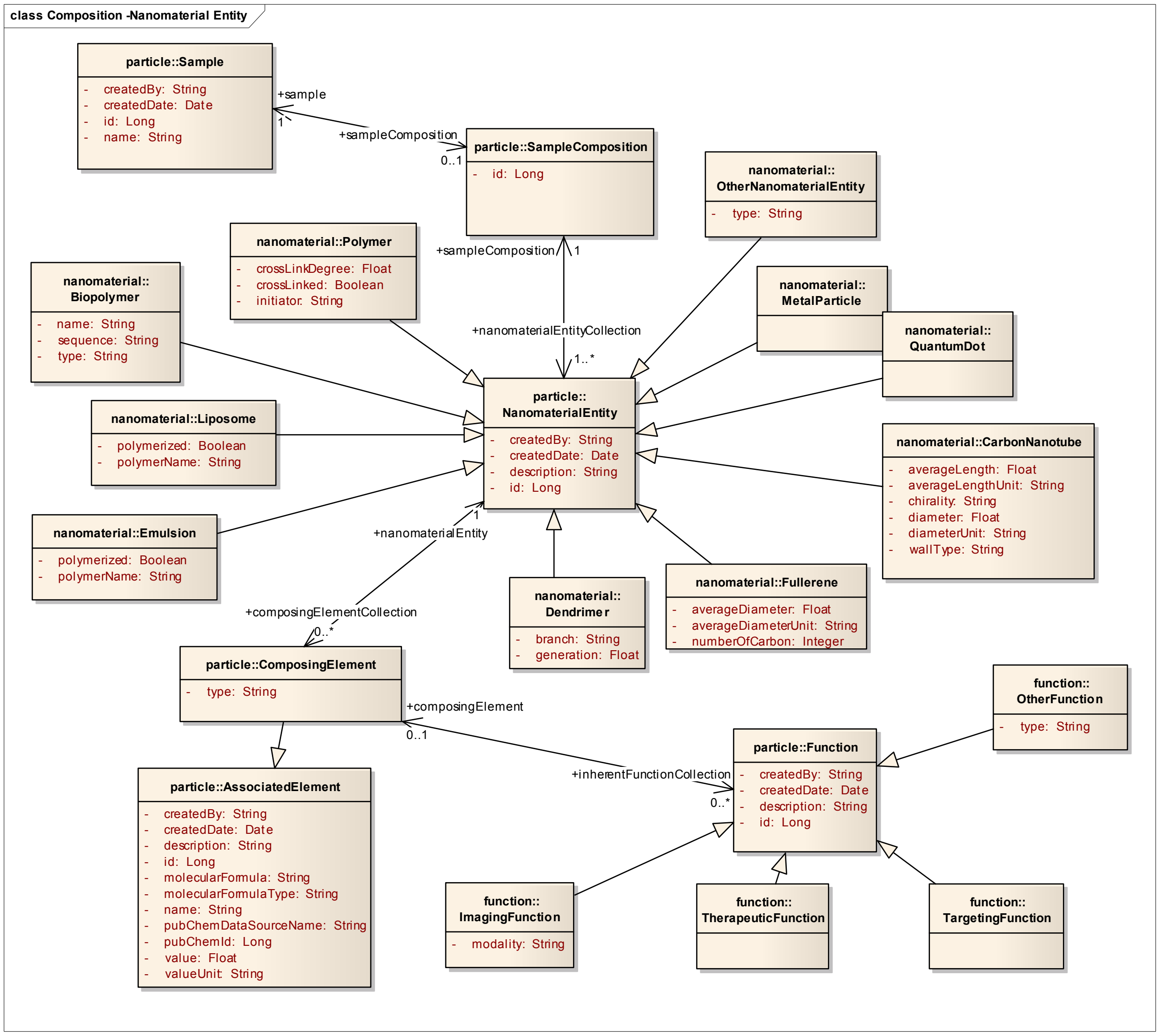


Figure 3-4: Composition – Nanomaterial Entity (nano-OM)

3.1.2.2 *Composition – Functionalizing Entity*

As shown in Figure 3-5, a FunctionalzingEntity is described as the functionalization material of a Sample. In the nano-OM, we describe the following types of FunctionalizingEntity: Antibody, Biopolymer, SmallMolecule, and others are categorized as OtherFunctionalizingEntity. Each FunctionalizingEntity has a collection of Function objects describing the functions a nanoparticle sample acquire as a result of functionalization. When describing functions of a nanoparticle sample in the context of FunctionalizingEntity, TargetingFunction is further described to have an optional collection of Target objects. There are three defined types of Target objects: Antigen, Gene, Receptor, and others categorized as OtherTarget.

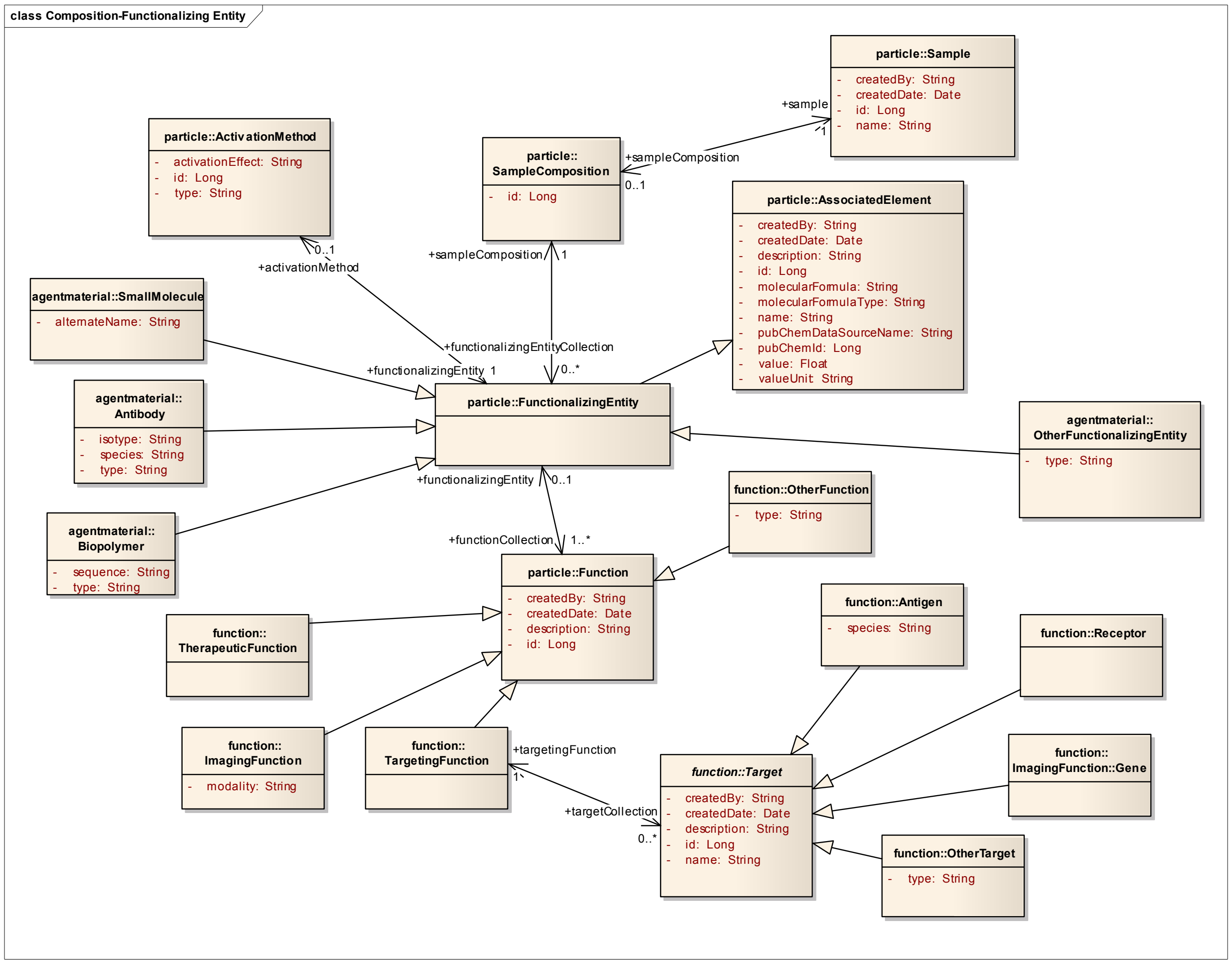


Figure 3-5: Composition – Functionalizing Entity (nano-OM)

3.1.3 Characterization Overview

As shown in Figure 3-6, a Characterization has an optional Protocol which optionally associates with File; an optional collection of ExperimentConfig, which has a Technique and an optional collection of Instrument; an optional collection of Finding, which has an optional collection of Datum with an optional collection of Condition, and an optional collection of File. Each File has an optional collection of Keyword objects. In the nano-OM, we describe PhysicoChemicalCharacterization, InvitroCharacterization, and InvivoCharacterization as three types of Characterization, with others categorized as OtherCharacterization.

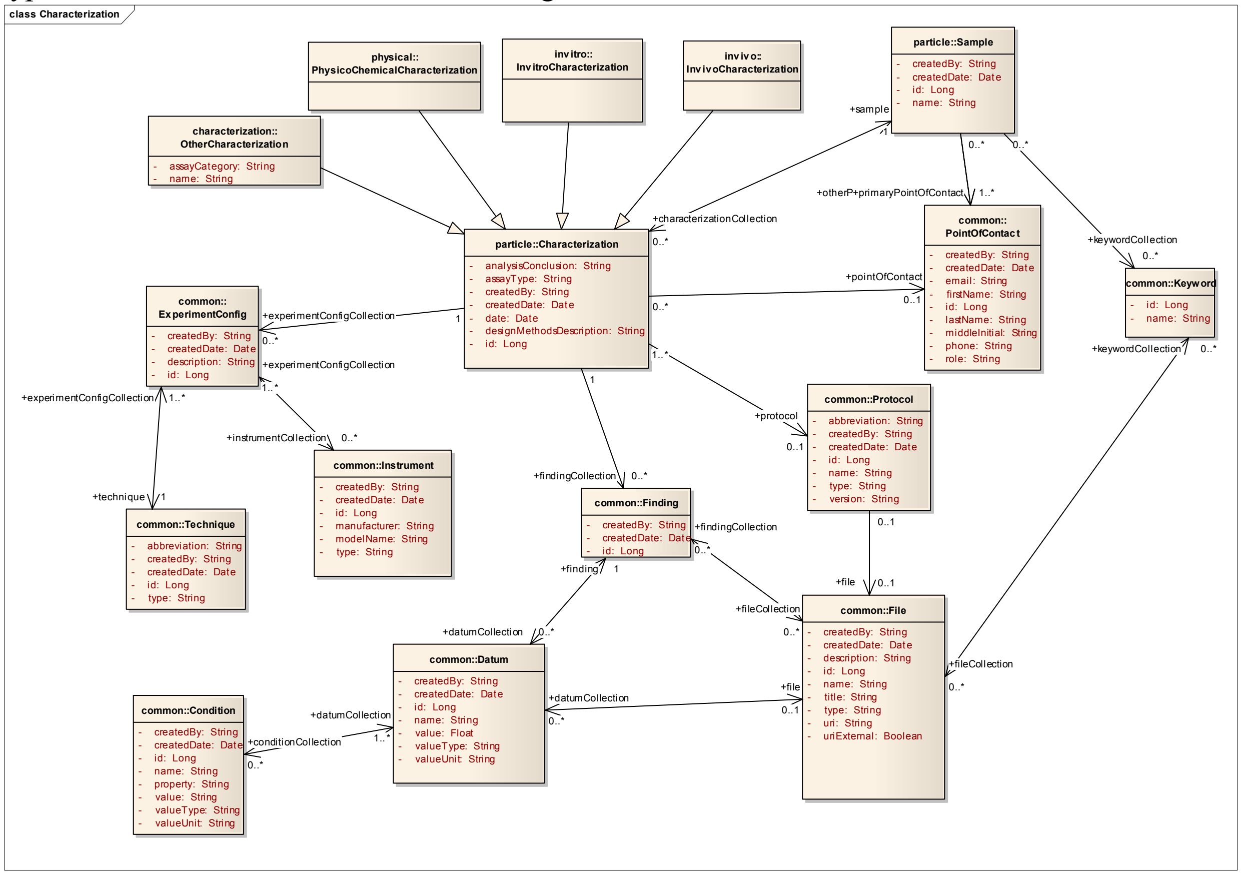


Figure 3-5: Characterization Overview (nano-OM)

3.1.3.1 *Physico-chemical Characterization*

As shown in Figure 3-6, we describe the following types of PhysicoChemicalCharacterization: MolecularWeight, PhysicalState, Purity, Reflexivity, Shape, Size, Solubility, and Surface.

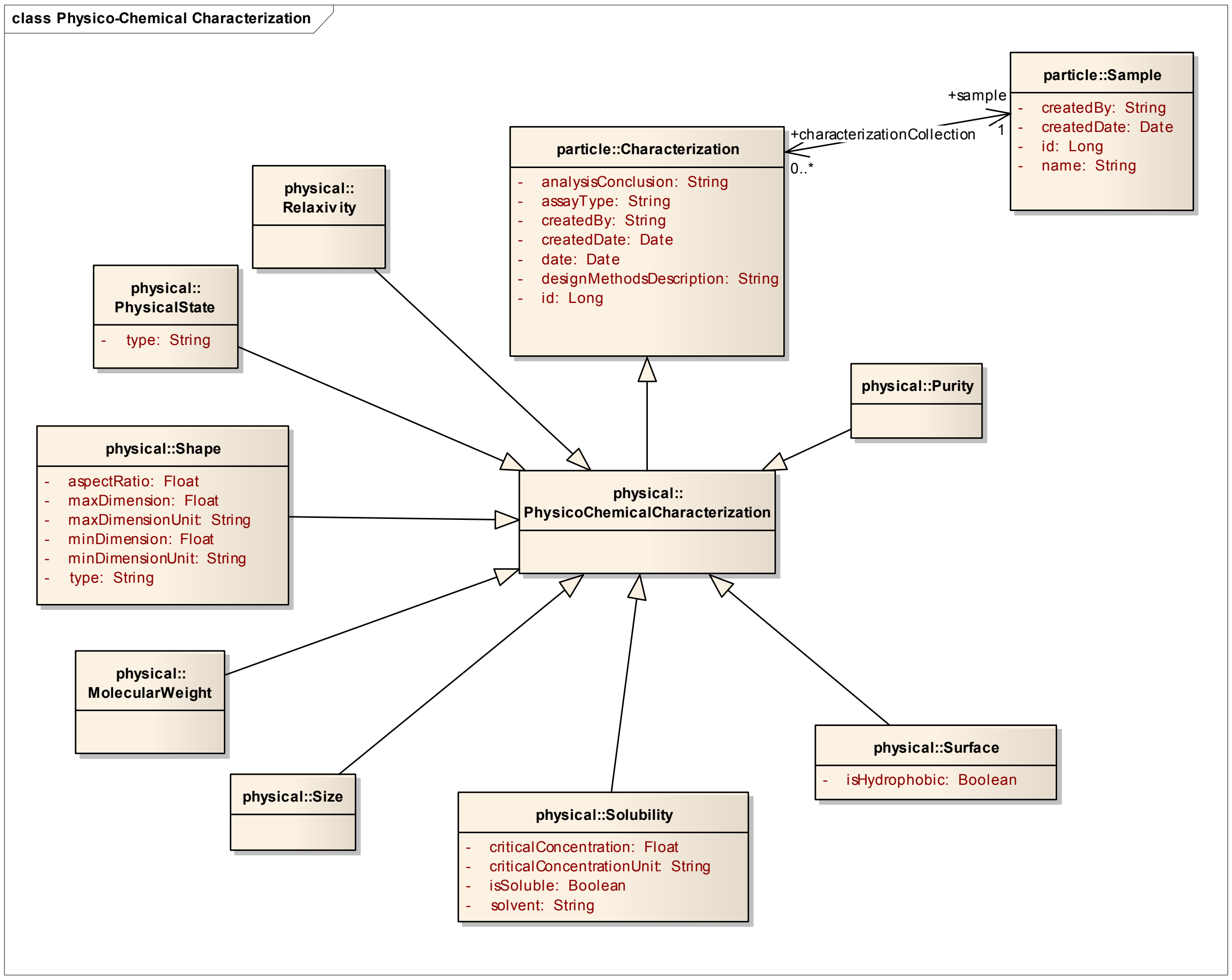


Figure 3-6: Physico-chemical Characterization (nano-OM)

3.1.3.2 *In Vitro Characterization*

As shown in Figure 3-7, we describe the following types of InvitroCharacterization: BloodContact, Cytotoxicity, EnyzmeInduction, ImmuneCellFunction, MetabolicStability, OxidativeStress, Sterility, Targeting, and Transfection.

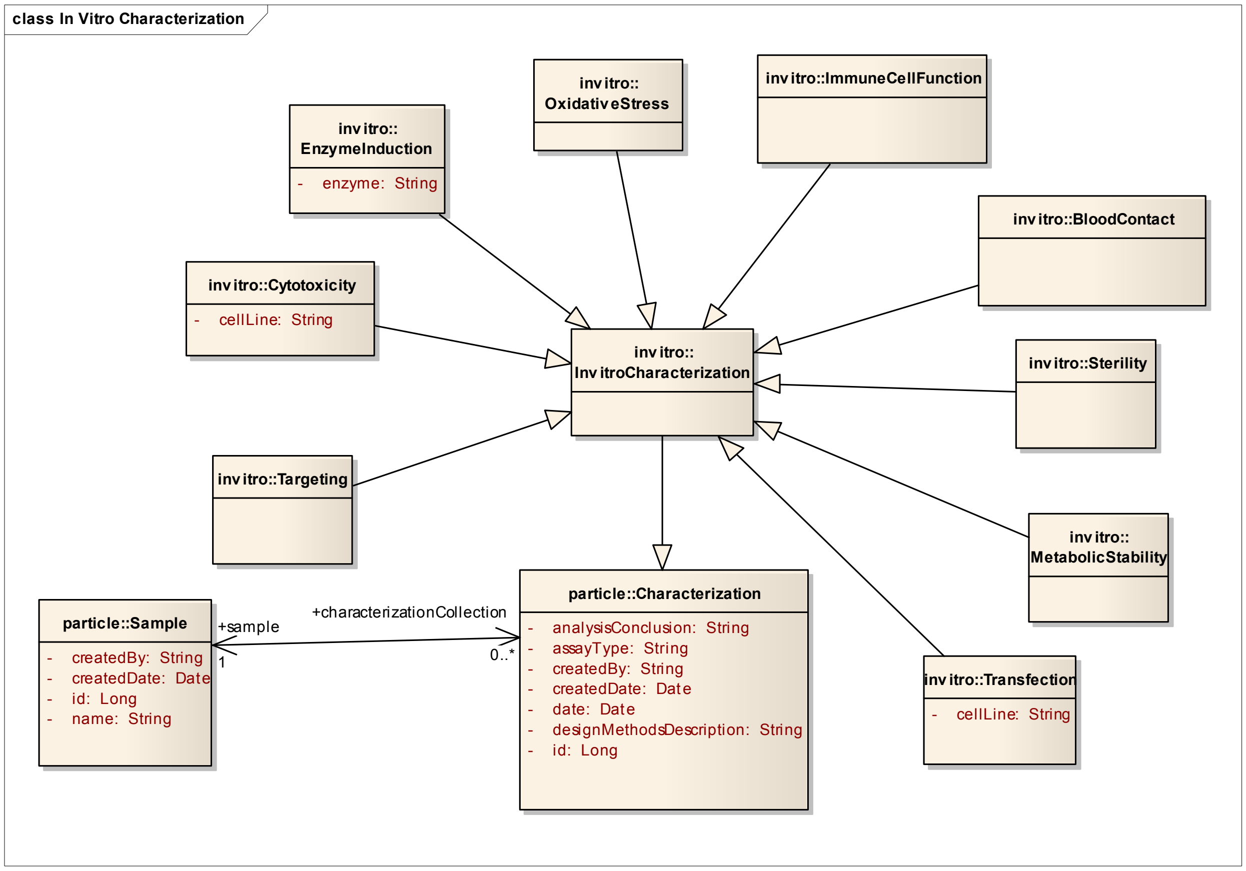


Figure 3-7: In Vitro Characterization (nano-OM)

3.1.3.3 *In vivo characterization*

In release 1.5, we only included two high-level in vivo characterizations: Pharmacokinetics and Toxicology as shown in Figure 3-8. A more detailed in vivo model has been designed through discussions with the caNanoLab In Vivo Characterizations Discussion Group, and will be implemented in future caNanoLab releases. More details about the in vivo model can be found on the caNanoLab wiki page https://wiki.nci.nih.gov/display/ICR/caNanoLab.

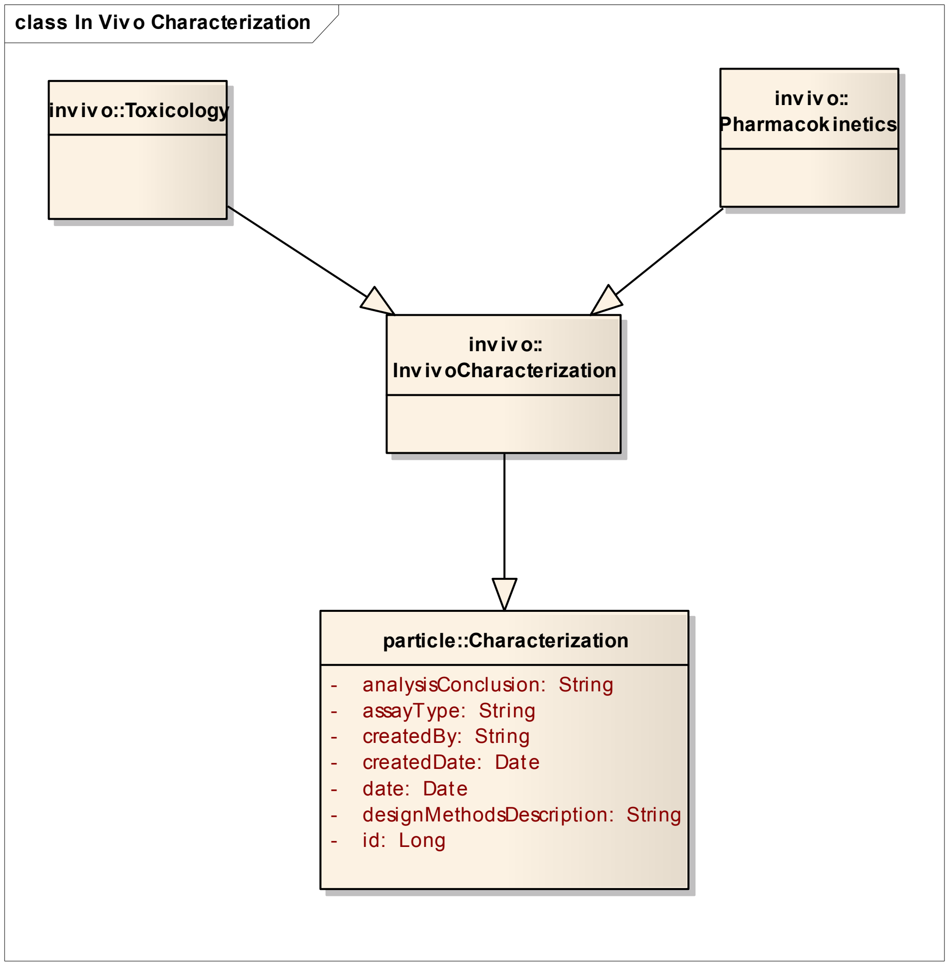


Figure 3-8: In Vivo Characterization (nano-OM)

3.2 DATA MODEL

The caNanoLab data model is closely coupled with the caNanoLab domain model. Tables are designed to work with the nano-OM through the Hibernate ORMs. Following the Hiberate ORM principles, there is a close mapping (almost one-to-one mapping) between an object and a table, as well as a close mapping between object relationships (e.g. association, inheritance) and referential integrity table relationships. Hibernate ORM offers different strategies for inheritance mappings. The strategies used in the caNanoLab data model design have been limited to those supported by the caCORE SDK. The caAdapter tool has been used to assist with object-relational mappings and annotations of the nano-OM using the Enterprise Architect tool.

The caNanoLab data model is categorized into two sub-models: nanoparticle data model and CSM data model. The following subsections provide details on each sub-model. Because there are a large number of database objects in the nanoparticle data model, it is difficult to fit them all in one ER diagram. The ER diagram is broken into several diagrams following the same sectioning scheme as in the domain model section.

3.2.1 Sample Overview

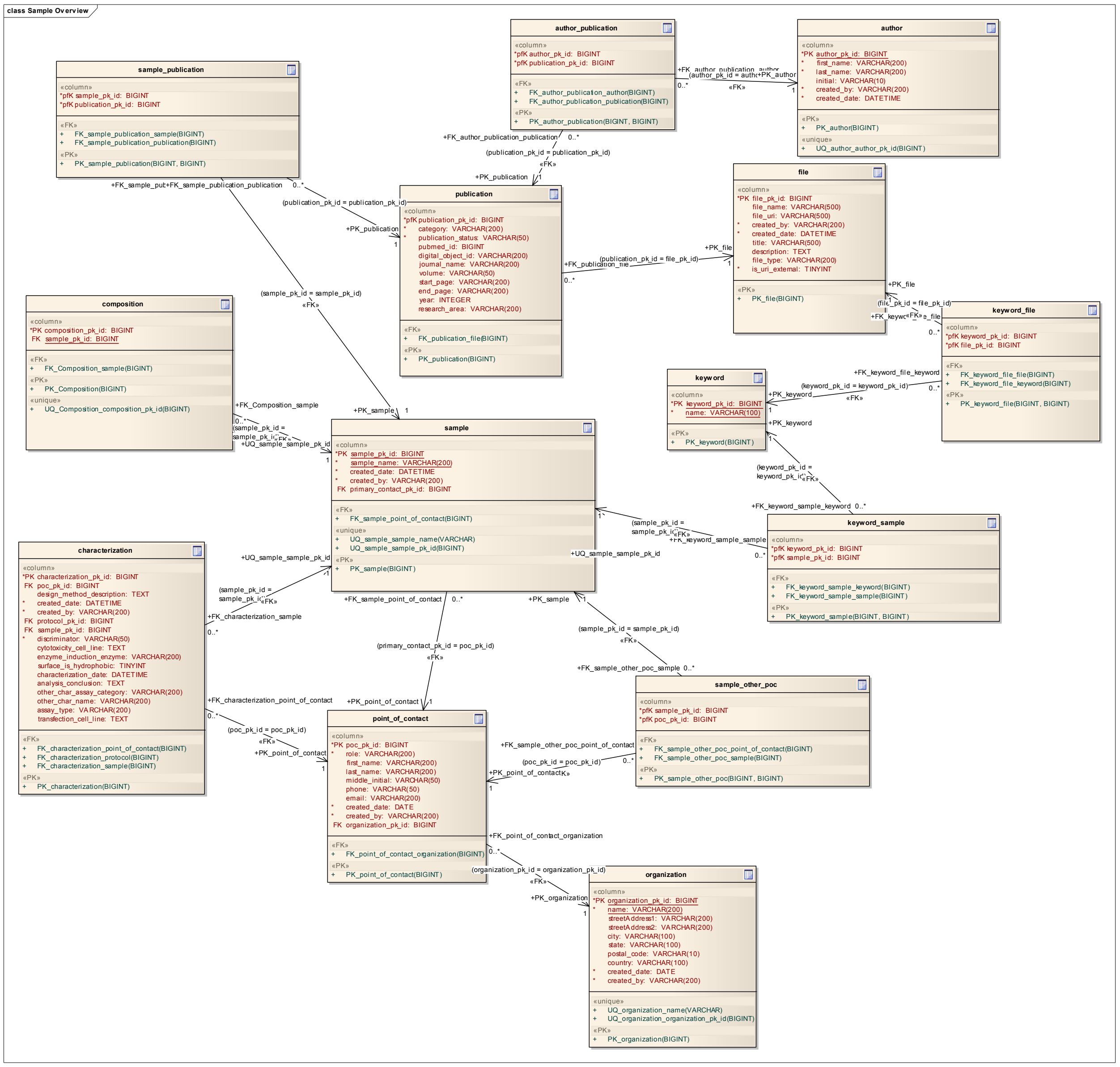


Figure 3-9: Sample Overview (Nanoparticle Data Model)

3.2.2 Composition Overview

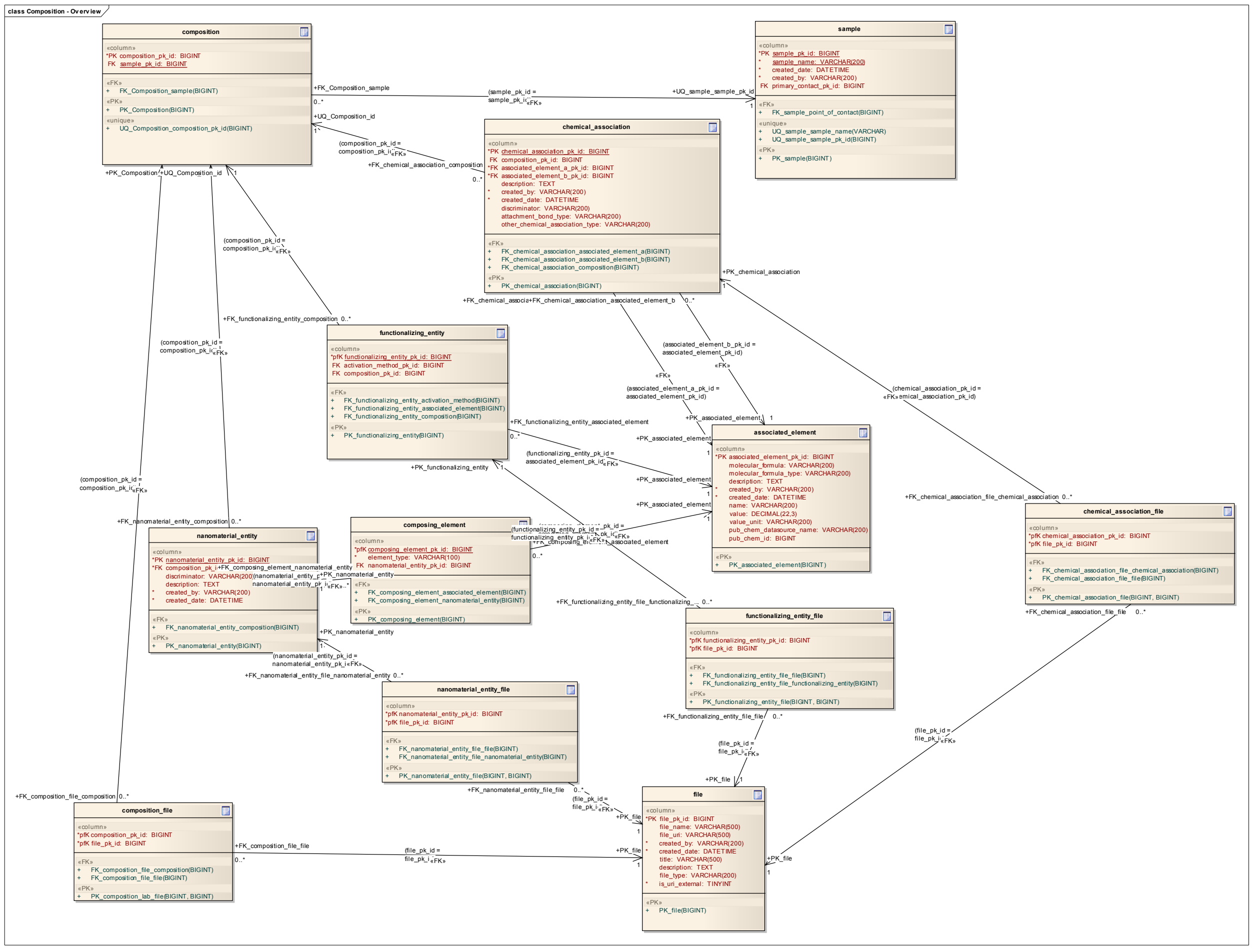

class Composition - Overview
composition
«column»
*PK composition_pk_id: BIGINT
FK sample_pk_id: BIGINT
«FK»
+ FK_Composition_sample(BIGINT)
«PK»
+ PK_Composition(BIGINT)
«unique»
+ UQ_Composition_composition_pk_id(BIGINT)
sample
«column»
*PK sample_pk_id: BIGINT
* sample_name: VARCHAR(200)
* created_date: DATETIME
* created_by: VARCHAR(200)
FK primary_contact_pk_id: BIGINT
«FK»
+ FK_sample_point_of_contact(BIGINT)
«unique»
+ UQ_sample_sample_name(VARCHAR)
+ UQ_sample_sample_pk_id(BIGINT)
«PK»
+ PK_sample(BIGINT)
chemical_association
«column»
*PK chemical_association_pk_id: BIGINT
FK composition_pk_id: BIGINT
*FK associated_element_a_pk_id: BIGINT
*FK associated_element_b_pk_id: BIGINT
description: TEXT
* created_by: VARCHAR(200)
* created_date: DATETIME
discriminator: VARCHAR(200)
attachment_bond_type: VARCHAR(200)
other_chemical_association_type: VARCHAR(200)
«FK»
+ FK_chemical_association_associated_element_a(BIGINT)
+ FK_chemical_association_associated_element_b(BIGINT)
+ FK_chemical_association_composition(BIGINT)
«PK»
+ PK_chemical_association(BIGINT)
functionalizing_entity
«column»
*pfK functionalizing_entity_pk_id: BIGINT
FK activation_method_pk_id: BIGINT
FK composition_pk_id: BIGINT
«FK»
+ FK_functionalizing_entity_activation_method(BIGINT)
+ FK_functionalizing_entity_associated_element(BIGINT)
+ FK_functionalizing_entity_composition(BIGINT)
«PK»
+ PK_functionalizing_entity(BIGINT)
associated_element
«column»
*PK associated_element_pk_id: BIGINT
molecular_formula: VARCHAR(200)
molecular_formula_type: VARCHAR(200)
description: TEXT
* created_by: VARCHAR(200)
* created_date: DATETIME
name: VARCHAR(200)
value: DECIMAL(22,3)
value_unit: VARCHAR(200)
pub_chem_datasource_name: VARCHAR(200)
pub_chem_id: BIGINT
«PK»
+ PK_associated_element(BIGINT)
nanomaterial_entity
«column»
*PK nanomaterial_entity_pk_id: BIGINT
*FK composition_pk_id: BIGINT
discriminator: VARCHAR(200)
description: TEXT
* created_by: VARCHAR(200)
* created_date: DATETIME
«FK»
+ FK_nanomaterial_entity_composition(BIGINT)
«PK»
+ PK_nanomaterial_entity(BIGINT)
composing_element
«column»
*pfK composing_element_pk_id: BIGINT
* element_type: VARCHAR(100)
FK nanomaterial_entity_pk_id: BIGINT
«FK»
+ FK_composing_element_associated_element(BIGINT)
+ FK_composing_element_nanomaterial_entity(BIGINT)
«PK»
+ PK_composing_element(BIGINT)
chemical_association_file
«column»
*pfK chemical_association_pk_id: BIGINT
*pfK file_pk_id: BIGINT
«FK»
+ FK_chemical_association_file_chemical_association(BIGINT)
+ FK_chemical_association_file_file(BIGINT)
«PK»
+ PK_chemical_association_file(BIGINT, BIGINT)
functionalizing_entity_file
«column»
*pfK functionalizing_entity_pk_id: BIGINT
*pfK file_pk_id: BIGINT
«FK»
+ FK_functionalizing_entity_file_file(BIGINT)
+ FK_functionalizing_entity_file_functionalizing_entity(BIGINT)
«PK»
+ PK_functionalizing_entity_file(BIGINT, BIGINT)
nanomaterial_entity_file
«column»
*pfK nanomaterial_entity_pk_id: BIGINT
*pfK file_pk_id: BIGINT
«FK»
+ FK_nanomaterial_entity_file_file(BIGINT)
+ FK_nanomaterial_entity_file_nanomaterial_entity(BIGINT)
«PK»
+ PK_nanomaterial_entity_file(BIGINT, BIGINT)
composition_file
«column»
*pfK composition_pk_id: BIGINT
*pfK file_pk_id: BIGINT
«FK»
+ FK_composition_file_composition(BIGINT)
+ FK_composition_file_file(BIGINT)
«PK»
+ PK_composition_lab_file(BIGINT, BIGINT)
file
«column»
*PK file_pk_id: BIGINT
file_name: VARCHAR(500)
file_uri: VARCHAR(500)
* created_by: VARCHAR(200)
* created_date: DATETIME
title: VARCHAR(500)
description: TEXT
file_type: VARCHAR(200)
* is_uri_external: TINYINT
«PK»
+ PK_file(BIGINT)
+FK_Composition_sample 0..*
(sample_pk_id = sample_pk_id) «FK»
+UQ_sample_sample_pk_id 1
+UQ_Composition_id 1
(composition_pk_id = composition_pk_id) «FK»
+FK_chemical_association_composition 0..*
+PK_Composition
+UQ_Composition_id 1
+FK_functionalizing_entity_composition 0..*
+FK_nanomaterial_entity_composition 0..*
+FK_composition_file_composition 0..*
+PK_chemical_association 1
+FK_chemical_association_associated_element_a
+FK_chemical_association_associated_element_b 0..*
(associated_element_a_pk_id = associated_element_pk_id) «FK»
(associated_element_b_pk_id = associated_element_pk_id) «FK»
+PK_associated_element 1
(chemical_association_pk_id = chemical_association_pk_id) «FK»
+FK_chemical_association_file_chemical_association 0..*
+FK_functionalizing_entity_associated_element 0..*
(functionalizing_entity_pk_id = associated_element_pk_id) «FK»
+PK_functionalizing_entity 1
(functionalizing_entity_pk_id = functionalizing_entity_pk_id) «FK»
(composing_element_pk_id = associated_element_pk_id) «FK»
+FK_composing_element_associated_element 0..*
+FK_composing_element_nanomaterial_entity 0..*
(nanomaterial_entity_pk_id = nanomaterial_entity_pk_id) «FK»
+PK_nanomaterial_entity 1
+FK_nanomaterial_entity_file_nanomaterial_entity 0..*
+FK_functionalizing_entity_file_functionalizing_... 0..*
+FK_functionalizing_entity_file_file 0..*
+FK_nanomaterial_entity_file_file 0..*
+FK_chemical_association_file_file 0..*
+FK_composition_file_file 0..*
(file_pk_id = file_pk_id) «FK»
+PK_file 1

Figure 3-10: Composition Overview (Nanoparticle Data Model)

3.2.2.1 *Composition – Nanomaterial Entity*

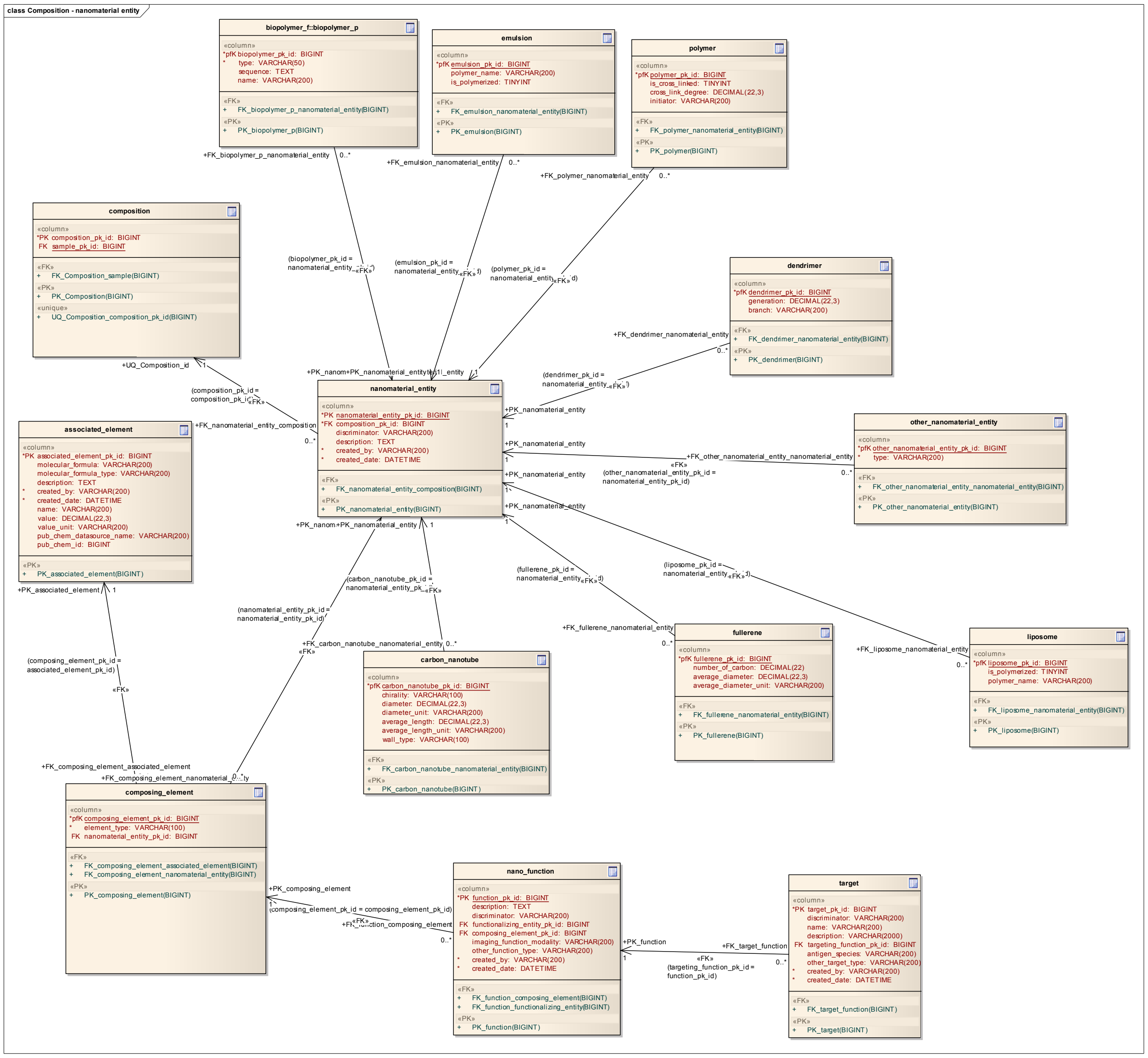

class Composition - nanomaterial entity
biopolymer_f::biopolymer_p
«column»
*pfK biopolymer_pk_id: BIGINT
* type: VARCHAR(50)
sequence: TEXT
name: VARCHAR(200)
«FK»
+ FK_biopolymer_p_nanomaterial_entity(BIGINT)
«PK»
+ PK_biopolymer_p(BIGINT)
emulsion
«column»
*pfK emulsion_pk_id: BIGINT
polymer_name: VARCHAR(200)
is_polymerized: TINYINT
«FK»
+ FK_emulsion_nanomaterial_entity(BIGINT)
«PK»
+ PK_emulsion(BIGINT)
polymer
«column»
*pfK polymer_pk_id: BIGINT
is_cross_linked: TINYINT
cross_link_degree: DECIMAL(22,3)
initiator: VARCHAR(200)
«FK»
+ FK_polymer_nanomaterial_entity(BIGINT)
«PK»
+ PK_polymer(BIGINT)
+FK_biopolymer_p_nanomaterial_entity 0..*
+FK_emulsion_nanomaterial_entity 0..*
+FK_polymer_nanomaterial_entity 0..*
composition
«column»
*PK composition_pk_id: BIGINT
FK sample_pk_id: BIGINT
«FK»
+ FK_Composition_sample(BIGINT)
«PK»
+ PK_Composition(BIGINT)
«unique»
+ UQ_Composition_composition_pk_id(BIGINT)
(biopolymer_pk_id = nanomaterial_entity «FK»)
(emulsion_pk_id = nanomaterial_entity «FK»)
(polymer_pk_id = nanomaterial_entity «FK» d)
dendrimer
«column»
*pfK dendrimer_pk_id: BIGINT
generation: DECIMAL(22,3)
branch: VARCHAR(200)
«FK»
+ FK_dendrimer_nanomaterial_entity(BIGINT)
«PK»
+ PK_dendrimer(BIGINT)
+FK_dendrimer_nanomaterial_entity 0..*
+UQ_Composition_id 1
+PK_nanom+PK_nanomaterial_entityte1l_entity 1
(dendrimer_pk_id = nanomaterial_entity «FK»)
(composition_pk_id = composition_pk_id «FK»)
nanomaterial_entity
«column»
*PK nanomaterial_entity_pk_id: BIGINT
*FK composition_pk_id: BIGINT
discriminator: VARCHAR(200)
description: TEXT
* created_by: VARCHAR(200)
* created_date: DATETIME
«FK»
+ FK_nanomaterial_entity_composition(BIGINT)
«PK»
+ PK_nanomaterial_entity(BIGINT)
+FK_nanomaterial_entity_composition 0..*
+PK_nanomaterial_entity 1
+PK_nanomaterial_entity 1
+PK_nanomaterial_entity 1
+PK_nanomaterial_entity 1
+FK_other_nanomaterial_entity_nanomaterial_entity 0..*
«FK» (other_nanomaterial_entity_pk_id = nanomaterial_entity_pk_id)
other_nanomaterial_entity
«column»
*pfK other_nanomaterial_entity_pk_id: BIGINT
* type: VARCHAR(200)
«FK»
+ FK_other_nanomaterial_entity_nanomaterial_entity(BIGINT)
«PK»
+ PK_other_nanomaterial_entity(BIGINT)
associated_element
«column»
*PK associated_element_pk_id: BIGINT
molecular_formula: VARCHAR(200)
molecular_formula_type: VARCHAR(200)
description: TEXT
* created_by: VARCHAR(200)
* created_date: DATETIME
name: VARCHAR(200)
value: DECIMAL(22,3)
value_unit: VARCHAR(200)
pub_chem_datasource_name: VARCHAR(200)
pub_chem_id: BIGINT
«PK»
+ PK_associated_element(BIGINT)
+PK_associated_element 1
+PK_nanom+PK_nanomaterial_entity 1
(carbon_nanotube_pk_id = nanomaterial_entity_pk «FK»)
(nanomaterial_entity_pk_id = nanomaterial_entity_pk_id)
(fullerene_pk_id = nanomaterial_entity «FK» d)
(liposome_pk_id = nanomaterial_entity «FK» d)
+FK_carbon_nanotube_nanomaterial_entity 0..*
«FK»
+FK_fullerene_nanomaterial_entity 0..*
+FK_liposome_nanomaterial_entity 0..*
(composing_element_pk_id = associated_element_pk_id)
«FK»
carbon_nanotube
«column»
*pfK carbon_nanotube_pk_id: BIGINT
chirality: VARCHAR(100)
diameter: DECIMAL(22,3)
diameter_unit: VARCHAR(200)
average_length: DECIMAL(22,3)
average_length_unit: VARCHAR(200)
wall_type: VARCHAR(100)
«FK»
+ FK_carbon_nanotube_nanomaterial_entity(BIGINT)
«PK»
+ PK_carbon_nanotube(BIGINT)
fullerene
«column»
*pfK fullerene_pk_id: BIGINT
number_of_carbon: DECIMAL(22)
average_diameter: DECIMAL(22,3)
average_diameter_unit: VARCHAR(200)
«FK»
+ FK_fullerene_nanomaterial_entity(BIGINT)
«PK»
+ PK_fullerene(BIGINT)
liposome
«column»
*pfK liposome_pk_id: BIGINT
is_polymerized: TINYINT
polymer_name: VARCHAR(200)
«FK»
+ FK_liposome_nanomaterial_entity(BIGINT)
«PK»
+ PK_liposome(BIGINT)
+FK_composing_element_associated_element
+FK_composing_element_nanomaterial_entity 0..*
composing_element
«column»
*pfK composing_element_pk_id: BIGINT
* element_type: VARCHAR(100)
FK nanomaterial_entity_pk_id: BIGINT
«FK»
+ FK_composing_element_associated_element(BIGINT)
+ FK_composing_element_nanomaterial_entity(BIGINT)
«PK»
+ PK_composing_element(BIGINT)
+PK_composing_element 1
(composing_element_pk_id = composing_element_pk_id)
«FK»
+FK_function_composing_element 0..*
nano_function
«column»
*PK function_pk_id: BIGINT
description: TEXT
discriminator: VARCHAR(200)
FK functionalizing_entity_pk_id: BIGINT
FK composing_element_pk_id: BIGINT
imaging_function_modality: VARCHAR(200)
other_function_type: VARCHAR(200)
* created_by: VARCHAR(200)
* created_date: DATETIME
«FK»
+ FK_function_composing_element(BIGINT)
+ FK_function_functionalizing_entity(BIGINT)
«PK»
+ PK_function(BIGINT)
+PK_function 1
+FK_target_function 0..*
«FK» (targeting_function_pk_id = function_pk_id)
target
«column»
*PK target_pk_id: BIGINT
discriminator: VARCHAR(200)
name: VARCHAR(200)
description: VARCHAR(2000)
FK targeting_function_pk_id: BIGINT
antigen_species: VARCHAR(200)
other_target_type: VARCHAR(200)
* created_by: VARCHAR(200)
* created_date: DATETIME
«FK»
+ FK_target_function(BIGINT)
«PK»
+ PK_target(BIGINT)

Figure 3-11: Composition – Nanomaterial Entity (Nanoparticle Data Model)

3.2.2.2 *Composition – Functionalizing Entity*

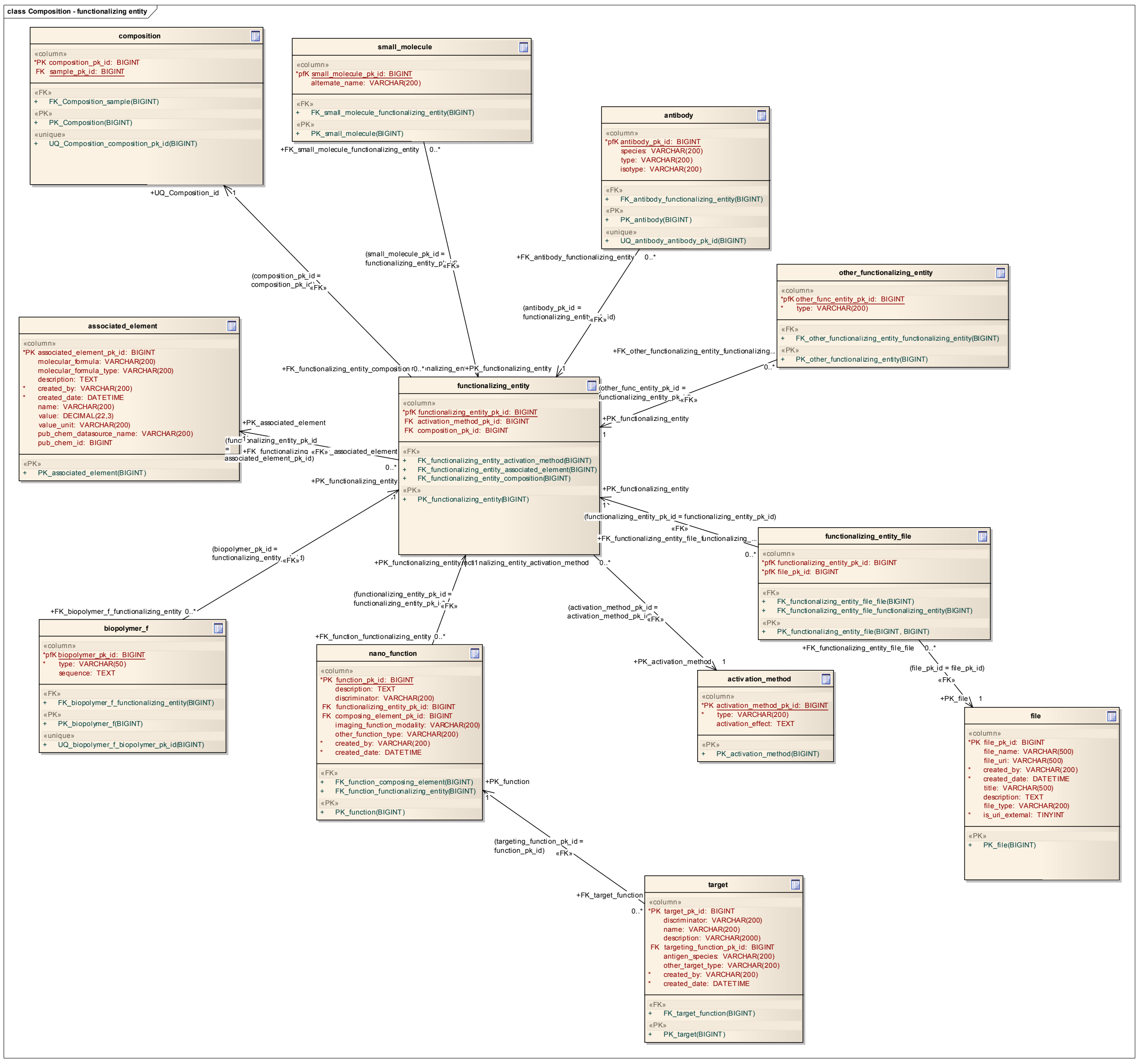

class Composition - functionalizing entity
composition
«column»
*PK composition_pk_id: BIGINT
FK sample_pk_id: BIGINT
«FK»
+ FK_Composition_sample(BIGINT)
«PK»
+ PK_Composition(BIGINT)
«unique»
+ UQ_Composition_composition_pk_id(BIGINT)
small_molecule
«column»
*pfK small_molecule_pk_id: BIGINT
alternate_name: VARCHAR(200)
«FK»
+ FK_small_molecule_functionalizing_entity(BIGINT)
«PK»
+ PK_small_molecule(BIGINT)
antibody
«column»
*pfK antibody_pk_id: BIGINT
species: VARCHAR(200)
type: VARCHAR(200)
isotype: VARCHAR(200)
«FK»
+ FK_antibody_functionalizing_entity(BIGINT)
«PK»
+ PK_antibody(BIGINT)
«unique»
+ UQ_antibody_antibody_pk_id(BIGINT)
other_functionalizing_entity
«column»
*pfK other_func_entity_pk_id: BIGINT
* type: VARCHAR(200)
«FK»
+ FK_other_functionalizing_entity_functionalizing_entity(BIGINT)
«PK»
+ PK_other_functionalizing_entity(BIGINT)
associated_element
«column»
*PK associated_element_pk_id: BIGINT
molecular_formula: VARCHAR(200)
molecular_formula_type: VARCHAR(200)
description: TEXT
* created_by: VARCHAR(200)
* created_date: DATETIME
name: VARCHAR(200)
value: DECIMAL(22,3)
value_unit: VARCHAR(200)
pub_chem_datasource_name: VARCHAR(200)
pub_chem_id: BIGINT
«PK»
+ PK_associated_element(BIGINT)
functionalizing_entity
«column»
*pfK functionalizing_entity_pk_id: BIGINT
FK activation_method_pk_id: BIGINT
FK composition_pk_id: BIGINT
«FK»
+ FK_functionalizing_entity_activation_method(BIGINT)
+ FK_functionalizing_entity_associated_element(BIGINT)
+ FK_functionalizing_entity_composition(BIGINT)
«PK»
+ PK_functionalizing_entity(BIGINT)
functionalizing_entity_file
«column»
*pfK functionalizing_entity_pk_id: BIGINT
*pfK file_pk_id: BIGINT
«FK»
+ FK_functionalizing_entity_file_file(BIGINT)
+ FK_functionalizing_entity_file_functionalizing_entity(BIGINT)
«PK»
+ PK_functionalizing_entity_file(BIGINT, BIGINT)
biopolymer_f
«column»
*pfK biopolymer_pk_id: BIGINT
* type: VARCHAR(50)
sequence: TEXT
«FK»
+ FK_biopolymer_f_functionalizing_entity(BIGINT)
«PK»
+ PK_biopolymer_f(BIGINT)
«unique»
+ UQ_biopolymer_f_biopolymer_pk_id(BIGINT)
nano_function
«column»
*PK function_pk_id: BIGINT
description: TEXT
discriminator: VARCHAR(200)
FK functionalizing_entity_pk_id: BIGINT
FK composing_element_pk_id: BIGINT
imaging_function_modality: VARCHAR(200)
other_function_type: VARCHAR(200)
* created_by: VARCHAR(200)
* created_date: DATETIME
«FK»
+ FK_function_composing_element(BIGINT)
+ FK_function_functionalizing_entity(BIGINT)
«PK»
+ PK_function(BIGINT)
activation_method
«column»
*PK activation_method_pk_id: BIGINT
* type: VARCHAR(200)
activation_effect: TEXT
«PK»
+ PK_activation_method(BIGINT)
file
«column»
*PK file_pk_id: BIGINT
file_name: VARCHAR(500)
file_uri: VARCHAR(500)
* created_by: VARCHAR(200)
* created_date: DATETIME
title: VARCHAR(500)
description: TEXT
file_type: VARCHAR(200)
* is_uri_external: TINYINT
«PK»
+ PK_file(BIGINT)
target
«column»
*PK target_pk_id: BIGINT
discriminator: VARCHAR(200)
name: VARCHAR(200)
description: VARCHAR(2000)
FK targeting_function_pk_id: BIGINT
antigen_species: VARCHAR(200)
other_target_type: VARCHAR(200)
* created_by: VARCHAR(200)
* created_date: DATETIME
«FK»
+ FK_target_function(BIGINT)
«PK»
+ PK_target(BIGINT)
+UQ_Composition_id 1
(composition_pk_id = composition_pk_id) «FK»
+FK_functionalizing_entity_composition 0..*
+FK_small_molecule_functionalizing_entity 0..*
(small_molecule_pk_id = functionalizing_entity_pk_id) «FK»
+PK_functionalizing_entity 1
+FK_antibody_functionalizing_entity 0..*
(antibody_pk_id = functionalizing_entity_pk_id) «FK»
+FK_other_functionalizing_entity_functionalizing... 0..*
(other_func_entity_pk_id = functionalizing_entity_pk_id) «FK»
+PK_functionalizing_entity 1
+PK_associated_element 1
(functionalizing_entity_pk_id = associated_element_pk_id) «FK»
+FK_functionalizing_entity_associated_element 0..*
+PK_functionalizing_entity 1
(biopolymer_pk_id = functionalizing_entity_pk_id) «FK»
+FK_biopolymer_f_functionalizing_entity 0..*
+PK_functionalizing_entity 1
(functionalizing_entity_pk_id = functionalizing_entity_pk_id) «FK»
+FK_functionalizing_entity_file_functionalizing... 0..*
+FK_functionalizing_entity_activation_method 0..*
(functionalizing_entity_pk_id = functionalizing_entity_pk_id) «FK»
+FK_function_functionalizing_entity 0..*
(activation_method_pk_id = activation_method_pk_id) «FK»
+PK_activation_method 1
+FK_functionalizing_entity_file_file 0..*
(file_pk_id = file_pk_id) «FK»
+PK_file 1
+PK_function 1
(targeting_function_pk_id = function_pk_id) «FK»
+FK_target_function 0..*

Figure 3-12: Composition – Functionalizing Entity (Nanoparticle Data Model)

3.2.3 Characterization View

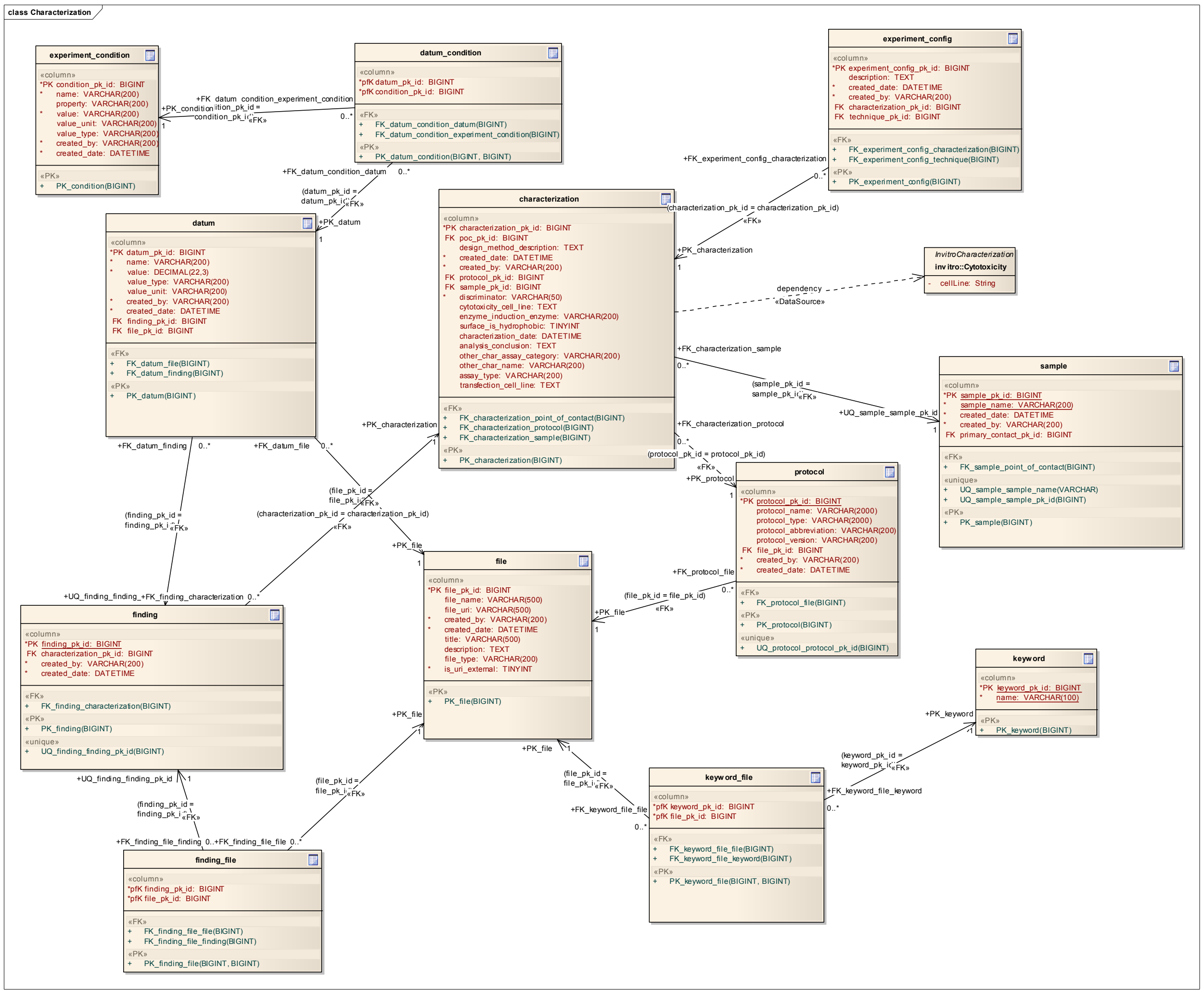

class Characterization
experiment_condition
«column»
*PK condition_pk_id: BIGINT
* name: VARCHAR(200)
property: VARCHAR(200)
* value: VARCHAR(200)
value_unit: VARCHAR(200)
value_type: VARCHAR(200)
* created_by: VARCHAR(200)
* created_date: DATETIME
«PK»
+ PK_condition(BIGINT)
datum_condition
«column»
*pfK datum_pk_id: BIGINT
*pfK condition_pk_id: BIGINT
«FK»
+ FK_datum_condition_datum(BIGINT)
+ FK_datum_condition_experiment_condition(BIGINT)
«PK»
+ PK_datum_condition(BIGINT, BIGINT)
+FK_datum_condition_experiment_condition
+PK_condition
(condition_pk_id = condition_pk_id)
«FK»
1
0..*
+FK_datum_condition_datum
0..*
(datum_pk_id = datum_pk_id)
«FK»
+PK_datum
1
experiment_config
«column»
*PK experiment_config_pk_id: BIGINT
description: TEXT
* created_date: DATETIME
* created_by: VARCHAR(200)
FK characterization_pk_id: BIGINT
FK technique_pk_id: BIGINT
«FK»
+ FK_experiment_config_characterization(BIGINT)
+ FK_experiment_config_technique(BIGINT)
«PK»
+ PK_experiment_config(BIGINT)
+FK_experiment_config_characterization
0..*
(characterization_pk_id = characterization_pk_id)
«FK»
+PK_characterization
1
characterization
«column»
*PK characterization_pk_id: BIGINT
FK poc_pk_id: BIGINT
design_method_description: TEXT
* created_date: DATETIME
* created_by: VARCHAR(200)
FK protocol_pk_id: BIGINT
FK sample_pk_id: BIGINT
* discriminator: VARCHAR(50)
cytotoxicity_cell_line: TEXT
enzyme_induction_enzyme: VARCHAR(200)
surface_is_hydrophobic: TINYINT
characterization_date: DATETIME
analysis_conclusion: TEXT
other_char_assay_category: VARCHAR(200)
other_char_name: VARCHAR(200)
assay_type: VARCHAR(200)
transfection_cell_line: TEXT
«FK»
+ FK_characterization_point_of_contact(BIGINT)
+ FK_characterization_protocol(BIGINT)
+ FK_characterization_sample(BIGINT)
«PK»
+ PK_characterization(BIGINT)
InvitroCharacterization
invitro::Cytotoxicity
- cellLine: String
dependency
«DataSource»
datum
«column»
*PK datum_pk_id: BIGINT
* name: VARCHAR(200)
* value: DECIMAL(22,3)
value_type: VARCHAR(200)
value_unit: VARCHAR(200)
* created_by: VARCHAR(200)
* created_date: DATETIME
FK finding_pk_id: BIGINT
FK file_pk_id: BIGINT
«FK»
+ FK_datum_file(BIGINT)
+ FK_datum_finding(BIGINT)
«PK»
+ PK_datum(BIGINT)
+FK_characterization_sample
0..*
(sample_pk_id = sample_pk_id)
«FK»
+UQ_sample_sample_pk_id
1
sample
«column»
*PK sample_pk_id: BIGINT
* sample_name: VARCHAR(200)
* created_date: DATETIME
* created_by: VARCHAR(200)
FK primary_contact_pk_id: BIGINT
«FK»
+ FK_sample_point_of_contact(BIGINT)
«unique»
+ UQ_sample_sample_name(VARCHAR)
+ UQ_sample_sample_pk_id(BIGINT)
«PK»
+ PK_sample(BIGINT)
+FK_characterization_protocol
0..*
(protocol_pk_id = protocol_pk_id)
«FK»
+PK_protocol
1
protocol
«column»
*PK protocol_pk_id: BIGINT
protocol_name: VARCHAR(2000)
protocol_type: VARCHAR(2000)
protocol_abbreviation: VARCHAR(200)
protocol_version: VARCHAR(200)
FK file_pk_id: BIGINT
* created_by: VARCHAR(200)
* created_date: DATETIME
«FK»
+ FK_protocol_file(BIGINT)
«PK»
+ PK_protocol(BIGINT)
«unique»
+ UQ_protocol_protocol_pk_id(BIGINT)
+FK_protocol_file
0..*
(file_pk_id = file_pk_id)
«FK»
+PK_file
1
+FK_datum_finding
0..*
+FK_datum_file
0..*
+PK_characterization
1
(file_pk_id = file_pk_id)
«FK»
(characterization_pk_id = characterization_pk_id)
«FK»
(finding_pk_id = finding_pk_id)
«FK»
+PK_file
1
+UQ_finding_finding_+FK_finding_characterization 0..*
finding
«column»
*PK finding_pk_id: BIGINT
FK characterization_pk_id: BIGINT
* created_by: VARCHAR(200)
* created_date: DATETIME
«FK»
+ FK_finding_characterization(BIGINT)
«PK»
+ PK_finding(BIGINT)
«unique»
+ UQ_finding_finding_pk_id(BIGINT)
file
«column»
*PK file_pk_id: BIGINT
file_name: VARCHAR(500)
file_uri: VARCHAR(500)
* created_by: VARCHAR(200)
* created_date: DATETIME
title: VARCHAR(500)
description: TEXT
file_type: VARCHAR(200)
* is_uri_external: TINYINT
«PK»
+ PK_file(BIGINT)
+PK_file
1
+UQ_finding_finding_pk_id
1
(finding_pk_id = finding_pk_id)
«FK»
(file_pk_id = file_pk_id)
«FK»
+FK_finding_file_finding 0..+FK_finding_file_file 0..*
finding_file
«column»
*pfK finding_pk_id: BIGINT
*pfK file_pk_id: BIGINT
«FK»
+ FK_finding_file_file(BIGINT)
+ FK_finding_file_finding(BIGINT)
«PK»
+ PK_finding_file(BIGINT, BIGINT)
+PK_file
1
(file_pk_id = file_pk_id)
«FK»
+FK_keyword_file_file
0..*
keyword_file
«column»
*pfK keyword_pk_id: BIGINT
*pfK file_pk_id: BIGINT
«FK»
+ FK_keyword_file_file(BIGINT)
+ FK_keyword_file_keyword(BIGINT)
«PK»
+ PK_keyword_file(BIGINT, BIGINT)
+FK_keyword_file_keyword
0..*
(keyword_pk_id = keyword_pk_id)
«FK»
+PK_keyword
1
keyword
«column»
*PK keyword_pk_id: BIGINT
* name: VARCHAR(100)
«PK»
+ PK_keyword(BIGINT)

Figure 3-13: Characterization View (Nanoparticle Data Model)

3.2.3.1 *Characterization – Physico-chemical Characterization*

The inheritance structure in the physico-chemical characterization portion of the nano-OM has been implemented using both the table-per-subclass and the table-per-hierarchy strategies of the Hibernate ORM, such that a single table `characterization` (along with the `discriminator` column) is used to capture information embedded in child classes of PhysicoChemicalCharacterization that don't have attributes of their own, and a separate table is used to capture child classes that have their own specific attributes.

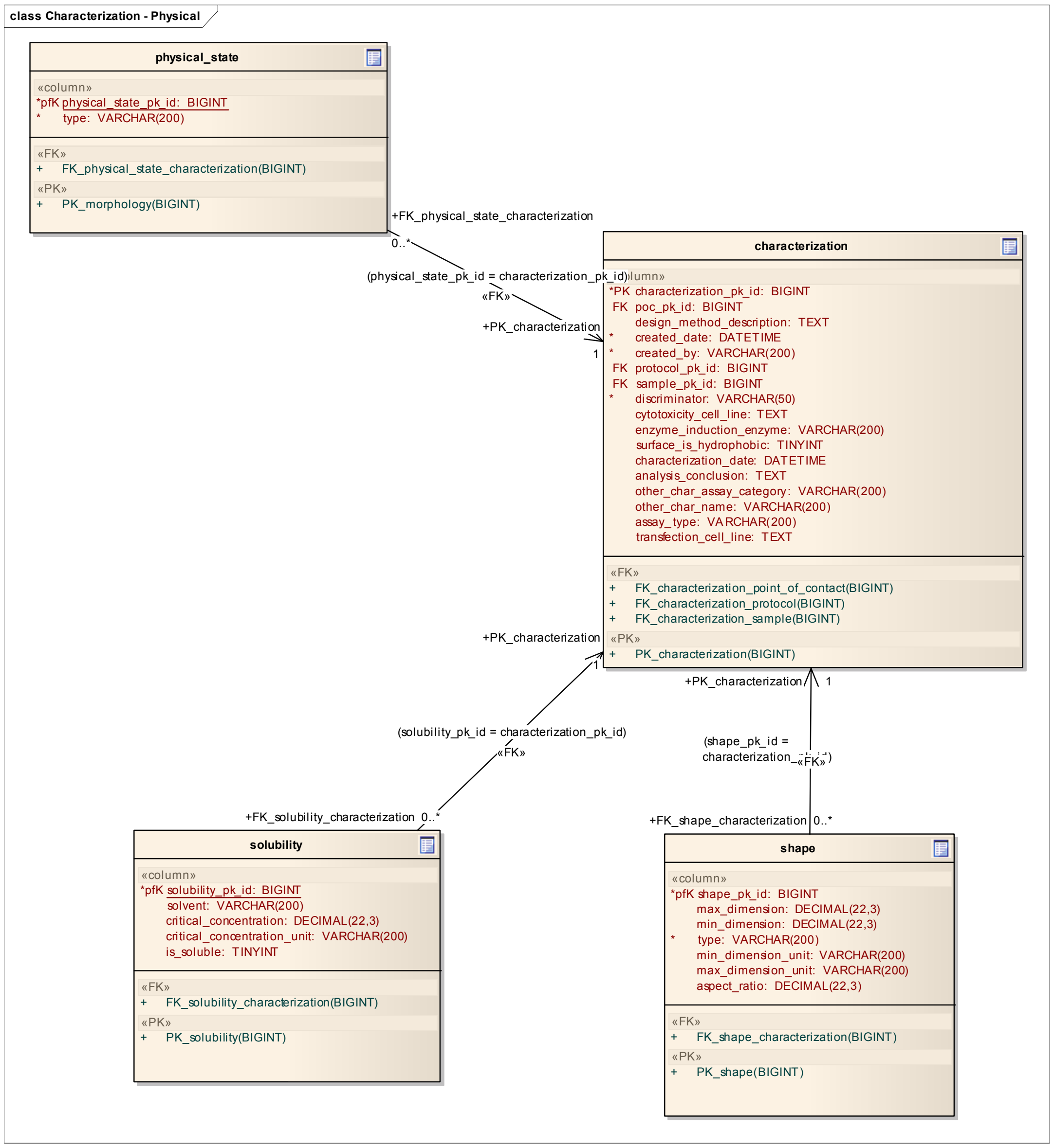

class Characterization - Physical
physical_state
«column»
*pfK physical_state_pk_id: BIGINT
* type: VARCHAR(200)
«FK»
+ FK_physical_state_characterization(BIGINT)
«PK»
+ PK_morphology(BIGINT)
+FK_physical_state_characterization
0..*
(physical_state_pk_id = characterization_pk_id)
«FK»
+PK_characterization
1
characterization
«column»
*PK characterization_pk_id: BIGINT
FK poc_pk_id: BIGINT
design_method_description: TEXT
* created_date: DATETIME
* created_by: VARCHAR(200)
FK protocol_pk_id: BIGINT
FK sample_pk_id: BIGINT
* discriminator: VARCHAR(50)
cytotoxicity_cell_line: TEXT
enzyme_induction_enzyme: VARCHAR(200)
surface_is_hydrophobic: TINYINT
characterization_date: DATETIME
analysis_conclusion: TEXT
other_char_assay_category: VARCHAR(200)
other_char_name: VARCHAR(200)
assay_type: VARCHAR(200)
transfection_cell_line: TEXT
«FK»
+ FK_characterization_point_of_contact(BIGINT)
+ FK_characterization_protocol(BIGINT)
+ FK_characterization_sample(BIGINT)
«PK»
+ PK_characterization(BIGINT)
+PK_characterization
1
+PK_characterization
1
(solubility_pk_id = characterization_pk_id)
«FK»
+FK_solubility_characterization 0..*
(shape_pk_id = characterization_
«FK»
+FK_shape_characterization 0..*
solubility
«column»
*pfK solubility_pk_id: BIGINT
solvent: VARCHAR(200)
critical_concentration: DECIMAL(22,3)
critical_concentration_unit: VARCHAR(200)
is_soluble: TINYINT
«FK»
+ FK_solubility_characterization(BIGINT)
«PK»
+ PK_solubility(BIGINT)
shape
«column»
*pfK shape_pk_id: BIGINT
max_dimension: DECIMAL(22,3)
min_dimension: DECIMAL(22,3)
* type: VARCHAR(200)
min_dimension_unit: VARCHAR(200)
max_dimension_unit: VARCHAR(200)
aspect_ratio: DECIMAL(22,3)
«FK»
+ FK_shape_characterization(BIGINT)
«PK»
+ PK_shape(BIGINT)

Figure 3-14: Characterization – PhysicoChemical Characterization View (Nanoparticle Data Model)

3.2.3.2 *Characterization – In Vitro Characterization and In Vivo Characterization*

The inheritance structure in the in vitro characterization and in vivo characterization portions of the nano-OM has been implemented using table-per-hierarchy strategy such that a single table `characterization` is used to capture information embedded all child classes of InvitroCharacterization, as well as all child classes of InvivoCharacterization through the use of the `discriminator` column.

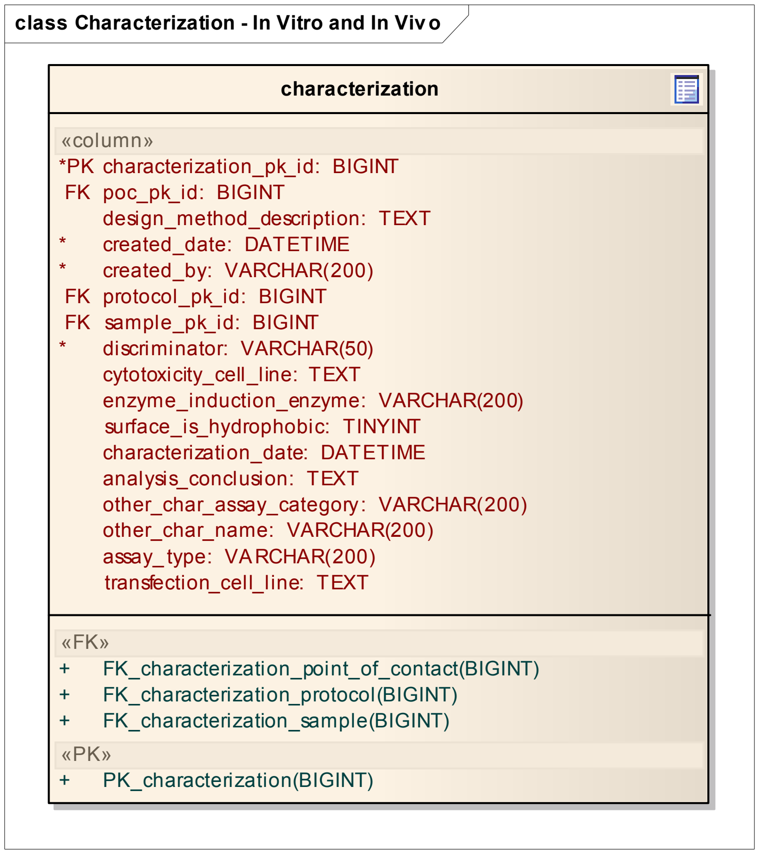
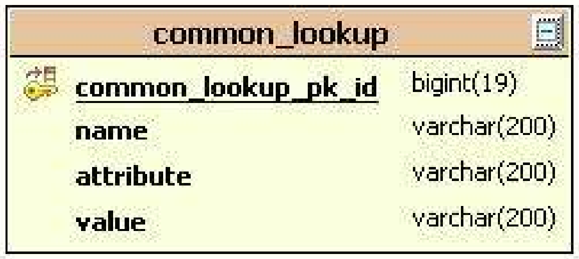

class Characterization - In Vitro and In Vivo
characterization
«column»
*PK characterization_pk_id: BIGINT
FK poc_pk_id: BIGINT
design_method_description: TEXT
* created_date: DATETIME
* created_by: VARCHAR(200)
FK protocol_pk_id: BIGINT
FK sample_pk_id: BIGINT
* discriminator: VARCHAR(50)
cytotoxicity_cell_line: TEXT
enzyme_induction_enzyme: VARCHAR(200)
surface_is_hydrophobic: TINYINT
characterization_date: DATETIME
analysis_conclusion: TEXT
other_char_assay_category: VARCHAR(200)
other_char_name: VARCHAR(200)
assay_type: VARCHAR(200)
transfection_cell_line: TEXT
«FK»
+ FK_characterization_point_of_contact(BIGINT)
+ FK_characterization_protocol(BIGINT)
+ FK_characterization_sample(BIGINT)
«PK»
+ PK_characterization(BIGINT)

Figure 3-15: Characterization – In Vitro and In Vivo Characterization View (Data Model)

3.2.4 Look Up Table

A lookup table `common_lookup`, as shown in Figure 3-15, has been created to store known controlled vocabularies that are displayed as values in the drop-down lists in the application. It is also used to store any user entered name-attribute-value trios. It is in the planning that user-entered data will be periodically reviewed and fed into the EVS to generate new EVS concepts. Please see https://gforge.nci.nih.gov/docman/view.php/69/20193/caNanoLab_1.5_common_lookup.xls for all the pre-defined controlled vocabularies stored in the `common_lookup` table.

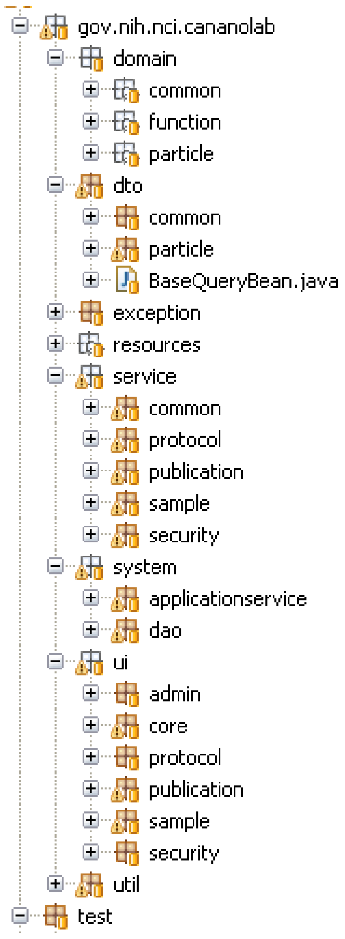


common_lookup	
common_lookup_pk_id	bigint(19)
name	varchar(200)
attribute	varchar(200)
value	varchar(200)

Figure 3-16: common_lookup Data Model

3.3 PACKAGE STRUCTURE

The caNanoLab root package is `gov.nih.nci.cananolab`. The source-code package structure is as shown in Figure 3-17. The packags are first divided according to the different tiers in the J2EE system: `domain, dto, exception, resources, service, system, ui, util`.



```
gov.nih.nci.cananolab
    domain
        common
        function
        particle
    dto
        common
        particle
        BaseQueryBean.java
    exception
    resources
    service
        common
        protocol
        publication
        sample
        security
    system
        applicationservice
        dao
    ui
        admin
        core
        protocol
        publication
        sample
        security
    util
test
```

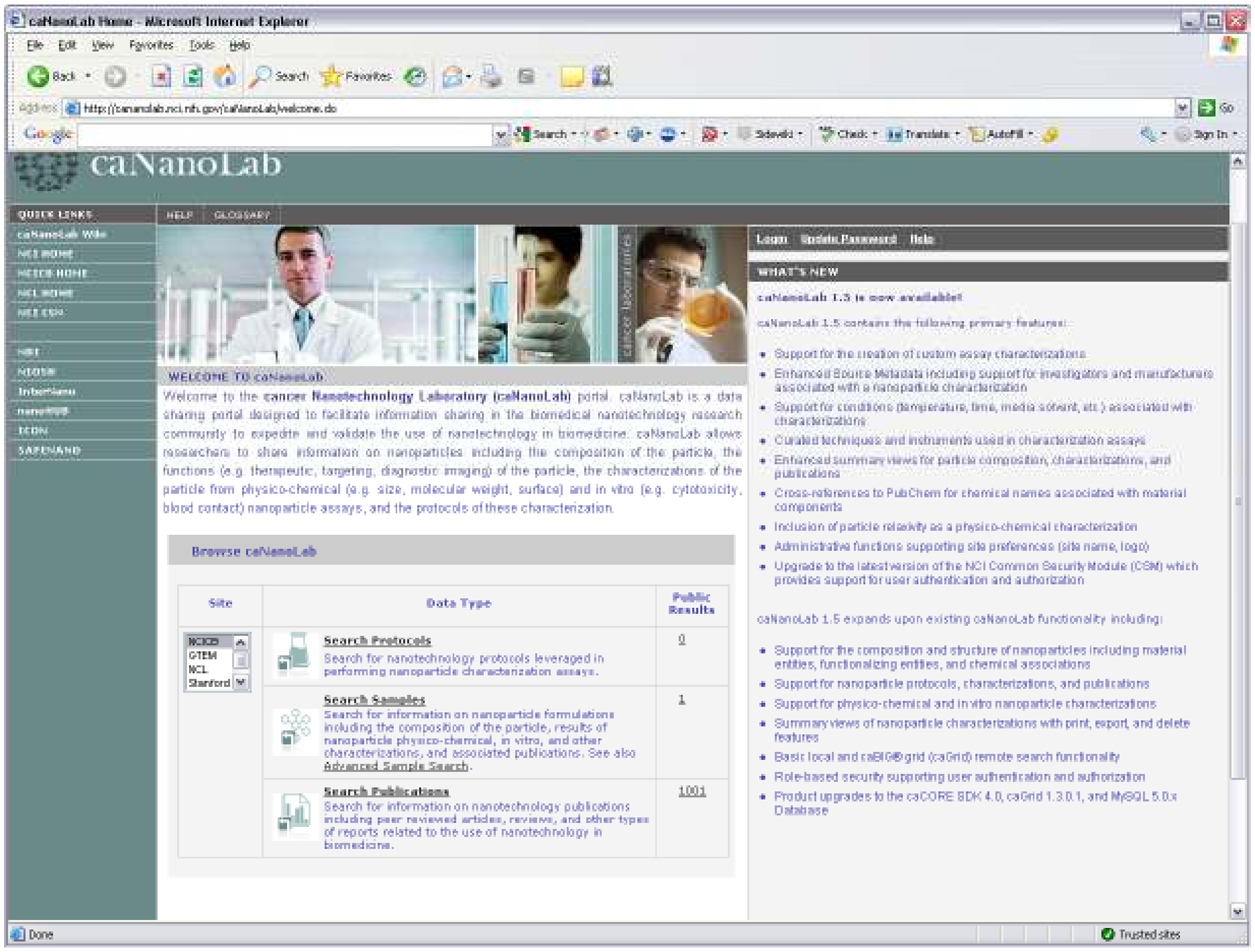
Figure 3-17: caNanoLab Source Package Structure

Each child package is further divided into sub-packages according to the different categories of nanoparticle information: common, protocol, sample, publications, and security.

3.4 USER INTERFACE

The caNanoLab user interface (UI) has been modeled based on the use cases gathered from NCL and CCNEs. Each page contains a header region containing the caNanoLab title and icon, a footer region containing links to application support, etc, a left menu region containing links to external related sites or to internal navigation links, a right content region containing contents of the caNanoLab application.

On the home page, as shown in Figure 3-18, without logging in to the system, users can seamlessly browse publicly available protocols, samples and publications available either in the local installation or in remote caNanoLab grid services.



WELCOME TO caNanoLab

Welcome to the **cancer Nanotechnology Laboratory (caNanoLab)** portal. caNanoLab is a data sharing portal designed to facilitate information sharing in the biomedical nanotechnology research community to expedite and validate the use of nanotechnology in biomedicine. caNanoLab allows researchers to share information on nanoparticles including the composition of the particle, the functions (e.g. therapeutic, targeting, diagnostic imaging) of the particle, the characterizations of the particle from physico-chemical (e.g. size, molecular weight, surface) and in vitro (e.g. cytotoxicity, blood contact) nanoparticle assays, and the protocols of these characterization.

Browse caNanoLab

Site	Data Type	Public Results
NCICB GTEM NCL Stanford	**Search Protocols** Search for nanotechnology protocols leveraged in performing nanoparticle characterization assays.	0
	Search Samples Search for information on nanoparticle formulations including the composition of the particle, results of nanoparticle physico-chemical, in vitro, and other characterizations, and associated publications. See also Advanced Sample Search.	1
	Search Publications Search for information on nanotechnology publications including peer reviewed articles, reviews, and other types of reports related to the use of nanotechnology in biomedicine.	1001

WHAT'S NEW

caNanoLab 1.5 is now available!

caNanoLab 1.5 contains the following primary features:

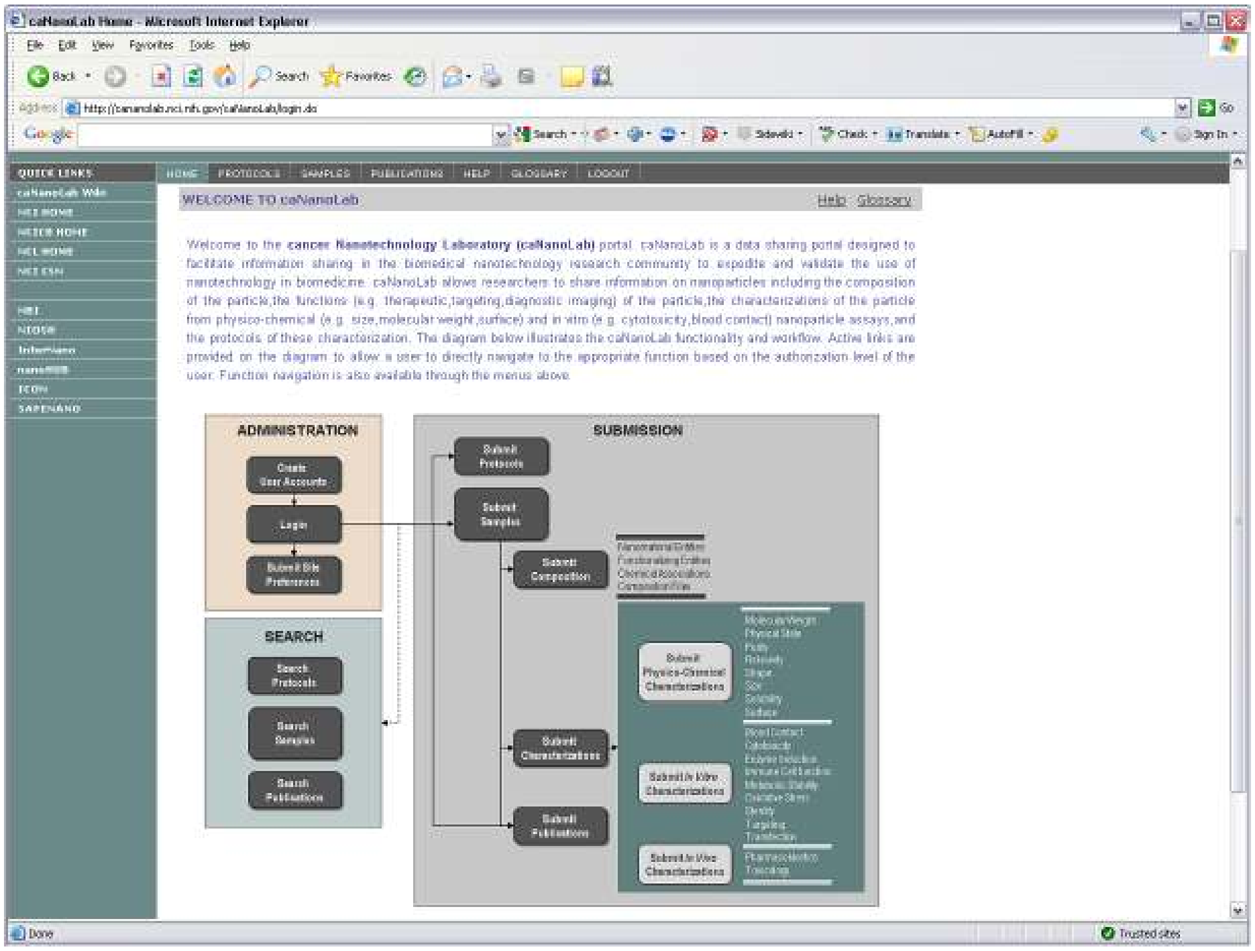
- Support for the creation of custom assay characterizations
- Enhanced Source Metadata including support for investigators and manufacturers associated with a nanoparticle characterization
- Support for conditions (temperature, time, media solvent, etc.) associated with characterizations
- Curated techniques and instruments used in characterization assays
- Enhanced summary views for particle composition, characterizations, and publications
- Cross-references to PubChem for chemical names associated with material components
- Inclusion of particle relaxivity as a physico-chemical characterization
- Administrative functions supporting site preferences (site name, logo)
- Upgrade to the latest version of the NCI Common Security Module (CSM) which provides support for user authentication and authorization

caNanoLab 1.5 expands upon existing caNanoLab functionality including:

- Support for the composition and structure of nanoparticles including material entities, functionalizing entities, and chemical associations
- Support for nanoparticle protocols, characterizations, and publications
- Support for physico-chemical and in vitro nanoparticle characterizations
- Summary views of nanoparticle characterizations with print, export, and delete features
- Basic local and caBIG® grid (caGrid) remote search functionality
- Role-based security supporting user authentication and authorization
- Product upgrades to the caCORE SDK 4.0, caGrid 1.3.0.1, and MySQL 5.0.x Database

Figure 3-18: caNanoLab Home Page

If a user decides to log into the application, at the top of the right content region, a menu bar with different tabs are displayed as shown in Figure 3-19. Different users with different access privileges can see different tabs. Please refer to section 3.6 for details on pre-defined security rules. Alternatively, users can click on the hotspots in the workflow diagram to navigate the system.



Welcome to the **cancer Nanotechnology Laboratory (caNanoLab)** portal. caNanoLab is a data sharing portal designed to facilitate information sharing in the biomedical nanotechnology research community to expedite and validate the use of nanotechnology in biomedicine. caNanoLab allows researchers to share information on nanoparticles including the composition of the particle,the functions (e.g. therapeutic,targeting,diagnostic imaging) of the particle,the characterizations of the particle from physico-chemical (e.g. size,molecular weight,surface) and in vitro (e.g. cytotoxicity,blood contact) nanoparticle assays,and the protocols of these characterization. The diagram below illustrates the caNanoLab functionality and workflow. Active links are provided on the diagram to allow a user to directly navigate to the appropriate function based on the authorization level of the user. Function navigation is also available through the menus above.

Figure 3-19: caNanoLab Web Interface Workflow

The caNanoLab web application user interface components are implemented using the Struts Framework. Struts ActionForms (declaratively defined as DynaActionForms) and action mappings are defined in the Struts configuration files under context root's `WEB-INF` folder:

```
struts-config.xml
struts-config-protocol.xml
struts-config-sample.xml
struts-config-publication.xml
struts-config-admin.xml
```

The Tiles definitions defining each page's layout and content are defined in the following files under the `WEB-INF` folder:

```
tiles-defs.xml
tiles-defs-protocol.xml
tiles-defs-sample.xml
tiles-defs-publication.xml
tiles-defs-admin.xml
```

The Struts Validator configuration files controlling how user entries at the input forms are validated are defined in the following files under the `WEB-INF` folder:

```
validator.xml
validator-rules.xml
validator-protocol.xml
validator-sample.xml
validator-publication.xml
```

The Struts Action classes handling each HTTP request are organized in the Java package `gov.nih.nci.cananolab.ui` under the `src` folder as follows:

```
gov.nih.nci.cananolab.ui.admin
gov.nih.nci.cananolab.ui.core
gov.nih.nci.cananolab.ui.protocol
gov.nih.nci.cananolab.ui.publication
gov.nih.nci.cananolab.ui.sample
gov.nih.nci.cananolab.ui.security
```

3.5 BUSINESS SERVICES

The business services in the caNanoLab system define business logics in submitting/searching nanoparticle samples and annotations, submitting/searching protocols, and submitting/searching publications. These business services are organized into the Java package `gov.nih.nci.cananolab.service` under the `src` folder as follows:

```
gov.nih.nci.cananolab.service.common
gov.nih.nci.cananolab.service.protocol
gov.nih.nci.cananolab.service.publication
gov.nih.nci.cananolab.service.sample
gov.nih.nci.cananolab.service.security
```

3.5.1 Local Implementation vs. Remote Implementation

Each business service is designed as a Java interface defining methods for persisting data into the caNanoLab data source and retrieving data from the data source. The interface has two implementation classes under the subpackage `impl`, one for communicating with a local database, one for communicating with a remote grid service. Since grid queries are read-only, the remote implementation classes don't implement any methods involving data persistence, and would throw an exception with a message "Not implemented for grid service".

In addition to the two implementation classes, there is also a helper class under the subpackage `helper` that captures the bulk of the Hibernate access codes (HQL, DetachedCriteria, etc.) that are used to retrieve data from the caNanoLab data source. Both the local implementation class of a business service interface and the grid service API (described in section 4) invoke the helper class for data retrieval. For example, the interface `gov.nih.nci.cananolab.service.sample.CompositionService` has two implementation classes, `impl.CompositionServiceLocalImpl` and a remote implementation class `impl.CompositionServiceRemoteImpl`, as well as a helper class `helper.CompositionServiceHelper`. With this design pattern for the business service, only one set of codes is required for the UI layer for both local searches and remote searches, and a Struts action class in the UI layer invokes either the local implementation or the remote implementation of the business service at run time and doesn't have to worry about the implementation details of the business service.

3.5.2 Customized caCORE SDK Application Service

In the caNanoLab system, the Hibernate transaction and session management are being encapsulated in the application service provided by the caCORE SDK. We customized the read-only caCORE SDK application service and caCORE SDK provide DAO with CRUD operations such that any business service can invoke the customized application service to make a CRUD call the data source.

The source codes related to the customized application service are packaged under the package `gov.nih.nci.cananolab.system` as follows:

- `applicationservice.CustomizedApplicationService`: Customized application service interface
- `applicationservice.impl.CustomizedApplicationServiceImpl`: implementation of the customized application service
- `dao.CustomizedORMDAO`: customized DAO interface defining CRUD operations to the data source
- dao.orm.CustomizedORMDAOImpl: implementation of the customized DAO interface.
- `dao.orm.CustomizedORMDAOFactory`: factory class to instantiate customized DAO.

The `application-config.xml` and `application-config-client.xml` files provided by the caCORE SDK are modified to reference customized application service.

3.6 FILE REPOSITORY STRUCTURE

In the caNanoLab system, users can upload files, and the uploaded files (protocol files, nanoparticle composition files, characterization files, publications) are stored on a network file system accessible to the server that hosts the application server serving the caNanoLab application. A pre-defined directory structure and file naming conventions are employed to prevent uploaded files with same names overwriting each other. When a file is uploaded, a timestamp in the format of `yyyyMMdd_HH-mm-ss-SSS` is added to the beginning of the filename as $NEW_FILE_NAME = $TIMESTAMP_$ORGINAL_FILENAME, e.g. 20061211_10-30-46-773_NCL200612A_fig 20.jpg. The new file name is saved to the appropriate directory on the file repository and file URI (`$DIRECTORY/$NEW_FILE_NAME`) is saved to the database. The following table describes the pre-defined directory structure for the different file types:

File Type	**Directory Structure**
`Publication`	`$ROOT/publications`
`Protocol`	`$ROOT/protocols`
`Characterization File`	`$ROOT/particles/$SAMPLE_NAME/$CHAR_NAME/`
`Nanomaterial Entity File`	`$ROOT/particles/$SAMPLE_NAME/nanomaterialEntity`
`Functionalizing Entity File`	`$ROOT/particles/$SAMPLE_NAME/functionalizingEntity`
`Chemical Association File`	`$ROOT/particles/$SAMPLE_NAME/chemicalAssociation`
`Composition File`	`$ROOT/particles/$SAMPLE_NAME/compositionFile`

Table 3-1: Uploaded File Directory Structure: $ROOT is the file repository root specified by users as a caNanoLab build property `file.repository.dir`; $SAMPLE_NAME is dynamically generated when a user uploads files associated with a sample; $CHAR_NAME is dynamically generated when users upload characterization files associated with a particular sample and a characterization name.

3.7 AUTHENTICATION AND AUTHORIZATION

User authentication and authorization in the caNanoLab system are implemented using CSM API version 4.1. In the caNanoLab system, the CSM database objects (prefixed with CSM) (shown in Figure 3-20) have been included under the same schema for storing the caNanoLab database objects. User accounts and assignment of users to user groups are maintained through a separate web application called UPT that comes bundled with the caNanoLab distribution. For the UPT tool that comes back caNanoLab installation, we made some customizations such that administrators can no longer update users passwords and can only reset a user's passwords by issuing an update to an existing user. A separate interface (visible on the caNanoLab log in page) has been implemented in the caNanoLab system to allow users to manage their own passwords.

As a part of the release 1.5 database seed data, a default user group `Public` is created and is assigned the `R` (read) role on protocols, samples and publications, meaning authenticated users belonging to the `Public` group can search publicly available protocols, samples and publications. During the caNanoLab application start up time, two default user groups are created: `Curator` and `Researcher`, where `Curator` and `Researcher` groups are prefixed with an application owner (e.g. NCL), specified as a build property, `application.owner`. During application startup time, the `Curator` group is automatically assigned to have `CURD` (create, update, read and delete privileges) role on sample, protocol and publication, meaning users belonging to this group can submit samples, protocols and publications to the system. When a `Curator` user submits a new sample with a new point of contact information into the system, the organization name entered in the point of contact information is automatically created as a new user group, and the new user group, the `Curator` group and the `Researcher` group are automatically assigned to have R (read) role on this newly created sample. When the `Curator` user marks the sample being visible to public within the caNanoLab application, any users would be able to access the sample without having to log in.

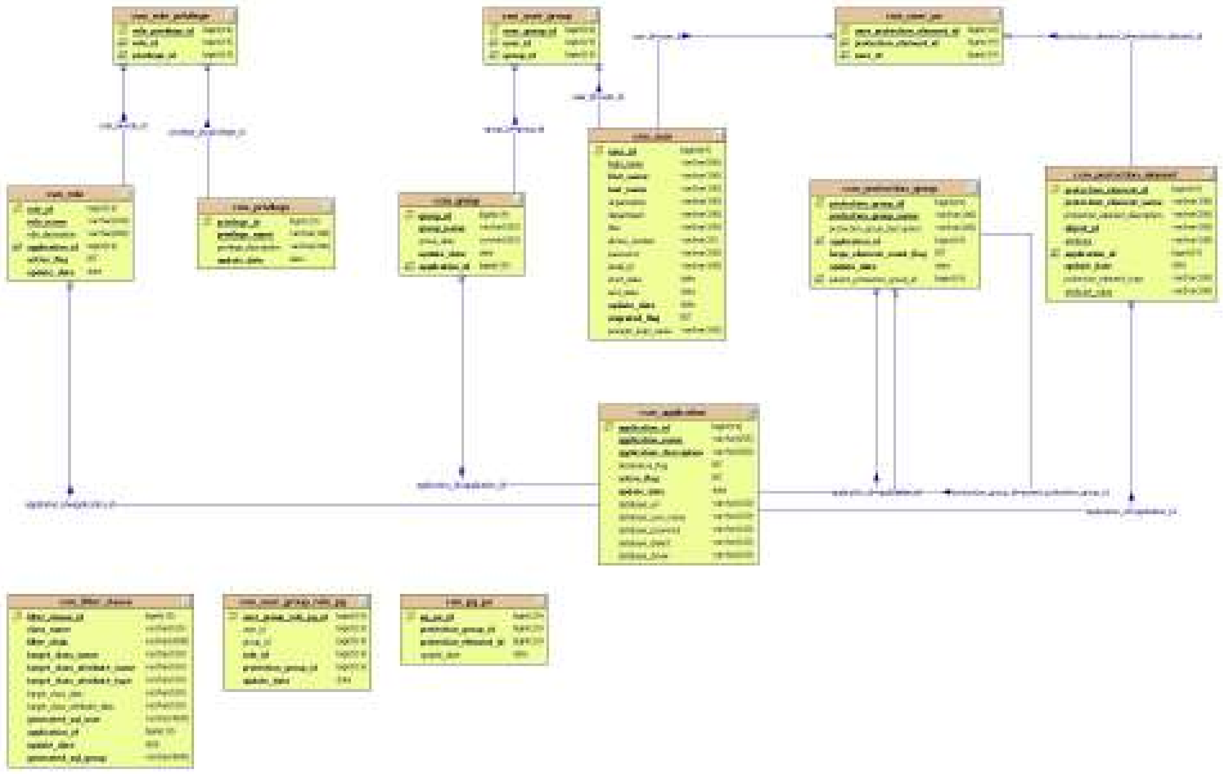


Figure 3-20: CSM Data Model

4. GRID-ENABLING ARCHITECTURE

Since release 1.1, the caNanoLab domain model has been grid-enabled such that users can execute CQLs against a deployed caNanoLab grid service to query for public data available through the caNanoLab domain model, or users can execute custom grid operations to retrieve public data returned by the operations. In release 1.5, the implementation of the caNanoLab grid service has been upgraded to caGrid 1.3 backed by caCORE SDK 4.0. As of release 1.5, five production caNanoLab grid services have been deployed and registered against the NCICB production index service. These production grid services are deployed at NCICBIIT, NCL, Washington University, Stanford University and Georgia Tech University.

4.1 CANANOLAB GRID ARCHITECTURE OVERVIEW

Figure 4-1 depicts the caNanoLab grid-enabling architecture using the caNanoLab grid service deployed at NCL and the grid client deployed at NCICBIIT as an example. In this example, the grid service deployed at NCL is considered the grid server side, and the caNanoLab web application deployed at NCICBIIT is considered as the client side of the NCL grid service.

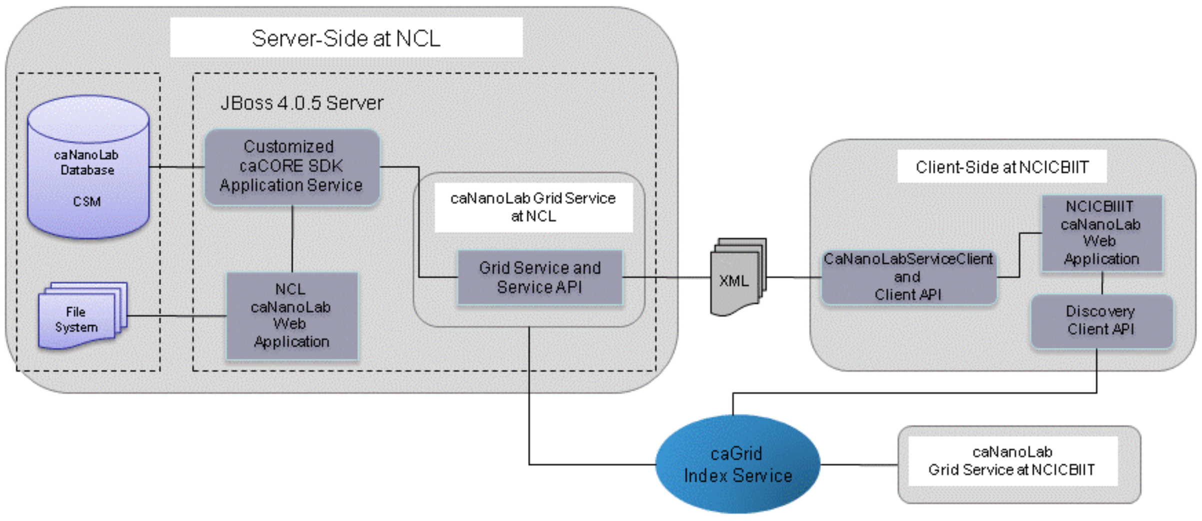

Server-Side at NCL
JBoss 4.0.5 Server
caNanoLab Database
CSM
File System
Customized caCORE SDK Application Service
NCL caNanoLab Web Application
caNanoLab Grid Service at NCL
Grid Service and Service API
XML
Client-Side at NCICBIIT
CaNanoLabServiceClient and Client API
NCICBIIT caNanoLab Web Application
Discovery Client API
caGrid Index Service
caNanoLab Grid Service at NCICBIIT

Figure 4-1: caNanoLab Grid-Enabling Architecture Diagram

When a user at the NCICBIIT web portal selects to query remote caNanoLab data resides in the NCL caNanoLab database, a CaNanoLabServiceClient object is instantiated at NCICBIIT with the grid service URL pointing to NCL. The CaNanoLabServiceClient offers APIs that allow the grid client to either execute CQLs or execute the custom grid operations. Both types of calls are translated into XMLs before the NCL grid service processes them. When the NCL grid service receives a CQL from the client, the CQL is being translated into an HQL in the grid service through a query processor. The application service (generated by the caCORE SDK) is called to execute the HQL against the NCL database to retrieve data from the database. Alternatively, when the NCL grid service receives a custom grid operation from the client, the grid service invokes the corresponding grid service API that constructs a customized HQL and passes it to the application service for processing. In both cases, the application service receives data objects back from the NCL database, and the grid service takes the retrieved data objects and serializes them into XML documents. These XMLs are passed back to the NCICBIIT CaNanoLabServiceClient object instantiated at NCICBIIT. The CaNanoLabServiceClient object takes the XML file and deserialized the XMLs back into objects that the NCICBIIT caNanoLab web portal understands.

Since it is required in caNanoLab that only public nanoparticle data are shared across the grid, we have implemented special mechanisms to filter out non-public data. Details are discussed in the next section.

4.2 CANANOLAB GRID SERVICE

The core of a caNanoLab grid-enabling architecture is the caNanoLab grid data service. The data service is generated using the caGrid 1.3 Introduce toolkit through local APIs provided by the caCORE SDK 4.0 backed caNanoLab data source.

4.2.1 Custom Grid Operations

Additional implementations of custom grid operations and public-filtering mechanism are added into the grid service implementation class `gov.nih.nci.cagrid.cananolab.service.CaNanoLabServiceImpl`. Custom operations are published as grid service metadata along with the caNanoLab domain model and are viewable to the general public.

4.2.1.1 *Unidirectional Associations*

Most custom grid operations are added because the caGrid infrastructure doesn't support inclusion of associations in the query results through CQL, and some associations defined in the caNanoLab object model are unidirectional. CQLs can not be written to retrieve the unidirectionally associated objects of an object. For example, the operation **`getFileByProtocolId`** is added to retrieve the File object associated with a Protocol object, and the association from Protocol to File is unidirectional. Following is list of all such operations (all throw `RemoteException` and is remitted here for brevity):

```
public gov.nih.nci.cananolab.domain.common.Finding[] getFindingsByCharacterizationId(
                    java.lang.String charId)

public gov.nih.nci.cananolab.domain.common.ExperimentConfig[]
getExperimentConfigsByCharacterizationId(
                    java.lang.String charId)

public gov.nih.nci.cananolab.domain.common.Keyword[] getKeywordsBySampleId(
                    java.lang.String sampleId)
```

```
public gov.nih.nci.cananolab.domain.common.File[] getFilesByCharacterizationId(
                    java.lang.String charId)

public gov.nih.nci.cananolab.domain.common.File[] getFilesByCompositionInfoId(
                    java.lang.String id, java.lang.String className)

public gov.nih.nci.cananolab.domain.common.Protocol getProtocolByCharacterizationId(
                    java.lang.String charId)

publicgov.nih.nci.cananolab.domain.common.PointOfContact
getPrimaryPointOfContactBySampleId(
                    java.lang.String sampleId)

public gov.nih.nci.cananolab.domain.common.PointOfContact[]
getOtherPointOfContactsBySampleId(
                    java.lang.String sampleId)

public java.lang.String[] getSampleNamesByPublicationId(
                    java.lang.String publicationId)

public gov.nih.nci.cananolab.domain.common.File getFileByProtocolId(
                    java.lang.String protocolId)
```

4.2.1.2 *Complex Queries*

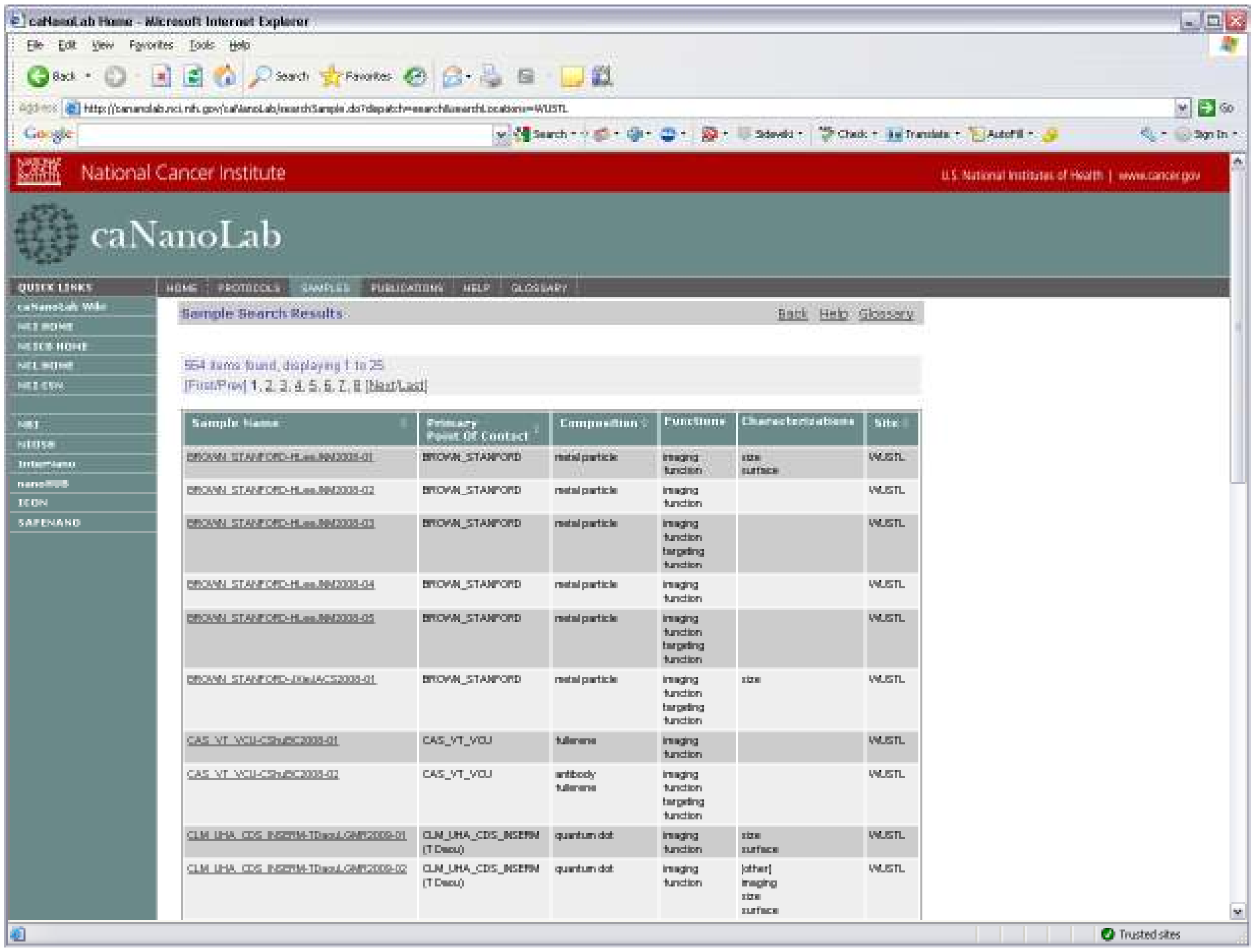
In addition to the operations supporting the retrieval of unidirectional associations, we also implemented a few custom operations to retrieve data based on the complex search criteria defined in the use cases of the caNanoLab portal. For these complex queries, constructing multi-level nested CQLs is not only inefficient, sometimes not even possible. Instead, we designed custom grid operations to return data from the complex queries. When implementing these operations, the actual Hibernate access codes are implemented in helper classes following the business delegate pattern. These helper classes were also used in the service objects for local complex searches used in the portal. The following is a list of such operations and associated helper operations:

```
public java.lang.String[] getSampleNames(
                    java.lang.String samplePointOfContact,
                    java.lang.String[] nanomaterialEntityClassNames,
                    java.lang.String[] functionalizingEntityClassNames,
                    java.lang.String[] functionClassNames,
                    java.lang.String[] characterizationClassNames,
                    java.lang.String[] words)

public java.lang.String[] getPublicationIdsBy(
                    java.lang.String publicationTitle,
                    java.lang.String publicationCategory, java.lang.String sampleName,
                    java.lang.String[] researchAreas, java.lang.String[] keywords,
                    java.lang.String pubMedId, java.lang.String digitalObjectId,
                    java.lang.String[] authors,
                    java.lang.String[] nanomaterialEntityClassNames,
                    java.lang.String[] functionalizingEntityClassNames,
                    java.lang.String[] functionClassNames)
```

4.2.1.3 *View Specific Queries*

The nano-OM is a rather complex domain model as there are many associations linked to a single object. To query for a fully loaded remote object (i.e. all associations are populated) on the grid, multiple CQLs or multiple custom grid operations would have to be executed across the grid. It may take minutes before all required information come back from the query. For a web application, this would not be desirable. For example, on the caNanoLab home page as shown in Figure 3-16, when a user clicks on the number of samples from a grid location, a set of remote queries would need be fired to in order to retrieve all samples from that grid location and all the associated information (sample point of contact, composition, function and characterizations) for these samples before the search results can be displayed in a table format as shown in Figure 4-2 below.



Sample Search Results

554 items found, displaying 1 to 25
[First/Prev] 1, 2, 3, 4, 5, 6, 7, 8 [Next/Last]

Sample Name	Primary Point Of Contact	Composition	Functions	Characterizations	Site
BROWN_STANFORD-HLee.NM2008-01	BROWN_STANFORD	metal particle	imaging function	size surface	WUSTL
BROWN_STANFORD-HLee.NM2008-02	BROWN_STANFORD	metal particle	imaging function		WUSTL
BROWN_STANFORD-HLee.NM2008-03	BROWN_STANFORD	metal particle	imaging function targeting function		WUSTL
BROWN_STANFORD-HLee.NM2008-04	BROWN_STANFORD	metal particle	imaging function		WUSTL
BROWN_STANFORD-HLee.NM2008-05	BROWN_STANFORD	metal particle	imaging function targeting function		WUSTL
BROWN_STANFORD-JXieJACS2008-01	BROWN_STANFORD	metal particle	imaging function targeting function	size	WUSTL
CAS_VT_VCU-CShuBC2008-01	CAS_VT_VCU	fullerene	imaging function		WUSTL
CAS_VT_VCU-CShuBC2008-02	CAS_VT_VCU	antibody fullerene	imaging function targeting function		WUSTL
CLM_UHA_CDS_INSERM-TDaouLGMR2009-01	CLM_UHA_CDS_INSERM (T Daou)	quantum dot	imaging function	size surface	WUSTL
CLM_UHA_CDS_INSERM-TDaouLGMR2009-02	CLM_UHA_CDS_INSERM (T Daou)	quantum dot	imaging function	[other] imaging size surface	WUSTL

Figure 4-2: Sample Search Results from a Grid Location

In order to return the nanoparticles back in a reasonable time frame for displaying in the caNanoLab web application, we implemented an alternative grid operation that returns a String array that captures all the information required to display in the table. This "hack" dramatically improves the query performance, although if the table format is to be changed in the future, the implementation of the grid operation would have to be modified.

```
public java.lang.String[] getSampleViewStrs(java.lang.String sampleName)
```

4.2.2 Public Filter

The caNanoLab grid service has been created as service that doesn't require authentication, but it is required that all data returned from the grid service are public data only. In order for the grid service to serve public data to the grid clients, the grid service side needs to filter out non-public data before passing the data back to the clients. Because data can be retrieved through CQLs or through grid operations, we implemented public filtering to satisfy both user scenarios.

For data retrieval through CQLs, we implemented a customized query processor `gov.nih.nci.cagrid.cananolab.sdk4query.processor.PublicDataSDK4QueryProcessor`, in which the method `queryCoreService` includes customized logic to append where clauses in the translated HQLs to filter out non public data. The customized query processor also invokes a customized version of CQL to HQL translator we implemented `gov.nih.nci.cagrid.cananolab.sdk4query.processor.PublicDataCQL2ParameterizedHQL`, in which we appended the select clause in the translated HQLs with information that can work with the appended where clause for translating CQLs with query modifiers. We also provided a fix in `PublicDataCQL2ParameterizedHQL` to correctly translate CQLs that involve object inheritance mapped through table-per-hierarchy mechanism (currently reported as a bug in the default CQL to HQL translator).

For data retrieval through custom caNanoLab grid service operations, the public data filtering logic is embedded in the helper classes that each grid service operation implementation invokes. By passing in a null user object into the helper class, the helper class in turn filters out non-public data and passes only public data back to the grid client.

4.2.3 Remote File Retrieval

In caNanoLab 1.5, users can retrieve remote protocol files and publications published from a grid service. Instead of passing the file content back from the grid service as byte arrays, we implemented grid operations to only return back public remote file IDs. Tests show that passing byte array of a file back from a caGrid data service is very memory intensive, and service performance is significantly impacted. When a remote grid service passes back a public file ID to a local caNanoLab application, the local application makes a direct http call to the remote caNanoLab application by constructing the correct download URL.

In the local caNanoLab application, a file can be downloaded to the client machine through a `download` dispatch method implemented in a Struts action class. For example, the download URL on the NCICB production site for downloading a public publication of ID 31784 stored at NCL would be: http://cananolab.nci.nih.gov/caNanoLab/publication.do?dispatch=download&fileId=3178496&lo

cation=NCL, under the assumption that the caNanoLab application and the caNanoLab grid service are deployed to the same JBoss container and share the same virtual host name.

APPENDIX A – ACRONYM LIST

Acronym	Description
ABCC	Advanced Biomedical Computing Center
caBIG	Cancer Biomedical Informatics Grid
caCORE	Cancer Common Ontological Representation Environment
caDSR	Cancer Data Standards Repository
caNanoLab	Cancer Nanotechnology Laboratory Analysis Bench
CCNE	Center of Cancer Nanotechnology Excellence
CSM	Common Security Module
CSS	Cascading Style Sheet
DAO	Data Access Object
DTO	Data Transfer Object
EIS	Enterprise Information System
EVS	Enterprise Vocabulary Services
J2EE	Java 2 Enterprise Edition
JSP	Java Server Page
JSTL	JavaServer Pages Standard Tag Library
NCICBIIT	National Cancer Institute Center for Biomedical Informatics and Information Technology
NCL	Nanotechnology Characterization Laboratory
ORM	Object Relational Mapping
POJO	Plain Old Java Object
SDK	Software Development Toolkit
SSL	Secure Socket Layer
UI	User Interface
UML	Unified Modeling Language
UPT	User Provisioning Toolkit
NPO	Nanoparticle Ontology