# caNanoLab – cancer Nanotechnology Laboratory

# Software Design Description

# Release 1.4

## 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 |

# TABLE OF CONTENTS

# 1. INTRODUCTION

The purpose of this document is to describe the design for the cancer Nanotechnology Laboratory (caNanoLab) Release 1.4.

## 1.1 PROJECT BACKGROUND

The caNanoLab effort is collaboration amongst 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, imaging) of the particle, the characterizations of the particle from physical (e.g. size, molecular weight) and in vitro (e.g. cytotoxicity, immunotoxicity) 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

## 1.2 IDENTIFICATION

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

- ***Title***: cancer Nanotechnology Laboratory
- ***Abbreviation:*** caNanoLab
- ***Version:*** 1.4
- ***Release Number:*** 1.4

## 1.3 REQUIREMENTS

The following is a list of high-level functional and technical requirements for caNanoLab 1.4:

| 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) 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 |
| **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 nanoparticle composition specific to each nanoparticle type |
| 2.2 | The system shall allow users to annotate and search nanoparticles based on nanoparticle characterizations, functions, keywords and summary/descriptions associated with characterizations |

| | |
|---|---|
| 2.3 | The system shall provide support for physical characterizations |
| 2.4 | The system shall provide support for in vitro characterizations |
| 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 instrument type, manufacturers associated with the characterizations |
| 2.7 | The system shall allow authorized users to set visibility of submitted assay results. |
| 2.8 | The system shall allow users to upload and retrieve nanoparticle reports |
| 2.9 | The system shall allow authorized users to set visibility of uploaded reports |
| 2.10 | The system shall allow users to associate reports 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 particle to other particles from the same particle source |
| 2.14 | The system shall allow authorized users to delete single and multiple characterizations |
| 2.15 | The System shall allow user to specify both base composition of a nanoparticle sample and the functionalization of a nanoparticle sample. |
| 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 sample composition. |
| 2.18 | The System shall allow user to specify multiple functions for a nanoparticle sample. |
| **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**<br><br>**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 nanoparticles and reports 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, nanoparticles and reports as well as the publicly available protocols, nanoparticles and reports stored in remote caNanoLab grid services. |
| 4.5 | The system shall be upgraded to caGrid 1.2 |
| 4.6 | The system shall search public composition data, physical 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.** |

**Table 1-1: Requirements Matrix**

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

## 1.4 SYSTEM OVERVIEW

The primary goals of the caNanoLab Release 1.4 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

The caNanoLab 1.4 effort 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 1.4 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 Scope Documents | CVS Server: cbiocvs2; module: cananolab; Directory: docs/Scope |

| | |
|---|---|
| caNanoLab Use Case Specification documents | CVS Server: cbiocvs2; Module: cananolab; Directory: docs/usecase |
| caNanoLab Architecture Checklist | CVS Server: cbiocvs2; Module: cananolab; Directory: docs/design |
| caNanoLab 1.4 Installation Guide | http://gforge.nci.nih.gov/frs/download.php/4395/caNanoLab_1.4_Installation_Guide.pdf |
| CSM 3.0 Developer's Guide<br><br>UPT 3.0 User's Guide | http://gforge.nci.nih.gov/frs/download.php/3714/CSM_Guide_ApplicationDevelopers.pdf<br><br>http://gforge.nci.nih.gov/frs/download.php/3716/UPT_User_Guide.pdf |
| caCORE SDK 4.0 Developer's Guide | http://gforge.nci.nih.gov/docman/view.php/148/8650/caCORE%20SDK%204.0%20Developer's%20Guide_101007.pdf |
| caBIG Compatibility Guidelines | https://gforge.nci.nih.gov/frs/download.php/3948/caBIG_Compatibility_Guidelines_v3.0_FINAL.pdf |
| caGrid 1.2 User Guide | http://gforge.nci.nih.gov/frs/download.php/3769/caGrid-1-2_Users_Guide.pdf |
| caGrid 1.2 Programmer Guide | http://gforge.nci.nih.gov/frs/download.php/3768/caGrid-1-2_Programmers_Guide.pdf |

**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 fosters a loosely coupled and tightly cohesive architecture in which objects in each tier are focused on specific architectural responsibilities yet, cleanly integrated with the other tiers. The architecture of a caNanoLab grid data service is described separately in section 4, not as a part of the caNanoLab system. The following architecture diagram shows the overall system architecture of the caNanoLab system. An overview of each tier is provided in the sections immediately following the diagram.

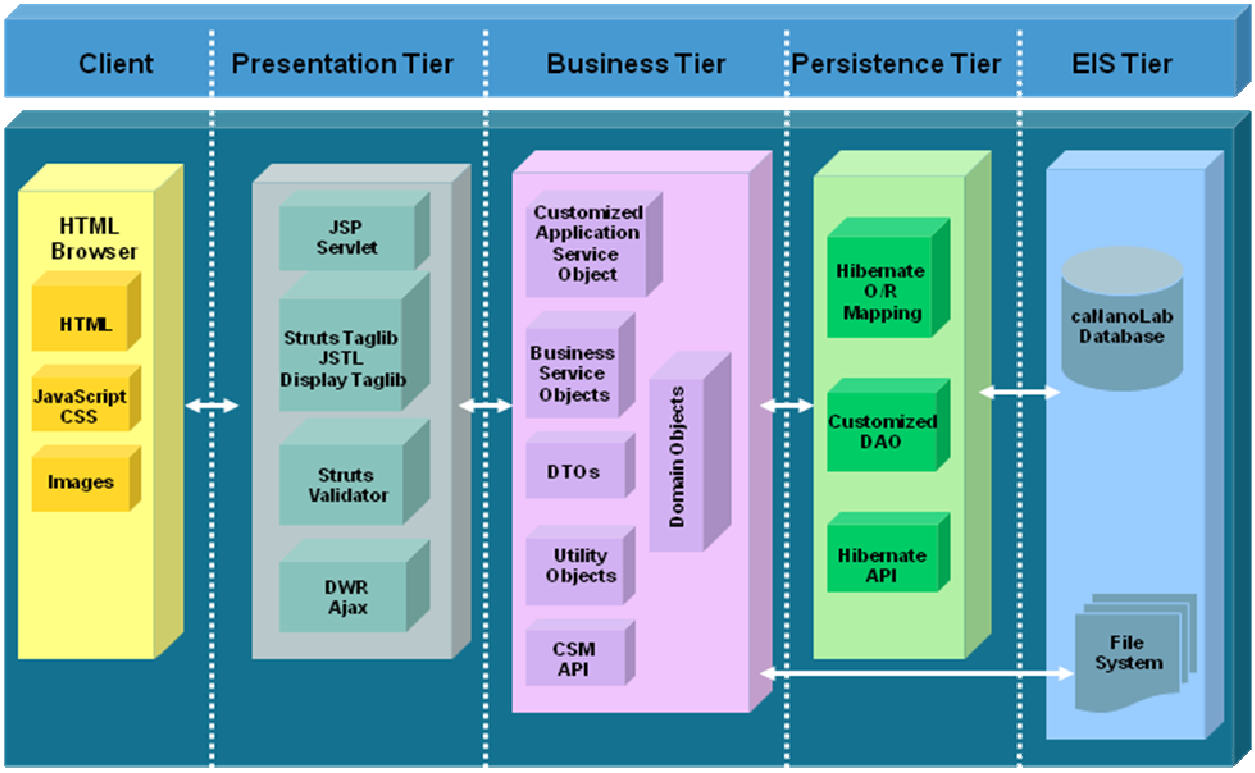


**Figure 2-1: System Architecture**

### 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. 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 nanoparticle samples and the compositions and the functionalizations, as well as the outcome (characterizations) of assays applied to these nanoparticles. The nano-OM has been modeled after the NCL's nanoparticle assay cascade and use cases from CCNEs. The following subsections describe details of the nano-OM. . 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 hierarchy diagram in Figure 3-1, the nano-OM is composed of the following two high-level packages:

- `particle` – Defines objects describing nanoparticle samples. This package includes two sub-packages: `particle.samplecomposition` and `particle.characterization`.
  - `particle.samplecomposition` – Describes the base composition, the functionalization, the chemical associations, and the functions of a nanoparticle sample.
  - `particle.characterization` - Describes the physical and the in-vitro characterizations that are applied to a nanoparticle sample. The characterization data include instruments used, protocols leveraged, characterization files (e.g. graphs, charts, spreadsheets, etc.) and derived data that provide insight into the complexities of the characterization.
- `common` – Defines objects that are commonly shared in the application, for example, `LabFile`, `Keyword`, etc.

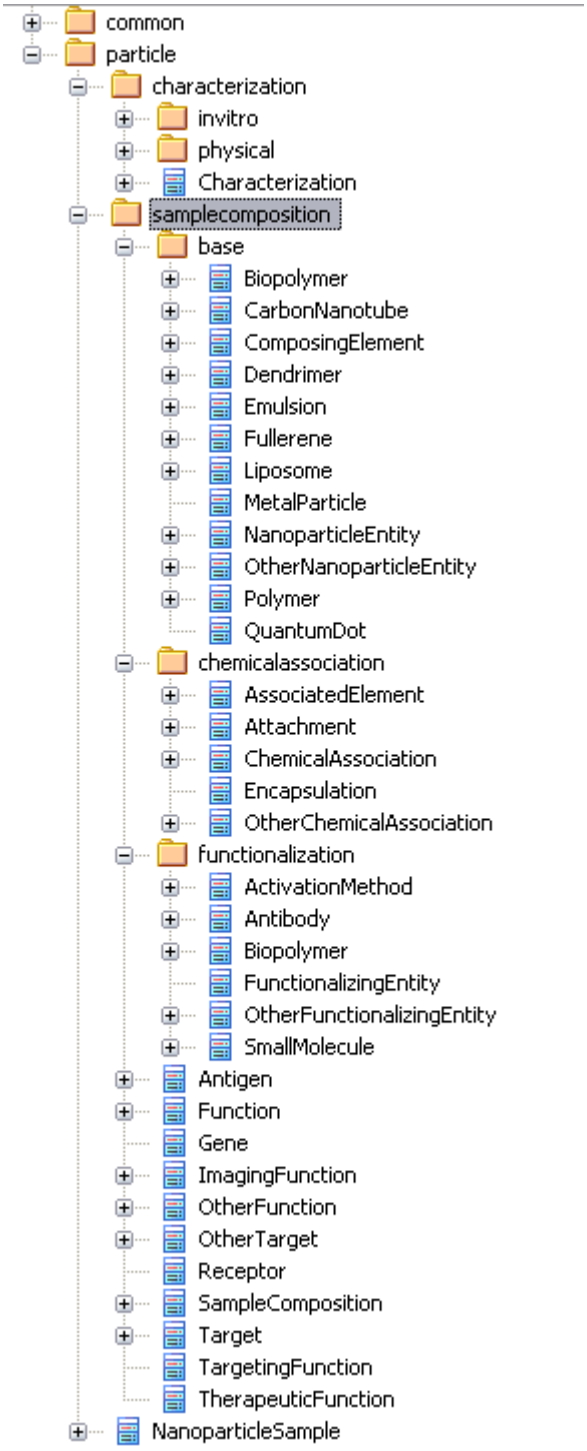


**Figure 3-1: nano-OM Package Hierarchy**

The subsections that follow illustrate nano-OM classes and attributes associated with each package in more details.

### 3.1.1 Nanoparticle Sample Overview

As shown Figure 3-2, at the highest level, a `NanoparticleSample` object has a `Source`, an optional `SampleComposition`, an optional collection of `Characterization` objects, and an optional collection of `Keyword` objects and an optional collection of `Report` objects.

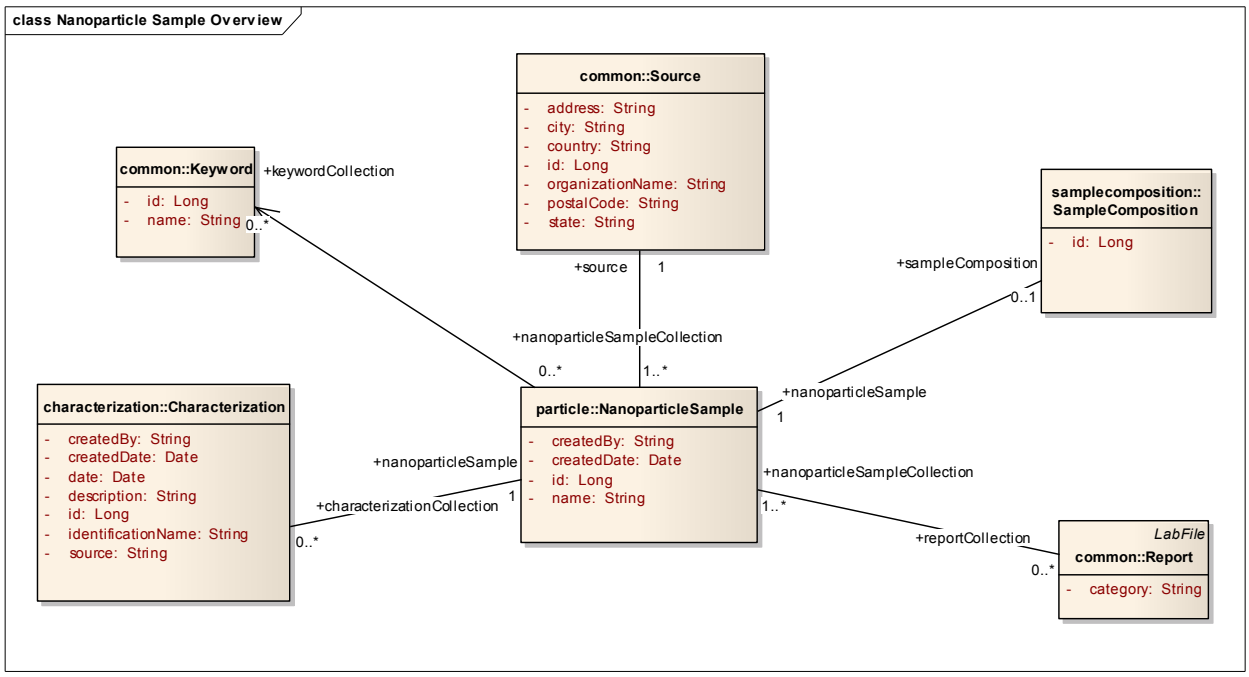


**Figure 3-2: Nanoparticle Sample Overview (nano-OM)**

### 3.1.2 Composition Overview

As shown in Figure 3-3, a `SampleComposition` object of a `NanoparticleSample` has a collection of `NanoparticleEntity` objects, an optional collection of `FunctionalizingEntity` objects, an optional collection of `ChemicalAssociation` (`Encapsulation`, `Attachment`, other) objects, and an optional collection of `LabFile` objects. Each `NanoparticleEntity` has an optional collection of `ComposingElement` objects, and each `ComposingElement` object has an optional collection of `Function` objects as inherent functions of the nanoparticle. Each `FunctionalizingEntity` has a collection of `Function` objects as the functions resulted from nanoparticle functionalization. Each `ChemicalAssociation` object has an optional collection of `LabFile` objects.

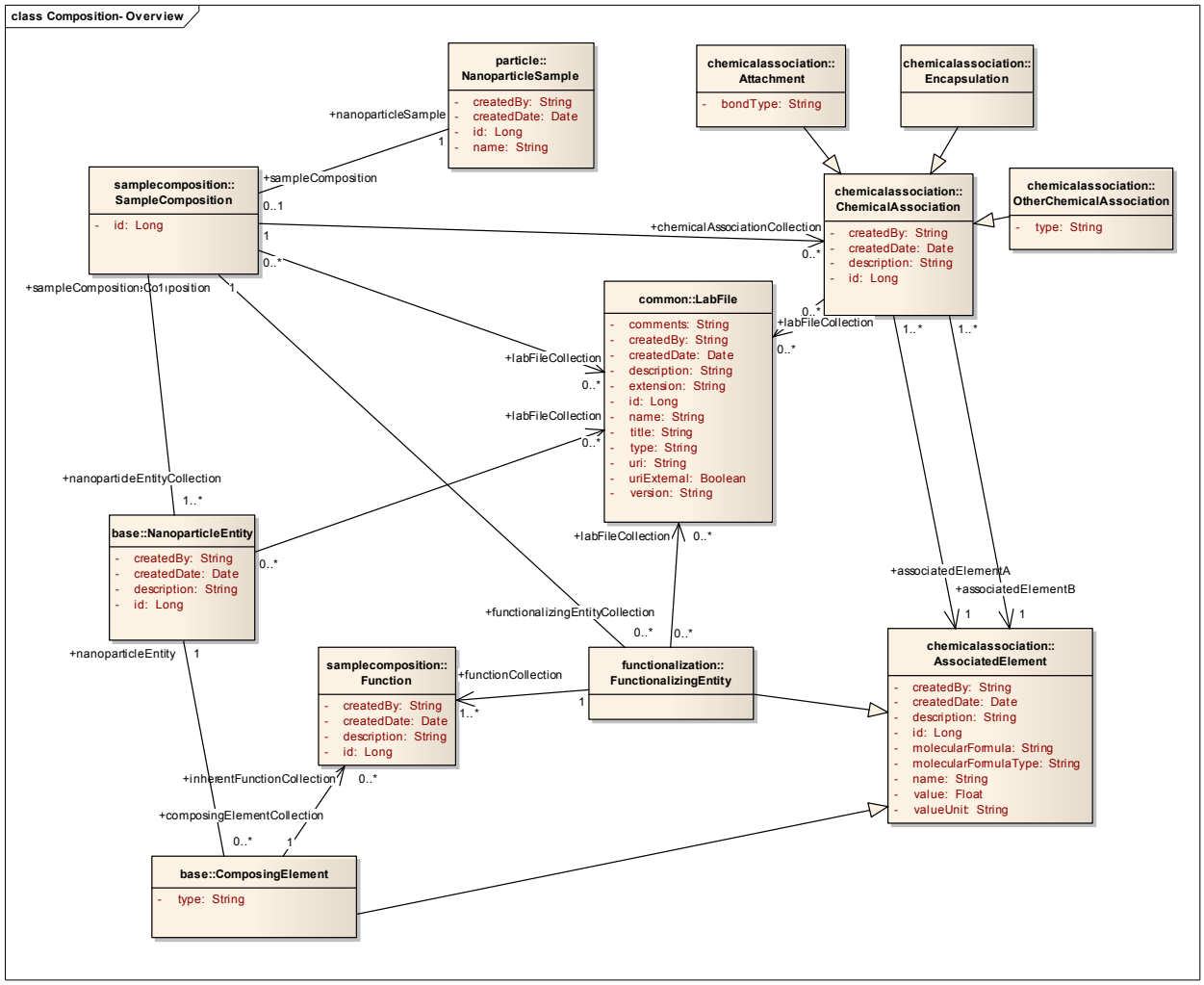


**Figure 3-3: Composition Overview (nano-OM)**

### 3.1.2.1 *Composition – Nanoparticle Entity*

As shown in Figure 3-4, we define eight types of `NanoparticleEntity`: `Biopolymer`, `CarbonNanotube`, `Dendrimer`, `Emulsion`, `Fullerene`, `MetalParticle`, `Polymer`, `QuantumDot`, and others are categorized as `OtherNanoparticleEntity`. Each `NanoparticleEntity` has an optional collection of `ComposingElement` objects. The inherence functions of a nanoparticle sample are described as a collection of `Function` objects associated with a `ComposingElement`. In the nano-OM, we describe three types of `Function` objects: `ImagingFunction`, `TargetingFunction` and `TheraupeticFunction`, with others categorized as `OtherFunction`.

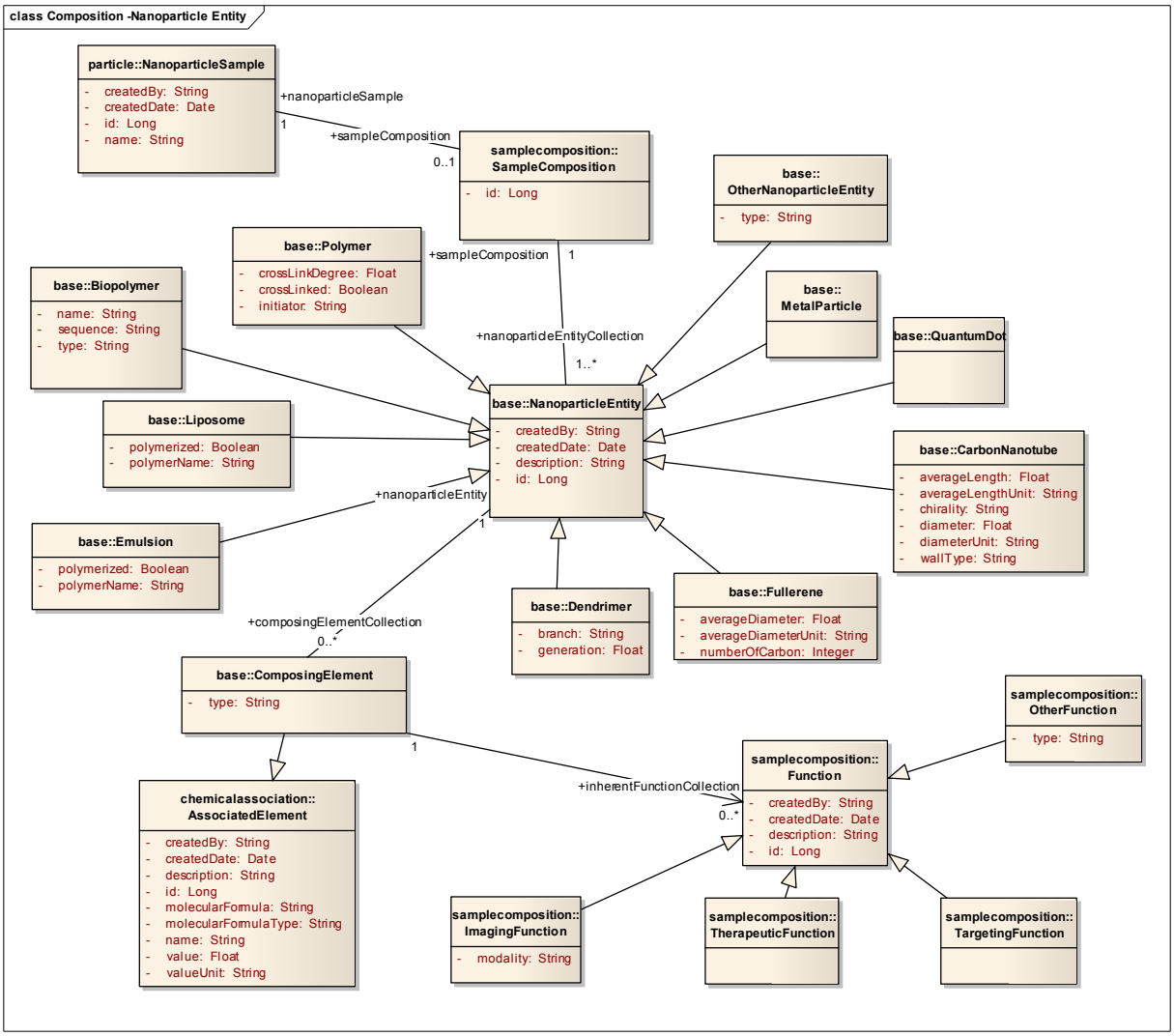


**Figure 3-4: Composition – Nanoparticle Entity (nano-OM)**

### 3.1.2.2 *Composition – Functionalizing Entity*

As shown in Figure 3-5, there are three 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 resulted from functionalization. When describing functions of a nanoparticle sample in the context of a `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`.

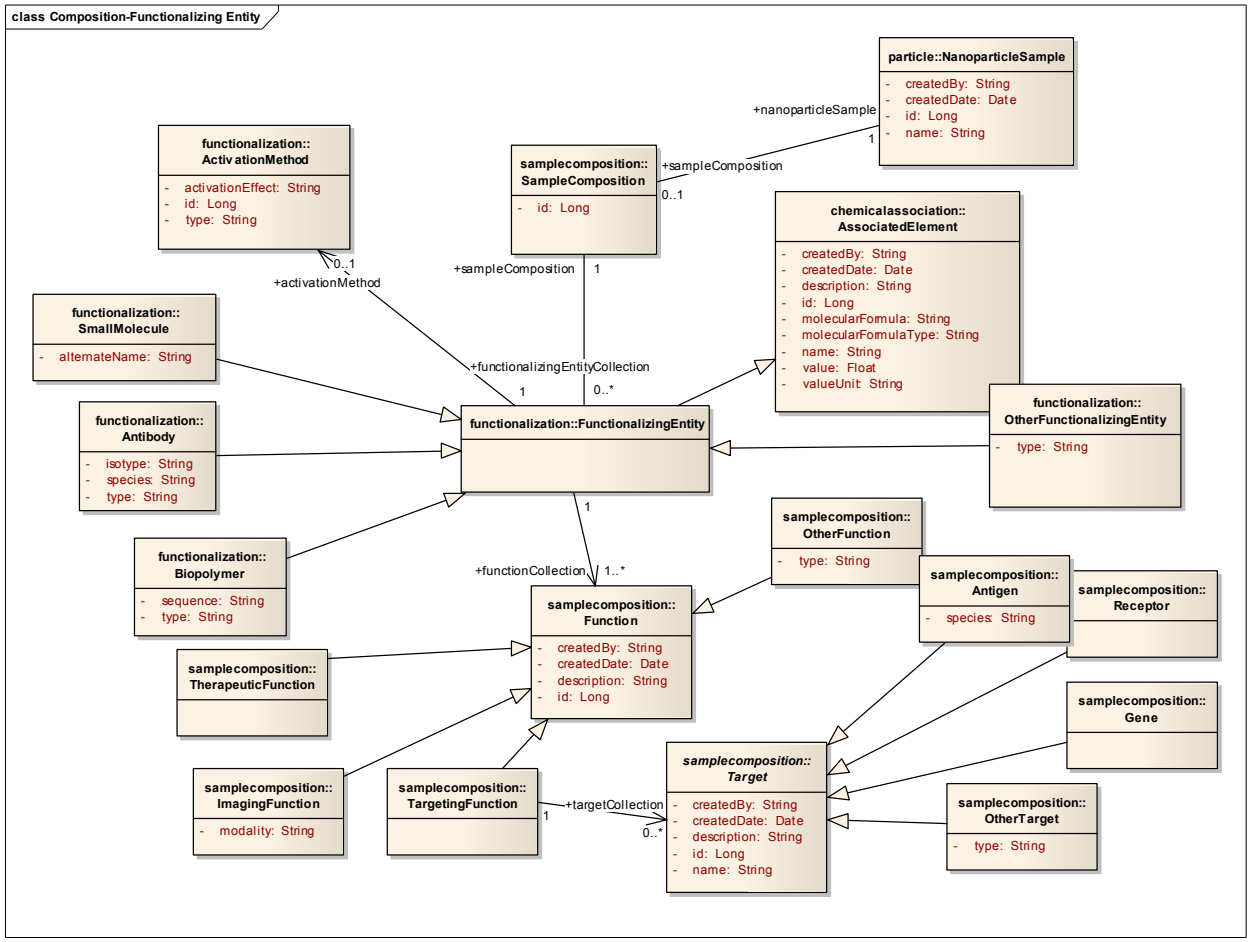


**Figure 3-5: Composition – Functionalizing Entity (nano-OM)**

### 3.1.3 Characterization Overview

As shown in Figure 3-6, a `Characterization` has an optional `ProtocolFile` and an associated `Protocol`, an optional `InstrumentConfiguration` and an associated `Instrument`, an optional collection of `DerivedBioAssayData` objects. Each `DerivedBioAssayData` has an optional `LabFile`, an optional collection of `DerivedDatum` objects. Each `LabFile` has an optional collection of `Keyword` objects. In the nano-OM, we describe `PhysicalCharacterization` and `InvitroCharacterization` as two types of `Characterization`. A third type `InvivoCharacterization` has also been designed to describe general particle pharmacokinetics, toxicology and efficacy, but is not yet implemented in the application. The draft design and draft wireframes can be found at http://gforge.nci.nih.gov/tracker/index.php?func=detail&aid=3823&group_id=69&atid=368.

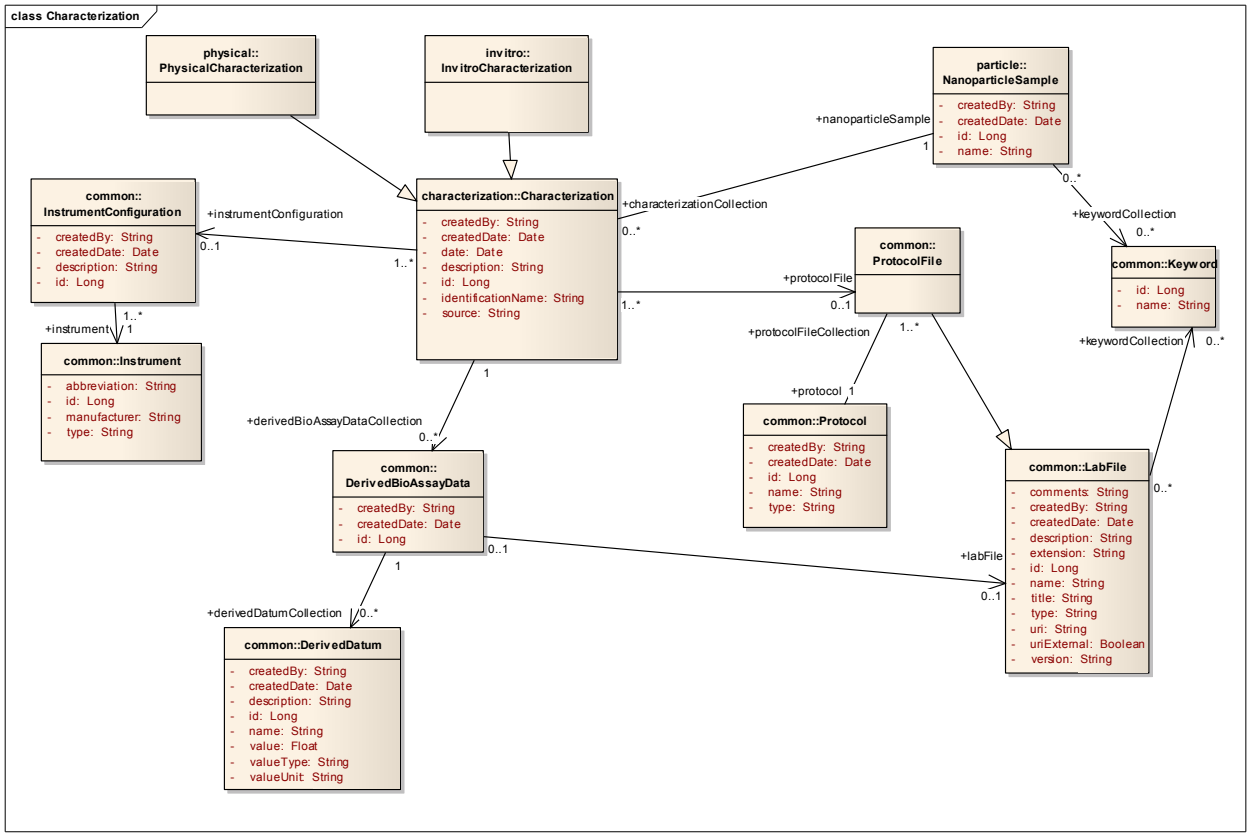


**Figure 3-5: Characterization Overview (nano-OM)**

### 3.1.3.1 *Physical Characterization*

As shown in Figure 3-6, there are seven types of `PhysicalCharacterization`: `MolecularWeight`, `PhysicalState`, `Purity`, `Shape`, `Size`, `Solubility`, and `Surface`. The `Surface` object is further described to have an optional collection of `SurfaceChemistry` objects.

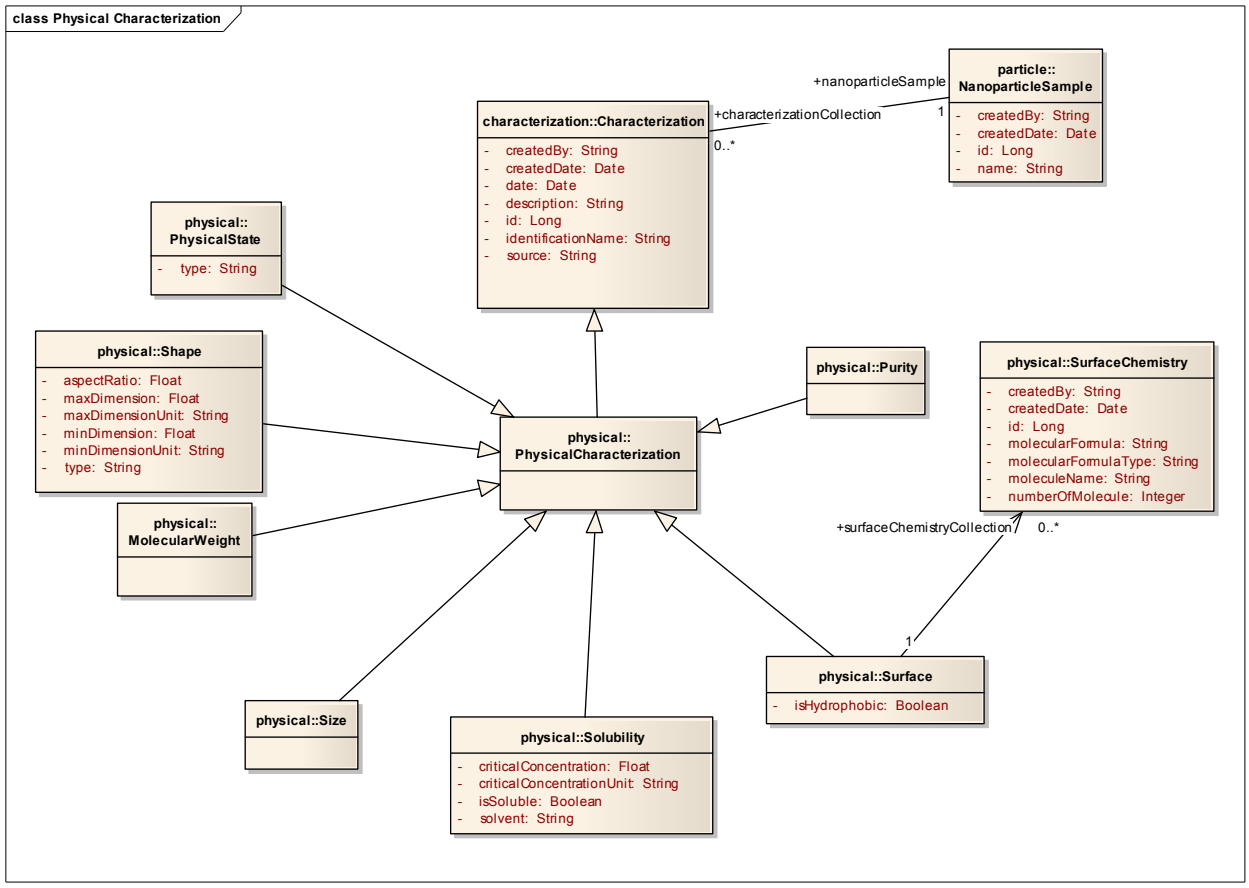


**Figure 3-6: Physical Characterization (nano-OM)**

### 3.1.3.2 *In Vitro Characterization*

As shown in Figure 3-6, a `Toxicity` object is a special kind of `InvitroCharacterization`, and there are four types of `Toxicity`: `OxidativeStress`, `EnzymeInduction`, `Cytotoxicity` and `Immunotoxicity`. `Cytotoxicity` can be `Caspase3Activation` or `CellViability`. `Immunotoxicity` can be `BloodContact` or `ImmuneCellFunction`. `BloodContact` can be `PlateletAggregation`, `Hemolysis`, `Coagulation` or `PlasmaProteinBinding`. `ImmuneCellFunction` can be `CFU_GM`, `ComplementActivation`, `Chemotaxis`, `Phagocytosis`,

`LeukocyteProliferation`, `OxidativeBurst`, `CytokineInduction` or `NKCellCytotoxicActivity`.

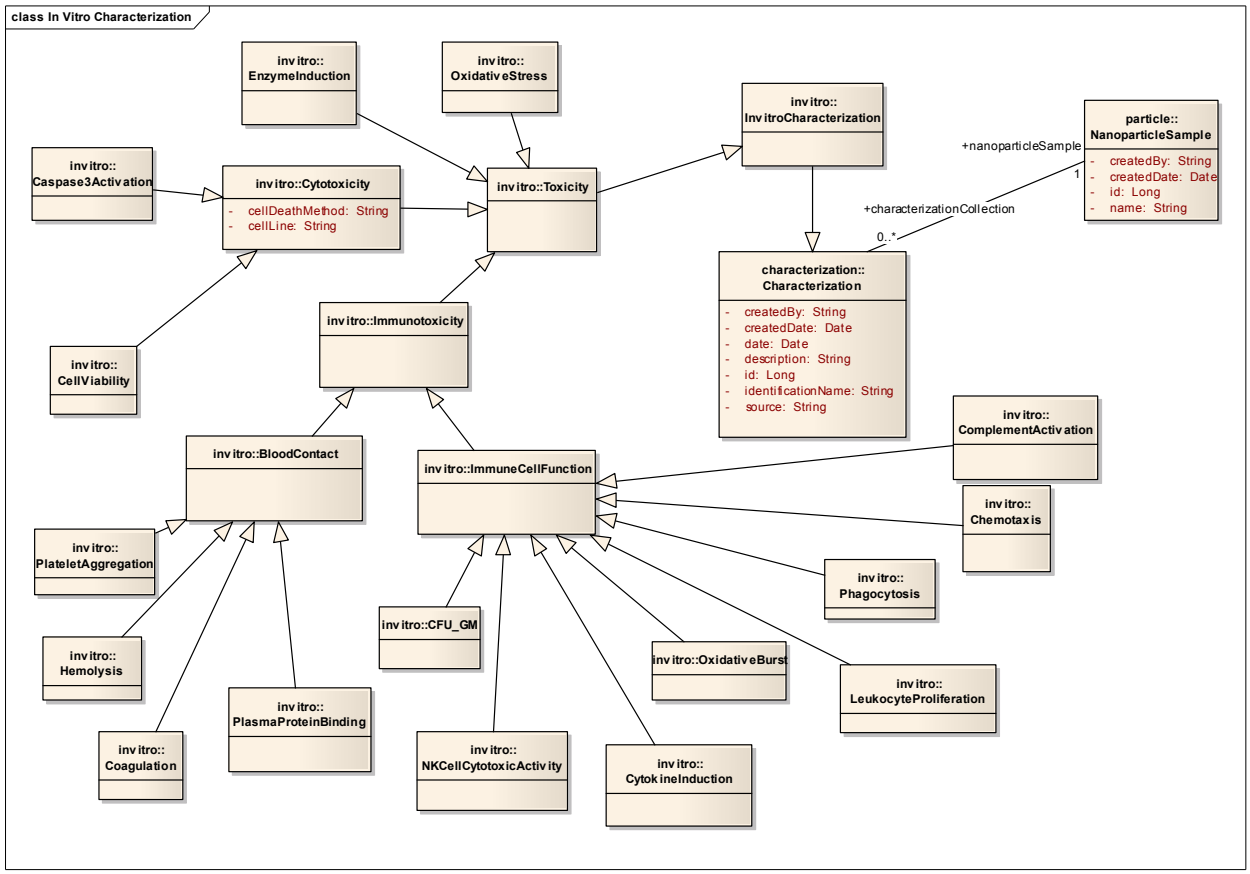


**Figure 3-7: In Vitro Characterization (nano-OM)**

## 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 Hibernate ORMs. Following the Hibernate 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 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 Nanoparticle Sample Overview

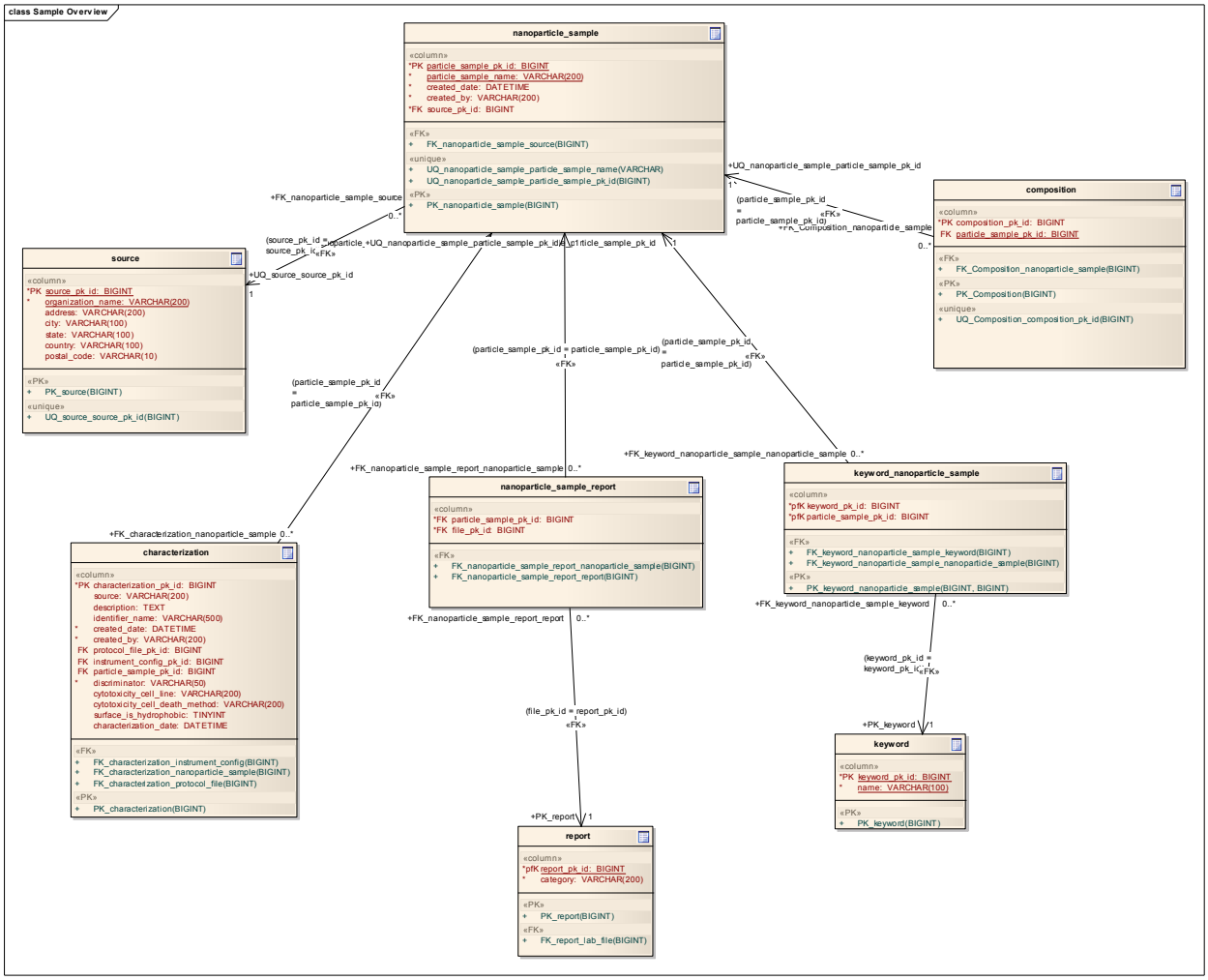


**Figure 3-8: Nanoparticle Sample Overview (Nanoparticle Data Model)**

## 3.2.2 Composition Overview

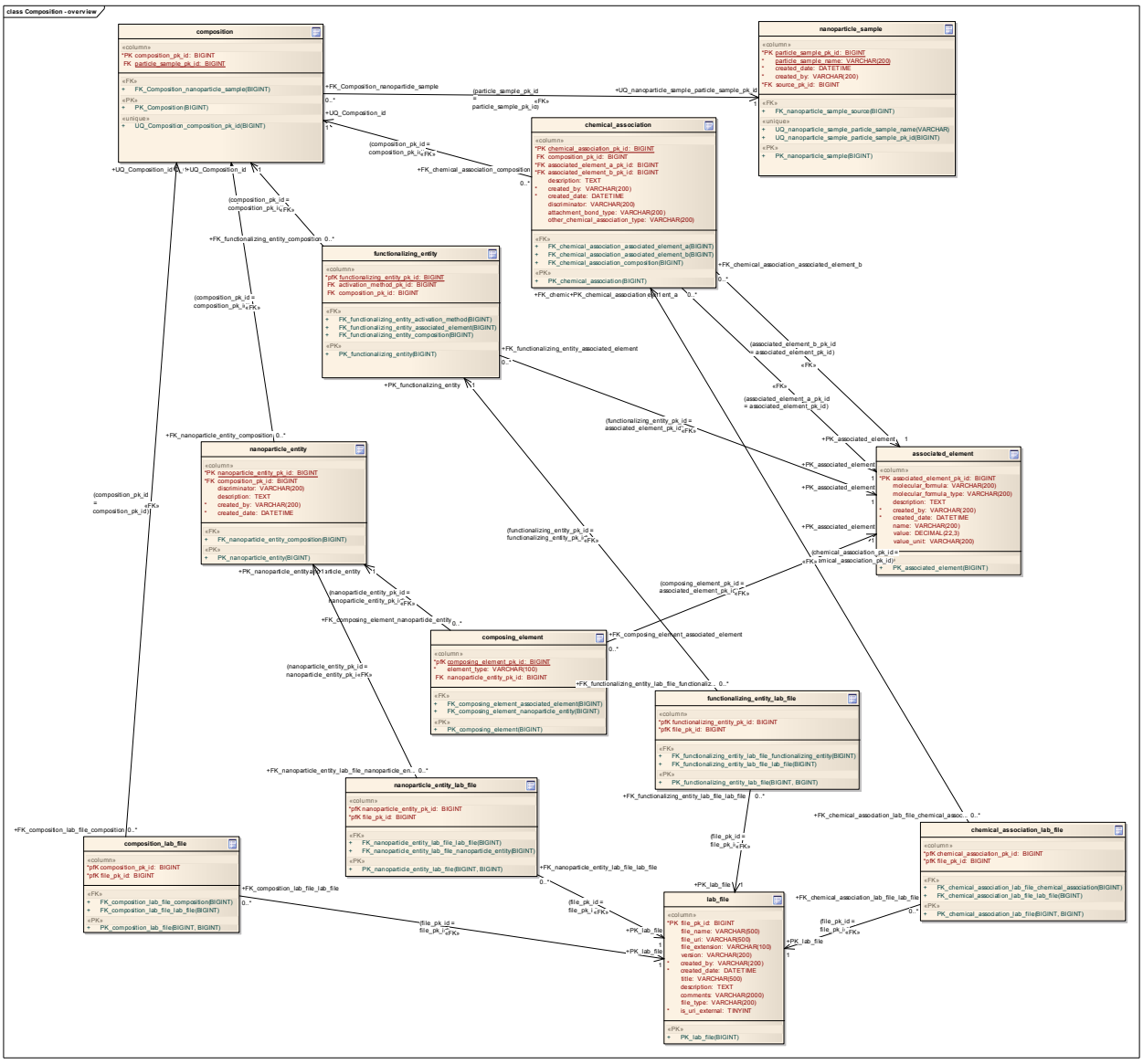


**Figure 3-9: Composition Overview (Nanoparticle Data Model)**

### 3.2.2.1 *Composition – Nanoparticle Entity*

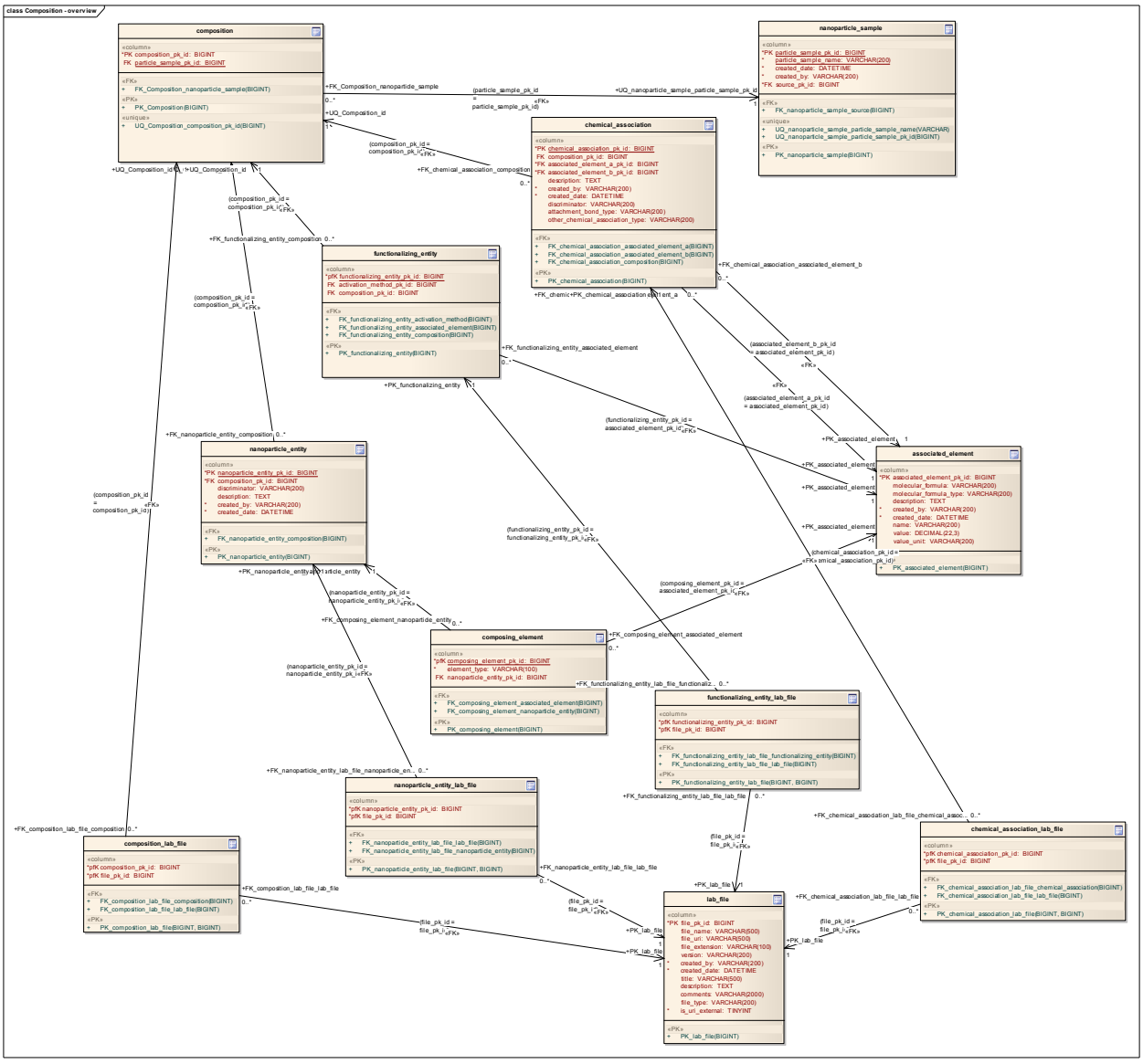


**Figure 3-10: Composition – Nanoparticle Entity (Nanoparticle Data Model)**

### 3.2.2.2 *Composition – Functionalizing Entity*

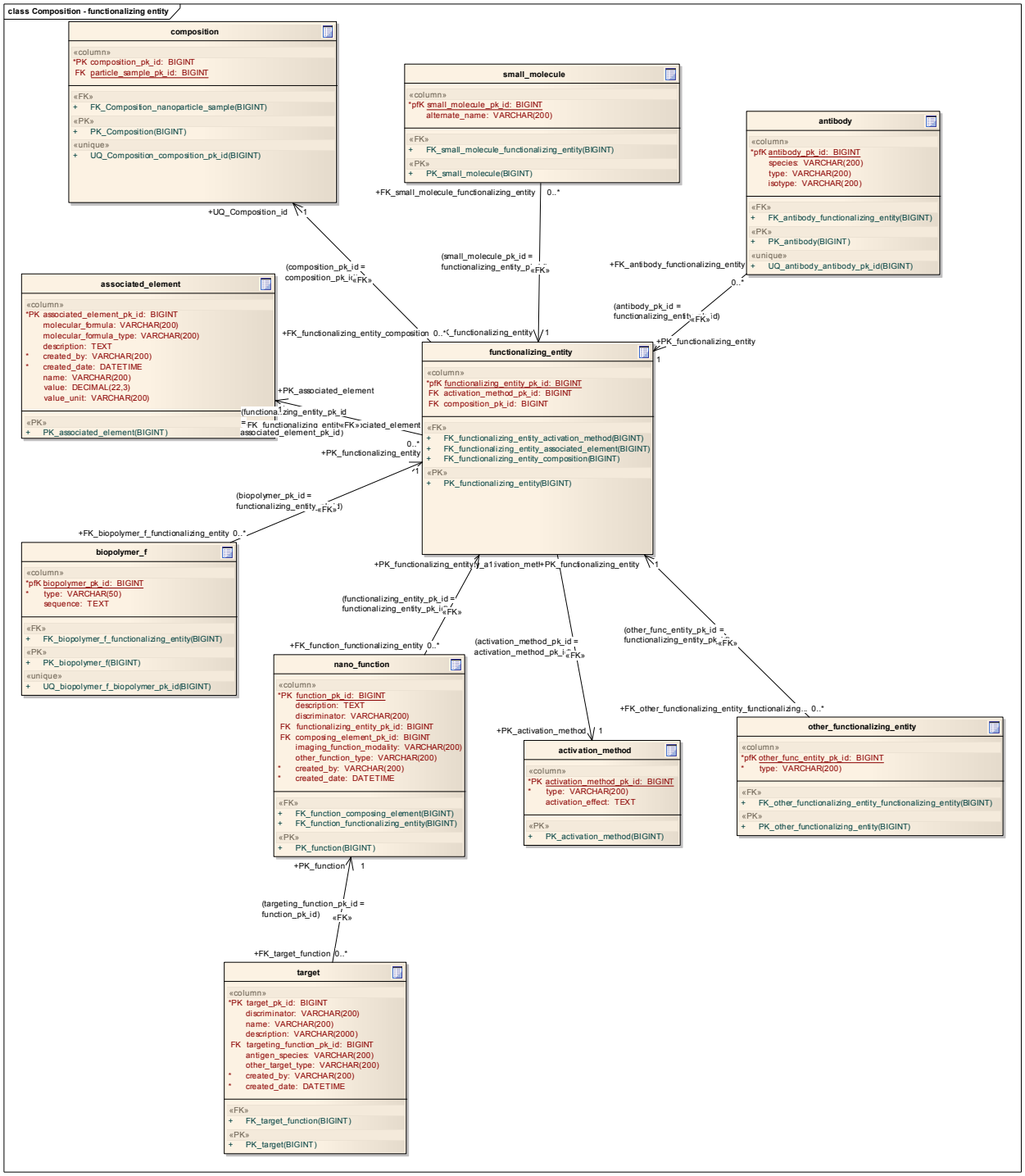


**Figure 3-11: Composition – Functionalizing Entity (Nanoparticle Data Model)**

### 3.2.3 Characterization View

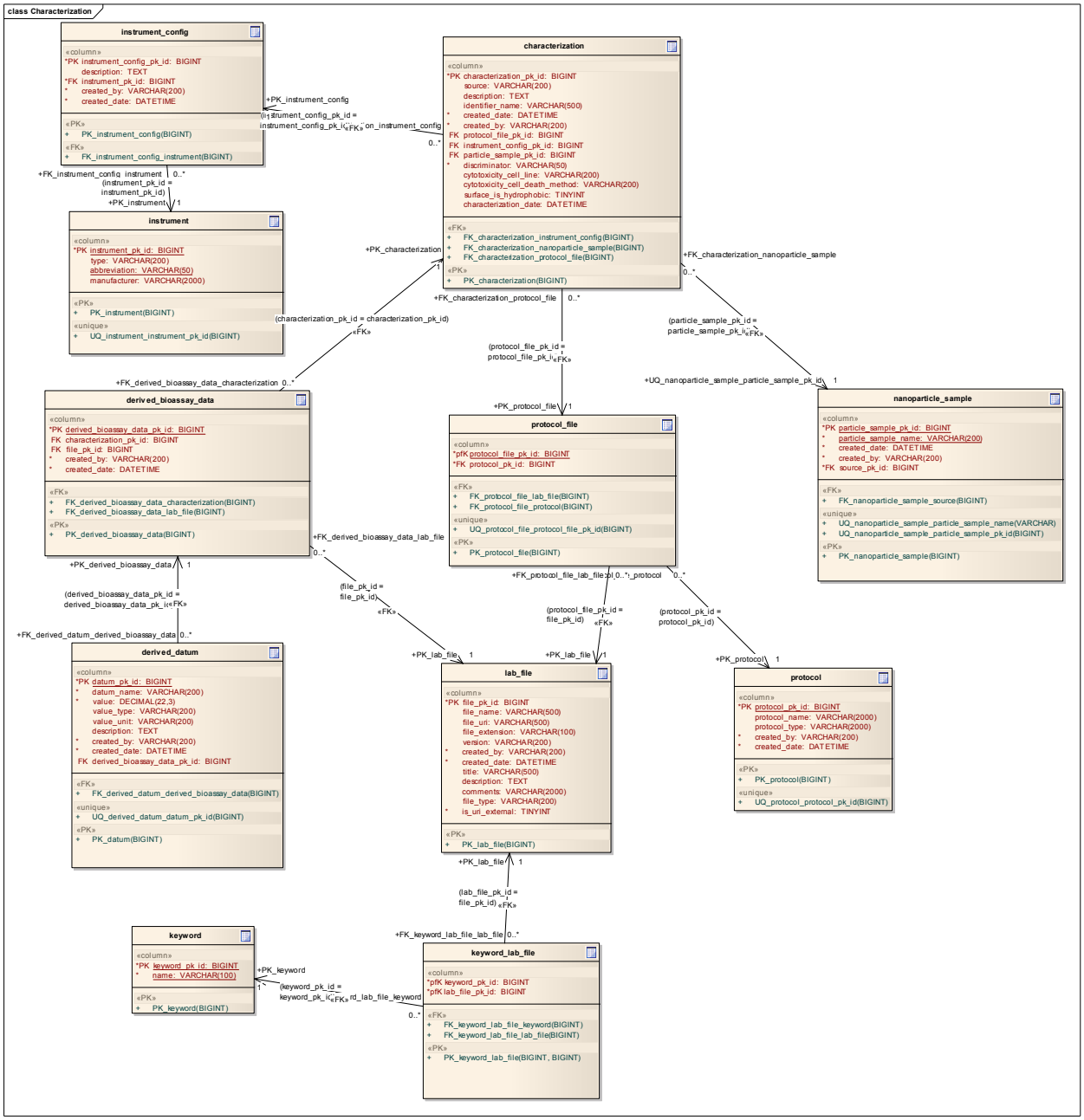


**Figure 3-12: Characterization View (Nanoparticle Data Model)**

### 3.2.3.1 *Characterization – Physical Characterization*

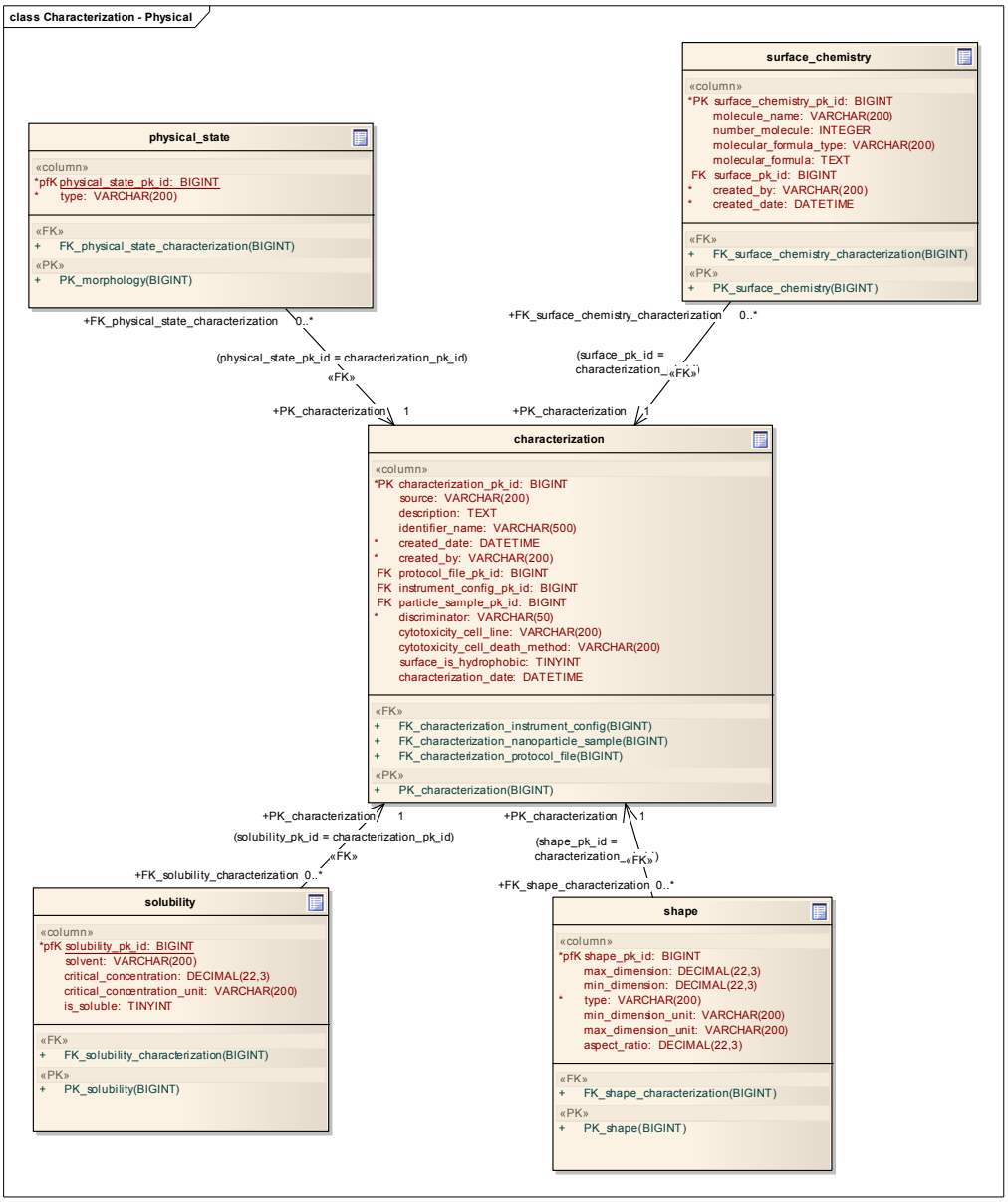


**Figure 3-13: Characterization – Physical Characterization View (Nanoparticle Data Model)**

### 3.2.3.2 *Characterization – In Vitro Characterization*

The inheritance structure in the in-vitro characterization portion of the nano-OM has been implemented using table-per-hierarchy strategy such that the same table `characterization` is used to capture information embedded all child classes of InvitroCharacterization through the use of the `discriminator` column.


**Figure 3-14: Characterization – In Vitro Characterization View (Nanoparticle Data Model)**

### 3.2.4 Look Up Table

A lookup table `common_lookup`, as shown in Figure 3-15, has been designed to store all pre-defined controlled vocabularies that are used as values in drop-down lists within the application.

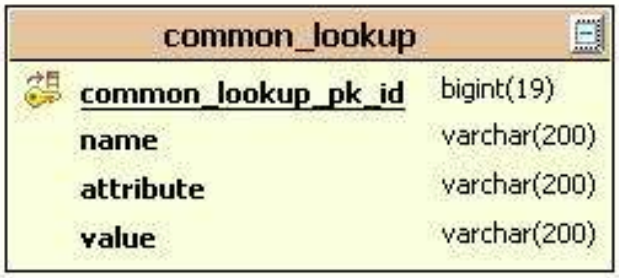


**Figure 3-15: common_lookup Data Model**

Pre-defined lookup values are stored in `name-attribute-value` trios and are looked up by the application to display in a drop-down list. For example, as shown in Table 3-1, four rows have been inserted into the table to capture the pre-defined derived datum names for Size characterization.

| name | attribute | Value |
|---|---|---|
| Size | derivedDatumName | PDI |
| Size | derivedDatumName | peak1 |
| Size | derivedDatumName | RMS-size |
| Size | derivedDatumName | Z-average |

**Table 3-1: An Example of Pre-Defined Rows in common_lookup**

Please see http://gforge.nci.nih.gov/frs/download.php/4396/common_lookup.xls for all the pre-defined controlled vocabularies stored in the `common_lookup` table. The table is also used to store any user-entered options through the `[Other]` option in the drop-down lists. For example, if a user entered a new derived data name `peak2` under Size characterization, a new `name-attribute-value` trio would be `Size-otherDerivedDatumName-peak2`.

It is in the planning that user-entered lookup values would be periodically reviewed and fed into the EVS to generate new EVS concepts. .

## 3.3 PACKAGE STRUCTURE

The caNanoLab root package is `gov.nih.nci.cananolab`. The source-code package structure is as shown in Figure 3-16. The packages are first divided according to the logical

divisions of different tiers in a J2EE system: `domain`, `dto`, `exception`, `resources`, `service`, `system`, `ui`, and `util`.

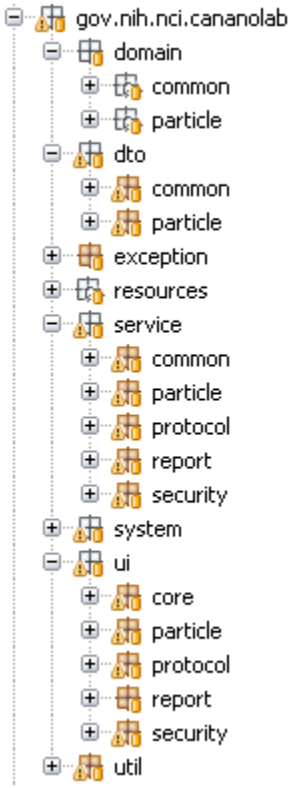


**Figure 3-16: caNanoLab Source Package Structure**

Each child package is further divided into sub-packages according to the logical divisions of different categories of nanoparticle information: `common`, `protocol`, `particle`, `report`, and `security`.

## 3.4 USER INTERFACE

The caNanoLab web application user interface has been modeled based on the use cases gathered from NCL and CCNEs.

### 3.4.1 Struts

The presentation-tier components are implemented using the Struts Framework. Struts ActionForms (declaratively defined as DynaActionForms) and action mappings are defined in the Struts configuration files:

```
struts-config.xml
struts-config-protocol.xml
struts-config-particle.xml
struts-config-report.xml
```

The Tiles definitions defining each page's layout and content are defined in the following files:

```
tiles-defs.xml
tiles-defs-protocol.xml
tiles-defs-particle.xml
tiles-defs-report.xml
```

The Struts Validator configuration files controlling how user entries at the input forms are validated are defined in the following files:

```
validator.xml
validator-rules.xml
validator-protocol.xml
validator-particle.xml
validator-report.xml
```

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

```
gov.nih.nci.cananolab.ui.core
gov.nih.nci.cananolab.ui.protocol
gov.nih.nci.cananolab.ui.particle
gov.nih.nci.cananolab.ui.report
gov.nih.nci.cananolab.ui.security
```

### 3.4.2 Web Design

Each caNanoLab web 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 3-17, without logging in to the system, users can seamlessly browse publicly available protocols, nanoparticles and reports available either in the local installation or in remote caNanoLab grid services.


**Figure 3-17: caNanoLab Home Page**

After a user logs into the application, a menu bar with different tabs is displayed at the top of the right content region, as shown in Figure 3-18. Different users with different access privileges can see different tabs. Please refer to section 3.7 for details on pre-defined security rules. Alternatively, users can click on hotspots in the workflow diagram to navigate through the system.

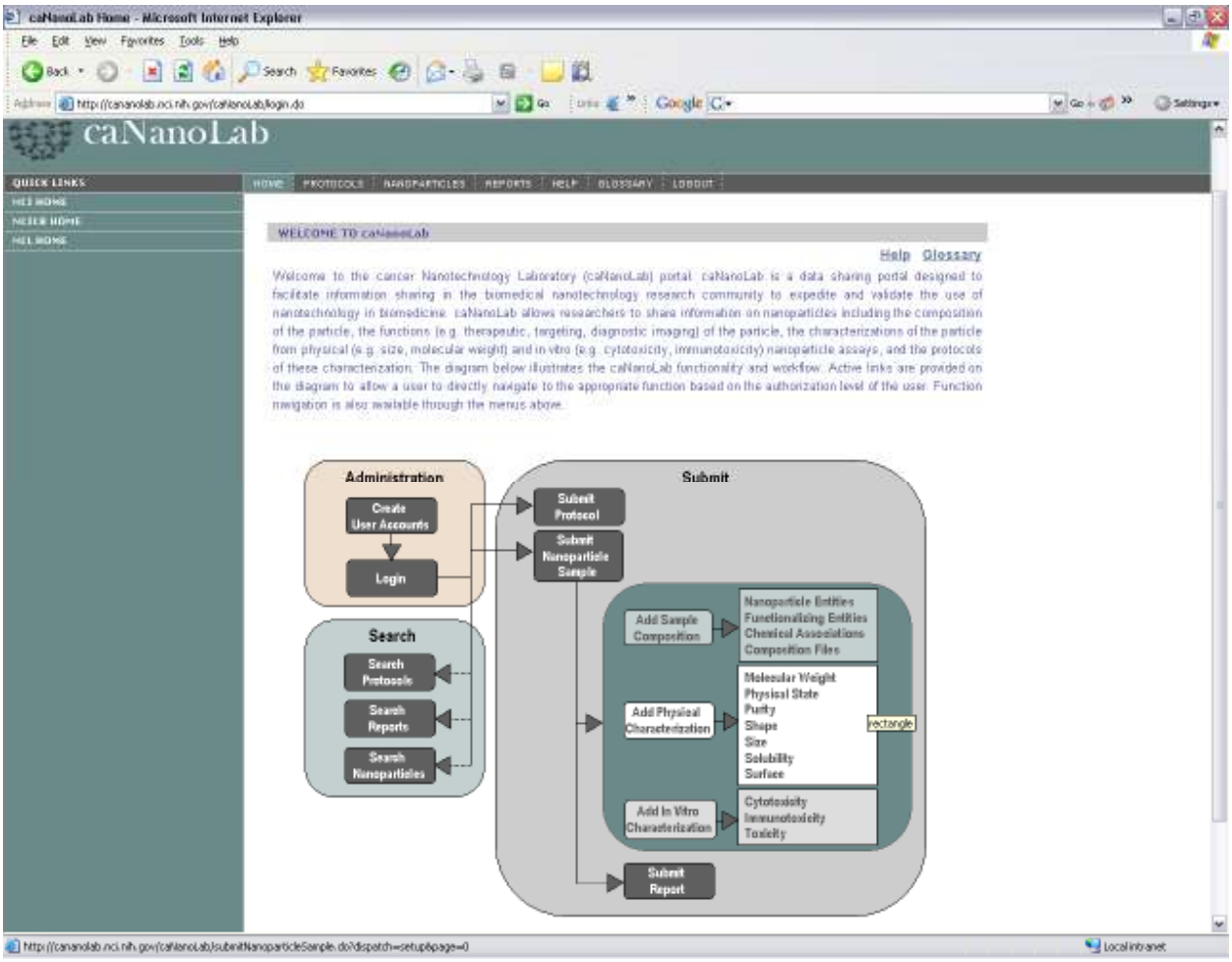


**Figure 3-18: caNanoLab Web Interface Workflow**

## 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 reports. These business services are organized into the Java package `gov.nih.nci.cananolab.service` as follows:

```
gov.nih.nci.cananolab.service.common
gov.nih.nci.cananolab.service.protocol
gov.nih.nci.cananolab.service.particle
gov.nih.nci.cananolab.service.report
gov.nih.nci.cananolab.service.security
```

### 3.5.1 Local Implementation vs. Remote Implementation

Each business service in the caNanoLab application is designed as a Java interface such that it specifies methods for persisting data into the caNanoLab data source and retrieving data from the data source, but leave the implementation details to the two separate implementation classes under the sub-package `impl`, one for communicating with a local database, one for communicating with a remote grid service. For example, the interface `gov.nih.nci.cananolab.service.particle.NanoparticleSampleService`, defining methods for persisting and search nanoparticle samples, has two implementation classes, `impl.NanoparticleSampleServiceLocalImpl` and a remote implementation class `impl.NanoparticleSampleServiceRemoteImpl`. The two implementation classes provide totally different ways of implementing the same method specification. In some cases, one method is only implemented in the local implementation class, not in the remote implementation class, and vice versa. For example, 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, a helper class under the sub-package `helper` has been designed to capture the bulk of the Hibernate access codes (Hibernate Query Languages, DetachedCriteria, etc.) that are required to retrieve data from a caNanoLab data source. Both the local implementation class of a business service interface and the grid service API (described in section 4) would invoke the helper class to complete the data retrieval. For example, the helper class `helper.NanoparticleSampleServiceHelper` is implemented for `gov.nih.nci.cananolab.service.particle.NanoparticleSampleService` such that the method `findNanoparticleSamplesBy` implements the Hibernate Query Language (HQL) necessary to retrieve a list of nanoparticle samples by a set of search criteria. This method is invoked by the local implementation class `impl.NanoparticleSampleServiceLocalImpl` to retrieve local nanoparticle samples from local caNanoLab data source, and is also invoked by the caNanoLab grid service method `getNanoparticleSamplesBy` to return nanoparticle samples from the caNanoLab data source associated with the grid service.

With the design pattern described for the business services, the UI layer is exposed to the same business interface for local searches and remote searches, can decide to invoke either one at run time. Any implementation changes made in the business tier would not be propagated to the UI layer and thus simplifies the implementation for the seamlessly search interface shown in Figure 3-17.

### 3.5.2 Customized caCORE SDK Application Service

In the caNanoLab system, the Hibernate transaction and session management are taken care of by the application service provided by the caCORE SDK. Since the caNanoLab application requires data persistence, we made customizations to the read-only caCORE SDK application service and DAO with CRUD (create, read, update and delete) operations such that a business service in the

caNanoLab system would invoke the customized application service to make a CRUD call to 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 also 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 and reports) 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 have been designed to prevent uploaded files overwriting each other. The following table describes the pre-defined directory structure for the different file types:

| File Type | Directory Structure |
|---|---|
| Report | $ROOT/reports |
| Protocol | $ROOT/protocols |
| Characterization Files | $ROOT/particles/$PARTICLE_NAME/$CHAR_TYPE/ |
| Nanoparticle Entity Files | $ROOT/particles/$PARTICLE_NAME/nanoparticleEntity |
| Functionalizing Entity Files | $ROOT/particles/$PARTICLE_NAME/functionalizingEntity |
| Chemical Association Files | $ROOT/particles/$PARTICLE_NAME/chemicalAssociation |

| | |
|---|---|
| Composition Files | $ROOT/particles/$PARTICLE_NAME/compositionFile |

**Table 3-2: Uploaded File Directory Structure:** $ROOT is the file repository root specified by users as a caNanoLab build property `file.repository.dir`; $PARTICLE_NAME is dynamically generated when a user uploads files associated with a nanoparticle; $CHAR_TYPE is dynamically generated when users upload characterization files associated with a particular particle and a characterization type.

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. Adding the timestamp prevents files with the same name to overwrite each other.

## 3.7 AUTHENTICATION AND AUTHORIZATION

User authentication and authorization in the caNanoLab system are implemented using CSM API version 3.1. We made some customizations to add encryptions to passwords. This feature was not available at the time of integration, but is later made available in later releases of CSM. We are yet to upgrade caNanoLab to integrate with the latest CSM release. The CSM database objects (prefixed with `csm`) (shown in Figure 3-18) 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. We also made some customizations to the UPT tool such that an initial password is assigned to the user when the user is first created , and an administrator is no longer the one maintaining users' passwords. The administrator can only reset users' passwords by issuing an update an existing user in the UPT tool. Users are to maintain their own passwords through a page inside the caNanoLab application.

During the caNanoLab application start-up time, three default user groups are created: PI, Researcher and Public, where PI and Researcher groups are prefixed with an application owner (e.g. NCL), specified as a build property, `application.owner`. During application startup time, the PI group is automatically assigned to be able to submit protocols, nanoparticles and reports. In addition, when a PI user submits a new nanoparticle sample with a new source into the system, the new source is automatically created as a new user group. The new user group, the PI group and the Researcher group are all automatically assigned to have read access on this newly created nanoparticle sample. When the PI user marks the nanoparticle sample being visible to public within the caNanoLab application, public users would be able to access the nanoparticle without having to log in.

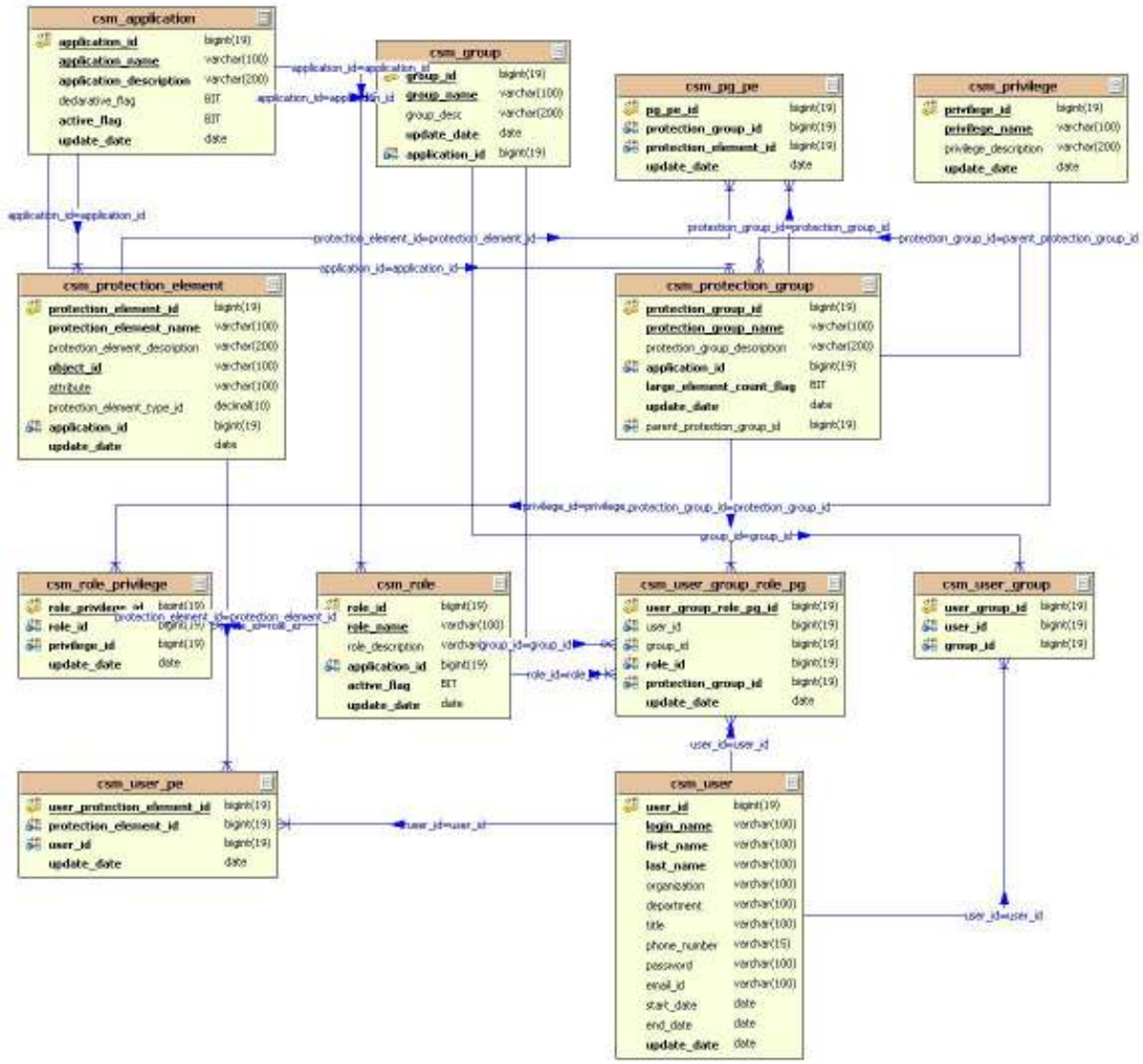


**Figure 3-18: CSM Data Model**

# 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. Users can also execute custom grid operations to retrieve public data made available by the grid operations. In release 1.4, the implementation of the caNanoLab grid service has been upgraded to caGrid 1.2 backed by the caCORE SDK 4.0. As of release 1.4, 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.

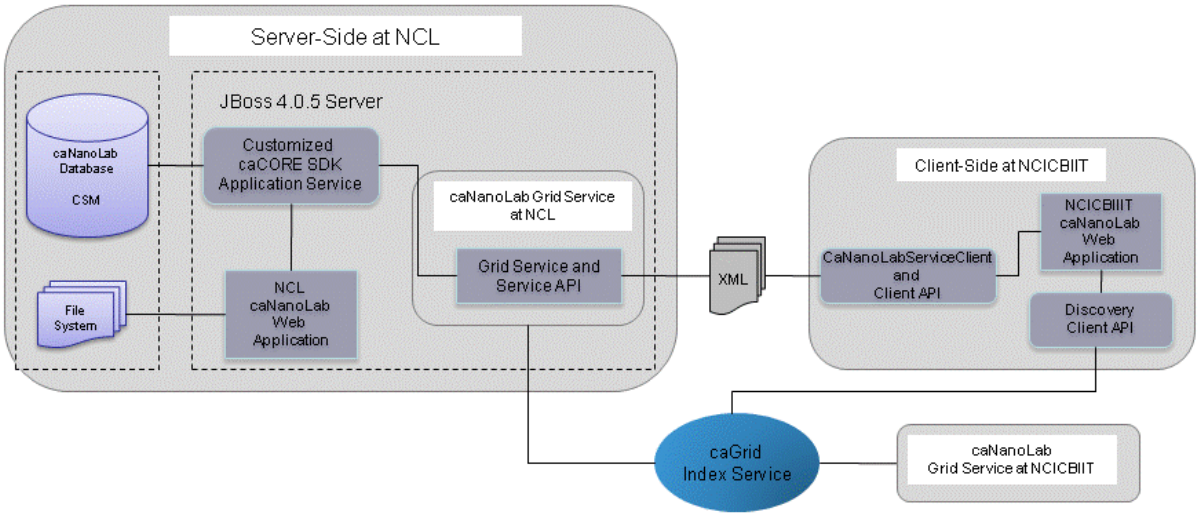


**Figure 4-1: caNanoLab Grid-Enabling Architecture Diagram**

When a user at the NCICBIIT web portal selects to query remote caNanoLab data residing 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 (provided by the caCORE SDK with caNanoLab customizations) is called to execute the HQL against the NCL database to retrieve

NCL nanoparticle data. Alternatively, when the NCL grid service receives a custom grid operation from the client, the grid service invokes the corresponding method in the grid service API, and the method in turn invokes a helper class (described in section 3.5.1) 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. The grid service takes the retrieved data objects and serializes them into XML documents and passes these XMLs to the `CaNanoLabServiceClient` object instantiated at NCICBIIT. The `CaNanoLabServiceClient` object takes the XMLs and deserialized them back into objects that the NCICBIIT caNanoLab web portal understands.

## 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 1.2 Introduce toolkit through local APIs provided by the caCORE SDK 4.0 backed caNanoLab data source. Through local Java APIs (newly available in caCORE SDK 4.0), the grid service is able to retrieve data directly from the associated data source in a more efficient manner than going through a remote Java API provided by a separate caCORE SDK generated web application, as in the case of the caNanoLab 1.1 grid service powered by caGrid 1.0 and caCORE SDK 3.1.

### 4.2.1 Custom Grid Operations

In release 1.4, we implemented many custom grid operations to satisfy the remote search use cases defined for the caNanoLab web portal. Most custom grid operations are added because the caGrid 1.2 infrastructure doesn't yet support inclusion of associations in the query results through CQL, and some associations defined in the caNanoLab object model are uni-directional. Some grid operations are added because for complex queries, CQLs are inefficient and hard to construct. Some grid operations are added to improve query performance. The following subsections describe each of these scenarios.

#### 4.2.1.1 *Uni-directional Associations*

CQLs as of caGrid 1.2, can only return attributes of an object of interest. All associated objects of an object of interest would have to be queried through separate CQLs. However, when the associations between two objects are uni-directionary, CQLs cannot be written to retrieve the uni-directionally associated objects of an object. For example, the operation `getActivationMethodByFunctionalizingEntityId` is added to retrieve the `ActivationMethod` object associated with a `FunctionalizingEntity` object, and the association from `FunctionalizingEntity` to `ActivationMethod` is uni-directional. Following is list of all such operations (all throw `RemoteException` and is remitted here for brevity):

```
public ActivationMethod getActivationMethodByFunctionalizingEntityId(
            String id)
public AssociatedElement getAssociatedElementAByChemicalAssociationId(
```

```
            String id)
public AssociatedElement getAssociatedElementBByChemicalAssociationId(
            String id)
public ChemicalAssociation[] getChemicalAssociationsByCompositionId(
            String id)
public DerivedBioAssayData[] getDerivedBioAssayDatasByCharacterizationId(
            String id)
public Function[] getFunctionsByFunctionalizingEntityId(String id)
public Function[] getInherentFunctionsByComposingElementId(String id)
public Instrument getInstrumentByInstrumentConfigurationId(String id)
public InstrumentConfiguration getInstrumentConfigurationByCharacterizationId(
            String id)
public Keyword[] getKeywordsByFileId(String id)
public Keyword[] getKeywordsByParticleSampleId(String id)
public LabFile getLabFileByDerivedBioAssayDataId(String id)
public LabFile[] getLabFilesByCompositionInfoId(String id, String className)
public ProtocolFile getProtocolFileByCharacterizationId(String id)
public SurfaceChemistry[] getSurfaceChemistriesBySurfaceId(String id)
public Target[] getTargetsByFunctionId(String id)
public DerivedDatum[] getDerivedDatumsByDerivedBioAssayDataId(
            String id)
```

#### 4.2.1.2 *Complex Queries*

In addition to the operations supporting the retrieval of uni-directional 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 these complex queries. As mentioned before, these grid operations would invoke helper classes to complete the query processing. These helper classes, also used in local queries, already contain HQLs to retrieve data based on complex search criteria from a caNanoLab data source. The following is a list of such grid operations:

```
public NanoparticleSample[] getNanoparticleSamplesBy(String particleSource,
            String[] nanoparticleEntityClassNames,
            String[] functionalizingEntityClassNames,
            String[] functionClassNames,
            String[] characterizationClassNames,
            String[] words)
public ProtocolFile[] getProtocolFilesBy(String protocolType,
            String protocolName,
            String protocolFileTitle)
public Report[] getReportsBy(String reportTitle,
            String reportCategory,
            String[] nanoparticleEntityClassNames,
            String[] functionalizingEntityClassNames,
String[] functionClassNames)
public String[] getCharacterizationClassNamesByParticleId(String id)
public String[] getFunctionalizingEntityClassNamesByParticleId(
            String id)
public String[] getFunctionClassNamesByParticleId(String id)
public String[] getNanoparticleEntityClassNamesByParticleId(String id)
```

#### 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 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 3-17, when a user clicks on the number of nanoparticles from a grid location, a set of remote queries would need be executed in order to retrieve all nanoparticles from that grid location and all the associated information (particle source, composition, function and characterizations) for these nanoparticles before the nanoparticle search results can be displayed in a table format as shown in Figure 4-2 below.

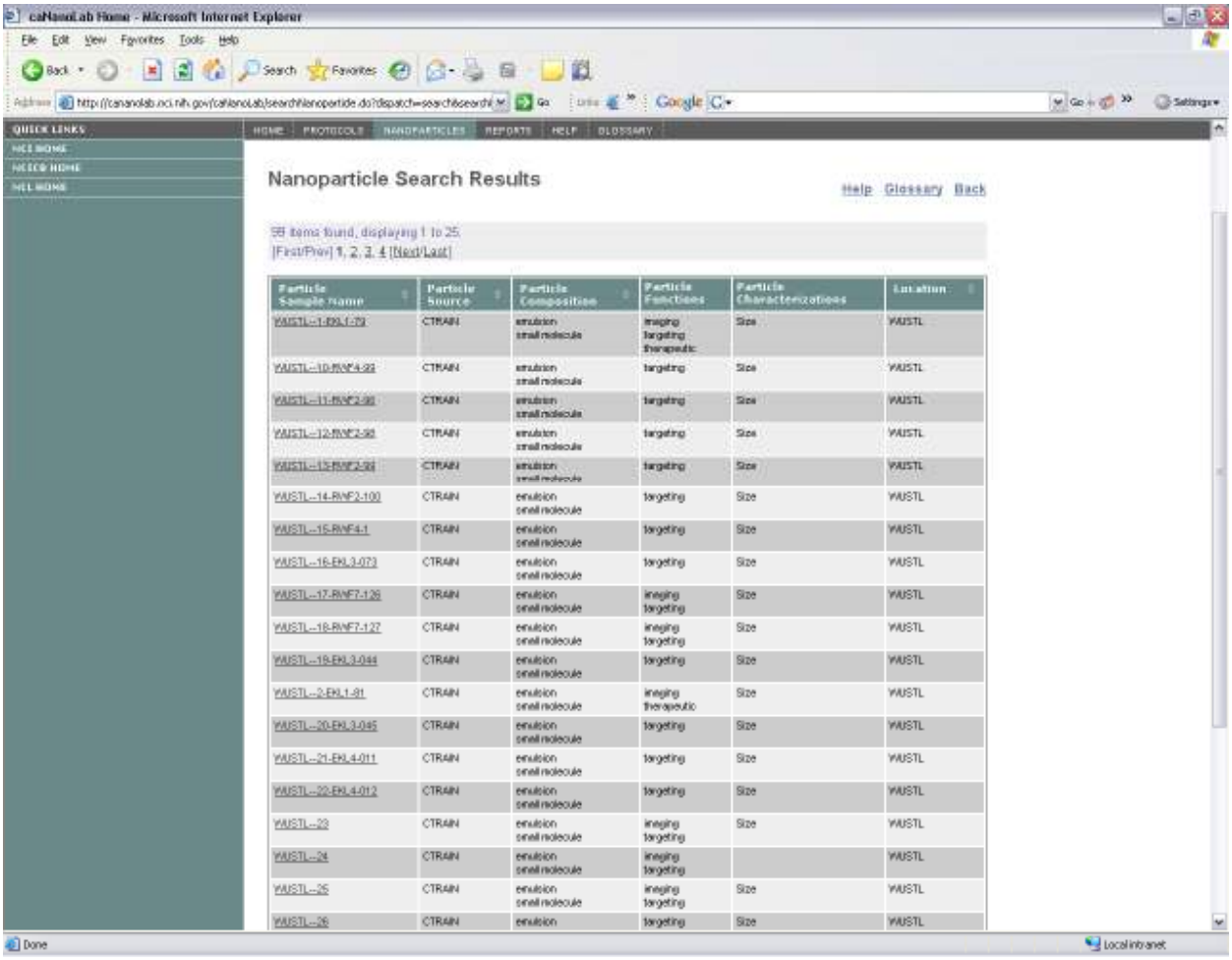


**Figure 4-2: Nanoparticle Search Results from a Grid Location**

In order to return the loaded nanoparticles back in a reasonable timeframe for displaying in the caNanoLab web application, we implemented an alternative grid operation that returns a String array instead of an array of `NaonparticleSample` objects as follows:

```
Public String[] getNanoparticleSampleViewStrs(String particleSource,
                  String[] nanoparticleEntityClassNames,
                  String[] functionalizingEntityClassNames,
                  String[] functionClassNames,
                  String[] characterizationClassNames,
                  String[] words)
```

On the server side, the nanoparticles are fully loaded and their information are translated into a 2D String matrix format specifically designed for displaying on the page shown in Figure 4-2. With this "hack", clients would only execute one grid operation, and grid query performance is dramatically improved. The obvious drawback of this approach is that if the view page format is to be changed, we'd have to adjust the implementation of this grid operation accordingly.

### 4.2.2 Public Filter

The caNanoLab grid service has been created as a 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. Given that the grid data can be retrieved through CQLs or through grid operations, we implemented the public filter in two different places.

As mentioned before, CQLs are processed by a query processor and then are translated into HQLs, we implemented a customized query processor `gov.nih.nci.cagrid.cananolab.service.CustomizedSDK4QueryProcessor` that is built on top of the default SDK query processor provided by the Introduce toolkit. In this customized query processor, we modified the implementation of the method `queryCoreService` such that additional logics are added to append a `where` clause to the translated HQLs when CQLs are for returning counts of objects, and additional logics are added to filter out non-public data when CQLs are for returning objects. The following methods in the class `gov.nih.nci.cagrid.cananolab.service.CaNanoLabServiceImpl` capture these additional logics used in the `queryCoreService` method:

```
List getPublicData(List rawObjects)
boolean isPublic(String dataId)
String modifyPublicHQL(String hql, String targetName,
            List<String> publicDataIdsAsNonNumbers,
            List<String> publicDataIdsAsNumbers)
List<Object> modifyPublicHQLParameters(List<Object> parameters,
            String targetName, List<String> publicDataIdsAsNonNumbers,
            List<String> publicDataIdsAsNumbers)
```

When custom grid operations are used to retrieve remote data, the query processor is not involved. We simply added the call to the method `getPublicData` shown above in the implementation of the grid operations to filter out non-public data before returning the data.

### 4.2.3 Remote File Retrieval

In caNanoLab 1.4, users can retrieve remote protocol files and reports from a grid location. 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.

In the 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 URL on the NCL production caNanoLab application to download a public report with id 31784 is: http://cananolab.abcc.ncifcrf.gov/caNanoLab/searchReport.do?dispatch=download&fileId=3178496&location=local. As long as the local caNanoLab application knows how to construct such a URL based on the remote public file ID, the local application would be able to make a direct HTTP link to the remote caNanoLab application to down the remote file content. To make this possible, we made the assumption that the remote caNanoLab application and the remote grid service are deployed to the same JBoss container and share the same virtual host name that make up the first part of the download URL.

# 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 | Cancer Center of Nanotechnology Excellence |
| CRUD | Create, Read, Update, Delete |
| CSM | Common Security Module |
| CSS | Cascading Style Sheet |
| DAO | Data Access Object |
| DTO | Data Transfer Object |
| EIS | Enterprise Information System |
| EVS | Enterprise Vocabulary Services |
| HQL | Hibernate Query Language |
| 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 |