Chemistry and OpenCMIS Technical Comparison

This is a technical comparison of the interfaces and classes present in both Chemistry and OpenCMIS.

Concepts

In Chemistry the session and the Connection are the same thing. The connection has different implementations depending on the way it’s connected to an underlying protocol. The connection implements methods from the high-level API, and also gives access to the low-level SPI implementing different methods.

In OpenCMIS the Session is a semi-generic context-like object (PersistentSessionImpl). Eventually, there will be two Session implementations. In the persistent model (almost) all changes are immediately passed to the repository. In the transient model all changes are cached until save() is called on the Session object. A Session can be "connected" using parameters to instantiate internally a low-level provider (CmisProvider). The provider holds configuration parameters that enable it to create a low-level SPI through a CmisSpiFactory. Through the SPI you can get to the various SPI *Service implementations.

Repository access

In Chemistry you get to a repository instance based on general repository parameters, and from it you can open connections with a username and password. The repository instance can be introspected (types, etc) without opening a session.

In OpenCMIS, you get a session factory, from which you open a session, from which you can get to the repository info (types, etc.). All connection parameters are passed to the createSession() method, including repository URL.

Registering a repository

- Chemistry:

  Map<String, Serializable> params = ...; // URL, optional user, password
  RepositoryService repositoryService = new APPRepositoryService(url, params);
  RepositoryManager.getInstance().registerService(repositoryService);

- OpenCMIS:

  No global registration. A JNDI-based method or dependency injection is suggested but not implemented.

Getting a repository / session factory

- Chemistry

  Repository repository = RepositoryManager.getInstance().getRepository("myrepo");

- OpenCMIS

  SessionFactory sessionFactory = SessionFactoryImpl.newInstance();

Getting a session / connection

- Chemistry

  Map<String, String> params = ...; // user, password
  Connection conn = repository.getConnection(params);

- OpenCMIS

  Map<String, String> params = ...; // URL, user, password
  Session session = sessionFactory.createSession(parameters);

Internal layer hierarchy (OpenCMIS)
(All classes and interfaces in bold are for public use. Everything else belongs to the internal machinery.)

- **SessionFactory**
  Interface of the session factory class.

- **SessionFactoryImpl**
  Factory class that creates Session objects from a given configuration.

- **PersistentSessionImpl**
  Implementation of the Session interface that follows the persistent model. Should be created with SessionFactoryImpl.

- **TransientSessionImpl**
  Implementation of the Session interface that follows the transient model. Should be created with SessionFactoryImpl.

- **CmisProviderHelper**
  Internal helper class that creates a CmisProvider object. It contains code that is shared by PersistentSessionImpl and TransientSessionImpl. It shouldn't be used by anybody else.

- **CmisProvider**
  The low-level client interface.

- **CmisProviderImpl**
  Implementation of the low-level client interface.

- **CmisProviderFactory**
  Factory class for CmisProvider objects. Although CmisProviderImpl can be instantiated directly, this factory sets some reasonable defaults and does a sanity check on the configuration. It is recommended to use this factory to create a CmisProvider object.

- **CmisSpi**
  Interface of the binding implementations. This interface is only interesting for binding developers. Applications use the CmisProvider or Session interfaces that hide the binding.

- **CmisAtomPubSpi**
  AtomPub binding implementation.

- **CmisWebServicesSpi**
  Web Services binding implementation.

From an application point of view it easy to use:

- If you want to use the client API, create a Session object with SessionFactoryImpl and don't bother about the rest.
- If you want to use the low-level provider API, create a CmisProvider object with CmisProviderFactory and don't bother about the rest.

### High-level APIs

From a connection/session you can get the root folder and express high-level operations

- **Chemistry**
  ```java
  Folder root = conn.getRootFolder();
  List<CMISObject> children = root.getChildren();
  ```

- **OpenCMIS**
  ```java
  Folder root = session.getRootFolder();
  PagingList<CmisObject> list = root.getChildren(1);
  ```

### Base object

Contains getters and setters for properties, with convenience methods.
Contains methods like delete() etc. that pass through to the SPI/provider.

- **Chemistry**
  The base interface is CMISObject. It flushes changes on save().

- **OpenCMIS**
  The base interface is CmisObject. It flushes property changes on updateProperties().
Specialized Objects

Implement additional object-oriented methods depending on the interfaces.

- **Chemistry**
  Folder, Document, Relationship, Policy

- **OpenCMIS**
  FileableCmisObject, Folder, Document, Relationship, Policy

Paging

- **Chemistry**
  ListPage: a page
  \[ \text{List} + \text{getHasMoreItems} + \text{getNumItems} \]
  Implemented by SimpleListPage. This is a data transport object.

- **OpenCMIS**
  PagingList: a list of pages which are themselves lists
  \[ \text{Iterable<List>} + \text{getNumItems} + \text{getMaxItemsPerPage} + \text{size} + \text{get(page)} \]
  AbstractPagingList is the base class. This is an active object that can fetch new pages by implementing a fetchPage() method that returns a FetchResult (which is equivalent to Chemistry’s ListPage). It also has a LRU cache for pages which is disabled by default.

Provider APIs

This is called “SPI” in Chemistry, and “Provider” in OpenCMIS.

Services interfaces

- **Chemistry**
  All CMIS services are implemented under the single interface SPI. The SPI uses classes and interfaces designed for Java.

- **OpenCMIS**
  From a provider you get the various CMIS services as different interfaces (RepositoryService, ObjectService, NavigationService, etc.) using getters. The interfaces and classes are generic and reflect the CMIS schema.

High-level vs low-level vs implementation

- **Chemistry**
  The high-level and SPI interfaces are mutualized (ex: org.apache.chemistry.RepositoryInfo).
  Florian> For some objects there are different interfaces on these two levels. For example, the step from ObjectEntry to CMISObject is comparable to OpenCMIS’ step from the provider API to the client API.
  Florian> JAXB objects will be necessary for Web Services, similar to OpenCMIS.

- **OpenCMIS**
  For the same concept OpenCMIS manipulates three different interfaces and their implementations:
  - the one in the high-level client API (ex: org.apache.opencmis.client.api.repository.RepositoryInfo, convenient access to data),
  - the one in the provider (ex: org.apache.opencmis.commons.provider.RepositoryInfoData, access to all extension points),
  - the one from JAXB (CmisRepositoryInfoType).

Common method parameters

- **Chemistry**
  The SPI bundles together a number of call parameters that are used often together: Inclusion contains properties and rendition filters, relationship inclusion, flags for allowable actions, policies, acls. An Inclusion is passed to the relevant SPI methods.

- **OpenCMIS**
  A default OperationContext on the session is used to specify these call parameters. A variant of the high-level methods taking an explicit OperationContext is also available. Furthermore, OperationContext controls the caching behavior of the objects retrieved by the call. In the provider interfaces everything is explicit, following the CMIS specification.

Object data

The base object contains information about one object: properties, allowable actions, relationships, renditions, etc.

- **Chemistry**
  ObjectEntry is the basic class.
  It also contains change info and path segments, depending on how it was retrieved.
OpenCMIS

ObjectData is the basic class. To provide it context, it is used by delegation is more complex constructions: ObjectInFolderData, ObjectInFolderContainer, ObjectInFolderList, ObjectParentData, ObjectList, etc. thus reflect the CMIS schema and allow access to all extension points.

Various enums

Relationship direction:

- **Chemistry**
  Defines them according to best Java use. For instance RelationshipDirection can be 'source', 'target', 'either' or null. There is no separate IncludeRelationships.

- **OpenCMIS**
  Mimicks JAXB. RelationshipDirection and IncludeRelationships are different.

Property type:

- **Chemistry**
  PropertyType is a class allowing definition of new types, for specialized backends.

- **OpenCMIS**
  PropertyType is an enum following JAXB.

Allowable actions:

- **Chemistry**
  AllowableActions is a set of QNames.

- **OpenCMIS**
  AllowableActions is a map from String (non-namespaced) to Boolean.