DataONE Architecture Documentation

Release 0.0.3

VDC Project, DataONE CCIT

November 16, 2009

CONTENTS

Change Log

20091113

- Numerous edits, some rearrangement of chapters

20091112

- General editing
- Setting up for PDF generation

20091110

- Incorporating many updates and edits
- Switch to Sphinx doc generator: easier to manage and better features

20090927

- Split usecases into separate document
- Added page breaks for rst2pdf

20090925

- Initial collation of all use cases into single document.
- Recast all diagrams to interaction diagrams using PlantUML automated tools

20090924

- Layout document structure, draft templates

This document describes portions of the DataONE infrastructure. Note that content is quite dynamic while the architecture design is being iterated for the first release.

Source of these documents is maintained in the project subversion repository.

A version of the entire document in PDF is also available for download.

CHAPTER

ONE

ABOUT THESE DOCUMENTS

These documents are generated using the excellent open source tools Sphinx for the overall document processing and generation, and PlantUML for the various UML diagrams.

1.1 Editing Content

All of these documents are formatted using reStructuredText, which is simply well formatted plain text. In many respects it is similar to the plain text formatting used in wiki pages, though enriched by the many nice features of the Sphinx document builder.

Formatting guidelines for Sphinx are available at http://sphinx.pocoo.org/contents.html

In general, it is best to edit in UTF-8 using unix style new lines. Note that white space is significant and is used primarily for determining which formatting sections different blocks of text belong to.

> **Warning:** Do not use tabs! Set your editor to replace tabs with spaces. Mine is setup with a tab = two spaces. If you do use tabs, then things can get a little screwy when others using spaces only edit your content, so it's best to use one or the other for indenting - and spaces have already been selected for these documents.

Any plain text editor will do the job, though my preference is TextMate on OS X (with the reStructuredText bundle installed it offers a nice preview feature). jEdit is another nice editor that I've used in the past, and there are a few plugins floating around to assist with reStructuredText formatting.

Follow the normal procedures for working with the subversion repository. After checking in, your edits should appear in the built out version of the docs within five minutes.

1.2 Building the Documentation

Sphinx and PlantUML are required to build the design documents from the source. Both of these have a dependency on graphviz for diagram generation.

Graphviz is available pre-built for many flavors of linux, windows, OS X, and can be installed on pretty much anything else that runs a c compiler. To install on ubuntu:

```
$ sudo apt-get install graphviz
```

Sphinx can be installed on any system with Python using setuptools

```
sudo easy_install -U Sphinx
```

Installing PlantUML is simply a matter of downloading the plantuml.jar file from the PlantUML site and putting it somewhere it can be executed. The script I use for running PlantUML is:

```
#!/bin/sh
CONFIG=/Users/vieglais/Projects/DataONE/svn/software/tools/trunk/docutils/plantuml.conf
java -jar ~/bin/plantuml.jar -config $CONFIG $1 $2 $3 $4 $5
```

The plantuml.conf file is in the DataONE subversion source tree.

To actually build a HTML copy of the documents:

```
$ cd ~/Projects/DataONE/svn/documents/Projects/VDC/docs/service-api/api-documentation
$ plantuml source source/UseCases
$ make html
```

To build a PDF copy, you will need to install latex and its dependencies. Then run:

```
$ make latex; make -f build/latex/Makefile all-pdf
```

CHAPTER

TWO

OVERVIEW

The DataONE system is fundamentally composed of three major components:

Member Nodes Existing and new data repositories. These systems offer heterogeneous implementations across a variety of data types and support a variety of metadata types. All data is stored on Member Nodes, and the DataONE infrastructure assists with coordinating distribution of data between MNs.

Coordinating Nodes The operational core of DataONE. The coordinating nodes catalog data and metadata, control the dispersion of data across Member Nodes, keep a replica of all metadata, offer services for distributed identity, and offer search and discovery capabilities.

Investigator Toolkit Initially provides a set of libraries for interacting with DataONE CNs and MNs (i.e. a "client library"). As the project develops, the ITK will grow to contain a rich set of applications and tools for interacting with the content available in DataONE (and potentially other Data Nets).

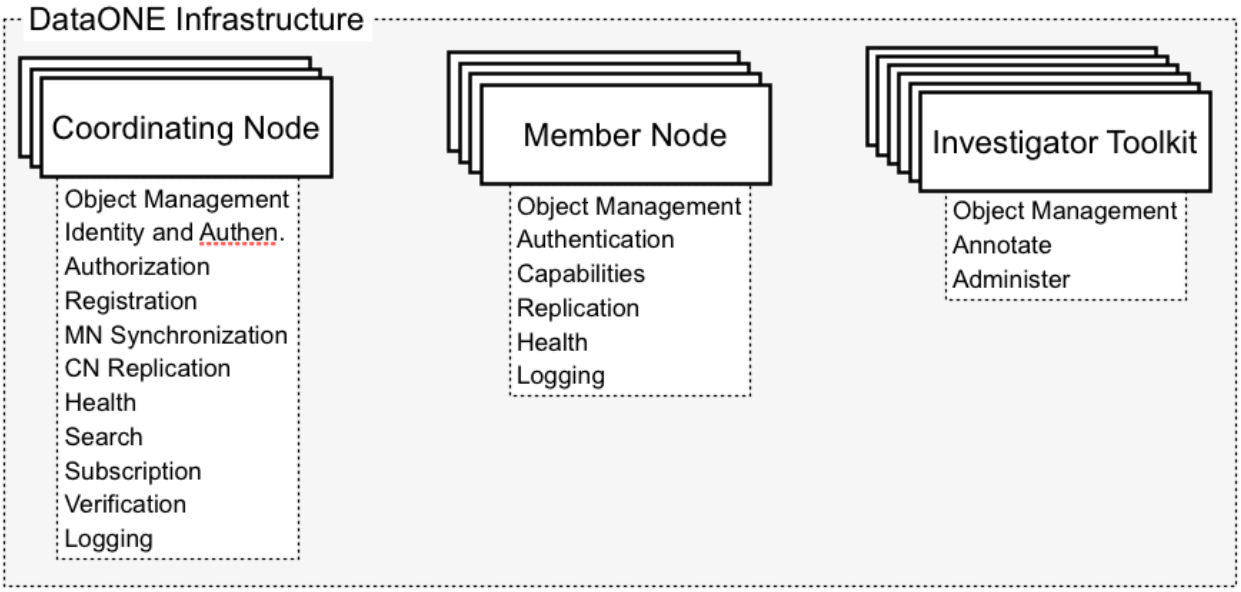


Figure 2.1: General overview of the major components of DataONE and the services they implement. It is invisaged that there will be very few CNs (3 in the first year), many more MNs (hundreds), and even more instances of the ITK.

In the initial version of DataONE, the system will be composed of three instances of Coordinating Nodes and at least three instances of Member Nodes.

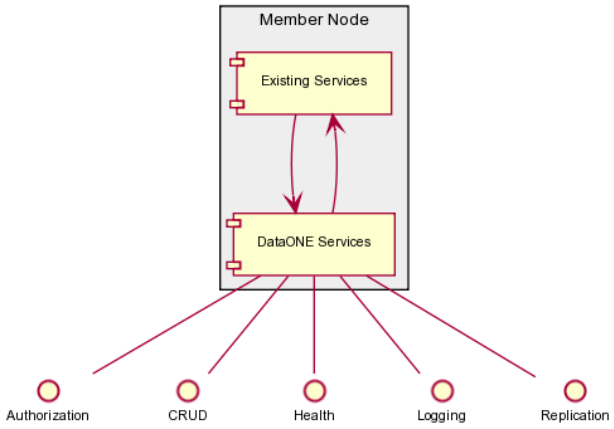


Figure 2.2: Simple diagram of the MN structure. DataONE service APIs are implemented by a component that interacts with the existing APIs of the Member Node.

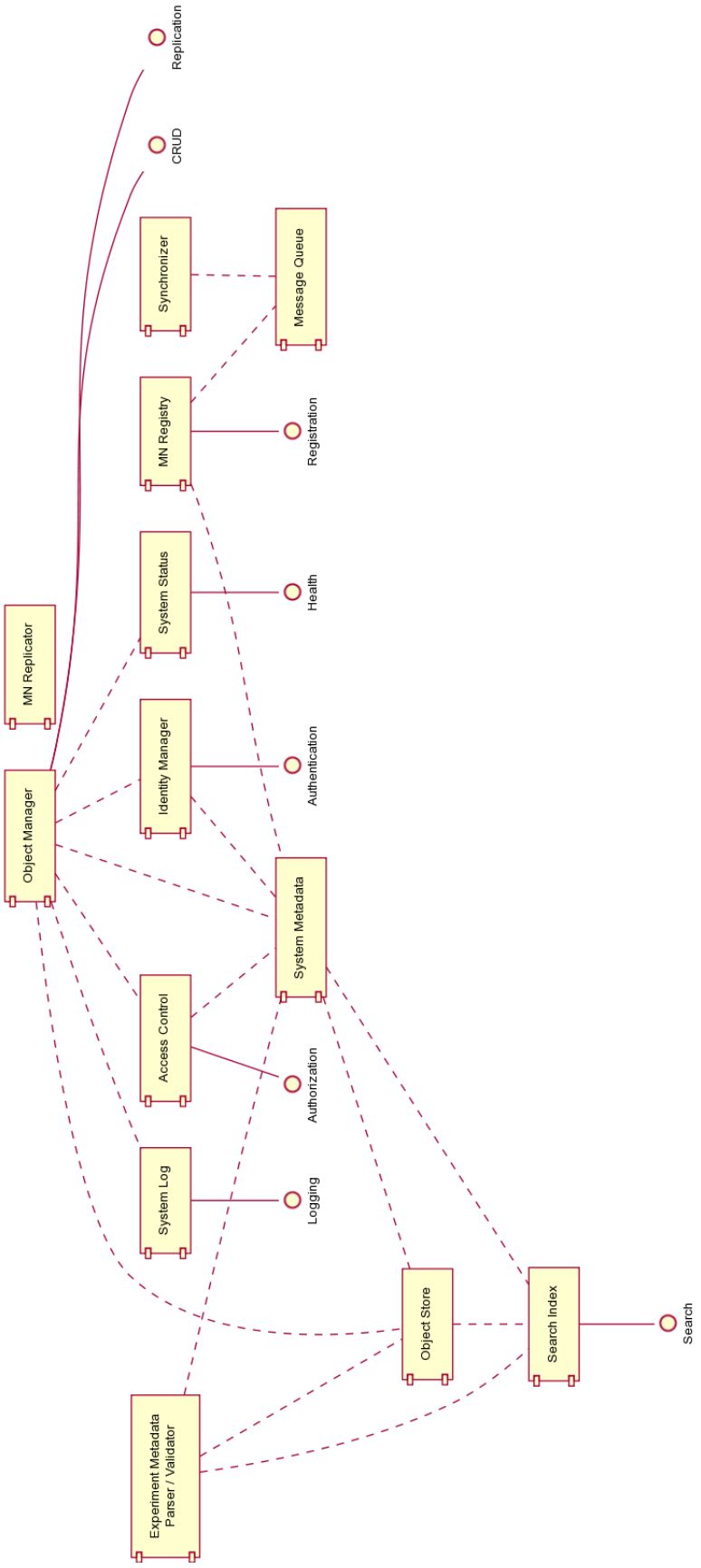


Figure 2.3: An estimate of CN components and the services they provide.

2.1 What is Data (DataONE Perspective)?

Author Initial version by Dave Vieglais <vieglais@ku.edu>

Status DRAFT - comments, edits, even complete rewrites welcome and expected

This document describes the concept of "data" within the first iteration of the DataONE system.

2.1.1 Overview

Data in the context of DataONE is a discrete unit of digital content that is expected to represent information obtained from some experiment or scientific study. The data is accompanied by "experiment metadata", which is a separate unit of digital content that describes properties of the data. The combination of data and accompanying metadata in DataONE is called a "data package". The data package and its components have identifiers that are guaranteed to be unique within all the DataONE infrastructure.

In the initial version of DataONE, data are treated as opaque sets of bytes and are stored on Member Nodes (MN). A copy of the experiment metadata is held by the Coordinating Nodes (CN) and is parsed to extract attributes that enable the discovery process.

This behavior is likely to change in the future to enable processing of the data with operations such as translation (e.g. for format migration), extraction (e.g. for rendering), and merging (e.g. to combine multiple instances of data that are expressed in different formats). Such operations rely upon a stable, accessible framework supporting reliable data access, and so are targeted after the initial requirements of DataONE are met and the core infrastructure is demonstrably robust.

2.1.2 Metadata Types

The following metadata formats are of interest to the DataONE project for the initial version and are representative of the types of content that will need to be stored and parsed.

In all cases the descriptive text was retrieved from the URL provided with the description. Hence, look at the documentation provided at the URL for authoritative descriptions of the particular standard.

Dublin Core

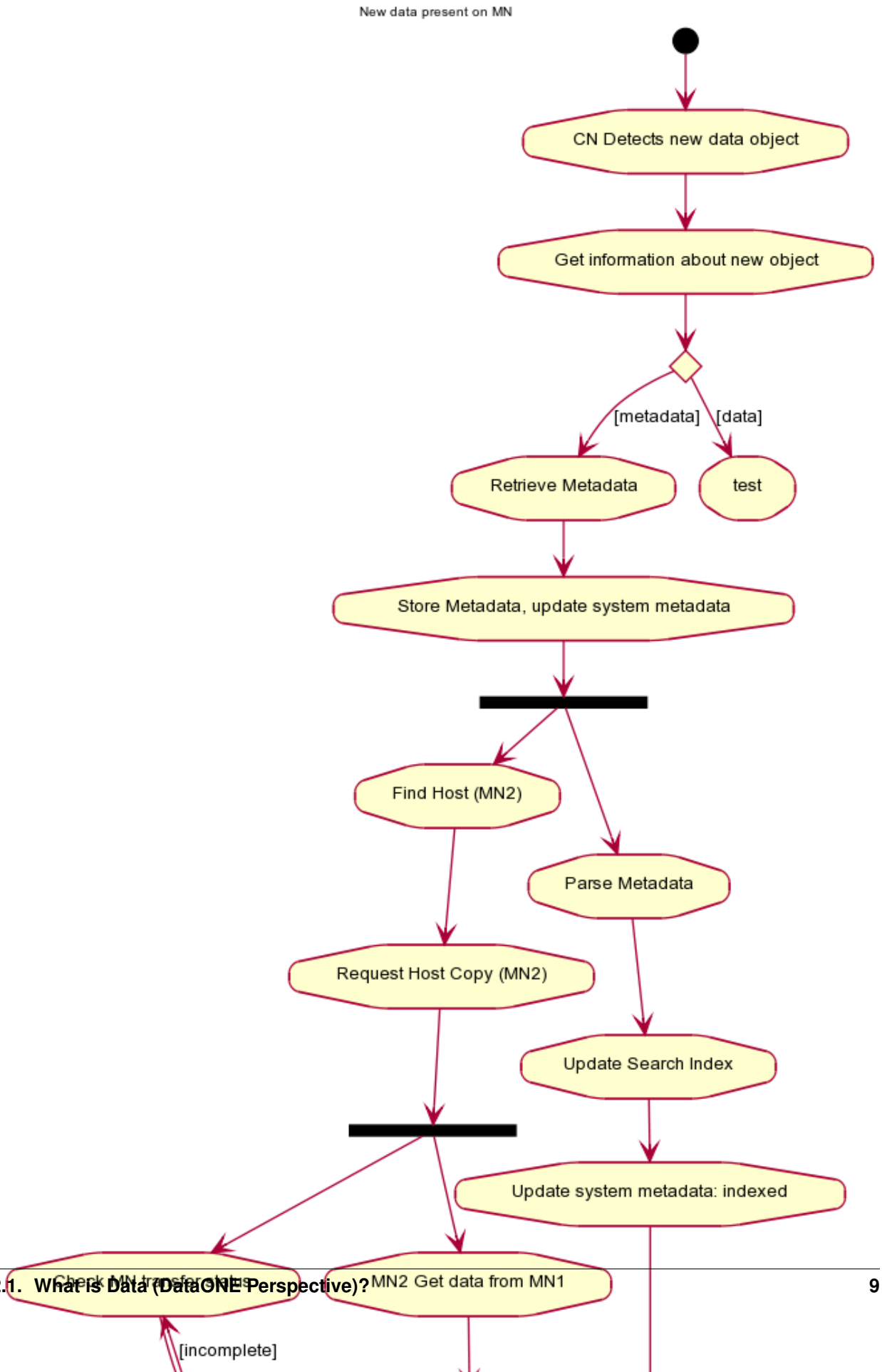
- http://dublincore.org/documents/dces/

The Dublin Core Metadata Element Set is a vocabulary of fifteen properties for use in resource description.

Darwin Core

- http://rs.tdwg.org/dwc/index.htm

The Darwin Core is body of standards. It includes a glossary of terms (in other contexts these might be called properties, elements, fields, columns, attributes, or concepts) intended to facilitate the sharing of information about biological diversity by providing reference definitions, examples, and commentaries. The Darwin Core is primarily based on taxa, their occurrence in nature as documented by observations, specimens, and samples, and related information. Included are documents describing how these terms are managed, how the set of terms can be extended for new purposes, and how the terms can be used. The Simple Darwin Core [SIMPLEDWC] is a specification for one particular way to use the terms - to share data about taxa and their occurrences in a simply structured way - and is probably what is meant if someone suggests to "format your data according to the Darwin Core".


New data present on MN
CN Detects new data object
Get information about new object
[metadata]
[data]
Retrieve Metadata
test
Store Metadata, update system metadata
Find Host (MN2)
Parse Metadata
Request Host Copy (MN2)
Update Search Index
Update system metadata: indexed
Check MN transfer status
MN2 Get data from MN1
[incomplete]

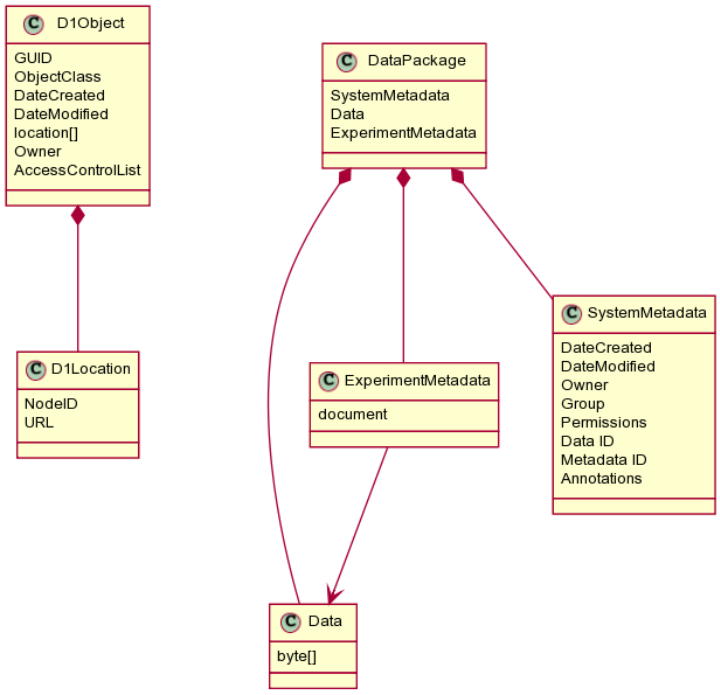


Figure 2.5: Sketch of a data package. System metadata attributes are likely to change significantly.

EML

- http://knb.ecoinformatics.org/software/eml

Ecological Metadata Language (EML) is a metadata specification developed by the ecology discipline and for the ecology discipline. It is based on prior work done by the Ecological Society of America and associated efforts (Michener et al., 1997, Ecological Applications). EML is implemented as a series of XML document types that can by used in a modular and extensible manner to document ecological data. Each EML module is designed to describe one logical part of the total metadata that should be included with any ecological dataset.

FGDC CSDGM

- http://www.fgdc.gov/metadata/geospatial-metadata-standards

The Content Standard for Digital Geospatial Metadata (CSDGM), Vers. 2 (FGDC-STD-001-1998) is the US Federal Metadata standard. The Federal Geographic Data Committee originally adopted the CSDGM in 1994 and revised it in 1998. According to Executive Order 12096 all Federal agencies are ordered to use this standard to document geospatial data created as of January, 1995. The standard is often referred to as the FGDC Metadata Standard and has been implemented beyond the federal level with State and local governments adopting the metadata standard as well.

```
-bio
(word document available for descriptions, Matt has XSD of FGDCbio)
(excel spreadsheet listing mapping,
 xslt: EML->FGDC (lossy), FGDC->EML)
(mapping available for EML -> DC (Duane))
```

GCMD DIF

Directory Interchange Format

- http://gcmd.nasa.gov/User/difguide/difman.html

The DIF does not compete with other metadata standards. It is simply the "container" for the metadata elements that are maintained in the IDN database, where validation for mandatory fields, keywords, personnel, etc. takes place.

The DIF is used to create directory entries which describe a group of data. A DIF consists of a collection of fields which detail specific information about the data. Eight fields are required in the DIF; the others expand upon and clarify the information. Some of the fields are text fields, others require the use of controlled keywords (sometimes known as "valids").

The DIF allows users of data to understand the contents of a data set and contains those fields which are necessary for users to decide whether a particular data set would be useful for their needs.

- Mapping to DC available at http://gcmd.nasa.gov/Aboutus/standards/dublin_to_dif.html

ISO 19137

http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=32555

ISO 19137:2007 defines a core profile of the spatial schema specified in ISO 19107 that specifies, in accordance with ISO 19106, a minimal set of geometric elements necessary for the efficient creation of application schemata.

It supports many of the spatial data formats and description languages already developed and in broad use within several nations or liaison organizations.

NEXML

http://nexml.org

The NEXUS file format is a commonly used format for phylogenetic data. Unfortunately, over time, the format has become overloaded - which has caused various problems. Meanwhile, new technologies around the XML standard have emerged. These technologies have the potential to greatly simplify, and improve robustness, in the processing of phylogenetic data:

Water ML

http://his.cuahsi.org/wofws.html

Web services are computer applications that interact with and exchange information with other applications over the internet. The CUAHSI-HIS uses a family of web services, called WaterOneFlow, that have been developed as a standard mechanism for the TRANSFER of hydrologic data between hydrologic data servers (databases) and users. Web services streamline the often time consuming tasks of extracting data from a data source, transforming it into a usable format and loading it in to an analysis environment. Web services format the data as XML and the specific variety of XML that is generated by the WaterOneFlow web services is known as CUAHSI WaterML. The specifics of WaterML are documented in the OGC™ Discussion Paper, CUAHSI WaterMLWeb services are computer applications that interact with and exchange information with other applications over the internet. The CUAHSI-HIS uses a family of web services, called WaterOneFlow, that have been developed as a standard mechanism for the TRANSFER of hydrologic data between hydrologic data servers (databases) and users. Web services streamline the often time consuming tasks of extracting data from a data source, transforming it into a usable format and loading it in to an analysis environment. Web services format the data as XML and the specific variety of XML that is generated by the WaterOneFlow web services is known as CUAHSI WaterML. The specifics of WaterML are documented in the OGC™ Discussion Paper, CUAHSI WaterML.

Genbank internal format

http://www.ncbi.nlm.nih.gov/Sitemap/samplerecord.html

ISO 19115

- http://en.wikipedia.org/wiki/ISO_19115

ISO 19115 "Geographic Information - Metadata" is a standard of the International Organization for Standardization (ISO). It is a component of the series of ISO 191xx standards for Geospatial metadata. ISO 19115 defines how to describe geographical information and associated services, including contents, spatial-temporal purchases, data quality, access and rights to use. The standard defines more than 400 meta data elements, 20 core elements.

- NA profile
- bio profile
- marine community metadata profile
- WMO profile

Dryad Application Profile

https://www.nescent.org/wg_dryad/Metadata_Profile

The Dryad metadata team has developed a metadata application profile based on the Dublin Core Metadata Initiative Abstract Model (DCAM) following the Dublin Core guidelines for application profiles. The Dryad application profile is being developed to conform Dublin Core Singapore Framework, a framework aligning with Semantic Web development and deployment.

ADN

- http://www.dlese.org/Metadata/adn-item/

The purpose of the ADN (ADEPT/DLESE/NASA) metadata framework is to describe resources typically used in learning environments (e.g. classroom activities, lesson plans, modules, visualizations, some datasets) for discovery by the Earth system education community.

GML Profiles

- http://en.wikipedia.org/wiki/Geography_Markup_Language#Profile

GML profiles are logical restrictions to GML, and may be expressed by a document, an XML schema or both.

NetCDF-CF-OPeNDAP

- http://opendap.org/
- http://www.oceanobs09.net/work/cwp_proposals/docs/100_Hankin_StandardsOceanDataInteroperability_CWPprop.doc

DDI

- Data Documentation Initiative
- http://www.ddialliance.org/

"The Data Documentation Initiative is an international effort to establish a standard for technical documentation describing social science data. A membership-based Alliance is developing the DDI specification, which is written in XML."

MAGE

- MicroArray and Gene Expression
- http://www.mged.org/Workgroups/MAGE/mage.html

"The group aims to provide a standard for the representation of microarray expression data that would facilitate the exchange of microarray information between different data systems."

ESML

- Earth Science Markup Language
- http://esml.itsc.uah.edu/

"Earth science data is archived and distributed in many different formats varying from character format, packed binary, "standard" scientific formats to self-describing formats. This heterogeneity results in data-application interoperability problems for scientific tools. The Earth Science Markup Language (ESML) is an elegant solution to this problem. ESML is an interchange technology that enables data (both structural and semantic) interoperability with applications without enforcing a standard format within the Earth science community."

CSR

- Cruise Summary Report
- http://www.oceanteacher.org/oceanteacher/index.php/Cruise_Summary_Report_%28CSR%29
- Oceanographic cruise data standard

The Cruise Summary Report (CSR), previously known as ROSCOP (Report of Observations/Samples Collected by Oceanographic Programmes), is an established international standard designed to gather information about oceanographic data. ROSCOP was conceived in the late 1960s by the IOC to provide a low level inventory for tracking oceanographic data collected on Research Vessels.

The ROSCOP form was extensively revised in 1990, and was re-named CSR (Cruise Summary Report), but the name ROSCOP still persists with many marine scientists. Most marine disciplines are represented in ROSCOP, including physical, chemical, and biological oceanography, fisheries, marine contamination/pollution, and marine meteorology. The ROSCOP database is maintained by ICES

2.1.3 Extracted Search Attributes

Data that exists within the DataONE system is always accompanied by experiment metadata in one of the supported metadata schemes (e.g. Dublin Core, EML, or another supported scheme) and expressed in one of the supported formats (e.g. XML, RDF-XML). The experiment metadata is also replicated with the data (where the Member Node (MN) supports the format and scheme) and additional copies are replicated across the Coordinating Nodes (CNs). The experiment metadata is parsed by the CNs to extract information for verification the content meets minimal metadata content and consistency requirements, and to support search operations.

The data and experiment metadata are tracked by additional metadata (the "system metadata") that exists only within the DataONE system . The combination of the three units is called a DataONE Package.

Essential Elements

- PI / Author
  - name authority service is desired for some control over names appearing in metadata
  - but for search, advantages to keep simple text representation
  - goal of gradual increase in specificity
- Keyword (uncontrolled keywords)
- Key concept -key concepts drawn from a set of ontologies
  - term
  - namespace
- Spatial bounding box (largest bounding rectangle)
- Spatial window (series of spatial envelopes representative of the spatial locations of where the data is collected from / relevant to.).
  - Spatial features (points, bounding boxes, polygons)
  - centroid
  - bounding box
  - polygon
  - (not largest extent)
  - (need to resolve the semantics of the bounding box search - e.g. if centroids are recorded but fall outside of bounding box search)
- Named places
  - term
  - type
  - context (Columbus OH, Columbus GA)
  - namespace of gazeteer
- Temporal window
  - Relative terms (e.g. paleo periods) need to be supported
  - Date ranges
  - Temporal coverage of the data set (e.g. searches - during, before, after)

- Full text search / Text search on abstract

Desirable Elements

- Title (2)
- Type of data (format) (2)
  - original data
  - summarized versions
  - method used for processing (to generate summary, or original data)
  - Resource type (spatial, models, observations, web service, ...)
- Scientific variables (from a controlled vocabulary) (1)
- Domain of data (physics, environmental, ...) (1)
- Biological taxonomic extents (1)
- Search by publication (1)

Some Others

- Generator of data (instrument, application)
- Related data (data sets, publications)
- Quality / level of curation
- Organizations involved in study
- Size of data (bytes)
- Number / location of replicas
- Dimensionality of data
- Units of measure (for sci variables)
- Identifier (GUID)
- Temporal coverage, low priority
  - (publication date)
  - creation date
  - last modified date
- Permissions on objects (e.g. available to read by user)

2.1.4 Structure of a Data Package

In the context of the D1 system, a unit of information is known as a "DataONE Package" (or D1 package, or D1P). A DataONE Package consists of:

- A discrete unit of digital content (the "data")
- Metadata describing the digital content (the "experiment metadata")
- D1 specific information about the package (the "system metadata")

Each DataONE Package has it's own unique identifier (guaranteed to be globally unique within the DataONE system). Additionally, each component of the D1 package has it's own unique identifier. A user of the DataONE system may thus retrieve the data, experiment metadata, or D1 metadata through the DataONE service interface `CN_crud.get()` method.

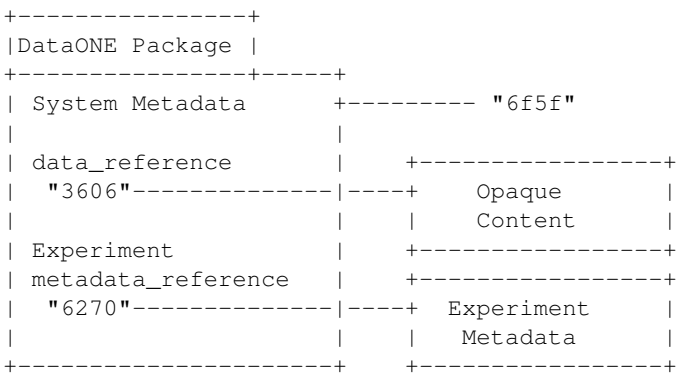


Figure 2. Conceptual structure of a DataONE Package with arbitrary identifiers associated with each element.

In the example presented in figure 2, the object with identifier "6f5f" refers to the system metadata of the data package. The system metadata contains information about the data package, including the location of it's content (the data and the experiment metadata identifiers) and other attributes such as the date and time the object first appeared in the DataONE system, last time modifications were made to the data package (the package may be modified, the content may not), the size of the components, ownership information, annotation references, and so forth.

Note: The identifiers used in these examples are random strings and are used for informational purposes only. They are not meant to endorse or imply any particular scheme.

DataONE System Metadata Elements

An instance of system metadata contains:

DateCreated Date and time (UTC) the instance was created in the DataONE system.

DateModified Date and time (UTC) the instance was last modified.

Owner Identifier of the owner of the instance.

Group Identifier of the group this instance belongs to. (Note: list of groups, pointer to a list of groups?)

Permissions Who can read, write the instance (similar to file permissions)

Data ID Reference to the data using the system wide identifier scheme

Metadata ID Reference to the experiment metadata for the data.

Annotations List of references to annotations about the data, experiment metadata, or system metadata. Each annotation would be some sort of triple - source, target and relation.

2.1.5 Accessing Content

Packages are accessed through the DataONE CRUD APIs which are implemented on Member Nodes and Coordinating Nodes.

The following text assumes a simple REST interface by way of example. The URL pattern for accessing content (`CN_crud.get()`) is:

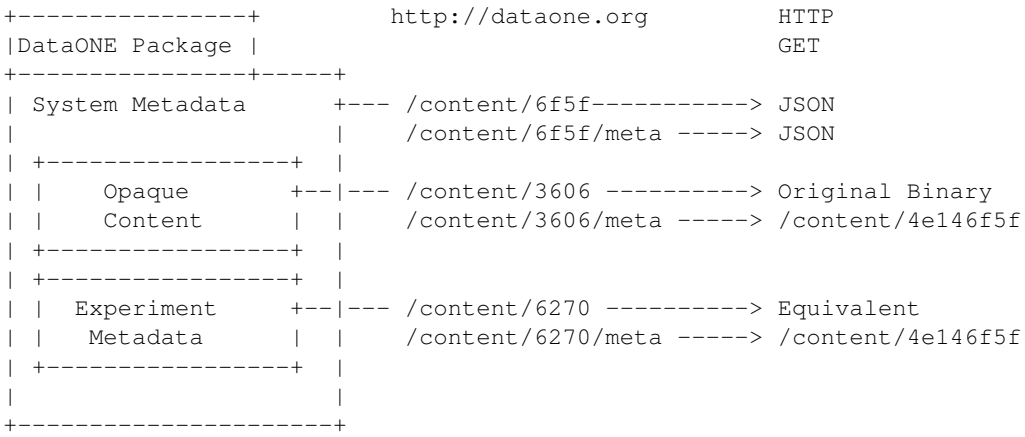
```
HTTP GET target=http://dataone.org/content/<identifier>/
```

The content type returned depends on the target:

Data Returns an exact copy of the bytes originally stored.

Experiment Metadata Returns a copy of the metadata document in the original markup.

System Metadata Returns a document expressed in JSON notation by default (other formats may be specified)



Some information about an identifier (the content size and mime-type) can be discovered prior to (or instead of) retrieval by invoking a HTTP HEAD operation on the same target URL:

```
HTTP HEAD target=http://dataone.org/content/<identifier>/
```

The `content-location` response header of the HEAD request always refers to the system metadata content which in turn always contains references to related components. That is, performing HEAD request on "3606" would return the size, age etc of the data object, but content-location would contain a URL that would resolve to the system metadata "6f5f" for the object.

Todo

The operational consequences of doing this do not appear to be well defined in the HTTP spec. Need to test against various browsers and libraries to evaluate behavior.

Metadata about any identifier used in the DataONE system may be retrieved through the HTTP operation:

```
HTTP GET target=http://dataone.org/content/<identifier>/meta/
```

In this case, the returned document will always be the DataONE system metadata.

Todo

The URLs used here are place holders and may not represent the actual access URLs used in the DataONE system.

2.1.6 Mutability

Data and experiment metadata are immutable for the first version of the DataONE system. As such, resolving the identifiers assigned to the data or the experiment metadata will always resolve to the same stream of bytes.

Todo

Byte stream equivalence of replicated experiment metadata would require that MNs record an exact copy of the metadata document received during replication operations in addition to the content that would be extracted and stored as part of the normal (existing) operations of a MN. Is this a reasonable requirement for MNs? Since MNs are required to store a copy of data, it seems reasonable to assume a copy of the metadata can be stored as well.

The DataONE `CN_crud.update()` method will fail if attempting to modify an instance of data or experiment metadata. Hence the update operation is only valid for the system metadata portion of a data package.

Todo

Is this reasonable? Likely that data owners may want to modify content after adding to DataONE. Perhaps allow a time period where data and metadata can be modified, after which it becomes more difficult (e.g. requiring administrator intervention or consensus from trusted authority)?

Deletion of content is only available to DataONE administrators (perhaps a curator role is required?).

Todo

Define the procedures for content deletion - who is responsible, procedures for contacting authors, timeliness of response.

2.1.7 Data Endianness

The data component of a D1 package is opaque to the D1 system (though this may change in the future), and so the endianness of the content does not affect operations except that it must be preserved. However, processing modules may utilize content from D1 and may be sensitive to the byte ordering of content. As such, the endianness of the data content should be recorded in the user supplied metadata (the experiment metadata), and where not present SHOULD be assumed to be least significant byte first (LSB, or small-endian).

Todo

Describe how endianness is specified in various experiment metadata formats.

2.1.8 Longevity

An original copy of the data is maintained for a long as practicable (ideally, the original content is never deleted). Derived copies of content, such as might occur when a new copy of a data object is created to migrate to a different binary format (e.g. an Excel 1.0 spreadsheet translated to Open Document Format) always create a new data object that will be noted as an annotation recorded in the system metadata of the data package.

2.1.9 Metadata Character Encoding

All metadata, including the experiment metadata and D1 package metadata MUST be encoded in the UTF-8 encoding. The DataONE `CN_crud.create()` and `CN_crud.update()` methods always expect UTF-8 encoded information, and so content that contains characters outside of the ASCII character set should be converted to UTF-8 through an appropriate mechanism before adding to DataONE.

2.1.10 Metadata Minimal Content

Experiment metadata MUST contain a minimal set of fields to be accepted by the DataONE system.

Todo

List and define the minimal set of fields with examples. A starting point would be the union of the required search properties and the information required for accurate citation.

2.2 Globally Unique Identifiers in DataONE

Globally Unique Identifiers (GUIDs) are utilized by many data providers to provide pointers to content. These identifiers can take many different forms from auto generated, essentially random character strings, to more well defined strings such as the URIs utilized by LSIDs and DOIs. The former typically require some external, well known system to locate the object referenced by the identifier. The later typically utilize some well known protocol that in a manner similar to HTTP URLs, a client application (i.e. an application that supports the protocol of the particular identifier scheme) is able to retrieve the object (and typically some metadata about the object) that the identifier refers to.

The entire set of potential Member Nodes is likely to utilize a wide range of identifier schemes, and it is unlikely that a given type of Member Node will change the identifier scheme they are utilizing. Hence, DataONE can not resort to reliance or promotion of any one scheme without risking the exclusion of potential participants. As a result, DataONE must be identifier agnostic.

DataONE will replicate data and associated metadata across many Member Nodes, and it is likely that not all MNs will support the same GUID scheme. This leads to the situation where in order to replicate content across MNs, there will need to be some sort of mapping between identifiers utilized by the different systems.

2.2.1 Suggested Strategy

1. DataONE supports all identifier schemes where the ID can be represented as a unicode string (this should be any identifier).
2. The original identifier first assigned by a Member Node is the identifier promoted as the authoritative identifier for that content. Other identifiers that may be assigned by MNs that don't support the original scheme will be mapped to the original.
3. If the original MN discontinues participation in DataONE, then the identifier originally used remains as the authoritative identifier.
4. Any identifiers in use by the DataONE system can be resolved at any node (CN or MN). A caching system (e.g. memcached) should be used to improve resolution performance (can be primed with existing IDs).

This strategy will enable the use of any identifier that can be represented by a string, and will persist the original identifier for the object regardless of what happens to the originating Member Node.

An obvious problem with this strategy though is that a particular object may have multiple identifiers associated with it, however since the original is persisted, it will be the primary identifier by which that content will be referenced, regardless of which node the object is located on.

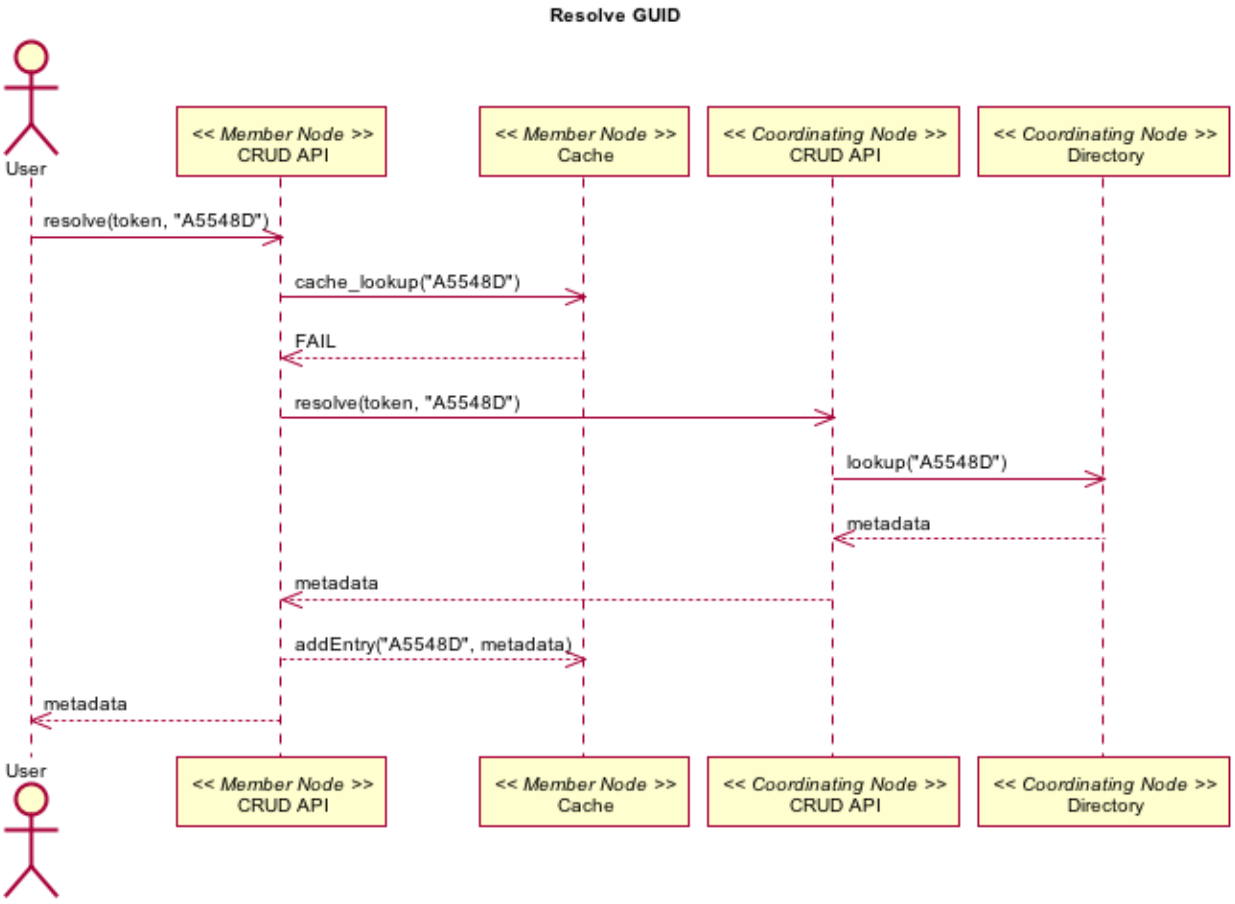
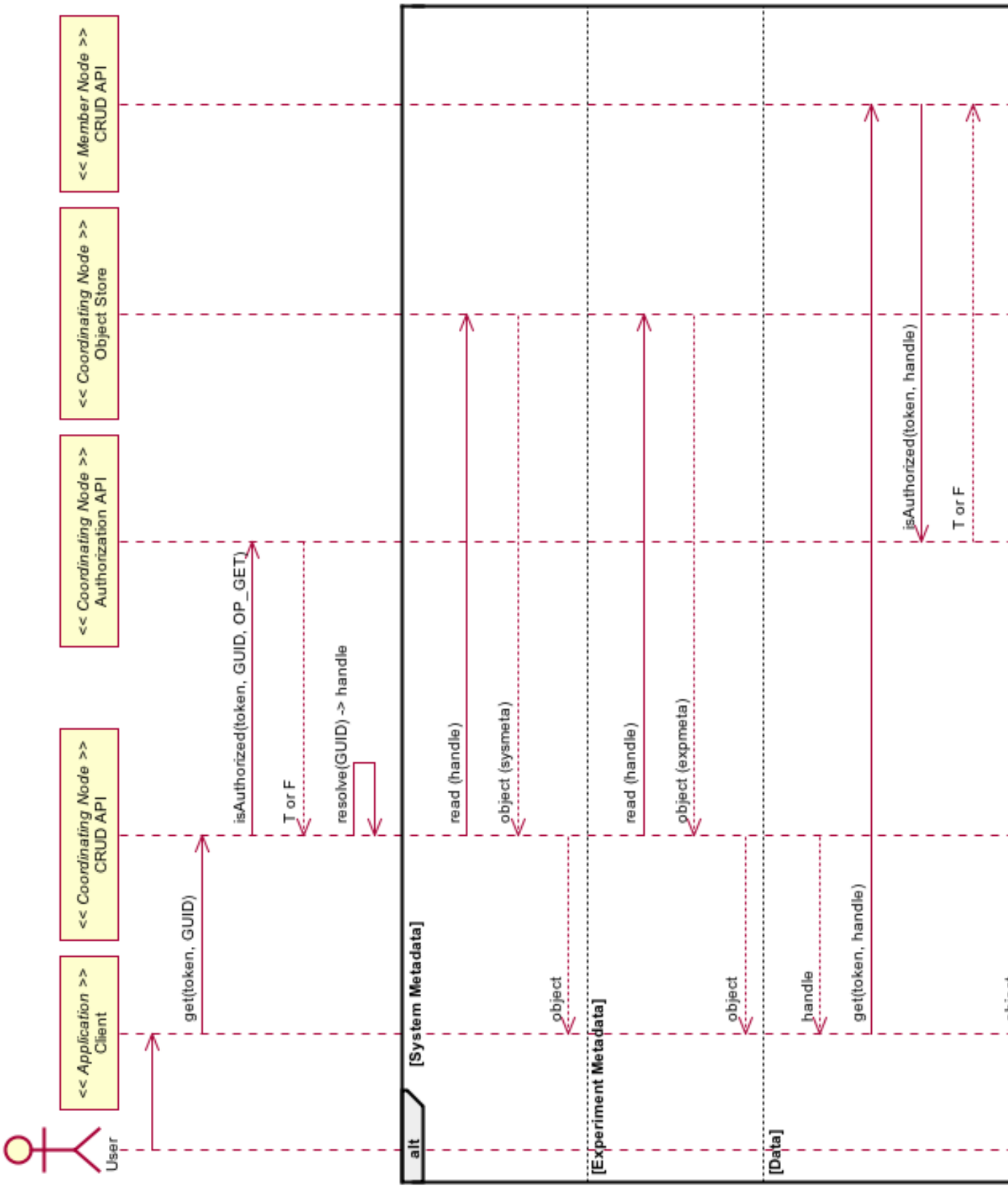


Figure 2.6: Resolving a GUID. In this scenario a user is trying to determine what the ID "A5548D" refers to, and uses the resolution service of a Member Node to that effect.


User
<< Application >>
Client
<< Coordinating Node >>
CRUD API
<< Coordinating Node >>
Authorization API
<< Coordinating Node >>
Object Store
<< Member Node >>
CRUD API
alt
[System Metadata]
[Experiment Metadata]
[Data]
get(token, GUID)
isAuthorized(token, GUID, OP_GET)
T or F
resolve(GUID) -> handle
read (handle)
object (sysmeta)
read (handle)
object (expmeta)
object
object
handle
get(token, handle)
isAuthorized(token, handle)
T or F

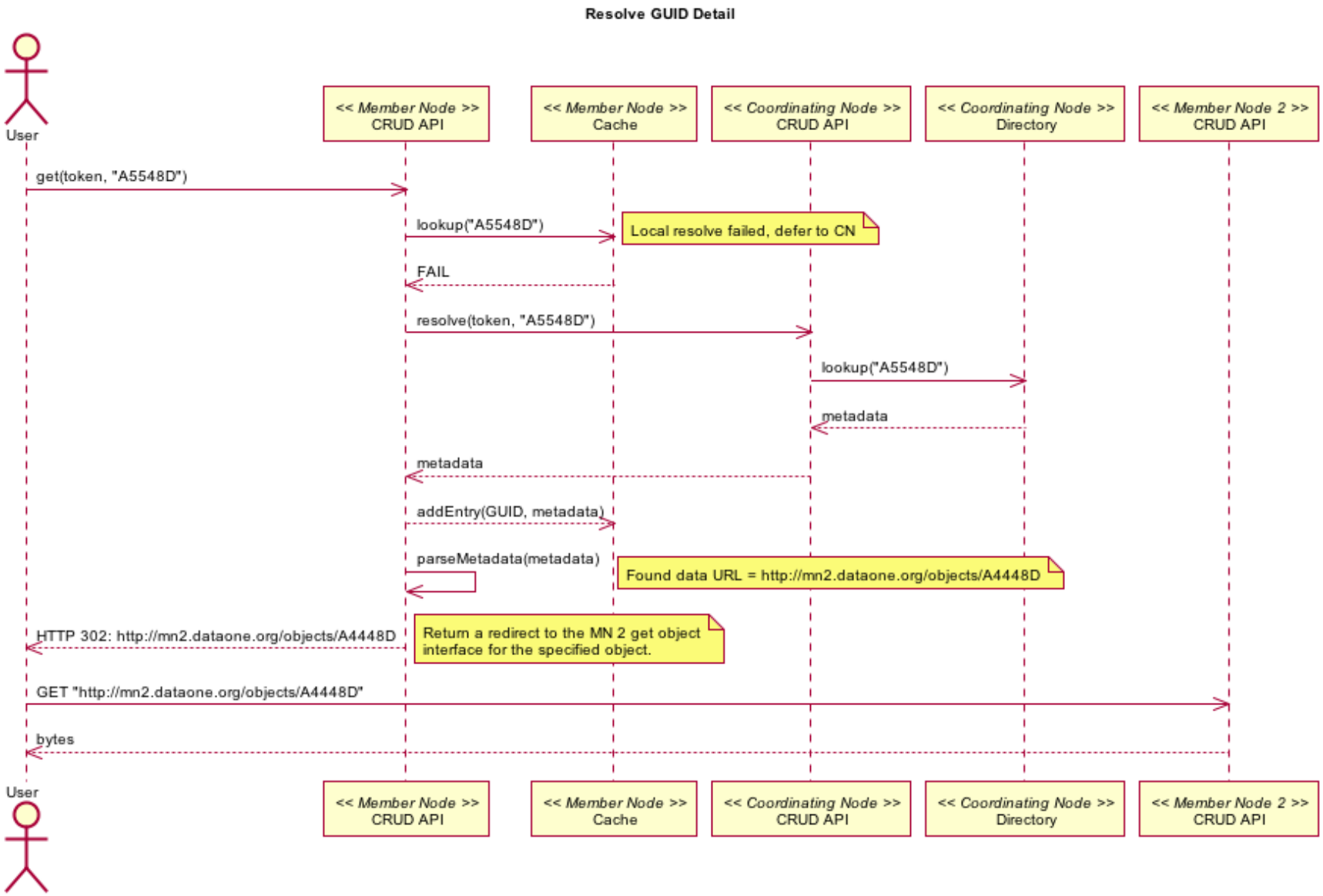


Figure 2.8: Detail for object retrieval of an object identified by a GUID. In this case, the User is requesting a data object from MN 1, though the data is actually located on MN 2.

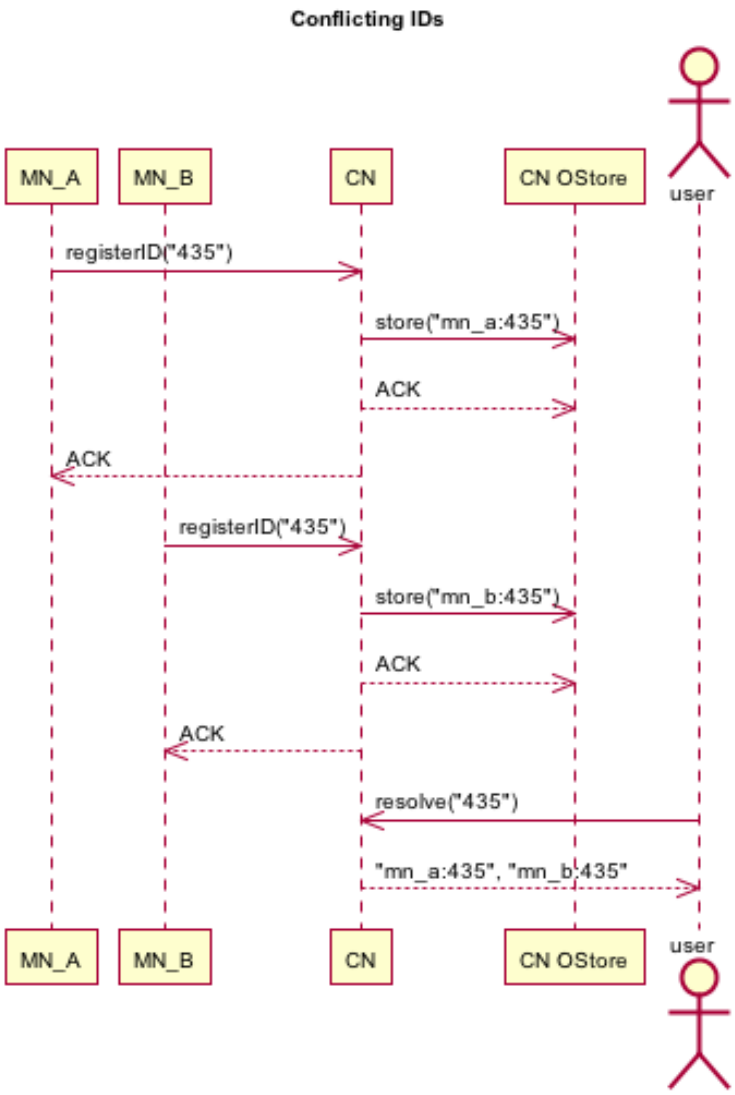


Figure 2.9: A scenario where two MNs happen to add different content to the system with the same identifier. Resolving the identifier without including the namespace results in two matches that must be interpreted by the client. The likelihood of such a scenario should be low, given that MNs should be utilizing identifier schemes that under ideal circumstances should not generate duplicate identifiers.

2.2.2 Notes from the 20090602 Albuquerque Meeting

These lightly edited notes were taken by Bruce Wilson of the group discussion about identifiers held during the VDC-TWG 20090602 Albuquerque Meeting.

Original notes are located in subversion at:

/documents/Projects/VDC/docs/20090602_04_ABQ_Meeting

Design Goals

From the DataONE perspective, an identifier is opaque. DataONE does not attach any meaning or resolution protocol based on the identifier.

A call to return the object associated with a particular identifier should always return either identically the same object or n/a if that object is no longer available. This raises a number of implementation issues, noted below. Particular issues include how to handle data which is regularly updated and things like status changes.

A member node may use its own internal identification scheme, but must be able to retrieve an object based on its DataONE globally unique identifier.

Member nodes may generate their own unique identifiers, such as DOI's, Handles, PURLs, or UUID's. The only requirement is that the identifier is unique across the space of DataONE. This implies that CN's must have functionality to

1. check that an identifier is unique and
2. to "reserve" or stub-out an identifier while the MN goes through the process of assembling the package to submit the object into DataONE.

When an object is replicated from one MN to another MN, the receiving MN must be able to accept and resolve the supplied DataONE identifier. That is, an object, no matter where it is within the DataONE network must be retrievable by its DataONE identifier, regardless of location. There was a lot of discussion on this point, and this is my interpretation of the conclusion. I believe we came out with the point that if a receiving member node assigns its own permanent identifier, then that creates more confusion, requires the MN to register that second ID with the CNs, and we can have confusion regarding the citation (for example) of the piece of data. It also makes tracking things like metrics, since the originating MN must then find out all other identifiers for the data and search for all of those. And while it can be argued that nobody ìownsî the data, there is (currently) a culture and need for the original archive to feel like it still can get credit for that investment.

A system doesn't need to maintain every version, but it does need to be able to identify every version.

Identifiers also apply to metadata as well as data.

Questions for Further Consideration

If a MN uses a DOI for a data set identifier, is it appropriate to include doi: in the identifier. For example, 10.3334/ORNLDAAC/840 is the DOI for a particular data set at the ORNL DAAC. Both "doi:10.3334/ORNLDAAC/840" and "10.3334/ORNLDAAC/840" can be presumed to be unique identifiers. Which should be used?

BEW: My personal preference is to use the one with the resolution protocol included. That does, however, make the identifier more of a "smart" identifier, which is generally problematic.

Where an identifier has a mechanism to resolve to multiple locations (such as is possible with an LSID and some DOI mechanisms) and that object is replicated from one MN to another MN, this would suggest that the originating MN needs to be notified of the additional location and has the option of registering the new location with the handle registration authority. This also means that if a replication is removed, the original MN should have the option of being notified, so that the resolution points are updated. Ideally, this should happen before the replica is removed (where possible), so that we eliminate (or at least minimize) the amount of time that an invalid resolution point is in someone else's system.

Where an identifier (such as a Handle) has a URL resolution, what should that resolution be? ORNL DAAC DOI's resolve to a web page where a user (after logging in) can see and download the components of the data set. Our opinion is that the DOI resolving to a human interpretable description of the object is more important than a machine interpretable resolution point. Some thought and guidance on this point for the overall DataONE community of practice is desirable.

Do we want/need a registry of name spaces? Where a MN uses a UUID (for example), there may not be a way to describe the name space for identifiers, unless the MN prefixes the UUID with some descriptor, which generally violates the general admonition about smart identifiers. It might, however, be helpful to have something like a set of regexps that describe the name space for a MN's identifiers, particularly if an automated way could be developed to look for potential collisions (non-null overlaps) between name spaces. BEW: My thought is that this is far from an initial feature, but the desirability of this as a possible future feature could have implications on the way we do things from the start.

Can the metadata standards support multiple globally unique identifiers? For example, what happens in the case that a MN starts down the DOI path and then switches to LSID's because of economic costs, for example, and goes back and assigns an LSID to historical data sets. Those data sets now have both an LSID and a DOI. Where is this in the metadata? Is there a mechanism for indicating the preferred ID and the alternate ID's? Likewise, how should things be handled when a MN decides to register an object with e.g. GCMD and the namespace that GCMD allows for identifiers does not allow for the MN's preferred identifier. Can a MN update the metadata to show an alternate key with the GCMD identifier (data set is also known as)? What is the implication for the metadata identifier in such a case? This is an update operation to the metadata, which implies that the metadata identifier is changed. How would one update the old metadata record to indicate that it is:

1. deprecated and
2. the id of the new metadata record?

The above also relates to the issue of establishing predecessor-successor relationships between identifiers. How should this be done across the system?

How do versions enter into the identifiers scheme? The general concept is that different versions of an object have different identifiers. What about having some type of an identifier that aggregates all versions of an object and which always points to the latest version of that object? How does D1 know that an object is a new version of an existing object? Update operation should take the old identifier and the new identifier. That would allow for the tracking of updates. A member node may track versions. Could create an interface specification for "latest version" where the CN calls the authoritative MN for the DS and asks for the identifier of the latest version of a particular identifier. Points back to the need for what amounts to meta-metadata - where the metadata object can be updated to indicate the status level of the data set (e.g. deprecated). Where is the identifier for something like World Ocean Data Base - this gets updated quarterly. They think of the fundamental unit as an observation point, which is either a location (e.g. buoy, possibly with different identifiers for different depths) or a leg of a trip, with multiple observations along a path.

For identifiers, we may need to specify the character space. What happens when a MN stores unique identifiers in a database field that supports just ASCII, but a different MN does it's unique identifiers in some other character set? PURL is a possible unique identifier, but we can get into cases now where URL's have characters from other language character sets (such as Arabic, Kanji, Ö)

What happens when a request for a replicated version of a data set comes to the replicate MN and the data set has been updated and the originating MN has not supplied the information about the update (e.g. they did an insert for the new version)?

How do we assign ID's for a continuous data stream or for a subset calculated on the fly? Does this mean that every request for a continuous data stream gets its own data set identifier, which then gets stored in the D1 system someplace? What is the value to the overall enterprise for storing the data set identifiers for each request, particularly in the context of something like a stream, where the on-the-fly processing is used to get a dynamic subset or dynamic reprojection? Examples of this sort of situation include the stream gauge data or the Atmospheric Radiation Measurement (ARM) archive. Ameriflux Flux tower data is a simpler case, in that they work on the basis of a site-year as a unit of data. The World Oceanic DataBase, however, operates on a location (and possibly depth) as a unit of data. Many of these are updated quarterly. Each unit of data has an identifier, unique within WODB, and WODB publishes a data stream that indicates what data packages were updated at what point in time. It is possible to determine whether a particular data package changed between two points in time. The differences are human interpretable, but it is not possible (in any generally automated fashion) to recreate the data stream for a particular data package at an arbitrary point in prior time.

Do the CN's need a method to determine the object type for an identifier? Do identifiers need to be unique across all types of identified objects?

2.3 Replication Notes

Author Dave V

This document contains notes about replication that may be of some use in DataONE.

2.3.1 General

1. Reduce complex transactions to be written by a single server
2. Monitor the server log
3. Be prepared for failure. The more data is distributed and scaled, the more likely failure will be encountered.

2.3.2 Coordinating Nodes

CNs need to keep a copy of all experiment metadata, and it is expected that each CN will be replicated with the others, so that each will appear the same.

There will be three CNs to start with.

There's basically three models for dealing with replication in this case:

- single master, two slaves
- ring - write one, it writes to the next, and so on.
- multi-master. Write to any, replication to others.

Multi-master replication would be the ideal option, as each CN would then offer identical functionality rather than the alternative of redirecting (presumably) 2/3 of requests to the master node.

There are basically two categories of information that are replicated:

- experiment metadata, which is information about a data set
- system metadata, which is information that is used to keep track of things such as the experiment metadata, MN capabilities and state, perhaps groups and access control lists, identity mappings

2.3.3 Member Nodes

Replication of data is achieved by copying content between Member Nodes until sufficient copies have been made to address concerns of access (x% of MNs may be offline and the content can still be reached) and efficiency of access (internet distance between client and the MN).

2.3.4 Resources

- rcs - venerable, functional
- inotify - the linux kernel file system monitor
- LDAP multi-master (OpenLDAP, 389 Directory Server)

- MySQL multi-master
- Microsoft's File System Replication Service and/or Distributed File System
- slony1 for Postgresql
- DRBD (dual master)
- GIT
- Mercurial
- Bazaar

2.4 Cross Domain Indexing and Access for Data and Metadata

Status Early Draft / notes

2.4.1 Problem

DataNet requires storage, search, and retrieval of information (data and metadata) from a wide variety of data services (e.g. Mercury, Metacat, OpenDAP, ...). All of these systems have different data service interfaces, support different metadata standards, and implement different query mechanism and syntaxes. Data must be replicated between service instances (Member Nodes, MN) and metadata must be replicated between all nodes (Coordinating Nodes, CN and Member Nodes) to ensure multiple copies exist to avoid data loss in the event of node failure and to improve access through geographic proximity.

A few general approaches to the problem include:

- translate the metadata to and from the format/model used internally by a MN
- treat the metadata document as an opaque object and just store it on the MNs, the CNs provide indexing service that locate copies of the metadata document
- MNs must implement a very general purpose metadata format, but may optionally make metadata available in more specific formats

2.4.2 Translation Approach

Translations between all metadata formats and the data service interfaces are implemented. In this scenario, metadata is translated to the native metadata format (or where multiple formats are supported, to the most appropriate form) supported by a MN and stored using the native API of the service. A common API provides the integration between all MNs, providing the basic operations necessary for managing and retrieving the content. Perhaps the most difficult component of this approach is the translation of metadata to the format supported internally by the service.

Problems:

- n x n bi-directional translations for metadata to be written, tested, and maintained.
- Metadata translation almost invariably leads to loss of information
- ...

Advantages:

- No or minimal changes to existing services (translation functions required).
- ...

2.4.3 Indexing Approach

Implement a common service API on all nodes that treats data and metadata as discrete units that can be read from and written to any node. The set of all nodes then becomes a large storage device. The CNs implement the processes which distribute content between all nodes (like a file system driver) to provide basic system level functionality. The actual metadata documents are opaque to the underlying storage system.

Metadata is not searched directly but is indexed by extracting content that matches semantically equivalent search terms. A trivial example is the use of the Dublin Core terms to search across all types of metadata. In this case, a "dublin core metadata extractor" extracts term values from a metadata document and updates an index that supports DC fields with the values and the document GUID. Searches on the index return the document GUID, which is then retrieved using the MN api.

Problems:

- Can not treat data available through service interfaces as a discrete unit (e.g. a MySQL service interface)
- Need parsers for all metadata formats to extract specific content
- New infrastructure (difficult to combine with existing services)
- Search capabilities on highly structured metadata may be limited
- ...

Advantages:

- No loss of information since there is no metadata translation, just extraction
- Format agnostic (system can store any type of discrete entity - basically anything that can be represented as a file)
- Search index can be highly tuned, multiple types of index can be implemented (e.g. topical domains)
- ...

2.4.4 Content Model Approach

Similar to the indexing approach, but in addition to the lowest common denomiator format, objects may make more detailed metadata/data available by advertising that they exhibit specific content models. These content models may be dictated by central DN1 community, or may be agreed upon by a small group of member nodes.

Problems:

- A central registry of data/metadata formats must be maintained.
- Burden is on member nodes to make sure they adhere to published content models.

Advantages:

- No loss of information since there is no metadata translation, just extraction
- Format agnostic (system can store any type of discrete entity - basically anything that can be represented as a file)
- Search index can be highly tuned, multiple types of index can be implemented (e.g. topical domains)
- Will work even for member nodes that only understand lowest-common-denominator formats, while nodes that understand more complex data/metadata will benefit from more specific searching and data management.
- Multiple communities can be accommodated, even if they have overlapping and/or inconsistent standards.

2.5 Authentication in DataONE

Author Dave Vieglais

Status Draft for comment

This document outlines some simplistic approaches for authentication of users of the DataONE system. It is not intended to be comprehensive but rather to offer a common starting point for discussion about the topic in the context of the DataONE infrastructure.

2.5.1 Overview

Users and services authenticate in DataONE for the following purposes:

- To confirm their identity

Identity is utilized in the DataONE infrastructure to:

- To indicate ownership of objects
- To confirm or deny access to an object (read / write) through some access control rules

Four scenarios of identify provider are investigated below:

1. Users are authenticated by Coordinating Nodes;
2. Users are authenticated by Member Nodes (a user identity managed by "their" MN);
3. Users are authenticated by a third party identity provider; and
4. Users are authenticated by any scheme with algorithmic verification available at any node.

The basic assumption in all cases is that the user authenticates using some credentials and a key (password or some other secret) and the identity provider returns a token that the user can pass around as proof that they have authenticated with the DataONE system at some time within the last XXX period of time.

2.5.2 Scenario 1: Authentication provided by CNs

- The Member Nodes must trust Coordinating Nodes as an identity provider
- The CNs must trust that MNs are forwarding credentials appropriately
- All accounts exist on and are managed by CNs

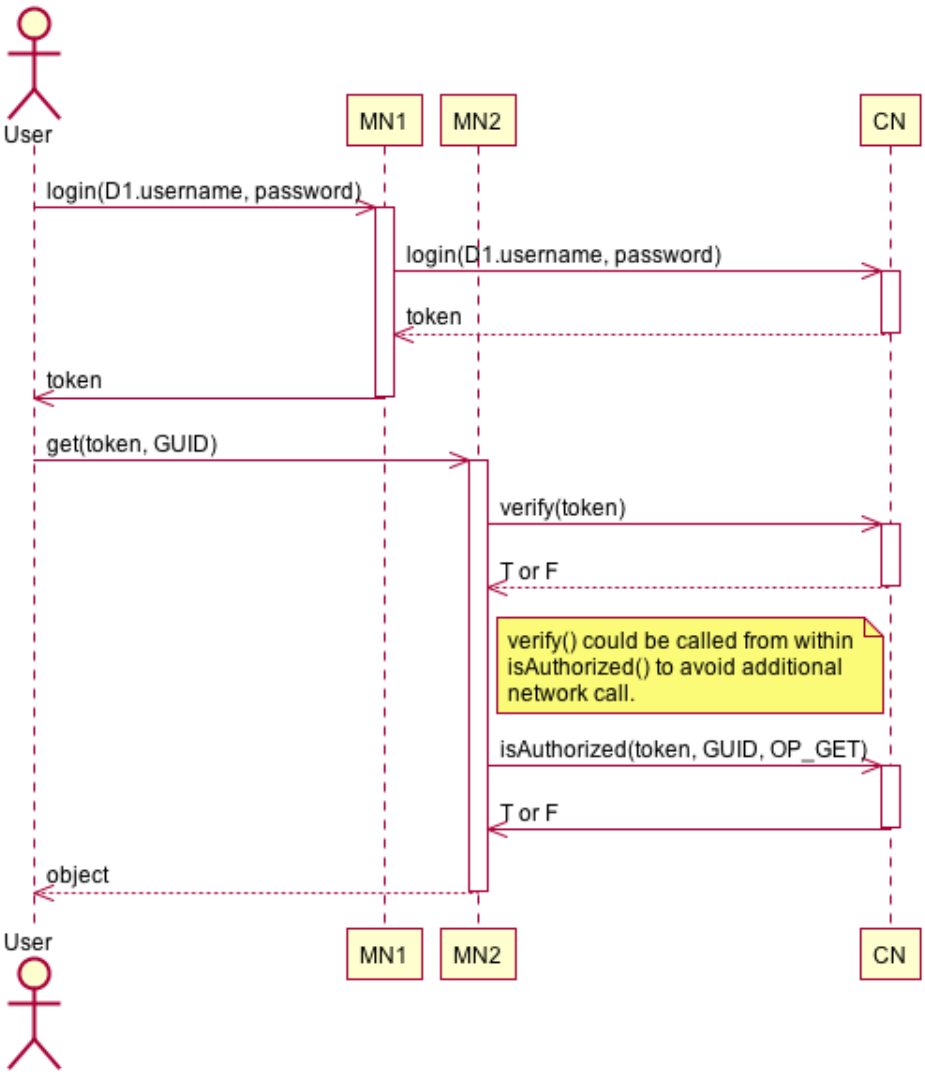


Figure 2.10: Authentication is provided by the CNs. The CNs may actually act as proxies for some other identity service, but from a DataONE perspective, the CNs are providing the authn/authz services.

2.5.3 Scenario 2. Authentication provided by MN

- token must have signature that indicates the issuing MN
- MN2 trusts MN1 as an authentication provider

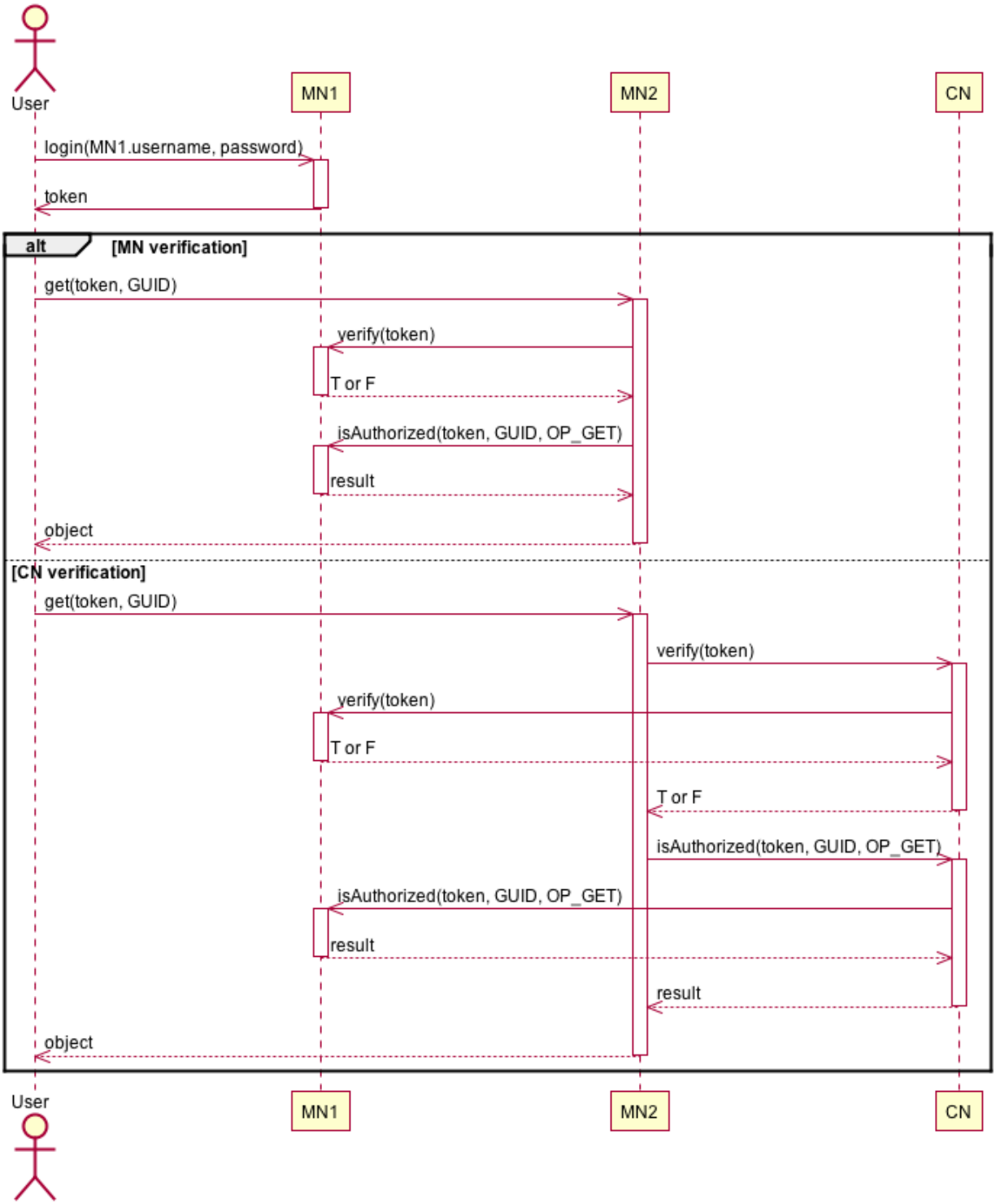


Figure 2.11: Authentication provided by a Member Node. With MN verification and authorization, those requests are forwarded to the appropriate MN (presumably the correct MN is indicated as an attribute of the token). The alternative is for verify / authz requests to always go to a Cn, which then forwards the request to the appropriate MN.

2.5.4 Scenario 3. Authentication through Third Party Identity Provider

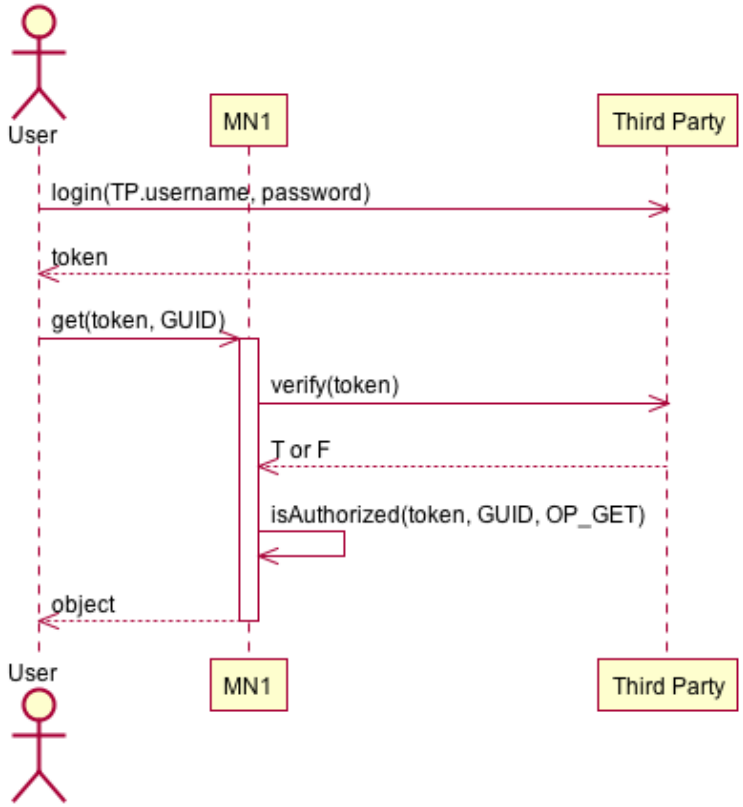


Figure 2.12: Authentication and verification provided by some third party, with authorization provided by the CNs.

- MNs and CNs must trust the third party identity provider
- Assuming access control rules are present on MN1

2.5.5 Scenario 4. Authentication Verified Algorithmically by Any Node

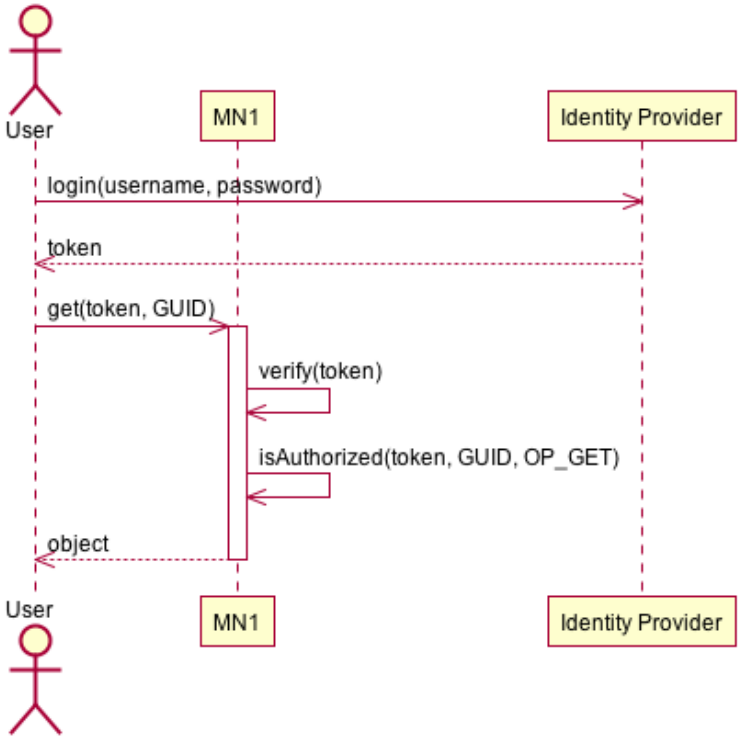


Figure 2.13: Tokens are verified by an algorithm. This requires trust in the algorithm.

- Minimizes communication between Nodes for identity verification
- All MNs and CNs must trust each other (to accept identity)
- Assuming access control rules are present on MN1

2.5.6 Notes from 20090602 ABQ Meeting

Access Control, Logging, Auditing -> Authentication + Identity Management

- Centralized directory (LDAP)
- Distributed directory (LDAP with referrals)
- Distributed management, replicated (LDAP with replication)
- GSI proxy certificates
- OpenID
- Shibboleth + InCommon

Verifying Identity

- Self registered (valid email, txt message, ...)
- Real person
- Member of xxx

Access Control

Principals

- User
- Member of group
- Public
- Authenticated user
- Site manager (for harvest, system operations, ...)
- Change request approval workflow
- group who have accepted intellectual rights

Things to control:

- Data
  - read
  - write
  - change perms
- Metadata
  - read
  - write
  - change perms
- Member Node Write
- Perform certain API calls (e.g. register node)
- Logs about {MNs, Data, ... }

Special People

- data curators
- MN curators / IMs

Expressing Permissions

- SAML
- eml-access
- Others to consider?

Attaching Permissions to Resources and Services

CHAPTER

THREE

USER REQUIREMENTS

The user requirements outlined in this document drove much of the process for identification of the *DataONE Functional Use Cases* described elsewhere. The functional use cases were identified and prioritized over a series of meetings funded by the NSF sponsored "Virtual Data Center" (VDC) INTEROP project.

3.1 UR 01. Core Functionality (proposed)

DataONE provides the distributed framework (which is comprised of Member and Coordinating Nodes as illustrated in Figure 1), sound management, and robust technologies that enable long-term preservation of diverse multi-scale, multi-discipline, and multi-national observational data. DataONE initially emphasizes observational data collected by biological (genome to ecosystem) and environmental (atmospheric, ecological, hydrological, and oceanographic) scientists, research networks, and environmental observatories. DataNetONE will be domain agnostic, progressively expanding to broader domains and building on infrastructure and interoperability with DataNet partners.

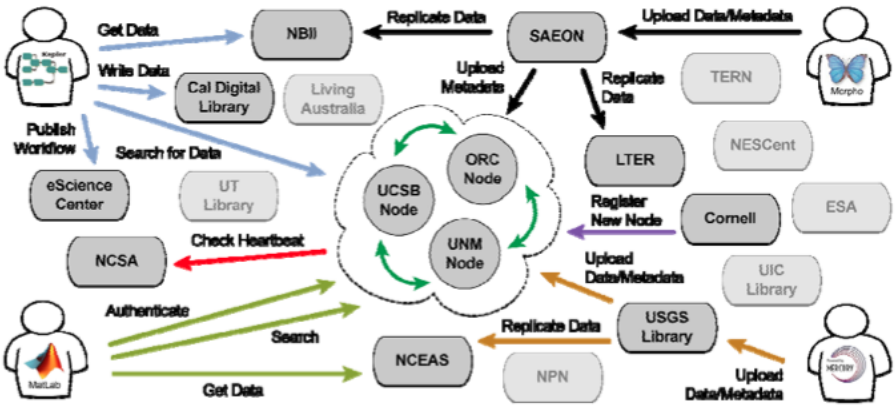


Figure 3.1: DataONE Member Nodes form a robust, distributed network via coordinating services provided by a set of Coordinating Nodes (i.e., Oak Ridge Campus, UC-Santa Barbara, and University of New Mexico) arranged in a high-availability configuration. Scientists and citizens interact with Member Nodes (e.g., South African Environmental Observation Network, California Digital Library, USGS National Biological Information Infrastructure) through software tools that utilize standardized interfaces. This structure supports many different usage scenarios, such as data and metadata management and replication (e.g., using Morpho [black arrows] or the Mercury system [orange arrows]), as well as analysis and modeling (e.g., using commercial software like Matlab [light green arrows] and open-source scientific workflow systems like Kepler [blue arrows]). Coordinating Nodes perform many basic indexing and data replication services to ensure data availability and preservation (e.g., node registration [purple arrow] and monitoring via heartbeat services (red arrow]).

3.2 UR 02. Value Added Services (proposed)

DataONE is not the end, but rather the means to enable scientists and citizens to address and better understand the difficult and complex biological, environmental, social, and technological challenges affecting human, ecosystem, and planetary sustainability. The comprehensive cyber-infrastructure allows novel questions to be asked that require harnessing the enormity of existing data and developing new methods to combine and analyze diverse data resources (Figure 2).

DataONE will accomplish its goals by making scientists, students, librarians, and citizens active participants in the data life cycle—especially, the data preservation process. By supporting community- derived interoperability standards and incorporating new value-added and innovative technologies (e.g., for semantic and geospatial information, scientific workflows, and advanced visualization) into the scientific process, DataONE will facilitate sophisticated data integration, analysis, interpretation, and understanding. A strong education and outreach program focuses on scientists and students learning to better and more easily manage, preserve, analyze, and visualize Earth observational data. Citizen scientists are actively engaged in data preservation and scientific discovery through their involvement in programs such as the USA National Phenology Network (USA-NPN) and numerous Cornell Laboratory of Ornithology citizen science efforts (e.g., eBird, Project FeederWatch).

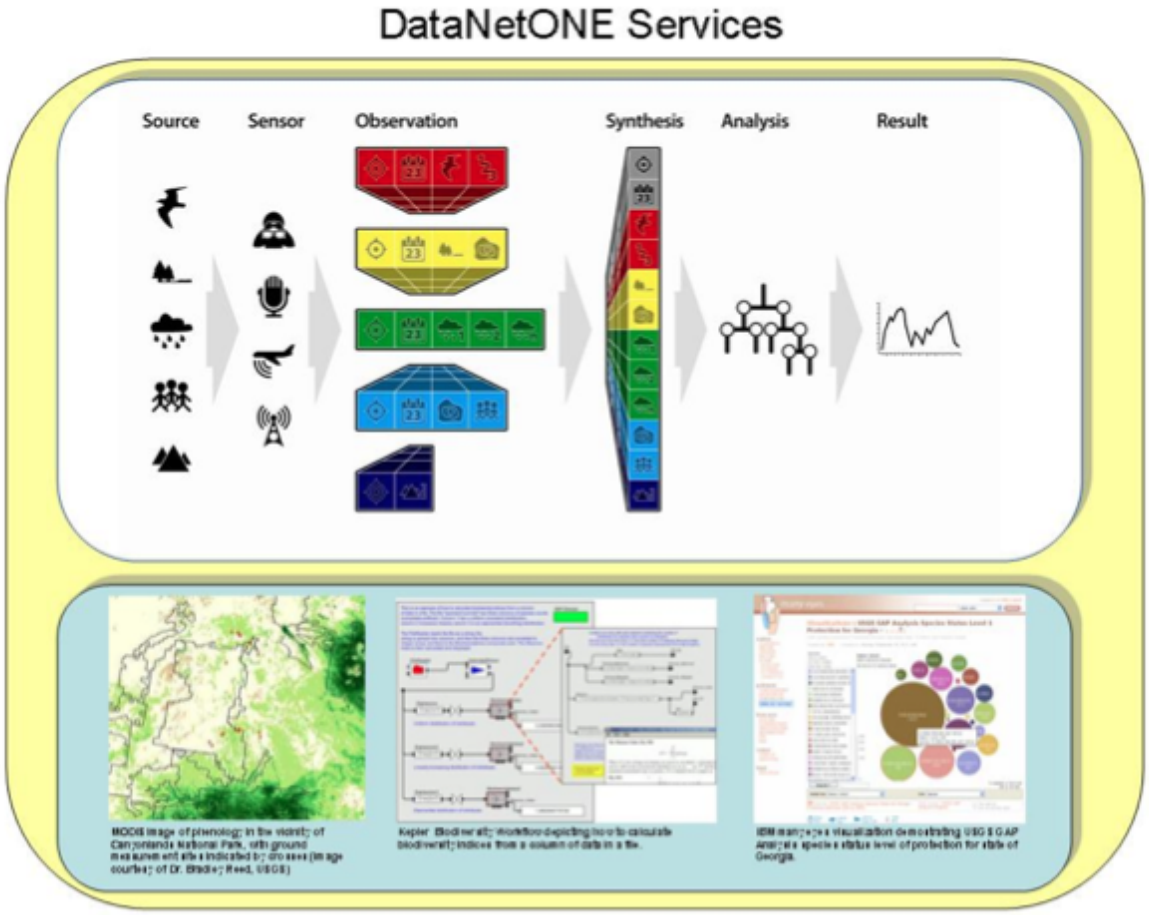


Figure 3.2: DataONE value-added services enable scientists to address novel questions through integrating disparate data sources (top), supporting geospatial data processing (lower left), and providing scientific workflow solutions like Kepler (lower middle) and high-level analyses and visualizations like IBM ManyEyes.

3.3 UR 03. System Architecture (proposed)

The DataONE architecture must embrace the highly dispersed and independent nature of data collection activities relevant to the environmental and earth sciences. Data are collected by tens of thousands of scientists around the world who have the expertise to describe and archive these data, as well as curate them. Attempting to centralize this curation function is inherently untenable and will not scale. Thus, DataONE will achieve both scalability and sustainability through a highly distributed system architecture (Figures 1 and 4) that utilizes the **DataNet Service Interface** to access uniform services provided and used by three types of cyber-infrastructure: (1) **Member Nodes** located at institutions distributed throughout academia, libraries, government agencies, and other organizations that provide local data storage, curation, and metadata for a set of data resources that are collected or affiliated with that institution; (2) **Coordinating Nodes** that are geographically-distributed (Figure 3) to provide a high-availability, fault-tolerant, and scalable set of coordinating services to the Member Nodes, including a complete metadata index and data replication services for all data in all Member Nodes; and (3) an **Investigator Toolkit** that provides a complete and evolving set of tools for data and metadata management by scientists and curators throughout the entire data life cycle (Figure 3). Initially, there will be three Coordinating Nodes geographically dispersed at ORC, UNM, and UCSB. A small number of additional coordinating nodes may be implemented as DataNetONE expands in scope, sustainable funding, and international presence.

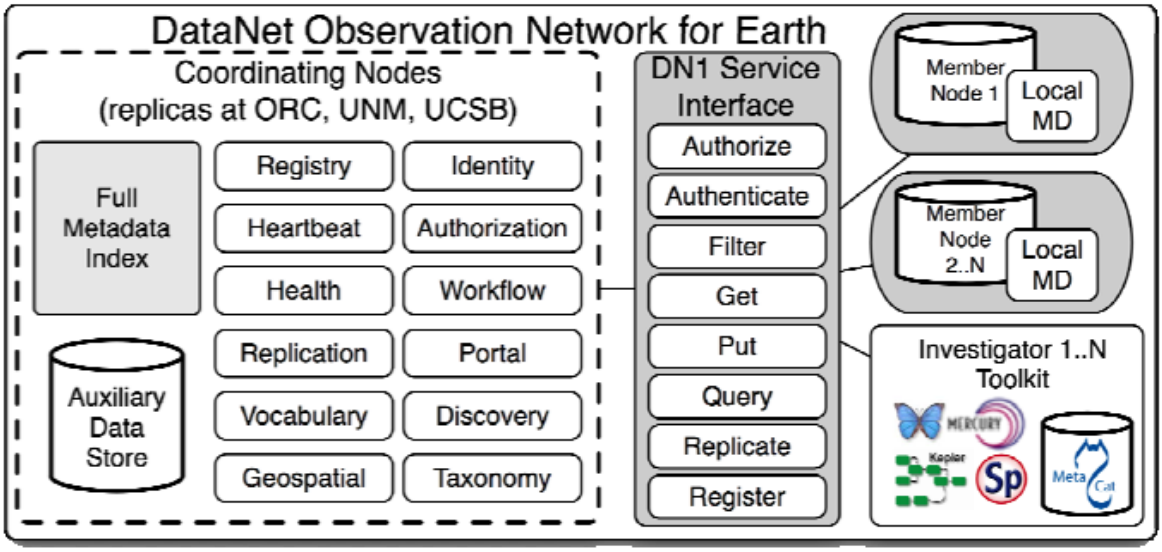


Figure 3.3: Main features of the DataNetONE architecture, emphasizing distributed data storage at Member Nodes and metadata indexing and services provided by Coordinating Nodes. Communication between Member Nodes (e.g., for replication), between Member Nodes and the Investigator Toolkit (e.g., for inserting data), and between Member Nodes and Coordinating Nodes (e.g., for metadata indexing) is all mediated via a common DataNetONE Service Interface that spans all node types.

3.4 UR 04. Spread and Impact of Invasive Species

Invasive species are:

- Expensive - response estimates up to $138bn per year in the US1
- Complex - many interactions and factors determine invasiveness and impact
- Overwhelming - border and import controls struggle with sheer volume of material and confusion for access and collation of relevant biodiversity information

Outcome is suboptimal consideration of potential biotic risks associated with imports of new products, packing, shipping methods or from new regions,

DataONE eliminates confusion for information access and integration, providing domain relevant knowledge to researchers, border agents and to inform policy makers on potential impact or savings associated with new or ongoing trade relationships.

Figure 3.4: Some examples of economically important invasive species in the United States.

Specifically, DataNetONE addresses:

- Volume – combines majority of digitized species occurrence information through GBIF,NBII, IABIN and other partnerships
- Structure - efficient access to data, metadata, and workflows from ecological, environmental, socio-economic data, and related studies and assessments (e.g. Metacat, Dryad, Mercury, NBII)
- Dynamic - service infrastructure for interpolation of data (e.g. species occurrence points to distribution regions and environmental requirements)
- Knowledge - semantic assisted discovery and integration of indirectly associated data
- Social interactions – transparent communications between researchers and implementers providing latest information possible for regions taxa, and trade routes, and fora of product reviews, models, and other related information

Figure 3.5: Japanese honeysuckle (Lonicera japonica) was brought to the United States from Asia as ground cover to prevent erosion. As an invasive species, it can topple small trees and schurbs by its weight. The resulting change in forest structure may negatively affect songbird populations.

3.5 Coordinating Node Requirements

This list of requirements for Coordinating Node functionality was obtained (probably) by a brainstorming session at the 20090602 ABQ VDC-TWG meeting.

Todo

Estimate some times for the unknowns here.

- IDs not discoverable until all CNs have confirmed copy
- Metadata searches return in max xxx secs
- CN can store and search > xxx metadata records
- CN can store and search multiple metadata standards - specifically (list Metadata_formats_of_interest.txt)
- CN can load-balance to maximize performance
- Any xxx CNs can be offline without affecting D1 services
- xxx concurrent transactions can be supported - different for different types of transaction e.g. delete vs. insert
- Each CN should be available xxx% time
- CN should expose MD in search within xxx secs of insert by MN
- CN services should be designed to be independently scalable
- CN services should be geographically replicated
- CN have complete metadata copies from all MNs
- D1 should be TRAC and/or DRAMBORA trusted repositories
- CN should respect replication policies of MNs
- CN should ensure that any given data set is available at any time (subject to policy restrictions) even when xxx% of MNs are offline
- CNs must validate that: - data are available at all replicas - replicas are valid
- CNs ensure that data are available in current formats (data format upgrade?)
- CNs should be economical to run and maintain
- CNs should be able to be remotely administered

- CNs should be secure and deflect malicious intent
- CNs should provide services to detect rogue data (viruses, copyright,)

CHAPTER

FOUR

USE CASES

4.1 Overview

An important early part in the process of system architecture is identifying the functional requirements that the resulting system should be able to address. Use cases are helpful for this as they describe how actors (typically entities external to the system) interact with system components. Interaction diagrams help define the interfaces and messages transmitted between components.

4.1.1 Use Case Layout

Each use case document is structured following a common template which includes the following elements.

Author Who generated the use case. This will usually be "VDC TWG".

Date Date the use case was identified and known subsequent modifications.

Version One of "Draft", "Review", "Published"

Goal Brief description of what is intended with the use case.

Summary An overview of the use case which includes the goal and major actors.

Use case diagram A diagram representation of the use case, somewhat following the UML.

Sequence diagram A sequence diagram that shows the operations involved in addressing the use case.

Actors The participants in the use case - includes people and systems.

Preconditions The conditions that must exist for the use case. For example, a particular use case may only be valid if the use has been authenticated.

Triggers Describes the even that causes the use case to be initiated.

Postconditions The state of the system after the use case - what conditions are true after the use case is completed.

Notes Miscellaneous notes and discussion items that can't be placed in the various categories. Comments should be annotated with time and author.

4.2 Implementation Priority

Submitted by DataONE Core Cyberinfrastructure Team / Virtual Data Center Technical Working Group

Revisions

- October 7, 2009; revised October 10, 2009; revised October 13, 2009

- 20091110: Text from Word format transferred to plain text (this document)

4.2.1 Introduction

The DataONE Core Cyberinfrastructure Team / Virtual Data Center Technical Working Group (CCIT/VDC TWG, hereafter CCIT) was charged with developing a prioritization of the approximately three dozen use cases developed to date. The prioritization was to be shared with the DataONE Leadership Team in order to come to agreement for the development priorities for years 1 through 5. This agreement was considered an important deliverable for the DataONE kickoff meeting in October 2009.

The CCIT reviewed existing documents, including the CI Preliminary Task List, the Service Interface Prioritization diagram, the DataONE – VDC June 2009 Technical Working Group Meeting Report, and the DataNetONE Implementation Plan (primarily Objective 4) – see References, below. The use cases are defined in the DataONE Architecture document available at https://129.24.0.15/ArchitectureDocs. Goals and milestones expressed in the CI Preliminary Task List were foundational for developing the proposed prioritization (see Table 1).

In some instances, use case implementations in the early years may be partial or limited implementations with work continuing in later years until completion. Examples of use cases that are likely to be improved over time are authentication, logging, search and retrieval, event notifications, etc. Work on other goals and milestones will begin as early as year 1, led by the appropriate working group, with the initial implementation in a subsequent year (e.g., scientific use cases; workflow support; ontology support). Certain goals and milestones (e.g., replication of data and metadata) will be met by evaluating alternatives and selecting a set of existing software applications. Additional details about Member Nodes will be provided in a separate document, forthcoming. This prioritization is mapped optimistically to the 5-year schedule: goals, milestones, and specific use cases that are the best candidates for potential deferral are indicated in the table and discussion below.

The CCIT reviewed the prioritization, distributed it to the Leadership Team, and discussed the prioritization during a videoconference held October 13, 2009. Improvements identified during the videoconference were incorporated into the document, which was then distributed at the DataONE kickoff meeting held October 20-22, 2009.

Goal / Milestone	Year 1	Year 2	Year 3	Years 4-5
Launch 3 Coord. / 3 Member Node network	X			
Initial GUID support	X			
Formalize service APIs	X			
Reference service / client implementations	X			
Authentication: short- then long-term	X1		X2	
Search, retrieval, metadata interoperability	X1	X2	X3	X4
DataONE User Interface	X1	X2	X3	X4
DataONE Investigator Toolkit	X1	X2	X3	X4
Replication of data and metadata	X			
Heartbeat / health monitor (basic to robust)*	X1	X2	X3	X4
Logging (basic to robust)*	X1	X2	X3	
Member Node registry	X1	X2		
Additional Member Nodes		X1	X2	X3
Data / metadata deposit, update, delete		X		
Identity provider (external or internal)		X		
Authorization		X		
Notification of DataONE events*		X1	X2	X3
Data usage policy support		X		
Web-based batch data uploads		X		
Launch robust public prototype			X	
Support scientific use cases			X	
Client discovery services			X	
Batch ingest for data / metadata			X	

Continued on next page

Table 4.1 – continued from previous page

Stress and load testing			X	
Coordinating node failover / load balancing*			X	
Data and metadata validation				X
Data and metadata migration				X
Workflow support				X
Ontology support				X
Provenance support				X
Support advanced scientific use cases				X
Capacity monitoring				X
Hardened infrastructure				X

Table 1: Goal / milestone summary, by year. Goals / milestones with asterisks could be deferred to the following year. X indicates the year of planned implementation. Xn indicates cyber-infrastructure goals to be met through successive releases. Shaded cells indicate activities undertaken by DataONE working groups and the CCIT in years prior to initial implementation.

Notes

Prioritizing system implementation to address the use cases involves several factors including:

- *Vision of the project*. The system is being designed with some overall goals described by the vision of the project proposal.
- *Requirements of the community*. The stakeholders that comprise the user and participant community quite likely has some opinion on functionalities of the system that are important to them. If these are not properly addressed, then the resulting system may appear as a failure to them.
- *Requirements of the sponsor*. The sponsor has laid out goals in the RFP that the project is responding to and also in the final agreement for conducting the work.
- *Dependencies between use cases*. Implementation of functionality to address some use cases will require implementation of some components not directly specific in a use case.
- *Resources available for implementation*. Some use cases may be identified as high priority, but would require resources that would prevent implementation of a number of other lower priority features.

4.2.2 Year 1 Use Cases

Goals and Milestones

The year 1 goals and milestones for DataONE include:

- Launching a network consisting of three Coordinating Nodes and three Member Nodes
- Initial support for GUIDs (globally unique identifiers)
- Formalize service APIs
- Provide reference service / client implementations
- Authentication using a short-term solution
- Search and retrieval of data from all Member Nodes to demonstrate basic metadata interoperability (initial release)
- DataONE user interface (initial release)

- DataONE Investigator Toolkit (initial release)
- Replication of data and metadata between Coordinating and Member Nodes, bootstrapped using existing repositories
- Heartbeat / health monitoring (initial release)
- Infrastructure (initial release)
- Member Node registry services (initial release)

Use Cases

Authentication Using Short-term Solution

Use Case 12 - User Authentication: Person, via client software, authenticates against Identify Provider to establish session token. Many operations in the DataONE system require user authentication to ensure that the user's identity is known to the system, and that appropriate access controls can be executed based on the identity.

Use Case 14 - System Authentication and Authorization: System Authentication/Authorization - Server authenticates and performs system operations (e.g. replication). This use case represents node-to-node authentication.

Search and Retrieval, Indexing, Read Data and Metadata, Metadata Interoperability

Use Case 1 - CRUD Read(): Get object identified by GUID (authenticated or not, notify subscriber of access). A client has an identifier for some object within the DataONE system and is retrieving the referenced object. The DataONE system must resolve the identifier and return the object bytes after checking that the user has read privileges on the object.

Use Case 02 - List GUIDs By Search: Get list of GUIDs from metadata search (anonymous and authenticated). A user performs a search against the DataONE system and receives a list of object identifiers (GUIDs) that match the search criteria. The list of GUIDs is filtered such that only objects for which the user has read permission will be returned. This use case assumes that the search is being performed by submitting a query against a CN.

Replication of Data and Metadata

Use Case 06 - MN Synchronize: Replicate / synchronize metadata record between Member Node and Coordinating Node. As data packages, or portions thereof, are created or modified, the metadata associated with those needs to be replicated to the coordinating nodes. The presence of new or changed information on a Member Node (MN) is made known to a Coordinating Node (CN) through the status information in a ping() response. If so indicated, the CN schedules a synchronization operation with the MN, a list of changed object GUIDS is retrieved by the CN, and the CN proceeds to retrieve and process each object identified. If new data packages are present on the MN, then a MN-MN synchronization process is scheduled.

Use Case 09 - Replicate MN to MN: Replicate data from Member Node to Member Node - (facilitated by Coordinating Node or independently but notifying Coordinating Node of operation).

Use Case 24 - MNs and CNs Support Transactions: Transactions - CNs and MNs should support transaction sets where operations all complete successfully or get rolled back (e.g., upload both data and metadata records). Implementation of this use case could be deferred to year 2.

Basic Heartbeat / Health Monitoring (candidate for deferral to year 2)

Use Case 10 - MN Status Reports: Coordinating Node checks "liveness" of all Member Nodes. Implementation of this use case could be deferred to year 2.

Basic Logging Infrastructure (candidate for deferral to year 2)

Use Case 16 - Log CRUD Operations: All CRUD operations on metadata and data are logged at each node. Implementation of this use case could be deferred to year 2.

Use Case 17 - CRUD Logs Aggregated at CNs: All CRUD logs are aggregated at coordinating nodes. Implementation of this use case could be deferred to year 2.

Basic Member Node Registry Services

Use Case 03 - Register MN: Register a new Member Node. This use case describes the technical process for addition of a new member node (MN) to the DataONE infrastructure. It is assumed that the appropriate social contracts have been formed and the MN is operational, ready to be connected.

4.2.3 Year 2 Use Cases

Goals and Milestones

The year 2 goals and milestones for DataONE include: - Deploy additional Member Nodes - Data and metadata deposit, update, and delete - Implement an external or internal identity provider - Implement authorization - Support notifications based upon DataONE events (initial release) - Support negotiated and approved data usage policies - Web-based interface for batch data uploads - Search and retrieval of data from all Member Nodes (release 2) - DataONE user interface (release 2) - DataONE Investigator Toolkit (release 2) - Heartbeat / health monitoring (release 2) - Logging infrastructure (release 2) - Member Node registry services (release 2)

Use Cases

Data and Metadata Deposit, Update, and Delete

Use Case 04 - CRUD (Create, Update Delete) Metadata: Create, update or delete metadata record on a Member Node. A user is creating a new metadata record on a Member Node (MN). The mechanism by which the user does this is out of scope for the DataONE system, so this use case continues from the point where a new Data Package is present on the MN. The metadata is retrieved by the CN using a pull mechanism (CN requests content from the MN).

Use Case 05 - CRUD (Create, Update Delete) Data: Create/Update/Delete data object in Member Node. May split out the update and delete portions to different use cases at some point in the future.

Use Case 23 - Owner Expunge Content: User can find out where all copies of my data are in the system and can expunge them. Implementation of this use case could be deferred to year 3.

Identity Provider

Use Case 15 - Account Management: User Account Management - Create new user account on Identity Provider (also edit, delete).

Support Authorization

Use Case 13 - User Authorization: ser Authorization - Client requests service (get, put, query, delete, ...) using session token.

Support Notifications Based Upon DataONE Events (candidate for deferral to year 3)

Use Case 21 - Owner Subscribe to CRUD Operations: Data owners can subscribe to notification service for CRUD operations for objects they own. Implementation of this use case could be deferred to year 3.

Use Case 28 - Derived Product Original Change Notification: Relationships/Versioning - Derived products should be linked to source objects so that notifications can be made to users of derived products when source products change. Implementation of this use case could be deferred to year 3.

Enhance the Logging Infrastructure

Use Case 16 - Log CRUD Operations: All CRUD operations on metadata and data are logged at each node.

Use Case 17 - CRUD Logs Aggregated at CNs: All CRUD logs are aggregated at coordinating nodes.

Use Case 18 - MN Retrieve Aggregated Logs: Member nodes can request aggregated CRUD log for {time period/object id/userid} for all of 'their' objects. Implementation of this use case could be deferred to year 3.

Use Case 19 - Retrieve Object Download Summary: General public can request aggregated download usage information by object id. Implementation of this use case could be deferred to year 3.

Use Case 20 - Owner Retrieve Aggregate Logs: Data owners can request aggregated CRUD log for {time period/object id} for all of 'their' objects. Implementation of this use case could be deferred to year 3.

Support Data Usage Policies

Use Case 08 - Replication Policy Communication: Communication of replication policy metadata between Member Nodes and Coordinating Nodes. The replication policy of Member Nodes (MN) indicates factors such as the amount of storage space available, bandwidth constraints, the types of data and metadata that can be managed, and perhaps access control restrictions. This information is used by Coordinating Nodes (CN) to balance the distribution of data packages throughout the DataONE system to achieve the goals of data package persistence and accessibility.

Use Case 31 - Manage Access Policies: Manage Access Policies - Client can specify access restrictions for their data and metadata objects. Also supports release time embargoes.

4.2.4 Year 3 Use Cases

Goals and Milestones

The year 3 goals and milestones for DataONE include: - Deploy additional Member Nodes - Launching a robust public prototype - Support selected scientific use cases - Authentication using a long-term solution - Implement client discovery services - Batch ingest support - Conducting stress and load testing - Implementing Coordination Node failover and load balancing - Search and retrieval of data from all Member Nodes (release 3) - DataONE user interface (release 3) - DataONE Investigator Toolkit (release 3) - Heartbeat / health monitoring (release 3) - Logging infrastructure (release 3) - Support notifications based upon DataONE events (release 2)

Use Cases

Authentication Using a Long-Term Solution

Use Case 12 - User Authentication: Person, via client software, authenticates against Identify Provider to establish session token. Many operations in the DataONE system require user authentication to ensure that the user's identity is known to the system, and that appropriate access controls can be executed based on the identity.

Use Case 14 - System Authentication and Authorization: System Authentication/Authorization - Server authenticates and performs system operations (e.g. replication). This use case represents node-to-node authentication.

Client Discovery Services

Use Case 33 - Search for Data: Clients should be able to search for data using CN metadata catalogs.

Use Case 34 - CNs Support Other Discovery Mechanisms (e.g. Google): Coordinating Nodes publish metadata in formats for other discovery services like Google/Libraries/GCMD/etc.

Batch Ingest

Use Case 07 - CN Batch Upload: Batch Operations - Coordinating Node requests metadata /data list from new Member Node and then batch upload (disable indexing for example to improve insert performance).

Coordination Node Failover and Load Balancing (candidate for deferral to years 4-5)

Use Case 29 - CN Load Balancing: Load Balancing - Requests to coordinating nodes are load balanced. Implementation of this use case could be deferred to years 4-5.

Complete Logging Infrastructure

Use Case 22 - Link/Citation Report for Owner: User can get report of links/cites my data (also can view this as a referrer log).

4.2.5 Years 4-5 Use Cases

Goals and Milestones

The year 4-5 goals and milestones for DataONE include:

- Deploy additional Member Nodes
- Data and metadata validation
- Data and metadata migration
- Workflow support
- Ontology support
- Provenance support
- Support for general and innovative scientific use cases (subsetting, translation, semantic interoperability, advanced visualization, etc.)
- Capacity monitoring
- Hardening of overall infrastructure into a robust system
- Search and retrieval of data from all Member Nodes (release 4)
- DataONE user interface (release 4)
- DataONE Investigator Toolkit (release 4)
- Heartbeat / health monitoring (release 4)
- Support notifications based upon DataONE events (release 3)

Use Cases

Data and Metadata Validation

Use Case 25 - Detect Damaged Content: System should scans for damaged/defaced data and metadata using some validation process.

Use Case 26 - Data Quality Checks: System performs data quality checks on data.

Data and Metadata Migration

Use Case 27 - Metadata Version Migration: CN should support forward migration of metadata documents from one version to another within a standard and to other standards.

Workflow Support

Use Case 11 - CRUD Workflow Objects: Create / update / delete / search workflow objects.

Provenance Support

Use Case 28 - Derived Product Original Change Notification: Relationships/Versioning - Derived products should be linked to source objects so that notifications can be made to users of derived products when source products change.

Use Case 32 - Transfer Object Ownership: User or organization takes over 'ownership' of a set of objects (write access for orphaned records).

Complete Heartbeat / Health Monitoring

Use Case 30 - MN Outage Notification: MN can notify CN about pending outages, severity, and duration, and CNs may want to act on that knowledge to maintain seamless operation.

4.2.6 References

CI Preliminary Task List, available at https://repository.dataone.org/documents/Meetings/20090210-mat-santa-barbara-mtg/CI_preliminary_tasklist_021809.xls

DataONE Architecture, available at: https://repository.dataone.org/documents/Projects/VDC/docs/service-api/api-documentation/build/html

DataNetONE Implementation Plan (Objective 4), available at: [need URI].

DataONE – VDC June 2009 Technical Working Group Meeting Report, https://repository.dataone.org/documents/Projects/VDC/docs/20090602_04_ABQ_Meeting/20090604MeetingReport.pdf

Service Interface Prioritization (diagram), available at https://repository.dataone.org/documents/Projects/VDC/docs/service-api/api-diagrams/service-api-layers.png

4.3 Use Case 01 - Get Object Identified by GUID

Author VDC TWG

Date

- 20091006 Added use case diagram
- 20090924 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Retrieve an object identified by GUID (authenticated or not, notify subscriber of access).

Summary A client has an identifier for some object within the DataONE system and is retrieving the referenced object. The DataONE system must resolve the identifier and return the object bytes after checking that the user has read privileges on the object.

Watchers (subscribers) of the object are notified of the retrieval (what are they notified of?).

Actors

- Client requesting object
- Coordinating Node
- Member Node

Preconditions

- Client has authenticated to at the desired level (e.g. client may not have authenticated, so access might be anonymous), and so an `auth_token | None` is available.

Triggers

- An object is requested from the DataONE system.

Post Conditions

- The client has a copy of the object bytes (or an error message in the case of failure)
- The log is updated with the results of the operation
- Watchers perform their processing

Notes

1. For the GET operation, the coordinating node does not need to check access, because the member node will do this check before delivering the results.
2. For the GET operation, should isAuth() be performed only by CNs? Relying on the MN system metadata requires trusted implementation of the MN system and consistency of system metadata across all MNs (which will be the case, though with uncertain latency). Requiring all isAuth() operations to be performed by CNs will increase trust in the operation (assuming the operation is not spoofed by a MN) though will increase load on CNs.
3. Data sent to watchers might include: timestamp, object identifier, user id, IP of client.

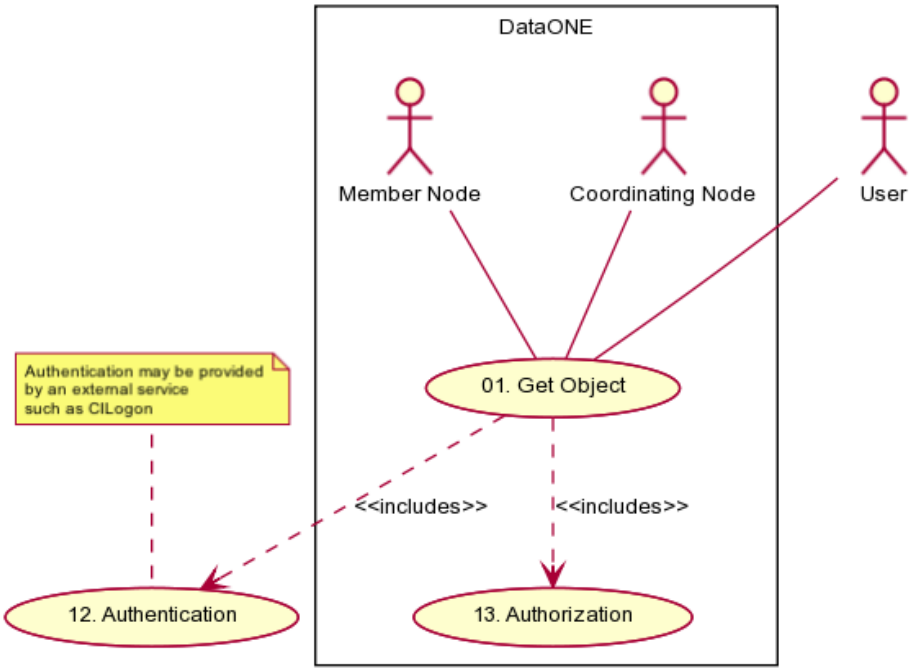


Figure 4.1: Use case 01 diagram showing actors and components involved in this action.

4.4 Use Case 02 - List GUIDs By Search

Author VDC TWG

Date

- 20090927 Provided more detail to use case
- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications

Version Draft

Goal Get list of GUIDs from metadata search (anon and authenticated).

Summary A user performs a search against the DataONE system and receives a list of object identifiers (GUIDs) that match the search criteria. The list of GUIDs is filtered such that only objects for which the user has read permission will be returned.

This use case assumes that the search is being performed by submitting a query against a CN.

Actors

- Client performing search operation
- Coordinating Node

Preconditions

- Client has authenticated to at the desired level (e.g. client may not have authenticated, so access might be anonymous).

Triggers

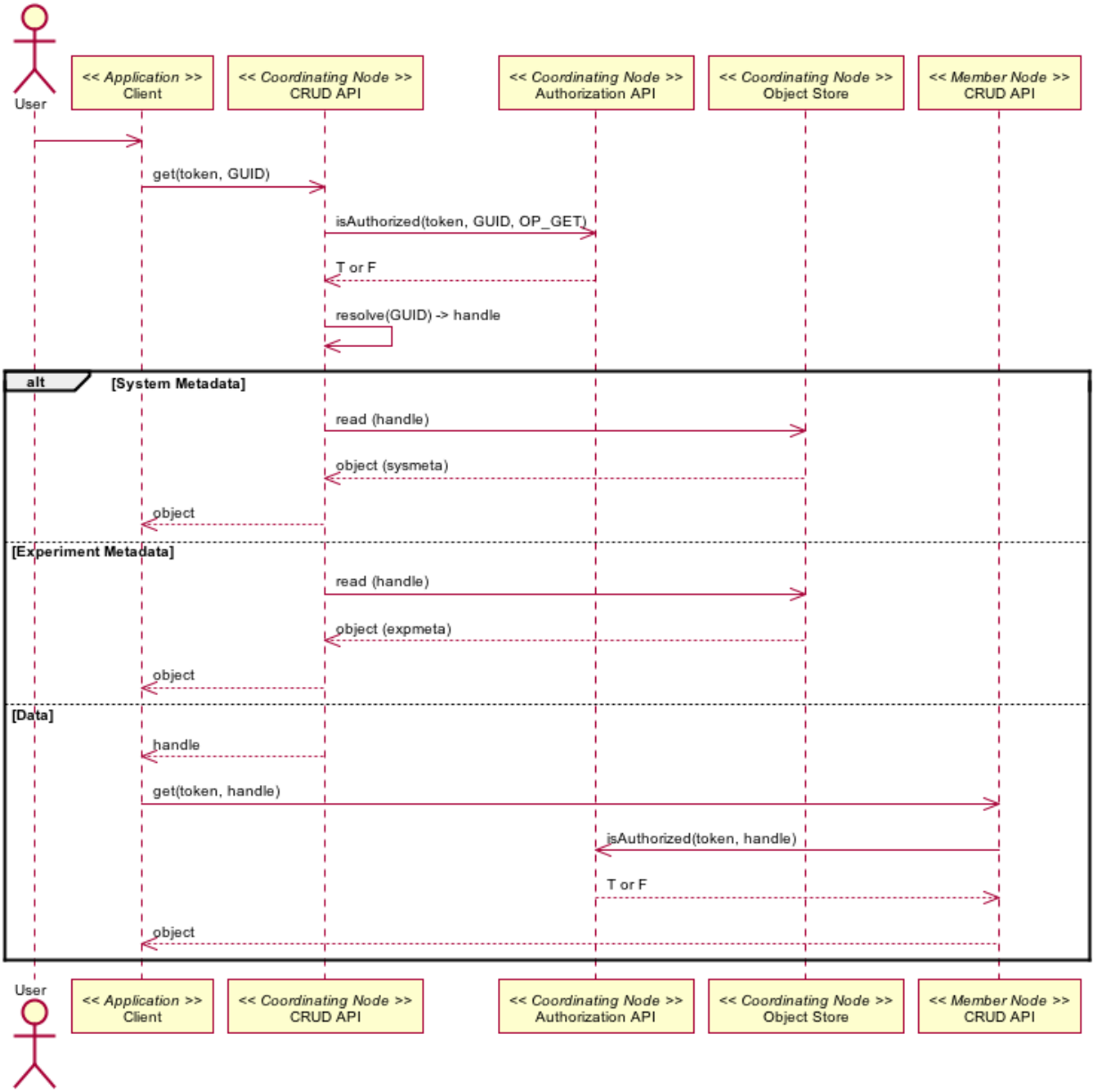


Figure 4.2: Sequence diagram for Use Case 01 illustrating the sequence for retrieving an object identified by a GUID from the DataONE system. Note the dichotomy in response depending on whether the object referenced by the GUID is a metadata document (record) or a data object. In the case of metadata, the metadata document is returned. In the case of data, a URL is returned that points drectly to a service endpoint for retrieving the bytes (perhaps instead always return metadata, for a data object it would be the system metadata?).

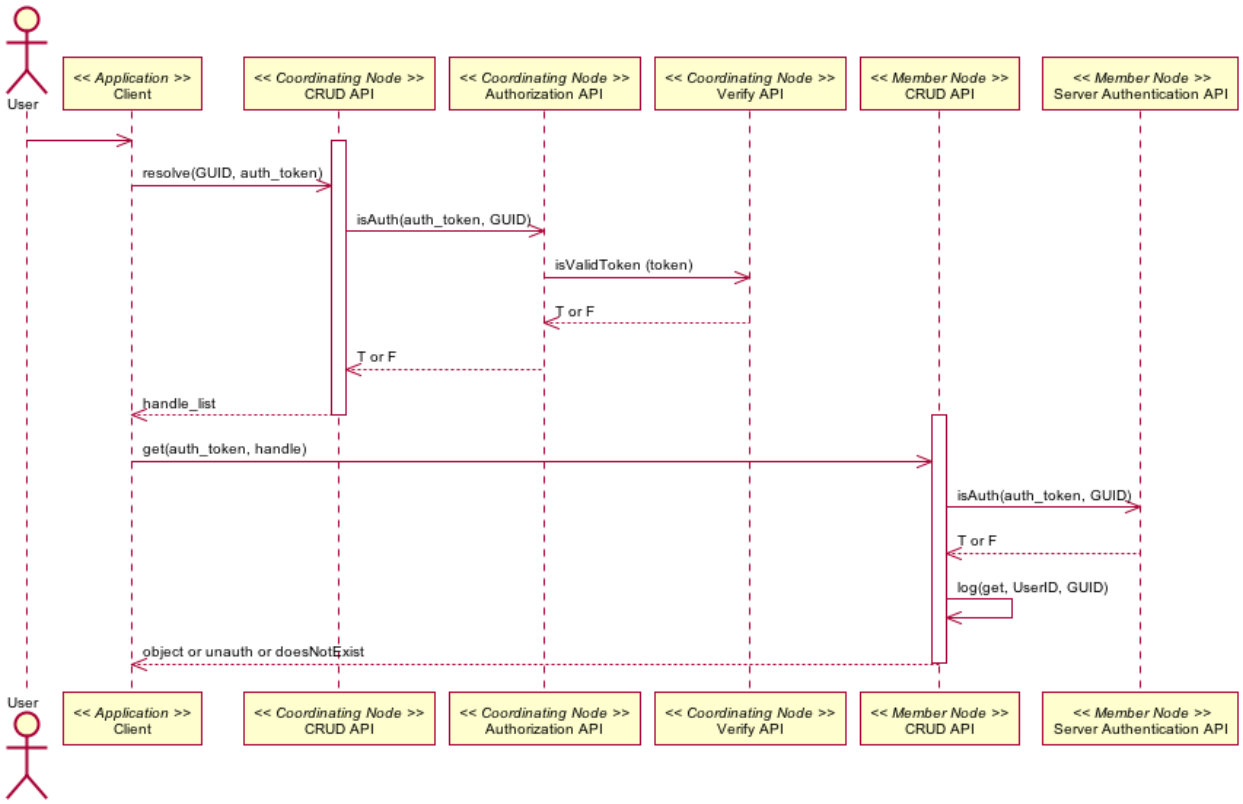


Figure 4.3: OLD: Sequence diagram for retrieving an object from DataONE.

- A search is performed against the DataONE system

Post Conditions

- The client has a list of GUIDs for which they have permission to read and match the supplied query, or an error message if the query was invalid.
- The log is updated with the request

Notes

- lots of stuff buried in query. What information/interface is needed for resolution/expansion of the query? For example a particular user may be identified with a particular community which uses a particular ontology. Where are things like ontologies maintained and where are the interfaces between these and searches maintained? Searches need to be able to respond to the dynamic nature of domain ontologies. Does the ontology need to have authorization tied to it? A use case would be an ontology which is under development and not ready for general distribution. Another case might be where the ontology contains representations of human subject information, meaning that "cancer survivors" includes this list of people. Or where the ontology provides geospatial information about restricted information (locations of USFS FIA plots). Similar issues apply to T&E species information.
- Group structure representation: A user is a member of zero to many groups, authorization may be carried with that group membership. This is down in the verify API, which needs to take into account group membership and group permission inheritance. Question: What is the resolution for conflicting access permissions?
- Q. Can a query be done at a member node? The diagram implies that the query is only done at a coordinating node. A member node may have a subset (or the full set) of metadata and the user may wish to do the search only at a member node, knowing that the subset is restricted to data relevant to a community of practice.

A. First iteration will provide query service on CNs, though nothing to prevent MNs implementing query as well.

- Where is the filtering done for the results, in terms of when does the result set get filtered to those which the user can see?

  Q. Does the model allow the query service to do two searches and union the results? The first query might be against the set of records known to be publicly readable, with a follow-on query against the set of records known to have restricted access. Is this envisioning the query service generating a set of results (against the restricted records) and passing that list to the authorization API, or is this envisioning a one by one situation. Or does the authorization API pass back to the query service the list of records that the user is authorized to see, and that gets anded with the result set.

  A. The resultset is a list of GUIDs that needs to be vetted for read access. Hence query results can be merged (union) or joined as the output from such operations is always a list of GUIDs that will be vetted for read access.

- How distributed is the database of users and their authorization records?

  Q. Does the coordinating node have to know all authorization records, with that being replicated? Or is some amount of the authorization maintained by (for example) the "owning" member node and the search system may need to contact multiple authorization authorities for the records in a particular result set? It also implies that if the authorization information is at the coordinating node, that the authorization authority needs a means to validate the replicated authorization information to ensure that the info at the cn is up-to-date.

  A. One scenario is that all authorization information is stored in the system metadata with each object. Then authorization can be performed by MNs or CNs.

- Q. Why does step 4 need to validate the token, given that the token was generated in step 2?

  A. This is normal security validation of all user inputs once those have gone out of the control of the issuing service. Also applies to cases where tokens have a specific lifetime, and the token may be valid, but expired. Authorization token lifetimes are important for both interactive sessions and for cases like workflows, where the session may run for days.

- What logging level is required for the operation? Should the search appear as a hit against each object? Just log the query (and perhaps who did it)?

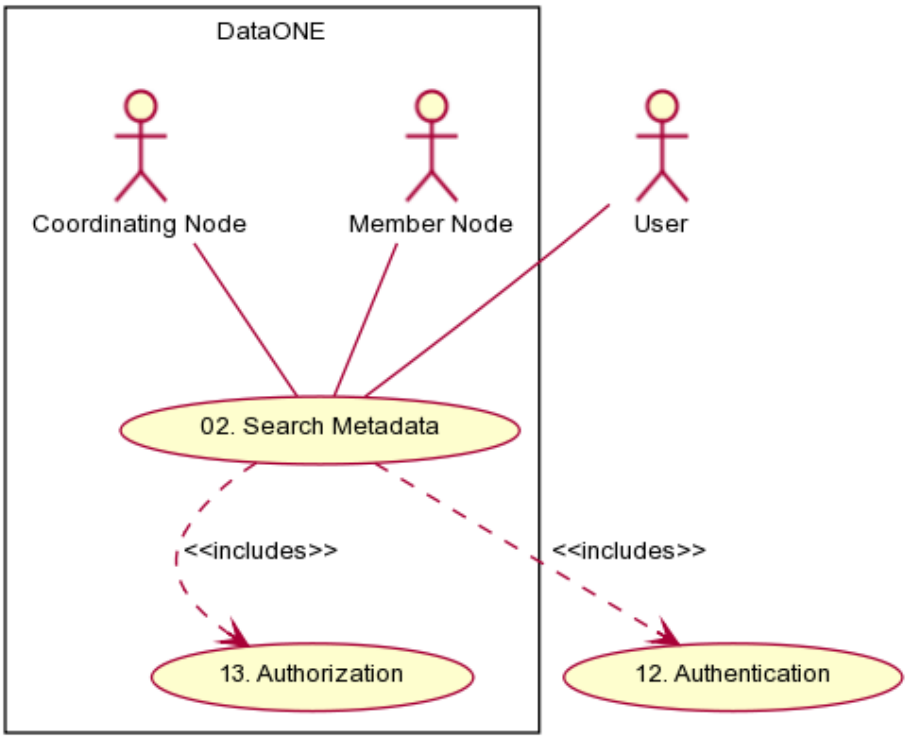


Figure 4.4: Use case 02.

4.5 Use Case 03 - Register MN

Author VDC TWG

Date

- 20090927 More detail added to use case description
- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Register a new Member Node.

Summary This use case describes the technical process for addition of a new member node (MN) to to the DataONE infrastructure. It is assumed that the appropriate social contracts have been formed and the MN is operational, ready to be connected.

The MN is identified by a URL which is the service endpoint. A DataONE administrator adds the MN URL to the MN registry. A CN retrieves the registration request from the message queue and queries the MN for capabilities, then verifies those capabilities match what was advertised. If everything is OK, the MN is made a live member of DataONE and replication of content on the MN is scheduled. The new MN also becomes a receiver for content replicated from other MNs.

Actors MN, CN, Administrator

Preconditions

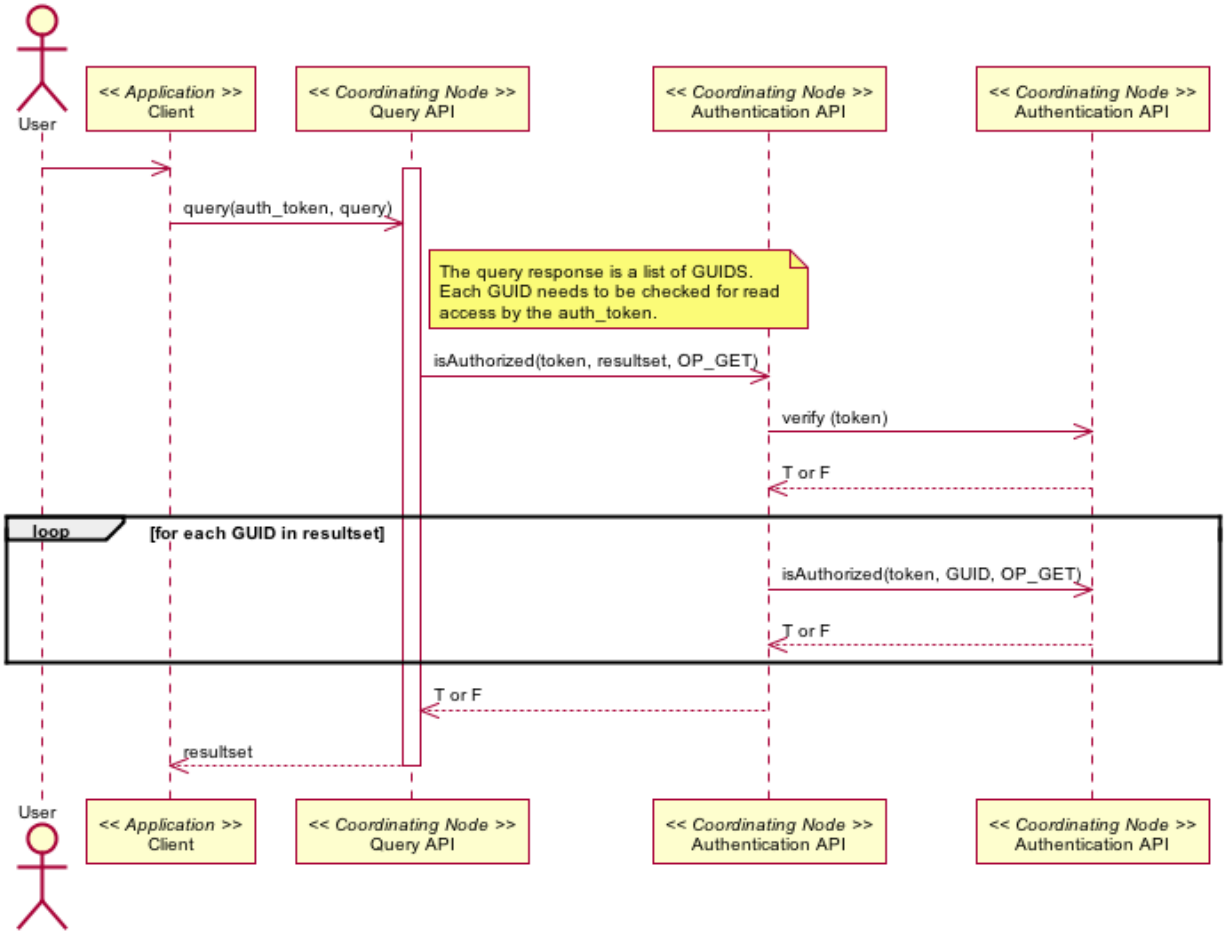


Figure 4.5: Interactions for use case 02.

- The MN is operational
- There is an agreement between the DataONE operators and the MN operator that the MN is to be added to DataONE
- The CNs are ready for receiving MN registrations

Triggers

- A new Member Node is ready to be brought online and a DataONE administrator initiates the process.

Post Conditions

- The new MN operates as part of the DataONE infrastructure
- DataONE infrastructure resources are incremented by the amount available at the new MN
- The operation outcomes are logged
- Synchronization of the MN commences as per scheduled operation
- Synchronize capabilities metadata across CNs
- Notify MN administrators of operation status (email?)
- The new MN is added to CN list of targets for replication

Notes

- Specify default replication policies
- Also check for version updates
- Should new nodes be registered with specified trust levels?
- Are there different levels of trust for member nodes?
- Data providers that use acceptable services can still be discovered and accessed without registering in the DataONE registry (Conflicting, as someone has to register it).

  However, a MN that has not registered but does expose DataONE services is not part of the DataONE infrastructure and so does not participate in replication or other DataONE services.
- Allow service providers to register their services, (such as data extraction services) (ala GEOSS), but include mapping to higher semantic model
- For well known services, registration system must be able to describe constraints (e.g., allowable inputs, outputs, algorithms that can be specified)
- Services can be parameterized
- Member node should be able to request the coordinating node to re-validate its capabilities list.

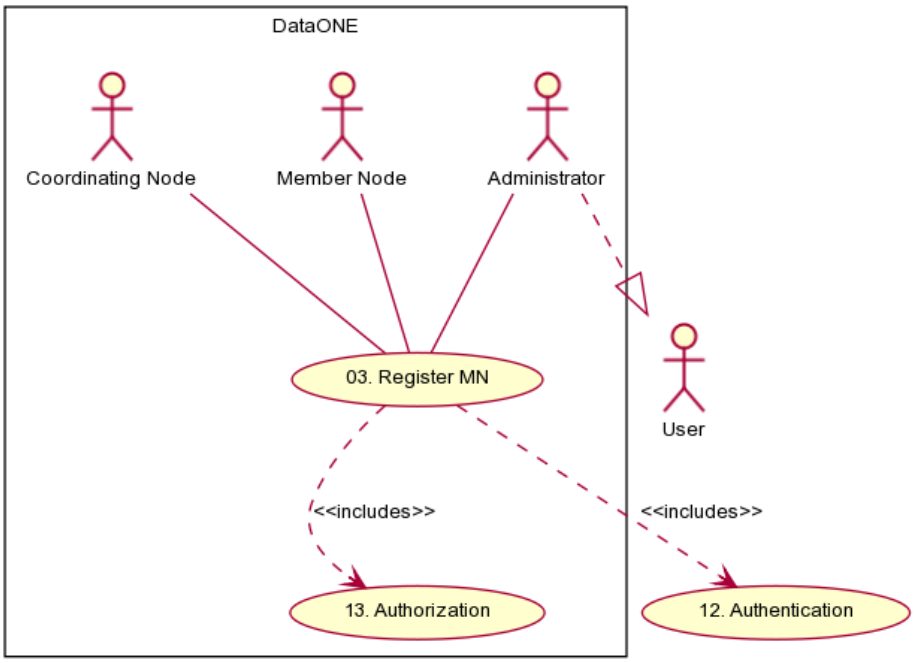


Figure 4.6: Use case 03.

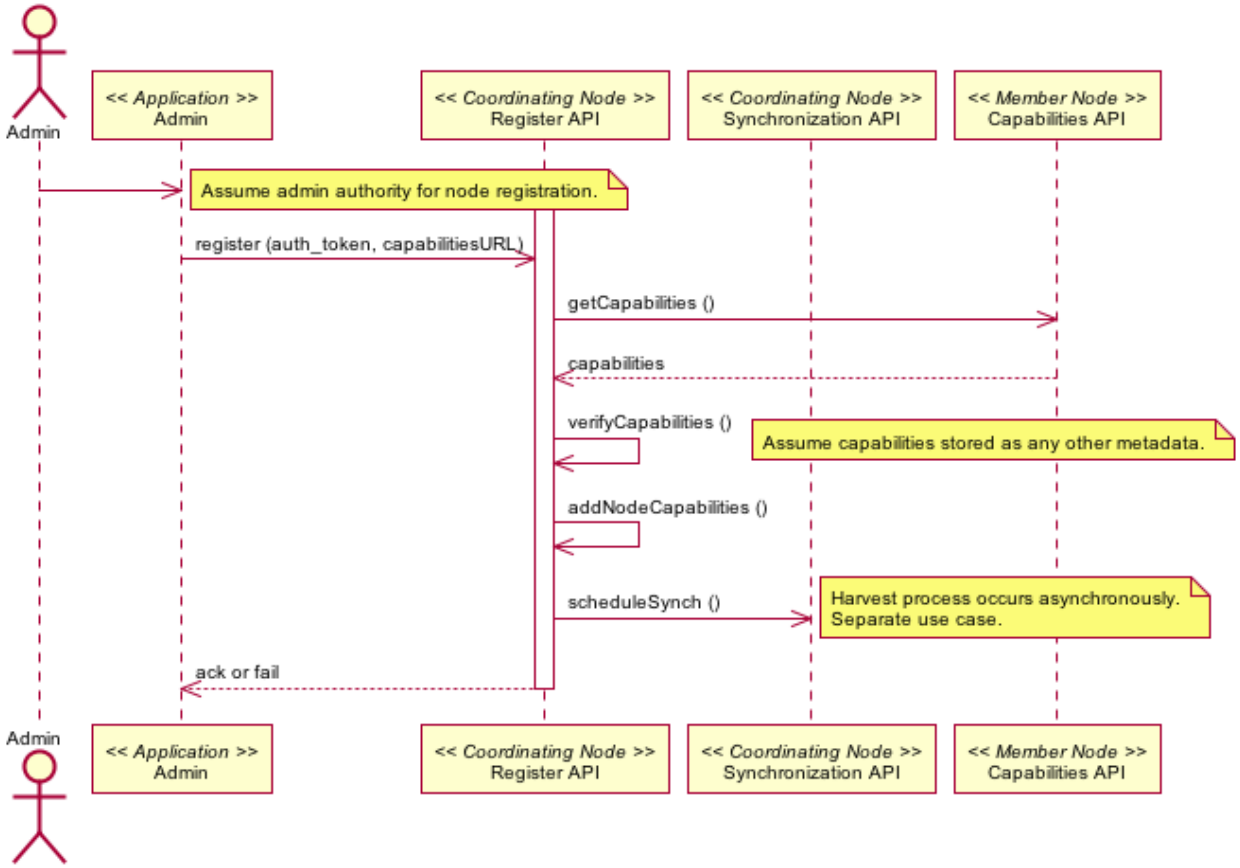


Figure 4.7: Interactions for use case 03.

4.6 Use Case 04 - CRUD (Create, Update Delete)

Author VDC TWG

Date

- 20090929 Filling out details
- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Create, update or delete metadata record on a Member Node.

Summary A user is creating a new metadata record on a Member Node (MN). The mechanism by which the user does this is out of scope for the DataONE system, so this use case continues from the point where a new Data Package is present on the MN.

The metadata is retrieved by the CN using a pull mechanism (CN requests content from the MN).

The availability of new content on a MN is indicated by a flag in the response to a ping() message issued by a CN. The CN then retrieves a list of changes from the MN and for each change, retrieves the experiment metadata, stores it and generates a search metadata document by extracting content from the experiment metadata. The changes to the CN (stored experiment metadata, update search metadata) are replicated to the other CNs (Use Case ###). The CN schedules replication of the data package across other MNs (*Use Case 06*).

Actors Member Node, additional Member Nodes, Coordinating Nodes

Preconditions

- Use is authenticated and so client application has an auth_token for the process.
- DataONE system operational
- MN is registered and is monitored

Triggers

- New data package is present on Member Node
- An existing data package is modified on a Member Node
- A data package is deleted from a Member Node

Post Conditions

- New data package is replicated across MNs
- Search metadata is updated
- Operation recorded in logs
- CN update of search metadata
- CN replication of metadata
- Watchers notified of change

Notes

- Are IDs are pegged to versions? If so, update might differ more radically from create.
- If delete is a metadata operation only, then will be very similar to update. But if it is a physical operation, it becomes more complex. Will there be an archival copy? Will member nodes that hold replicas be "forced" to destroy the replicas?

- Is submit from a member node a push or pull?
- Should queuing of a submission within a member node be required for success? (We assume yes, to guarantee replication of submissions).
- Can data packages be deleted or modified?

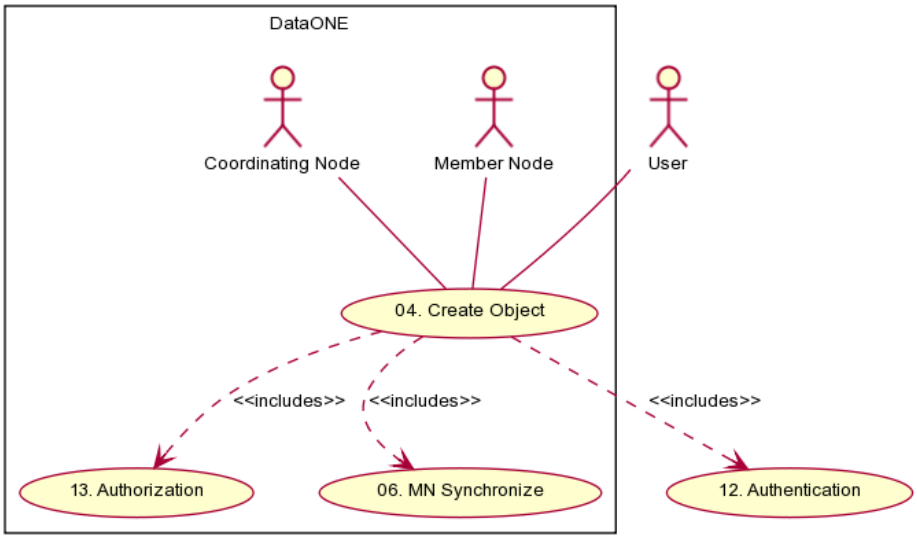


Figure 4.8: Use case 04.

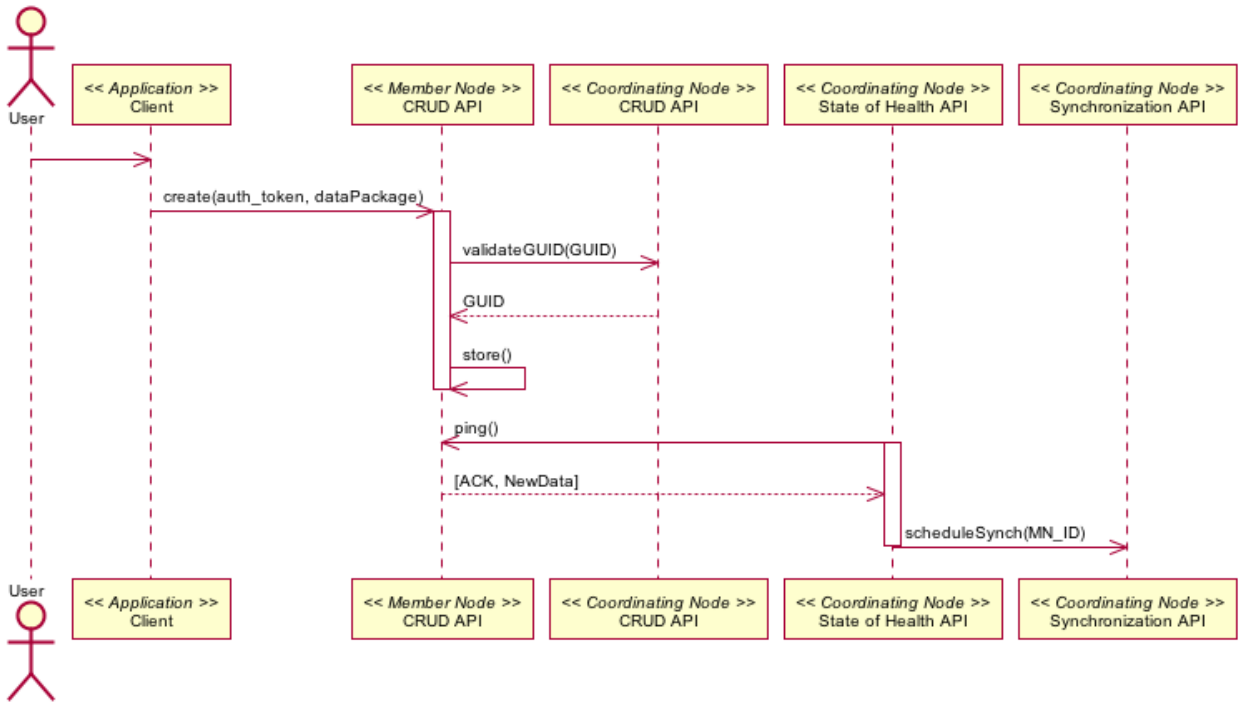


Figure 4.9: Create, update, delete, search metadata or data object in Member Node.

4.7 Use Case 05 - CRUD (Create, Update Delete)

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Create/Update/Delete data object in Member Node.

Summary This is a dupe of *Use Case 04* for now. May split out the update and delete portions to different use cases at some point in the future.

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

- See all questions above (Use Case #4) for metadata operations.
- client should be able ask CN for a list of acceptable MNs to upload data with specific characteristics (e.g., size, institution, theme)

4.8 Use Case 06 - MN Synchronize

Author VDC TWG

Date

- 20090930 Filled out content, changed mode to pull from MN
- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Replicate / synchronize metadata record between Member Node and Coordinating Node.

Summary

As data packages, or portions there of are created or modified, the metadata associated with those needs to be replicated to the coordinating nodes.

The presence of new or changed information on a Member Node (MN) is made known to a Coordinating Node (CN) through the status information in a `ping()` response. If so indicated, the CN schedules a synchronization operation with the MN, a list of changed object GUIDS is retrieved by the CN, and the CN proceeds to retrieve and process each object identified.

If new data packages are present on the MN, then a MN-MN synchronization process is scheduled.

Actors Member Node, Coordinating Node

Preconditions

- Member Node has new or changed data packages

Triggers

- New or changed content is present on a Member Node

Post Conditions

- MN is in sync with CN
- Log
- Watchers for changes to MN, CN, or DataONE system

Notes

- Overloading ping() for notification of change seems a little bit of a hack, though changing the name from ping() to status() makes it seem OK.

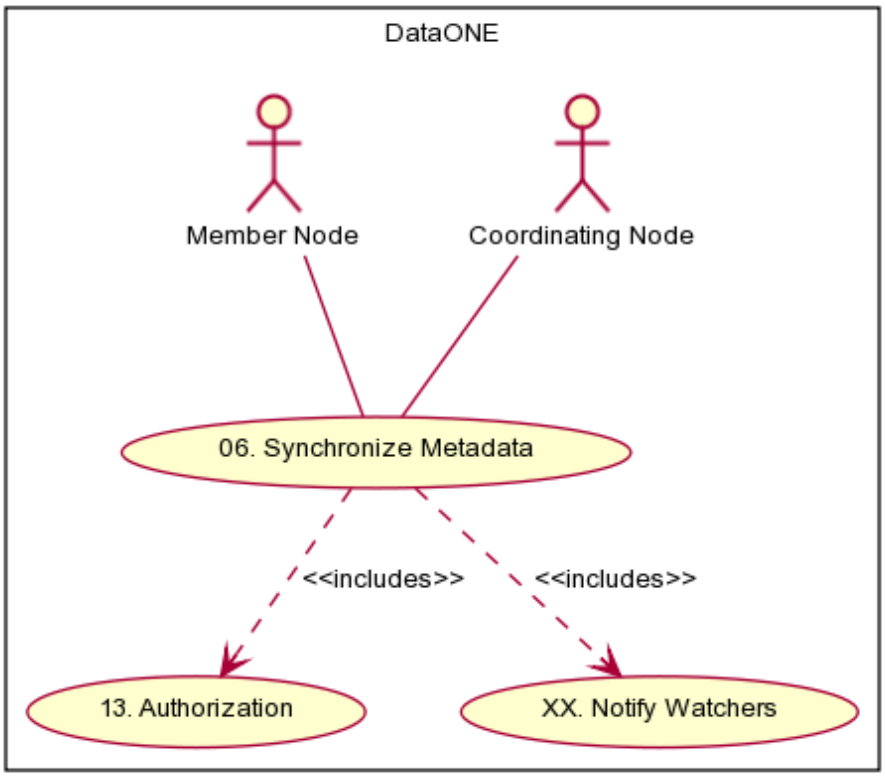


Figure 4.10: Use case 06.

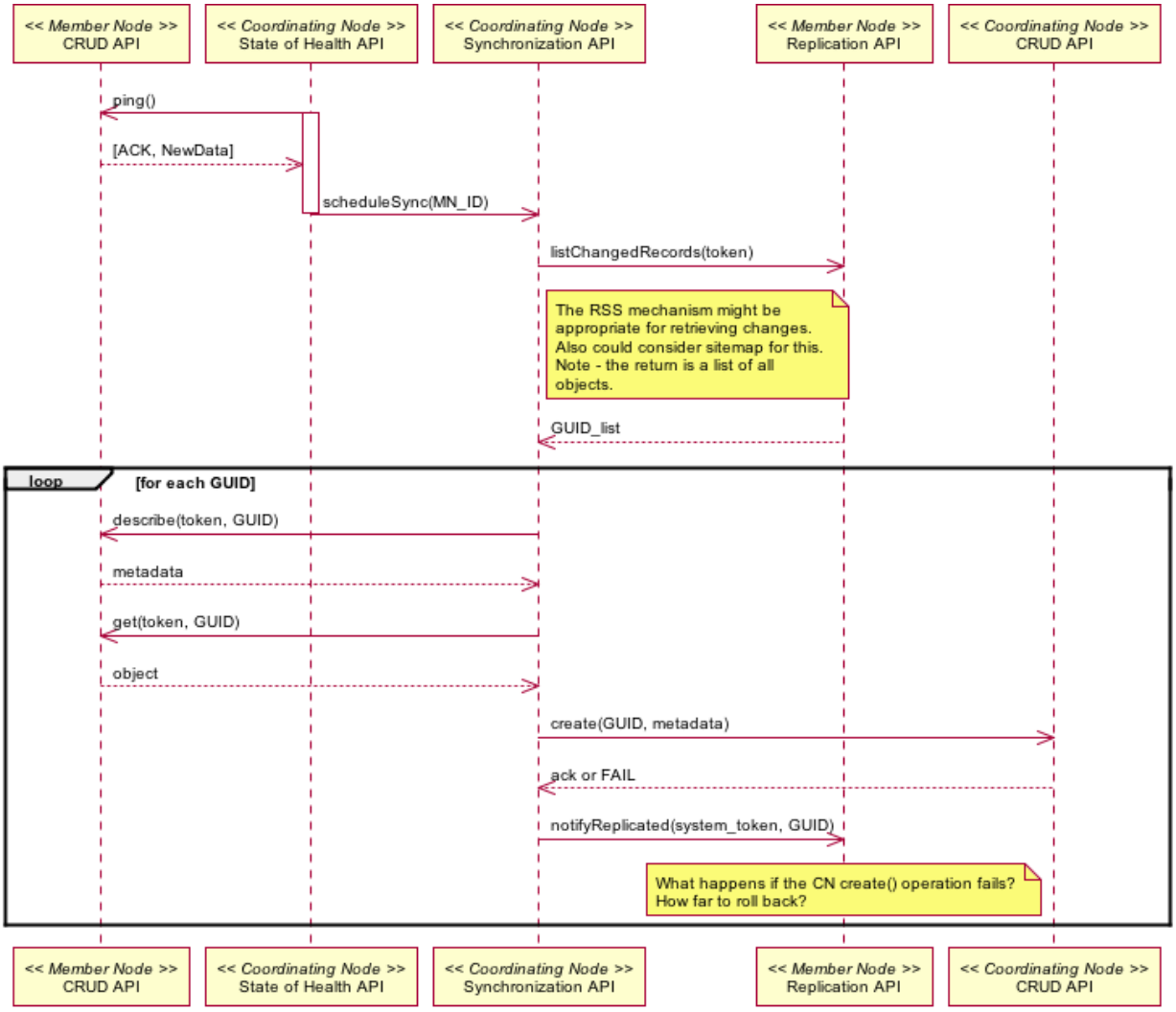


Figure 4.11: Synchronize metadata between Member and Coordinating Node

4.9 Use Case 07 - CN Batch Upload

Author VDC TWG

Date

- 20090930 Flagged as dupe
- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Batch Operations - Coordinating Node requests metadata /data list from new Member Node and then batch upload (disable indexing for example to improve insert performance).

Summary

This is a dupe of *Use Case 06* after refactoring that use case use a pull mechanism to retrieve changed records. The intent is to streamline the retrieval of large amounts of content from a MN, which perhaps involves turning of indexing and other potentially time consuming operations to speed the upload process.

Actors

- CN
- MN

Preconditions

- Large amount of metadata data is required to be uploaded to the CN

Triggers

- Indication of a large amount of content to retrieves from a MN

Post Conditions

- The upload is completed, CN synchronization is completed, MN replication is scheduled.

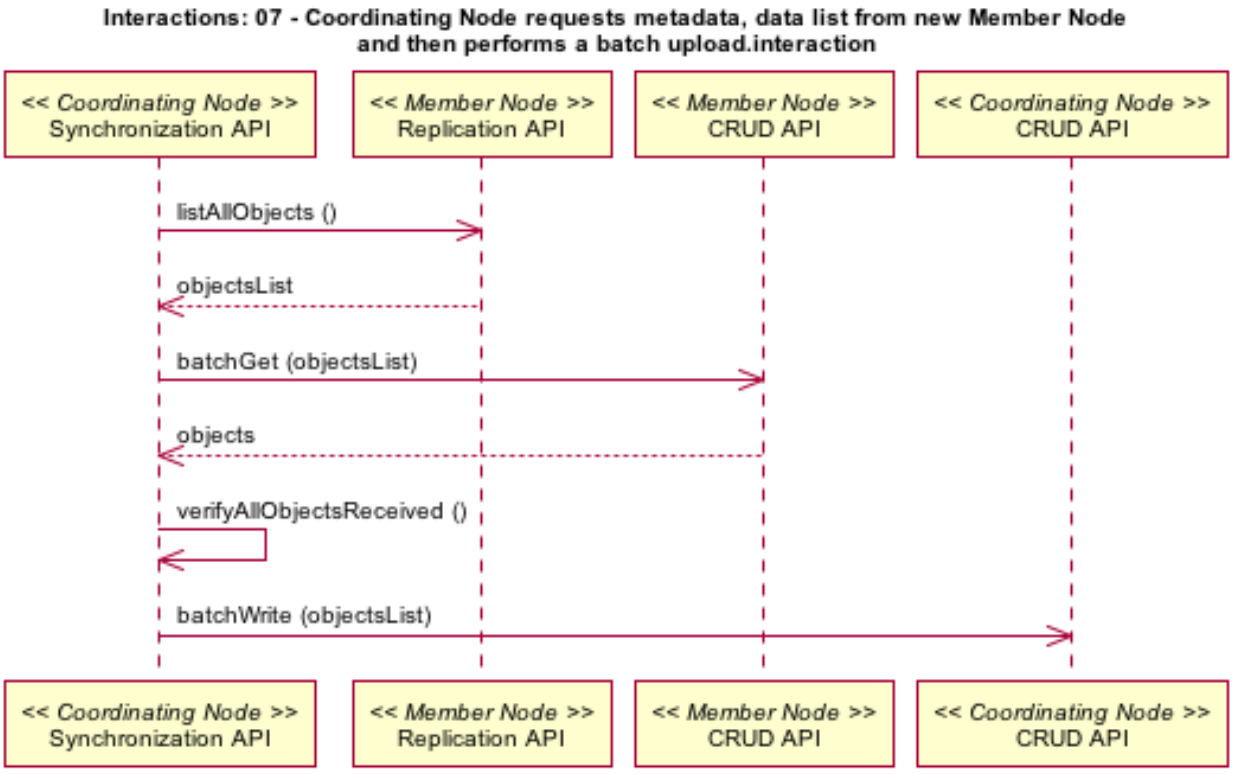


Figure 4.12: Interactions for use case 07.

4.10 Use Case 08 - Replication Policy Communication

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Communication of replication policy metadata between Member Nodes and Coordinating Nodes.

Summary

> The replication policy of Member Nodes (MN) indicates factors such as the amount of storage space available, bandwidth constraints, the types of data and metadata that can be managed, and perhaps access control restrictions. This information is used by Coordinating Nodes (CN) to balance the distribution of data packages throughout the DataONE system to achieve the goals of data package persistence and accessibility.

Actors Member Node, Coordinating Node

Preconditions

- Member Node is registered with a Coordinating Node
- Member Node is operational

Triggers

- A Member Node changes available capacity, bandwidth or some other operating characteristic
- A Member Node changes available services
- A Member Node software stack is updated (e.g. new version)

Post Conditions

- The DataONE system is updated with current state of available resources
- Change in replication policy may trigger a re-synchronization (e.g. if storage capacity shrinks or is enlarged)
- Operation is logged
- Watchers are notified in change in DataONE system property

Notes

- The goal as stated originally for this use case was "Communication of replication policy metadata among Member Nodes and Coordinating Nodes."
- There should be some restriction on how often replication policy can be changed to avoid thrashing that might occur for example, if a MN flips between significant differences in storage capacities (low to high, then low again).

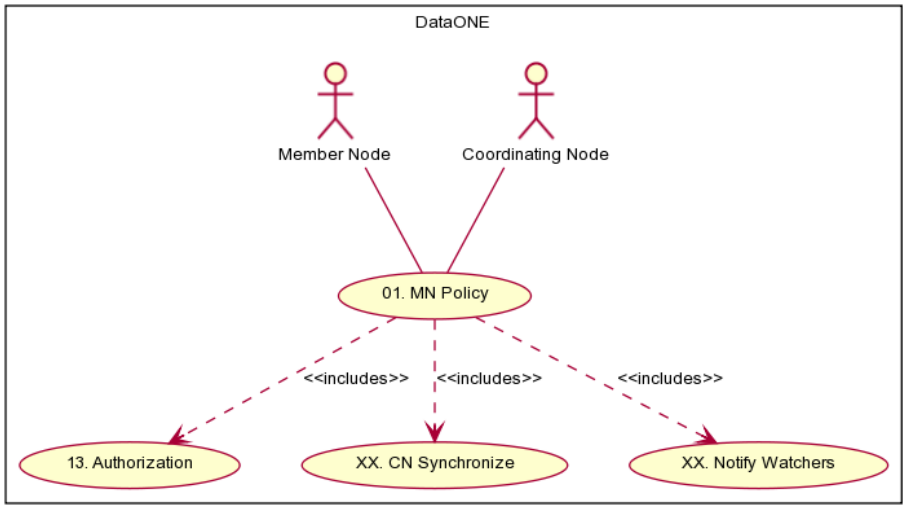


Figure 4.13: Use case 08.

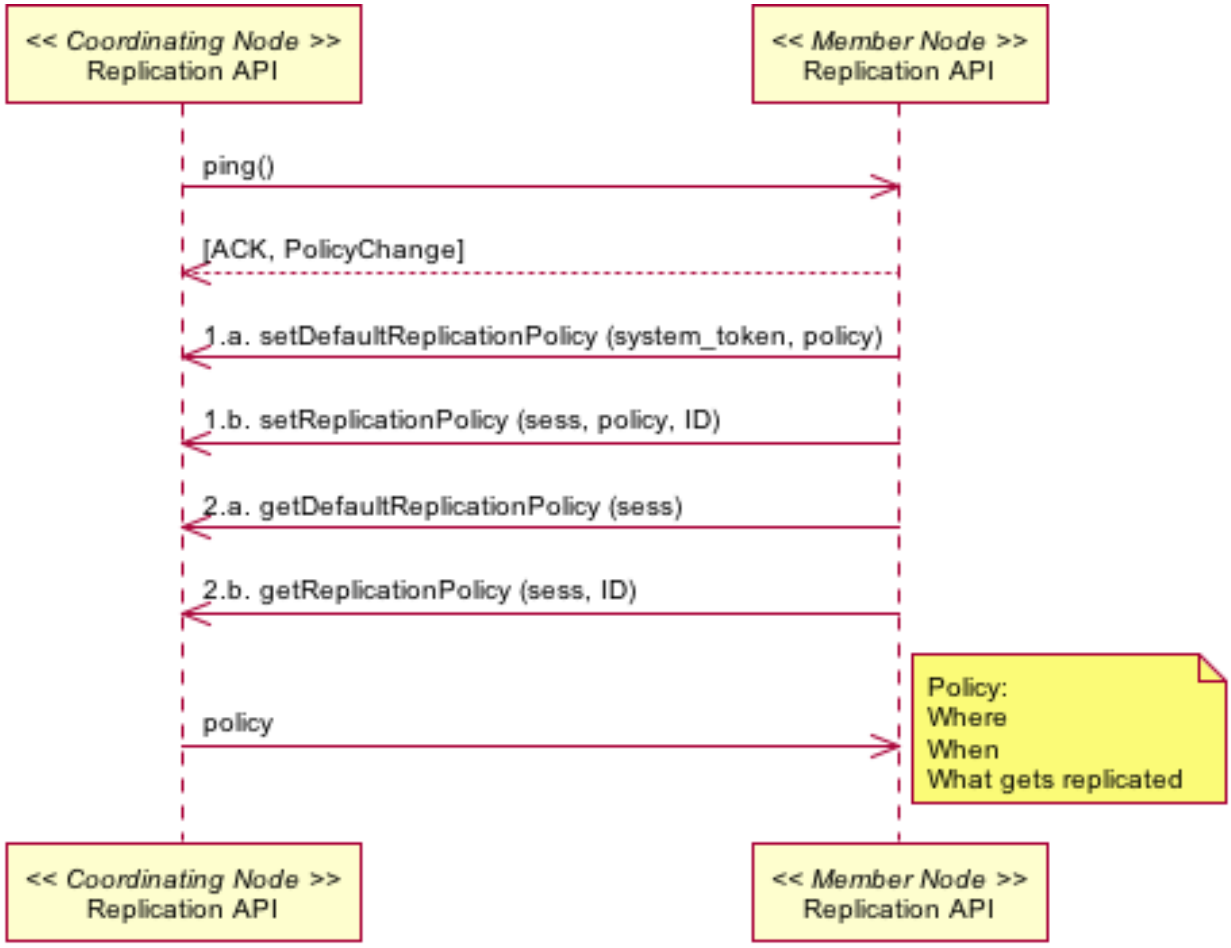


Figure 4.14: Communication of Replication Policy Metadata

4.11 Use Case 09 - Replicate MN to MN

Author VDC TWG

Date

- 20091001 Filled out description a bit more
- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Replicate data from Member Node to Member Node.

Summary

Replication of content between Member Nodes (MN) is done to improve persistence of information (avoid data loss with loss of MN) and to improve accessibility (more choices for content retrieval can lower bandwidth requirements for any particular MN). The process of replication may be facilitated by a Coordinating Node (CN), or perhaps only between MNs. The interaction diagram presented here follows the former notion, where replication is facilitated by a CN.

A full copy of a data package is made during the replication process, so the original experiment metadata and data is copied to the recipient MN. An important question is whether that content is stored in it's original form (for data this must be the case) or translated into a form that is supported by the MN (likely to be the case for experiment metadata).

If experiment metadata is translated into a local dialect by a recipient MN, then it is important that the original metadata is preserved on the CNs, as it is always possible that the original MN where the content was published may go offline or be removed from the DataONE system.

Actors Member Node x2, Coordinating Node

Preconditions

- Content is present on a Member Node

Triggers

- New data is available on a Member Node
- Information on a Member Node is altered
- Capabilities of a Member Node changes (accepting more or less content)
- Replication policy of DataONE or a Member Node changes

Post Conditions

- Content is present on one more Member Node
- System metadata is updated to reflect the change
- Watchers are notified of the change
- Member Node and Coordinating Node logs are updated

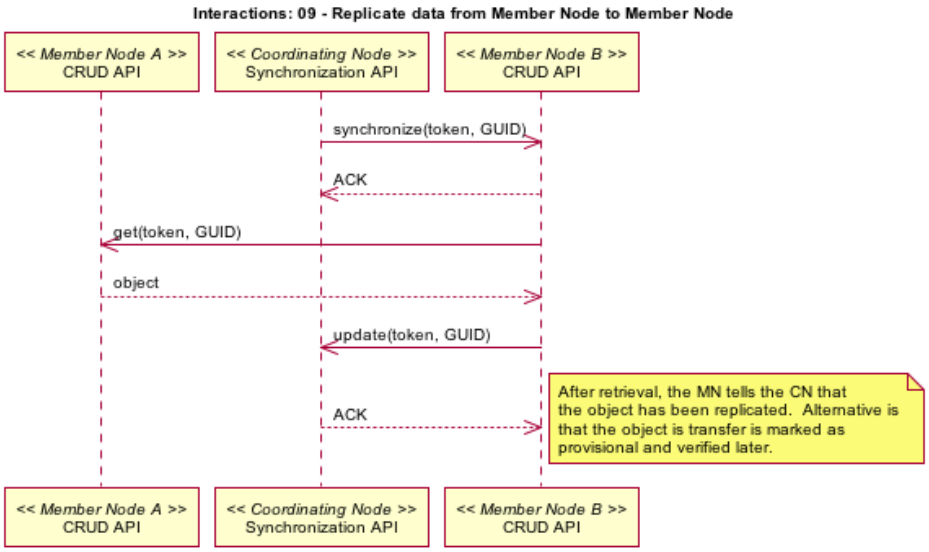


Figure 4.15: Interactions for use case 09. The diagram describes transfer of a single object from MN_B to MN_A as directed by a CN. It isassumed that the object does not exist on MN_B and the object has been identified as requiring synchronization by the CN checking its status in the system metadata.

4.12 Use Case 10 - MN Status Reports

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Coordinating Node checks "liveness" of all Member Nodes - checks ping, service x, load, space, bandwidth, transaction rate, ...

Summary

As part of the infrastructure monitoring process, a CN checks on the "liveness" of all Member Nodes. This operation will be performed with moderate frequency (perhaps in the order of once every few minutes).

The responses from the MNs should report sufficient metrics to gauge their performance in absolute terms and also relative to other MNs.

Actors CN, MN

Preconditions

- CNs are operational
- MN is registered

Triggers

- Timed interval
- Registration change

Post Conditions

- Status metadata is updated at CNs

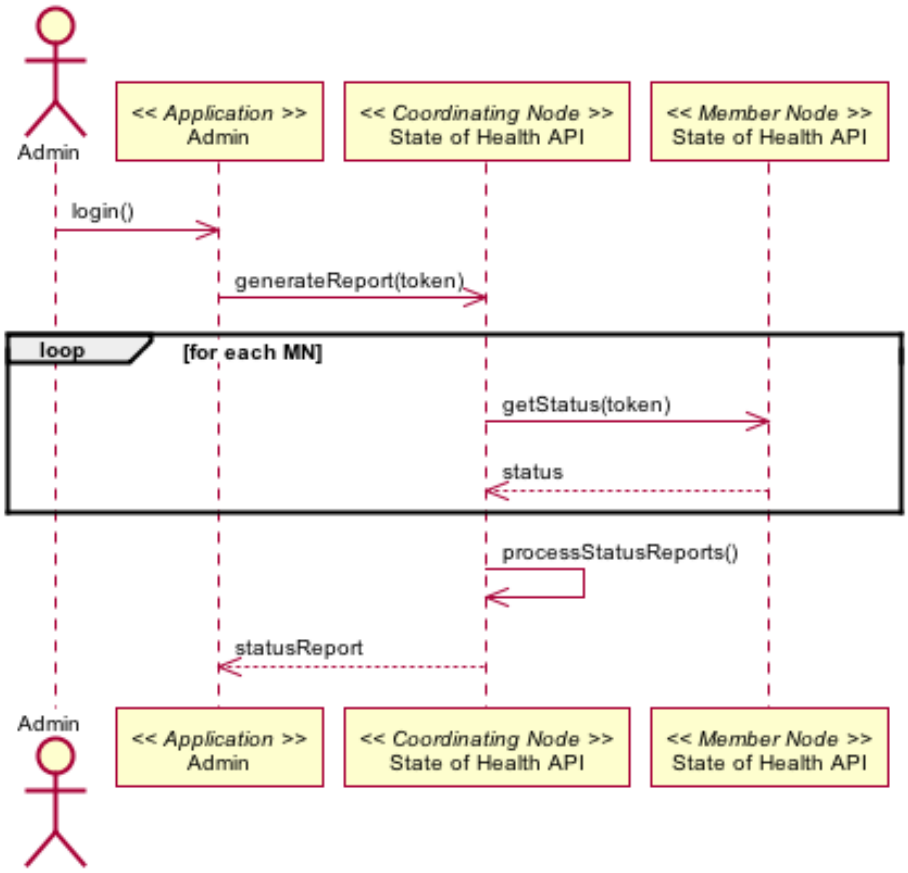


Figure 4.16: Interactons for use case 10. Describes a synchronous process for generating a report. Most likely, a better mechanism would be for a background process to be accumulating the necessary statistics from Member Nodes, then the generateReport function would operate on the accumulated data rather than forcing a fresh copy.

4.13 Use Case 11 - CRUD Workflow Objects

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Create / update / delete / search workflow objects.

Summary

This use case describes the management of a workflow object within the DataONE infrastructure.

It is not clear why this should be treated any differently from the management of regular objects.

Actors

- User
- Client application
- Member Node
- Coordinating Node

Preconditions

- DataONE system is operational
- User has authenticated with the DataONE system

Triggers

- User adds, deletes, or modifies a workflow object

Post Conditions

- Workflow object is stored on a Member Node
- Metadata about workflow object is stored on Coordinating Nodes
- Replication of workflow object across multiple Member Nodes is initiated

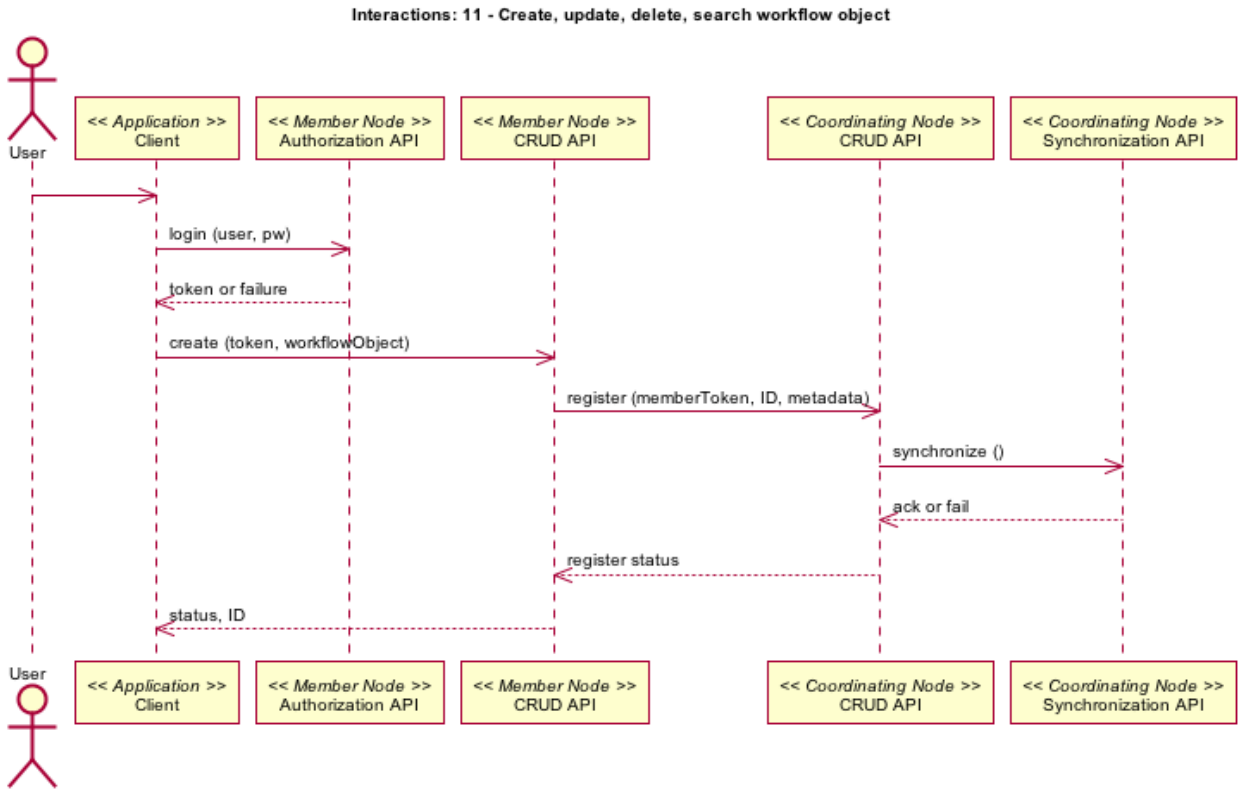


Figure 4.17: Interactions for use case 11.

4.14 Use Case 12 - User Authentication

Author VDC TWG

Date

- 20091001 Filled out more details
- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal User Authentication - Person via client software authenticates against Identify Provider to establish session token.

Summary Many operations in the DataONE system require user authentication to ensure that the user's identity is known to the system, and that appropriate access controls can be executed based on the identity.

The actual identity and authentication framework may exist outside of DataONE (one possible example might be the emerging CILogon service).

The authentication process might occur using either a Coordinating Node or a Member Node (if it implements the DataONE authentication API).

See also some notes about discussion at *Authentication in DataONE*

Actors User, Member Node, Coordinating Node, Authentication System

Preconditions

- User is not authenticated in the system

Triggers

- A user logs on to the DataONE system.
- A user tries to perform an operation in the DataONE system that requires authentication.

Post Conditions

- An `auth_token` is returned to the user (application) that can be used for future interactions with the DataONE system.
- In the event of authentication failure, the `auth_token` will provide identity equivalent to an anonymous user.
- The authentication operation is recorded in MN or CN logs (depending on where the authentication attempt was made)

Notes

- To be a bit more abstract, it might be good to use something like login(user, credentials) or login(subject) for the method signature. Here "subject" would be similar to the javax.security.auth.Subject concept. (PEA)
- Can logins also happening against a MN, which simply proxies the request up to the CN? (PEA)
- Need to specify a time to live for auth_tokens. Is this part of the authentication sub-system? If so, can the TTL be altered by the DataONE system?

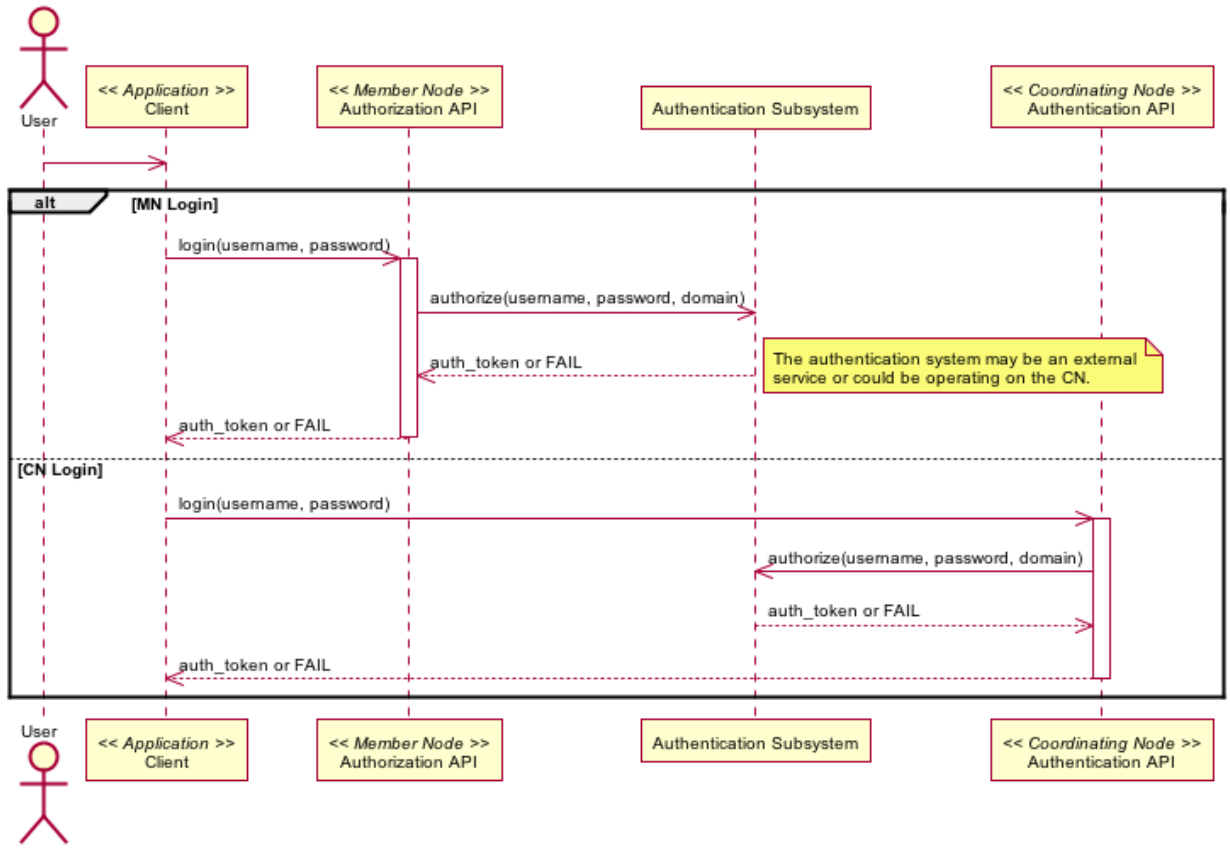


Figure 4.18: Interactions for use case 12.

4.15 Use Case 13 - User Authorization

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Determine if a user is authorized to perform some action on an object (e.g. get, put, query, delete, ...).

Summary

User authorization is the procedure that is followed to determine if the authenticated user is authorized to perform some operation. The result will always be True or False.

Note: Need to define which objects require access control / authorization

Actors

- User
- Client application
- Member Node
- Coordinating Node

Preconditions

- User is at some level of authentication (may be anonymous or logged in)

Triggers

- User is requesting some operation to be performed on an object
- An operation acting as a proxy for a user performs some operation on an object

Post Conditions

- The operation continues if the user is authorized or terminates raising an unauthorized exception.

4.16 Use Case 14 - System Authentication and Authorization

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal

A system process authenticates and performs some system operation (e.g. replication). In this example, a Member Node authenticates to enable transfer of content from MN A to MN B.

Summary

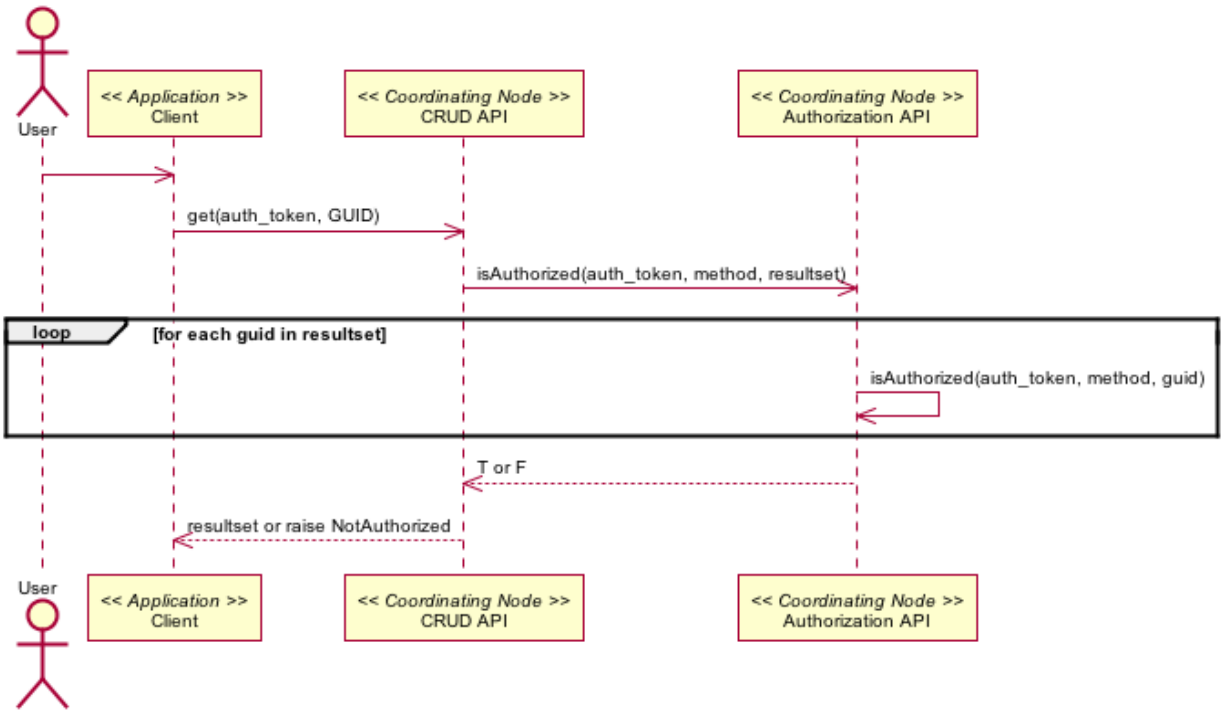


Figure 4.19: Interactions for use case 13.

System operations are required for many operations including the management of content across Member Nodes and replication between Coordinating Nodes. This use case describes the system level interactions required to perform a system level operation.

Actors

- Coordinating Node
- Member Node(s)

Preconditions

- Operational system

Triggers

- A system level operation is invoked.

Post Conditions

- The operation is completed
- The authentication token is optionally disposed

4.17 Use Case 15 - Account Management

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

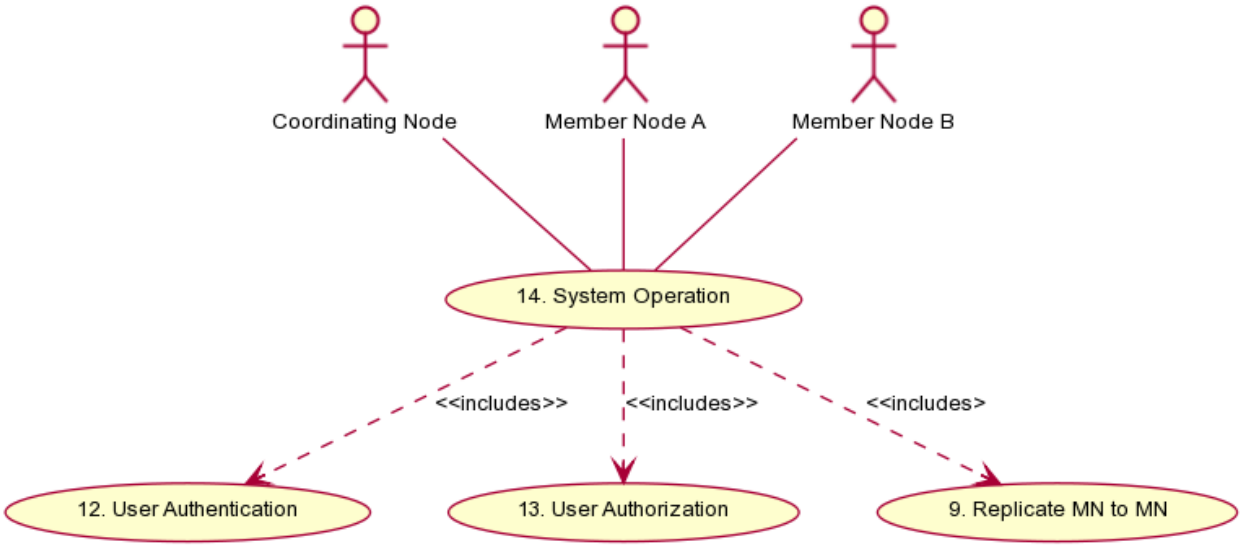


Figure 4.20: Use case 14.

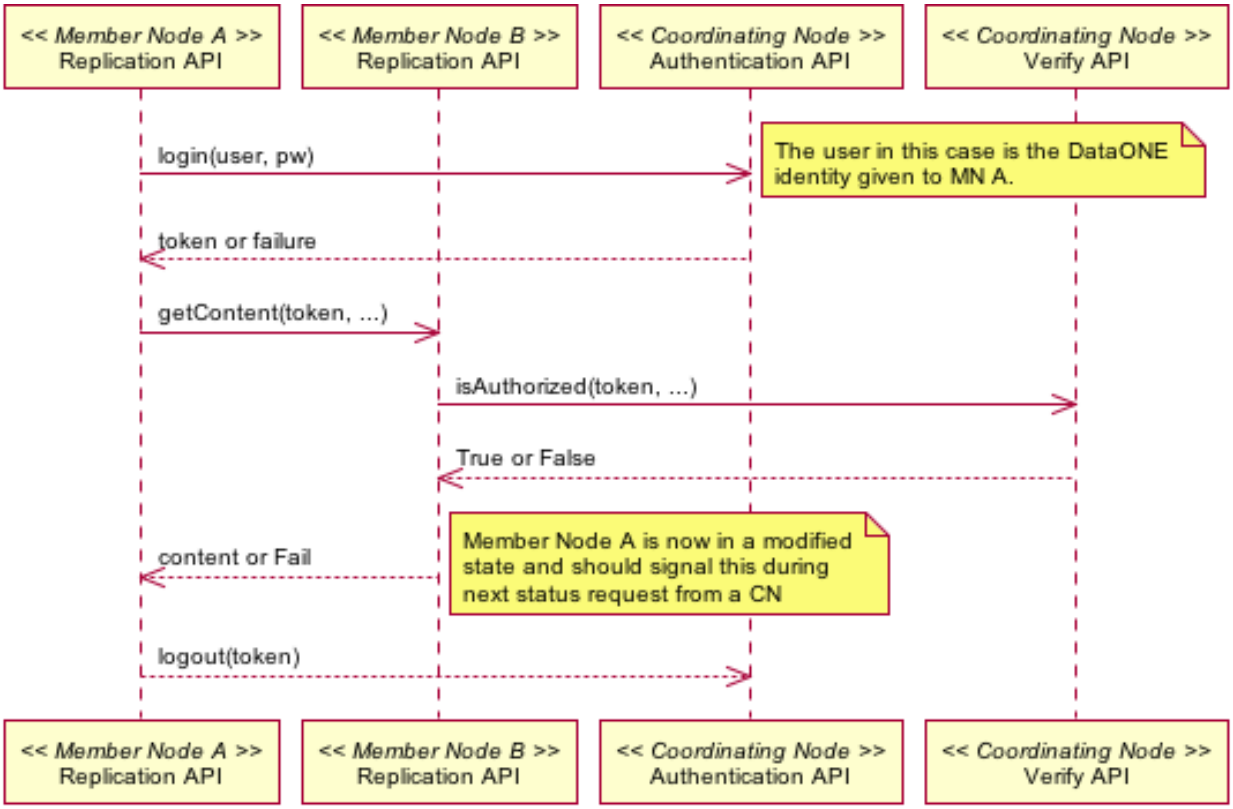


Figure 4.21: Interactions for use case 14.

Version Draft

Goal

Edit a user account. This includes creating, deleting, editing.

User Account Management - Create new user account on Identity Provider (also edit, delete, ...).

Summary

Perform basic account management operations. This process can be quite complex depending on the identity provider in use and the security policies that need to be addressed.

The use case and interaction presented here assumes a simplistic operation that relies only upon email verification for the new account creation. A more sophisticated interaction might include administrative approval of the new account, selection of an identity provider to use, and assignment of roles based on the level of approval and the nature of the selected identity provider (i.e. trustworthiness of identity provider).

Actors

- New User
- Administrator
- Identity provider
- Coordinating Node

Preconditions

- System is operational and policy is in place to accept new users.

Triggers

- A new user account is requested.

Post Conditions

- New account is created (if accepted)
- Access control rules for new account are specified
- Account information is replicated across CNs

Notes

- By default, accounts have no real privileges. To get higher privileges, users may have to jump through more hoops (such as verifying their association with a project/institution)
- Presumably, if we are using external identity providers this user account management functionality isn't provided by the CN. Right? (PEA)

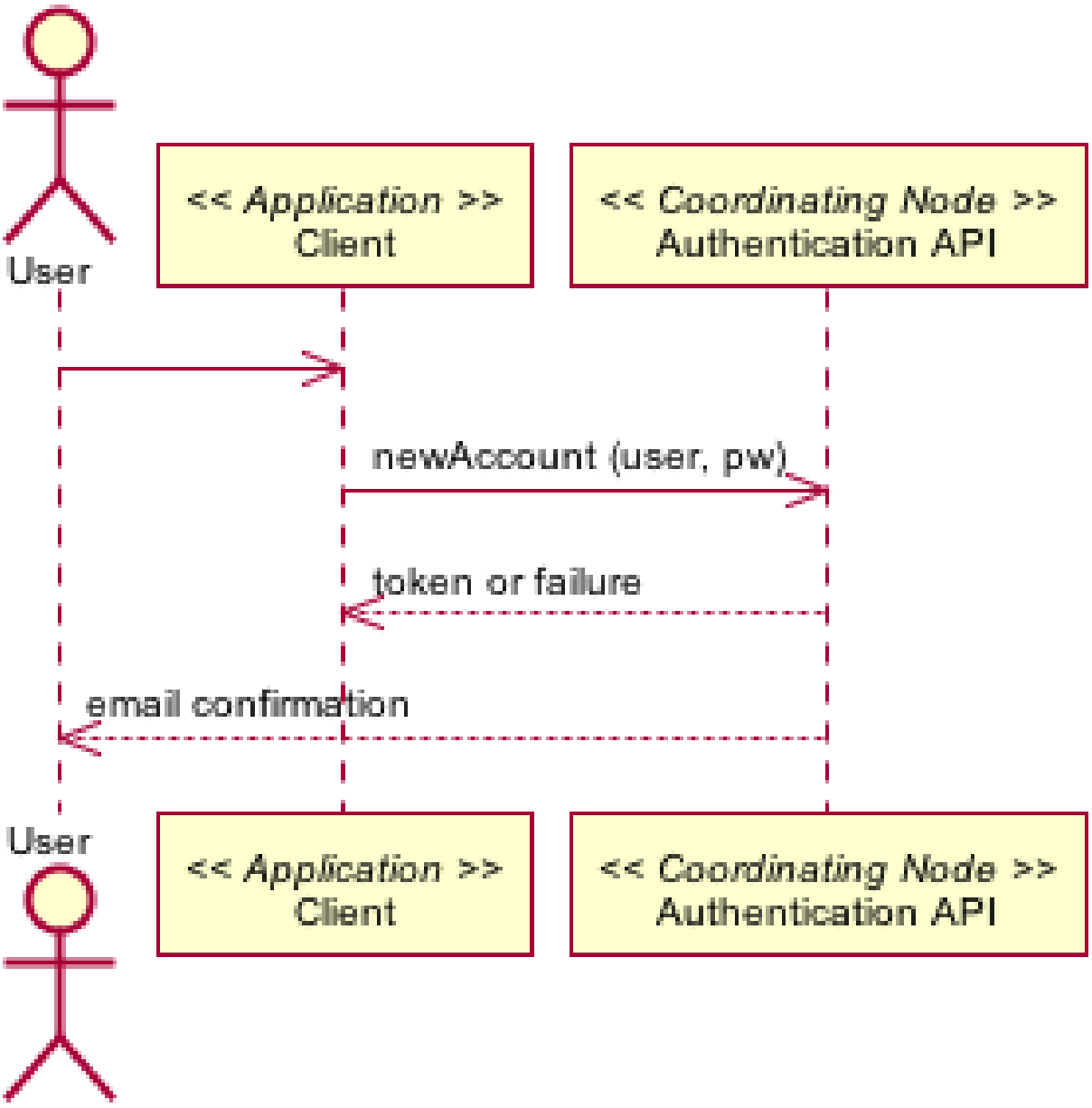


Figure 4.22: Interactions for use case 15.

4.18 Use Case 16 - Log CRUD Operations

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal All CRUD operations on metadata and data are logged at each node.

Summary

> Logging is important for monitoring performance and stability of the system as well as for ensuring appropriate attribution is given to contributors of content. This use case describes the action of logging a user request for content.

Actors

- User
- Member Node

Preconditions

- User has authenticated to desired level

Triggers

- User requests, modifies, or creates an object.

Post Conditions

- The operation is recorded in the log for the node.

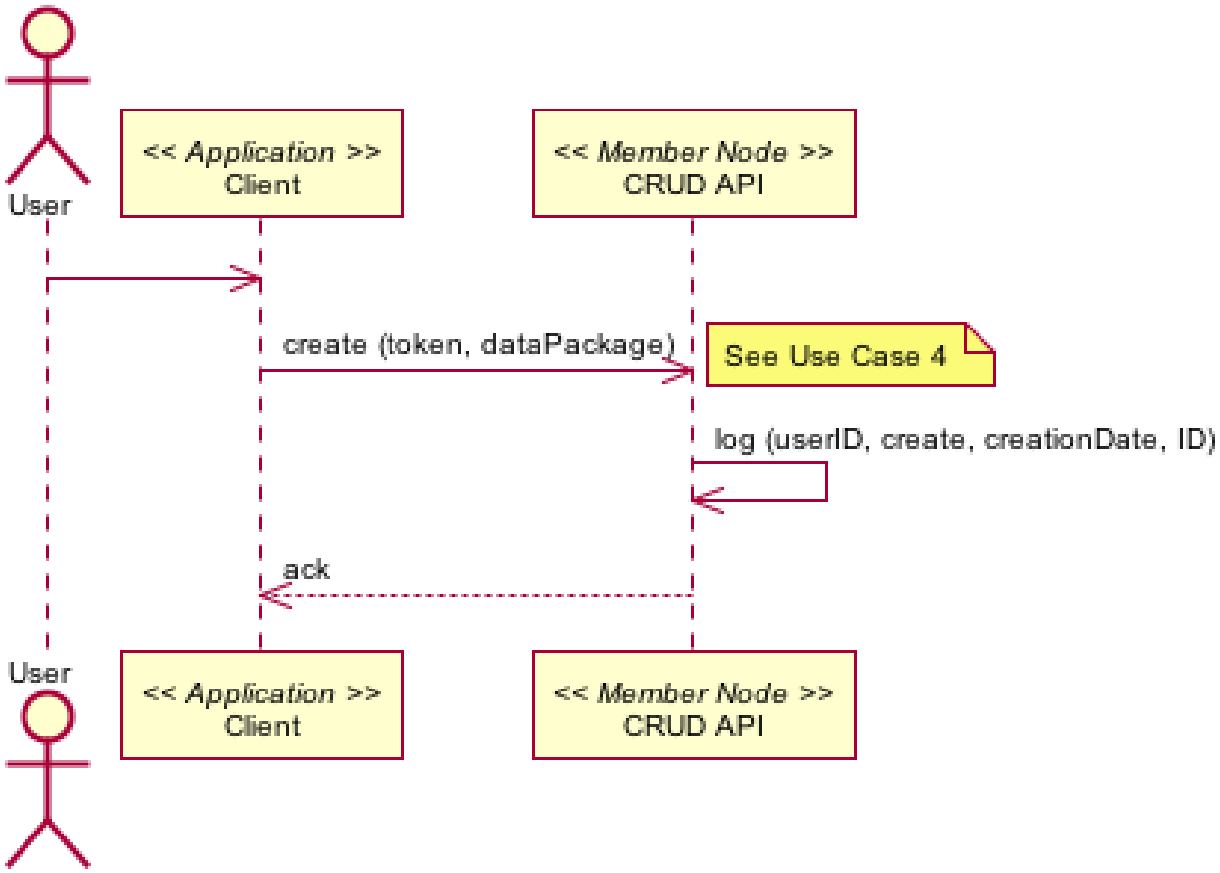


Figure 4.23: Interactions for use case 16.

4.19 Use Case 17 - CRUD Logs Aggregated at CNs

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal All CRUD logs are aggregated at coordinating nodes.

Summary

> Aggregation of CRUD operation logs is used to compute use statistics and other summary types of operations. The original log information should be viewable by administrators for some period of time.

Actors

- Member Node(s)
- Coordinating Node

Preconditions

- Log files are available to retrieve from Member Nodes

Triggers

- Timed event, logs are aggregated ever xxx days.

Post Conditions

- Aggregate logs are stored on Coordinating Node
- Logs are synchronized between CNs
- Log processing is triggered on CNs

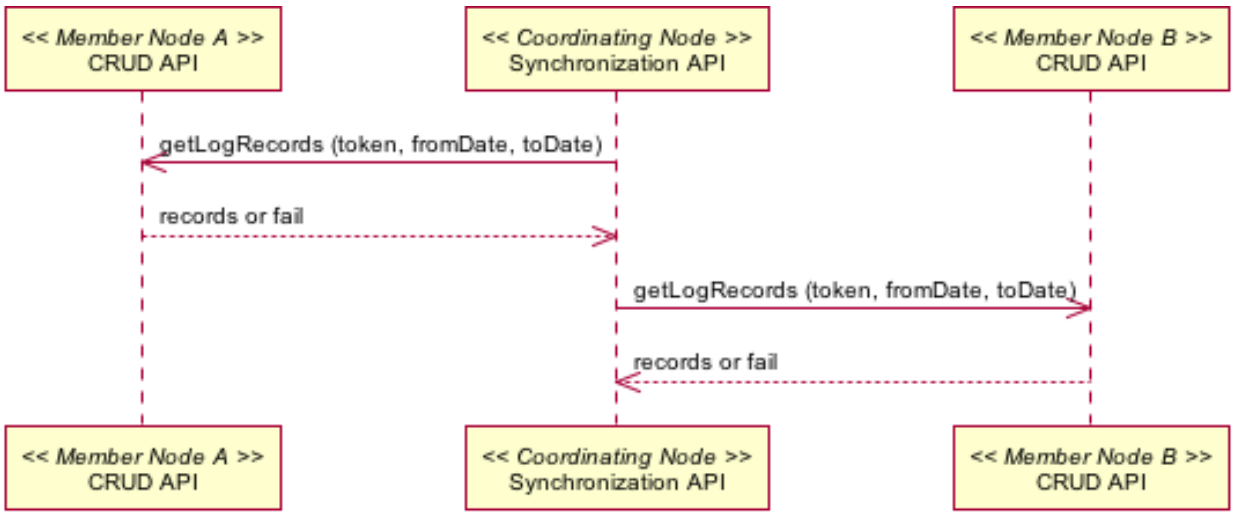


Figure 4.24: Interactions for use case 17.

4.20 Use Case 18 - MN Retrieve Aggregated Logs

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Member nodes can request aggregated CRUD log for {time period/object id/userid} for all of 'their' objects.

Summary

> A Member Node should be able to request use information for any objects for which that Member Node is the origin (or defacto origin in the case where the original MN was decommissioned).

Actors

- Coordinating Node
- Member Node

Preconditions

- Aggregate logs are accumulated by CNs

Triggers

- MN Administrator requests aggregate logs
- Timed event

Post Conditions

- Aggregate log information is available to the Member Node

Notes

This operation should really be a function for the administrative functions of the Investigator Toolkit. It is reasonable to expect that (especially for the first iteration of the infrastructure) MN administrators will want to peruse and evaluate the log data. In later stages, this information could be associated with content, so use statistics appear with search results.

4.21 Use Case 19 - Retrieve Object Download Summary

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal General public can request aggregated download usage information for any object ID (with anonymous read access)

Summary

This operation provides an indication of the popularity for an object in DataONE, somewhat equivalent to a citation index for a publication.

Actors

- Anonymous user
- Coordinating Node

Preconditions

Use statistics have been recorded for a given object

Triggers

- Any user requests use logs for an object

Post Conditions

- Requesting user has a copy of use statistics for an object (if authorized).

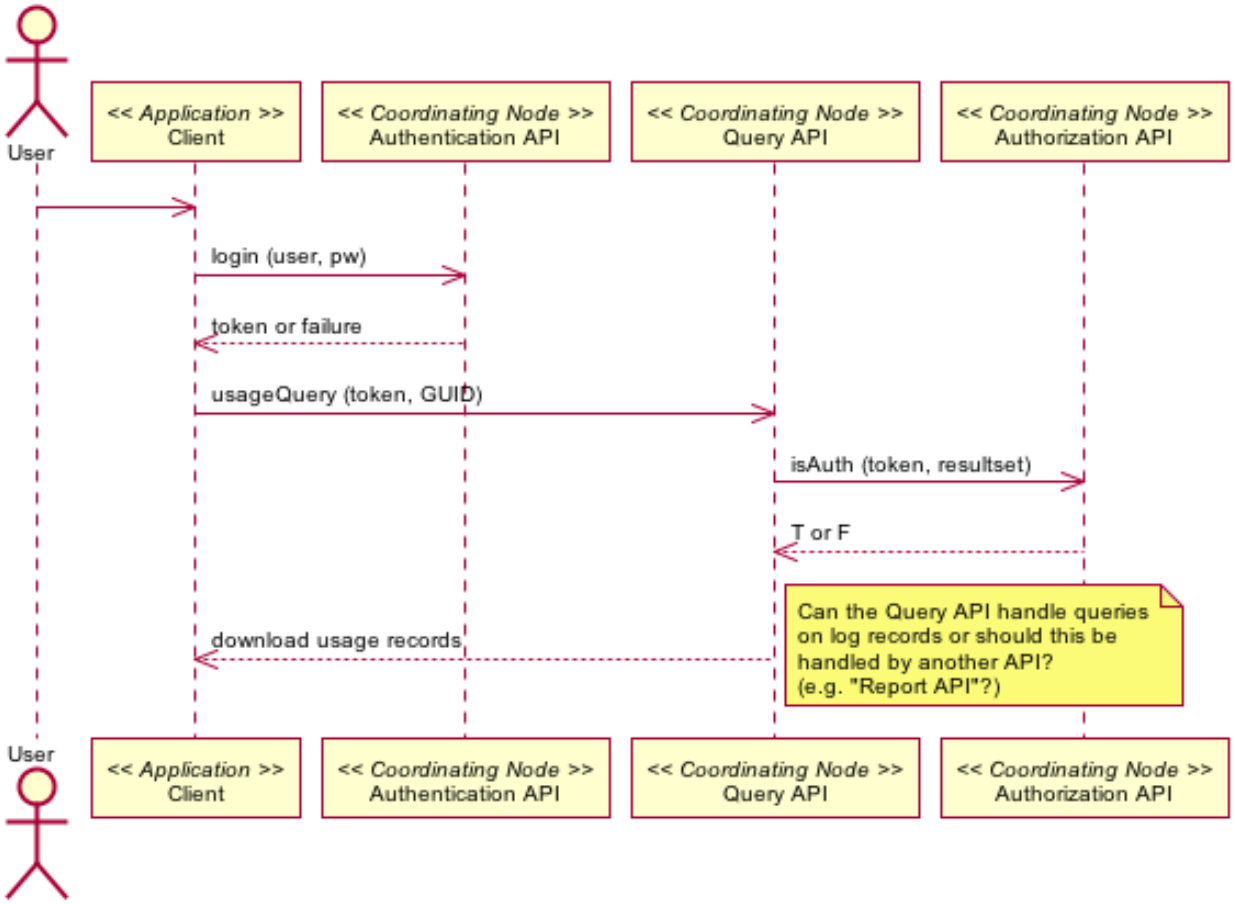


Figure 4.25: Interactions for use case 19.

4.22 Use Case 20 - Owner Retrieve Aggregate Logs

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Data owners can request aggregated CRUD log for {time period/object id} for all of 'their' objects.

Summary

> This operation provides a mechanism for data contributors to retrieve use information for content that they own. This is similar to `Use Case 19` except that the requesting use is authenticated, and a query on objects owned by the user is used to generate a list of GUIDs that indicate log information to be retrieved.

Actors

- Data owner
- Coordinating Node

Preconditions

- User has authenticated
- Log statistics are available

Triggers

- Authenticated user requests use information

Post Conditions

- User has a copy of use inforomation

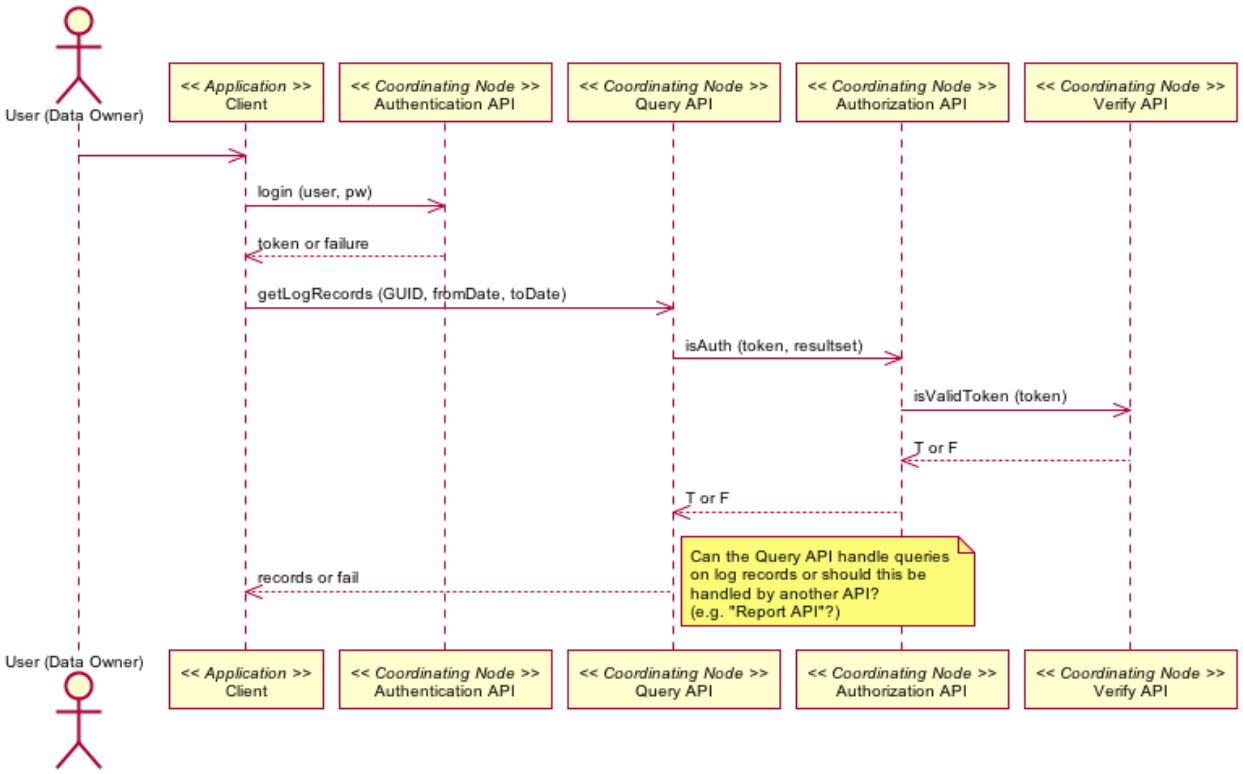


Figure 4.26: Interactions for use case 20.

4.23 Use Case 21 - Owner Subscribe to CRUD Operations

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Data owners can subscribe to notification service for CRUD operations for objects they own.

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

xxx

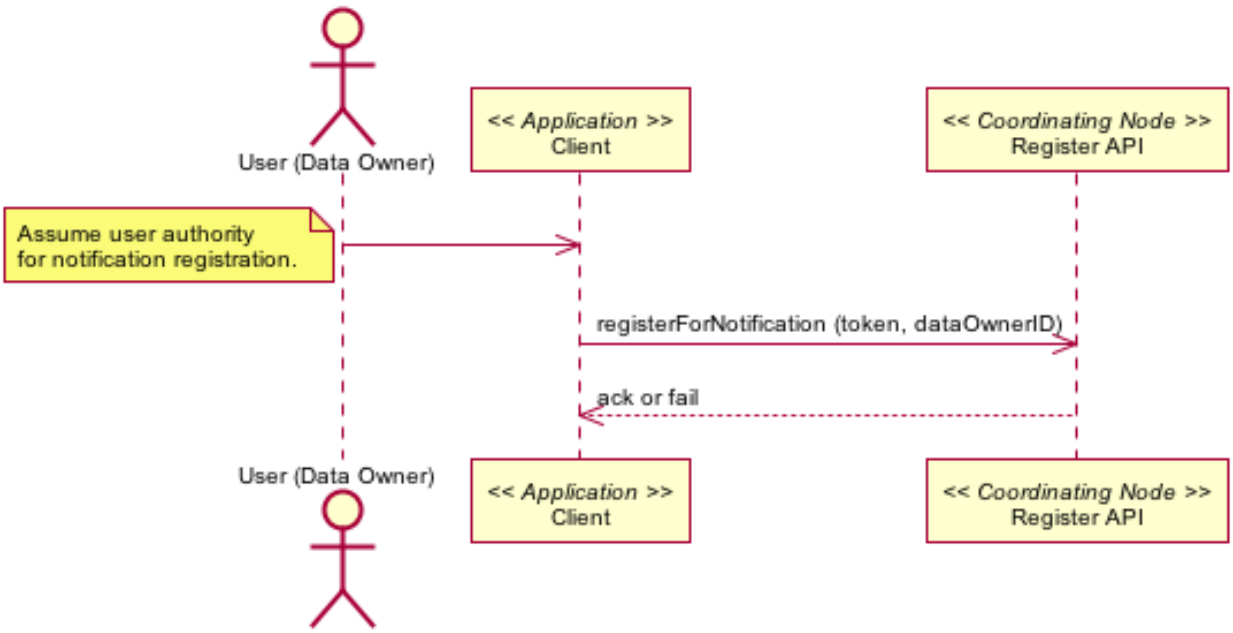


Figure 4.27: Interactions for use case 21.

4.24 Use Case 22 - Link/Citation Report for Owner

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal User can get report of links/cites my data (also can view this as a referrer log).

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

xxx

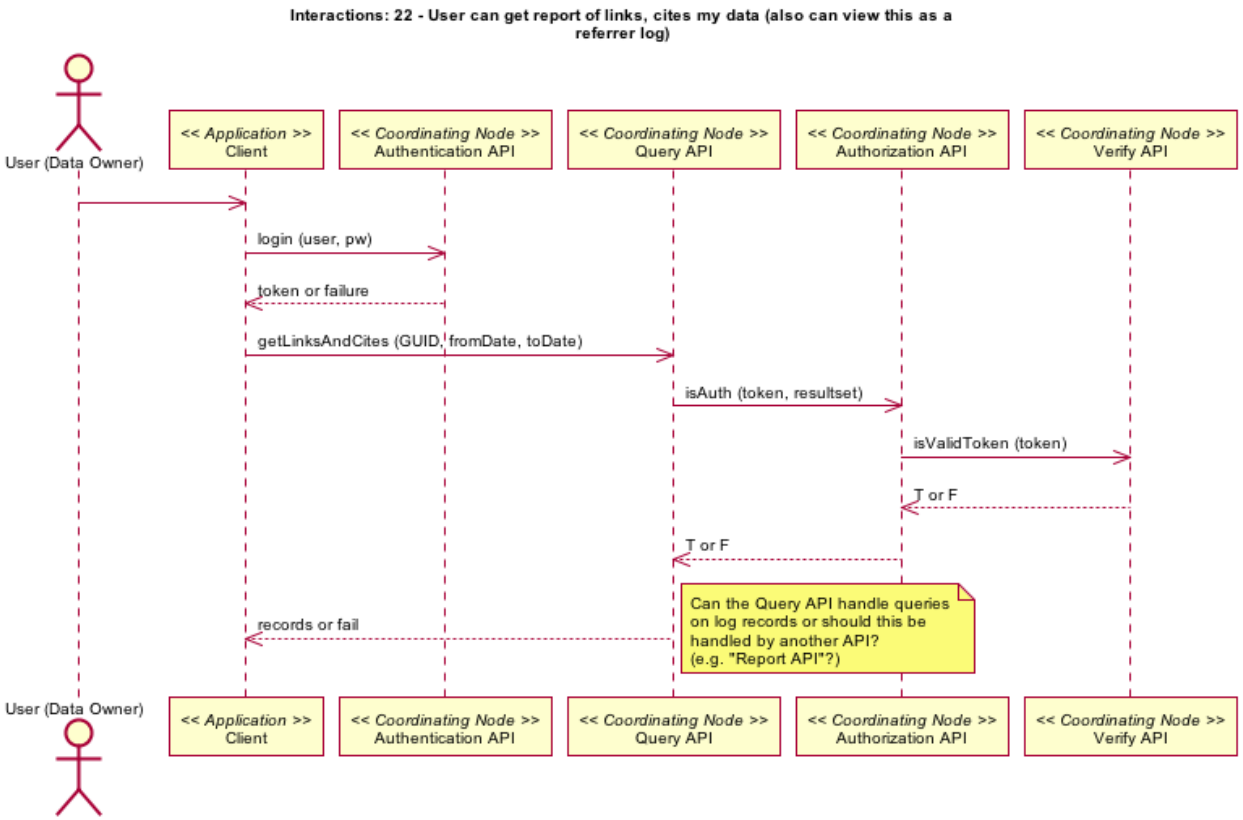


Figure 4.28: Interactions for use case 22.

4.25 Use Case 23 - Owner Expunge Content

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal User can find out where all copies of my data are in the system and can expunge them.

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

- Dealing with inappropriate usage of (ie, copyrighted objects that are stored)

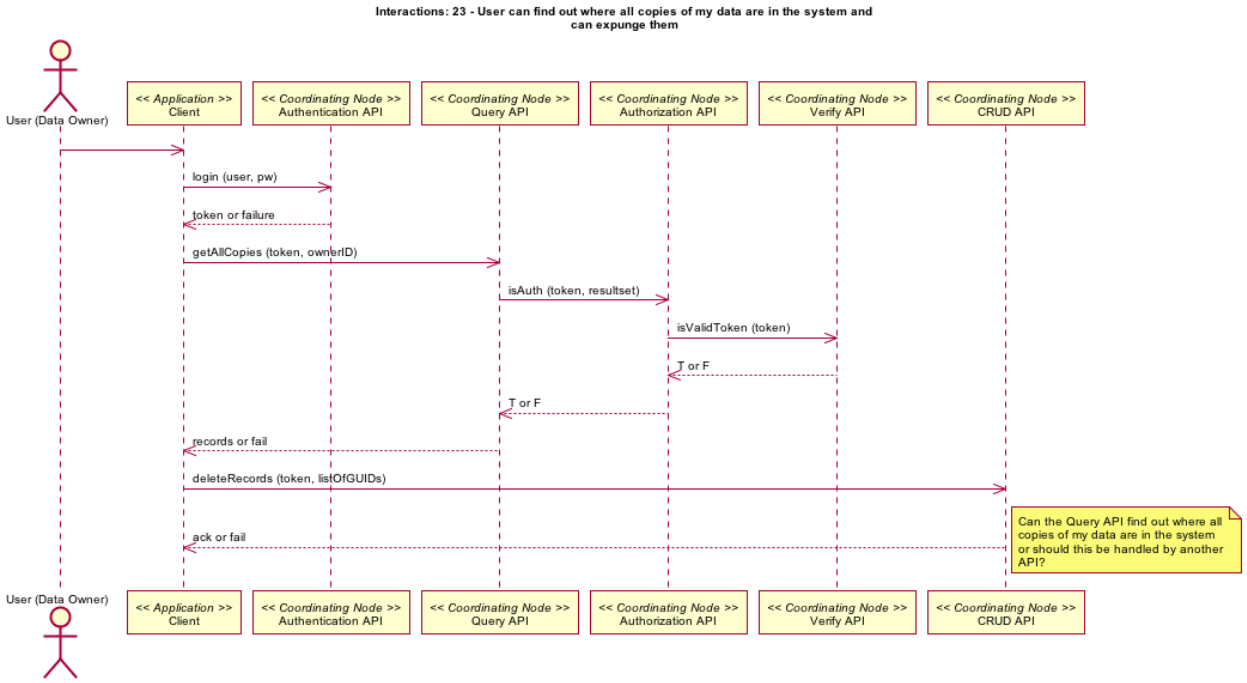


Figure 4.29: Interactions for use case 23.

4.26 Use Case 24 - MNs and CNs Support Transactions

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Transactions - CNs and MNs should support transaction sets where operations all complete successfully or get rolled back (e.g., upload both data and metadata records).

Summary

Transactions help ensure stability of the system, so that the chance of partial change to content is minimized.

Note that this use case is really stated as a requirement of the DataONE system.

Actors

- Member Nodes
- Coordinating Nodes

Preconditions

- DataONE operational
- Some data / metadata transfer operation is due.

Triggers

- Metadata transfer is in process and there is a network failure

Post Conditions

- The transfer changes should be rolled back to the state before the operation.

Notes

- Does this include just one service, or sets of services that can be included in a transaction?
- Can transactions span multiple MNs? MNs and CNs?

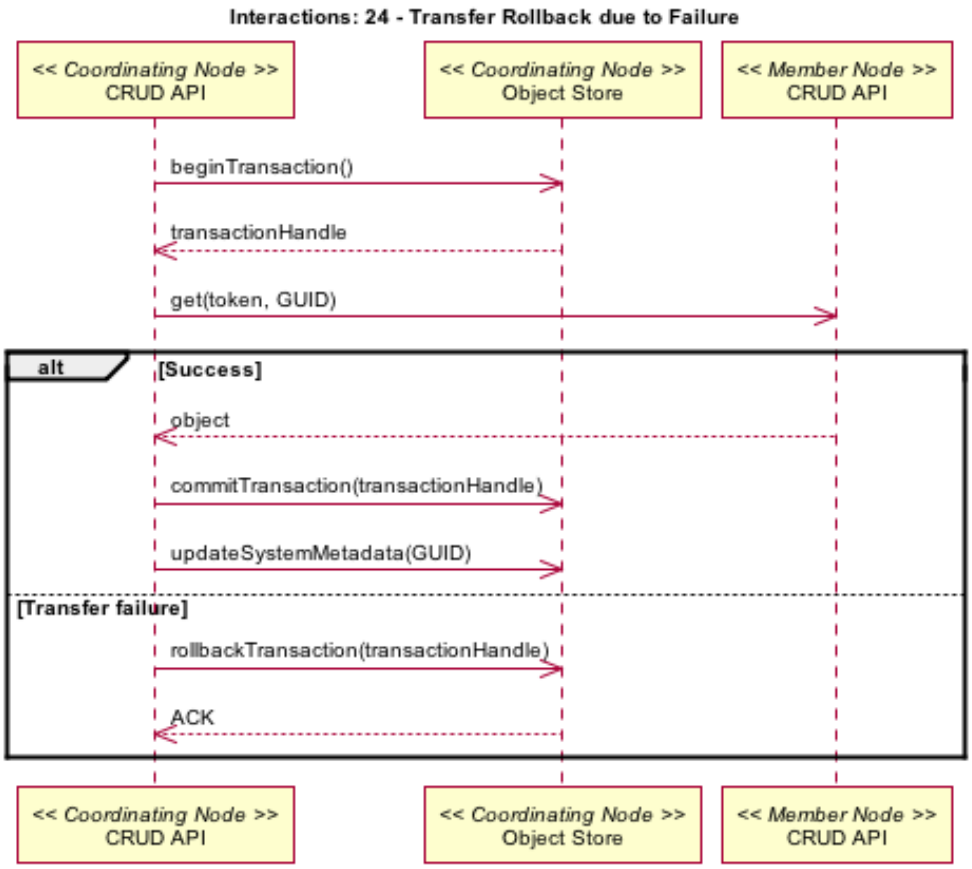


Figure 4.30: Interactions for use case 24. Simple transaction control around a get operation by a CN against and MN.

4.27 Use Case 25 - Detect Damaged Content

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal System should scans for damaged/defaced data and metadata using some validation process.

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

xxx

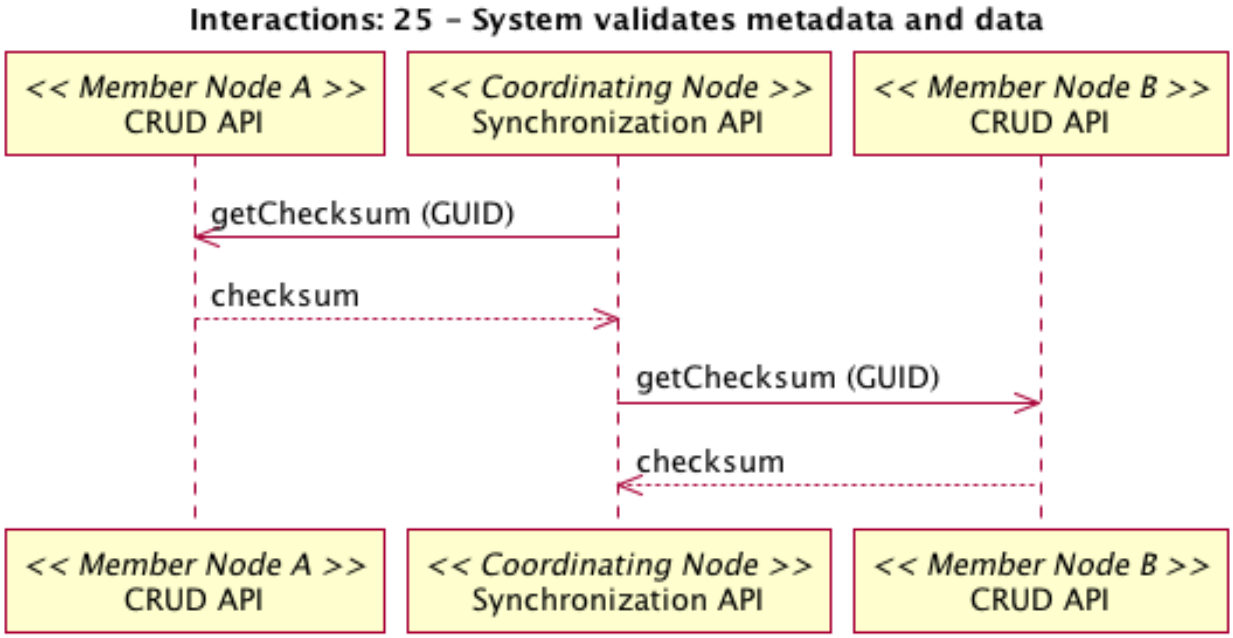


Figure 4.31: Interactions for use case 25.

4.28 Use Case 26 - Data Quality Checks

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal System performs data quality checks on data.

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

- Is this more for verifying the quality of metadata? Otherwise how does it differ from *Use Case 25*?

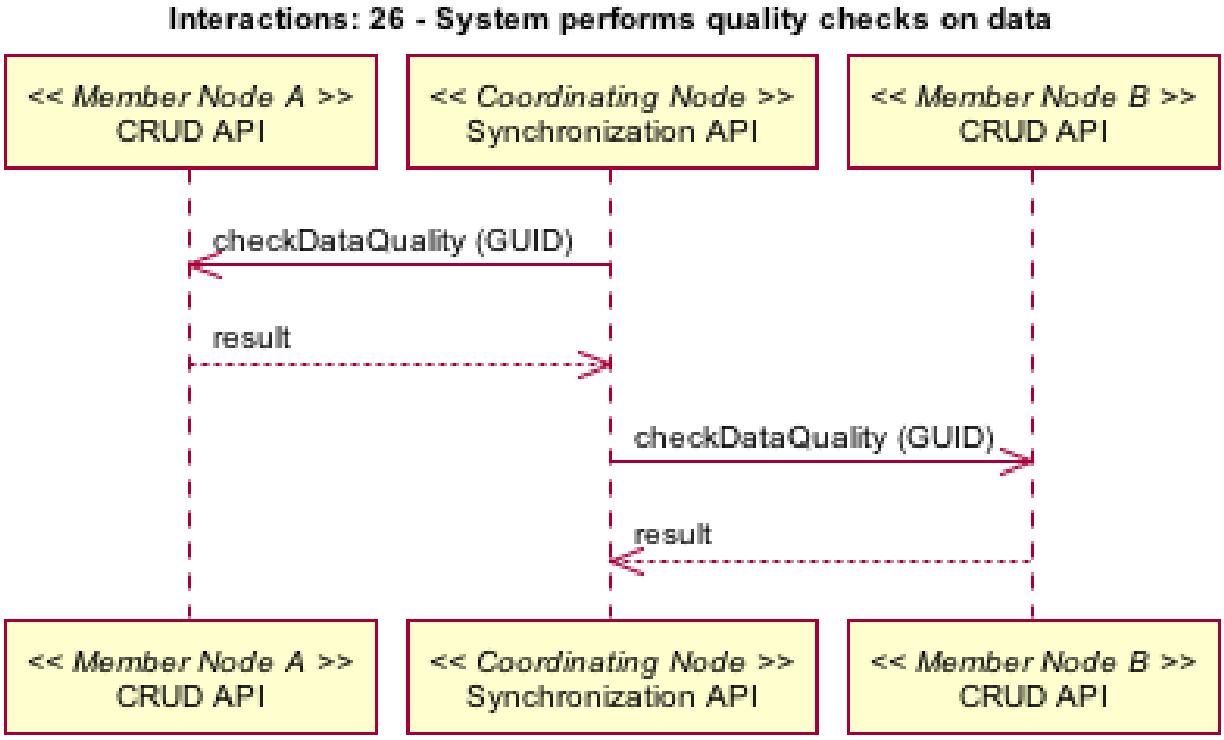


Figure 4.32: Interactions for use case 26.

4.29 Use Case 27 - Metadata Version Migration

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal CN should support forward migration of metadata documents from one version to another within a standard and to other standards.

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

- Should this apply to data as well?

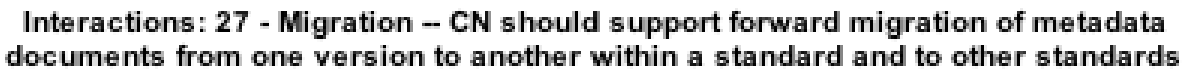


Figure 4.33: Interactions for use case 27.

4.30 Use Case 28 - Derived Product Original Change Notification

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Relationships/Versioning - Derived products should be linked to source objects so that notifications can be made to users of derived products when source products change.

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

- Who can assert these relationships? The relevant object owners? Anybody?
- How are relationships removed?

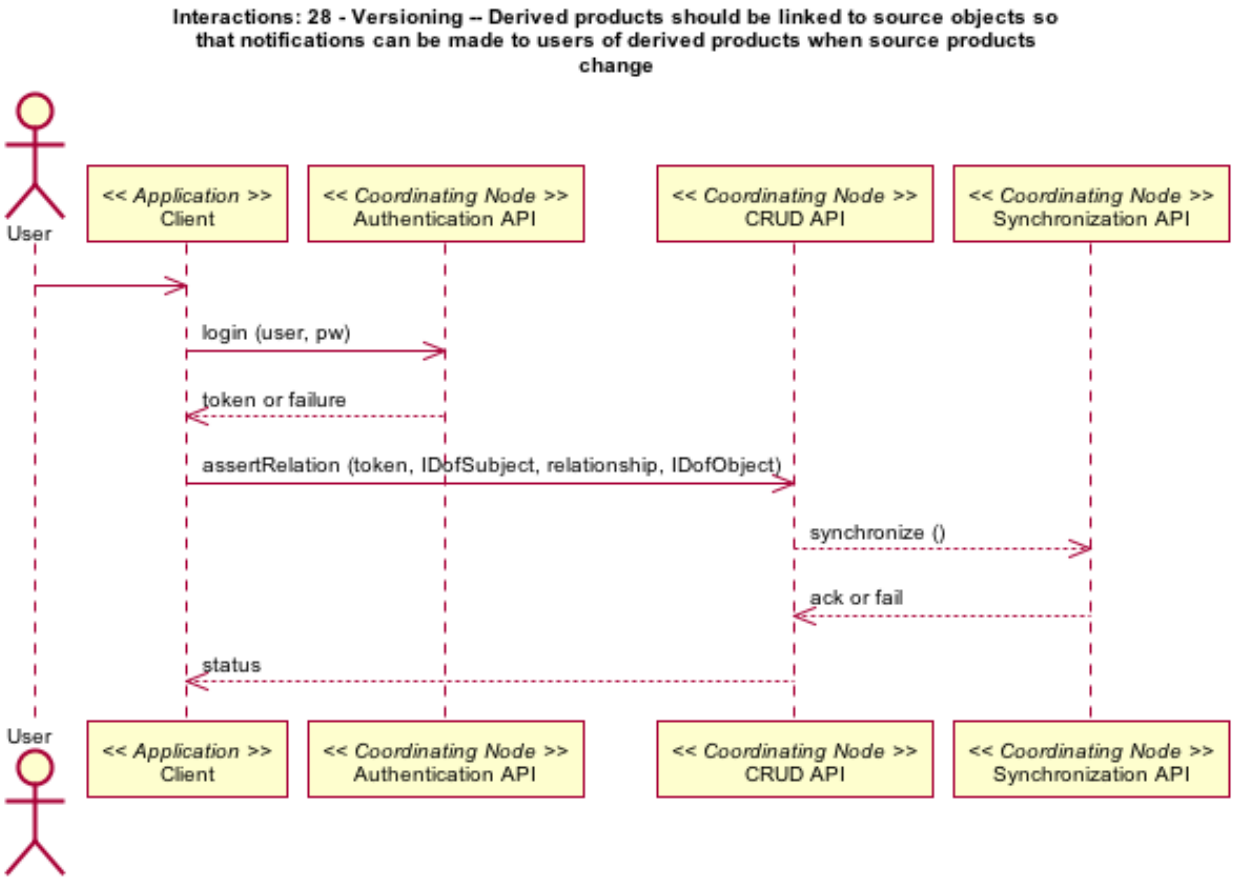


Figure 4.34: Interactions for use case 28.

4.31 Use Case 29 - CN Load Balancing

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Load Balancing - Requests to coordinating nodes are load balanced.

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

xxx

Figure 4.35: Interactions for use case 29.

4.32 Use Case 30 - MN Outage Notification

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal MN can notify CN about pending outages, severity, and duration, and CNs may want to act on that knowledge to maintain seamless operation.

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

xxx

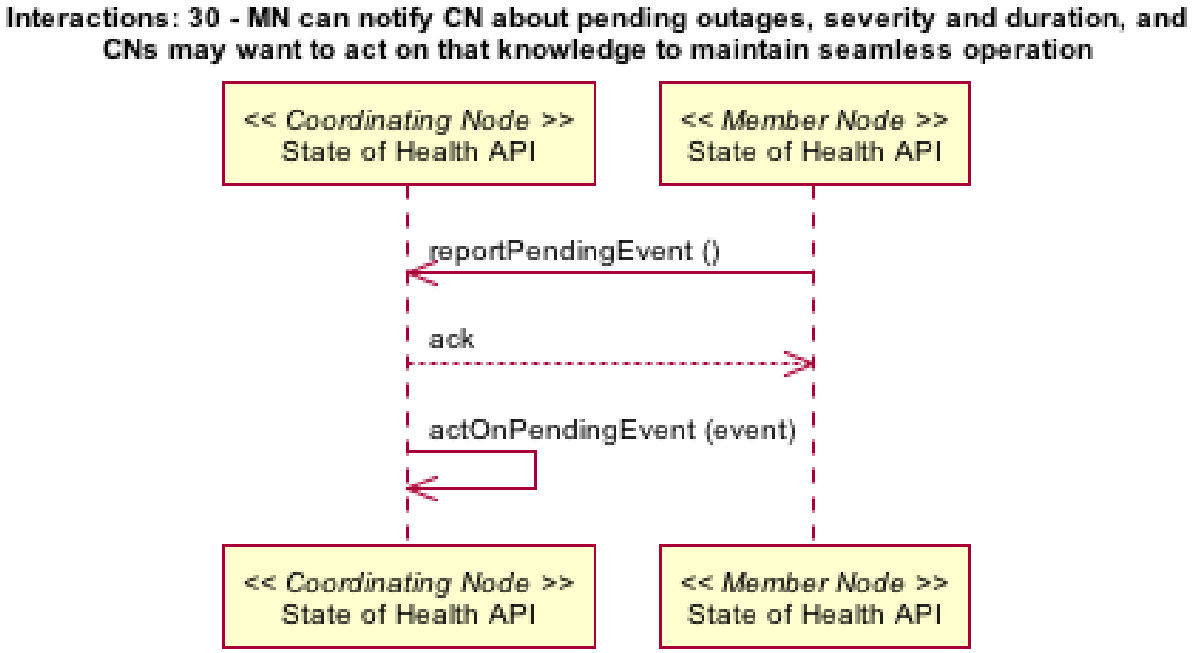


Figure 4.36: Interactions for use case 30.

4.33 Use Case 31 - Manage Access Policies

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Manage Access Policies - Client can specify access restrictions for their data and metadata objects. Also supports release time embargoes.

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

- Users can be members of groups that can participate in access directives.
- I have removed the phrase "and replication" from the use case statement because `Use Case 08` deals with setting replication policies. (PEA)
- Step #1, should have a signature of setAccess(token, GUID, accessPolicy). Even though the diagram says "Assume user authority for specifying restrictions", practically speaking we will need to verify that authority and the user's identify with a token. Also "accessLevel" sounds very limited, and access policy implies a possibly more sophisticated access policy delineation, including embargoes.

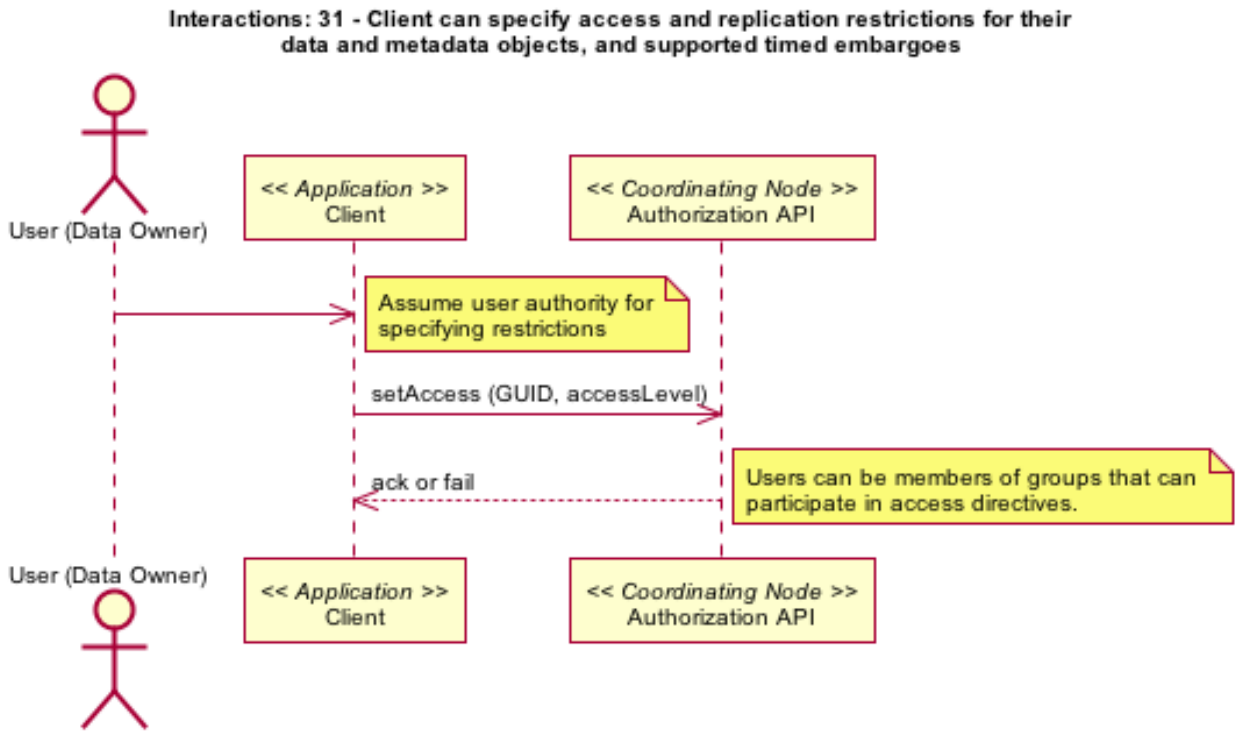


Figure 4.37: Interactions for use case 31.

4.34 Use Case 32 - Transfer Object Ownership

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal User or organization takes over 'ownership' of a set of objects (write access for orphaned records)

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

- Is there an authority for any given data/metadata object?

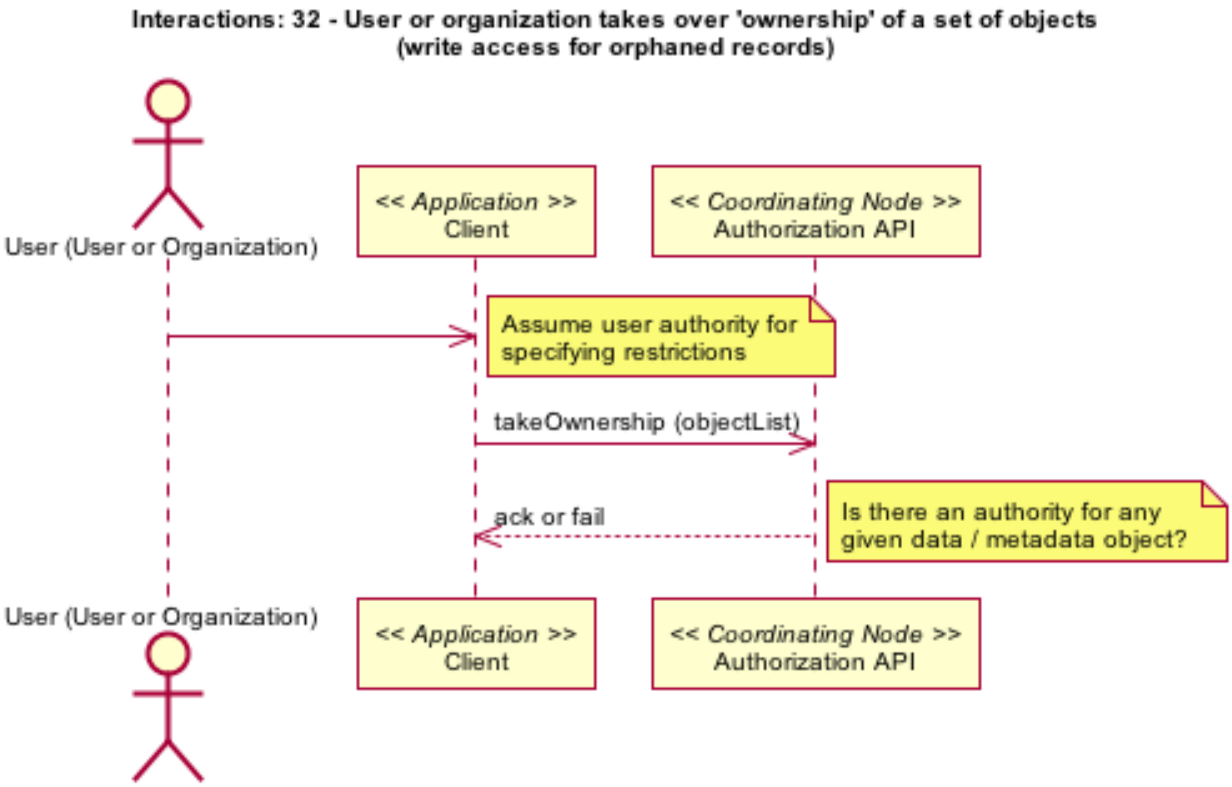


Figure 4.38: Interactions for use case 32.

4.35 Use Case 33 - Search for Data

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Clients should be able to search for data using CN metadata catalogs.

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

xxx

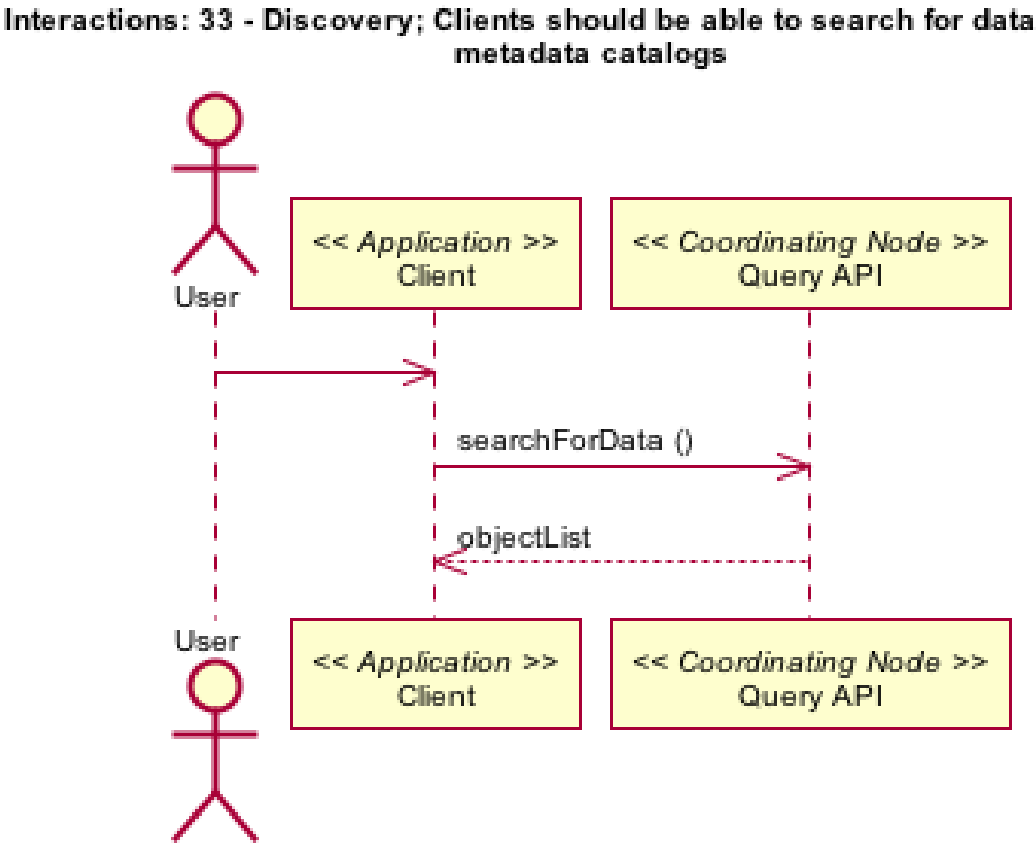


Figure 4.39: Interactions for use case 33.

4.36 Use Case 34 - CNs Support Other Discovery Mechanisms (e.g. Google)

Author VDC TWG

Date

- 20090925 Adding to use case description and documentation
- 20090114 (TWG meeting). Subsequent various modifications.

Version Draft

Goal Coordinating Nodes publish metadata in formats for other discovery services like Google/Libraries/GCMD/etc.

Summary

xxx

Actors xxx

Preconditions xxx

Triggers xxx

Post Conditions xxx

Notes

xxx

Figure 4.40: Interactions for use case 34.

CHAPTER

FIVE

DATAONE APPLICATION PROGRAMMING INTERFACES

These APIs represents the interfaces exposed by the Coordinating and Member Nodes and the Investigator Toolkit. They do not describe interfaces used only internally by those components.

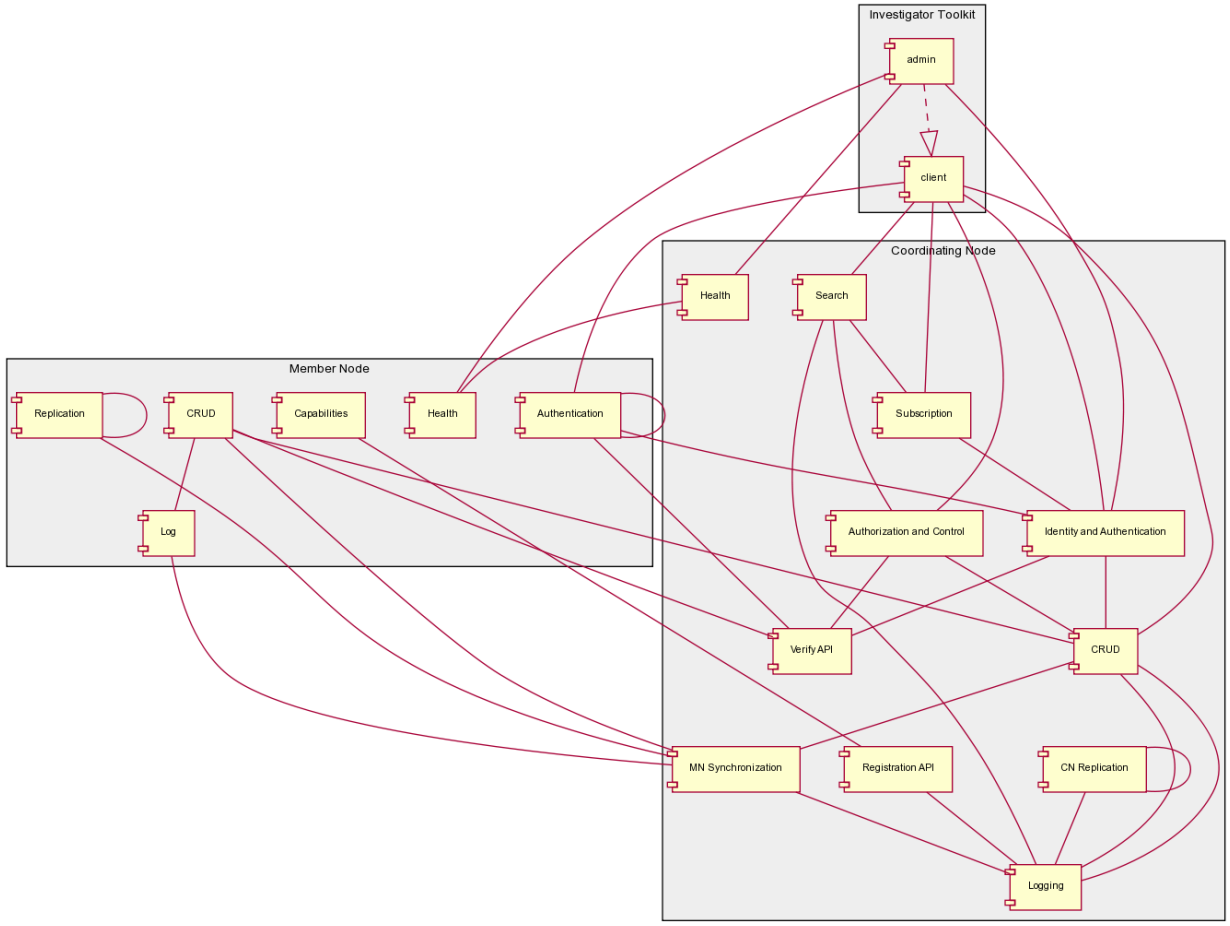


Figure 5.1: Component diagram showing the various APIs and their interactions (a line indicates communication between APIs).

5.1 Member Node APIs

These APIs are implemented by Member Nodes.

Todo

The APIs need to be edited for consistency of method signatures and naming. Update the sequence diagrams to reflect the edits to the API.

5.1.1 Authorization API

Provides a mechanism for a User to authenticate themselves with the DataONE system and verify that an operation is permitted for an authenticated User. See the document *Authentication in DataONE* for more details on some authentication options.

```
login(user, password, [auth_system]) -> token
logout(token)
isAuthorized (token, method, GUID) -> boolean
verify(token) -> boolean
```

exception `InvalidCredentials`

exception `AuthenticationTimeout`

exception `InvalidToken`

`login` (*user, password, [auth_system]*)

Authenticates a user for interaction with the DataONE system. The returned token is trusted throughout the DataONE system as a reliable statement as to the identity of the authenticated user.

Parameters

- *user* – User name for authentication
- *password* – Secret used for verifying identity
- *auth_system* – Place holder for support of multiple authentication services for a single user.

Return type A composite of the user id and authentication token.

Raises InvalidCredentials The supplied credentials were invalid (password + username don't jive)

Raises AuthenticationTimeout A response from the authentication service was not received before the operation timed out.

Note: `login()` may delegate the actual login to an authentication service, and a given user may utilize several authentication services. If this is the case then an additional parameter identifying the authentication service to use will be required.

`logout` (*token*)

Revokes trust in the supplied token, making it invalid for further use within the DataONE system.

Parameter *token* – The authentication token returned by `login()`

Return type No return

`isAuthorized` (*token, method, GUID*)

Indicates if the operation identified by `method` on the object identified by GUID is permitted by user identified by `token`.

Parameters

- *token* – The authentication token returned by `login()`
- *method* – Enumerated set of values indicating at least "read" and "write"
- *GUID* – An identifier or list of identifiers that are to be tested for access.

Return type True if the operation is permitted, otherwise False. If GUID is a list, then the return is the equivalent of boolean AND of the test results for all elements in the list.

Raises InvalidToken The supplied token is invalid.

Raises MN_crud.NotFound The object identified by GUID was not found.

Note: Should perhaps add convenience methods for "canRead()" and "canWrite()" to verify that a user is able to read / write an object.

verify (*token*)

Test if the provided token is valid.

Note that there is a potential security issue here as open access to this method enables a malicious user to potentially probe a large number of generated tokens to determine properties of the system, and perhaps even generate a token that matches a system token. Hence another level of restriction (perhaps network level blocking / limiting) is necessary.

Parameter *token* – The authentication token returned by `login()`

Return type True if the supplied token is valid, otherwise False

5.1.2 CRUD API

```
describe(token, GUID) -> objectMetadata
get(token, GUID) -> object
batchGet(token, GUIDs) -> ?
checkDataQuality (GUID)
create (token, dataPackage) -> GUID
create (token, workflowObject)
getChecksum (GUID) -> checksum
getLogRecords(token, fromDate, toDate)
log (create, creationDate, ID)
register (memberToken, ID, metadata)
```

exception NotFound

exception NotAuthorized

exception UnsupportedType

exception InsufficientResources

describe (*token, GUID*)

Describes the object located at the endpoint of a GUID. This is necessary to enable a client (e.g. a CN doing synchronization of new content) to determine what type of information is being referenced before it is registered in the system metadata. Helps to avoid accidental download of data objects rather than metadata.

It may also be possible / feasible / appropriate to utilize content negotiation in the HTTP GET method as part of the `get()` operation to retrieve this information (or perhaps HTTP HEAD).

Parameters

- *token* – Identity with privileges to read the GUID.
- *GUID* – Identifier for the object in question.

Return type Metadata describing what the GUID refers to.

get (*token, GUID*)

Retrieve an object identified by GUID from the Member Node.

Parameters

- *token* – The authentication token returned by `login()`
- *GUID* – The identifier for the object to be retrieved.

Return type Bytes of the specified object. For data objects, this will be the actual bytes of data. For metadata objects, this will be the representation of the object as provided by the Member Node.

Raises NotAuthorized The supplied identity does not have READ permission on the object.

Raises NotFound The object specified by GUID does not exist.

Note: There may be multiple representations of content (e.g. metadata as XML, RDF, or some other format). May need to add another parameter to specify the format, or perhaps rely on HTTP content negotiation for this.

batchGet (*token, GUIDs*)

Identical to get() except that a list of identifiers is requested.

The issue here is how to serialize the results. There are some conceptual mechanisms for handling structured markup such as XML, but no consistent approaches for working with data blobs. Perhaps this method should apply only to system metadata, for which the serialization can be defined independently of all the other participating nodes?

Parameters

- *token* – The authentication token returned by `login()`
- *GUIDs* – The list of identifiers to retrieve.

Return type Bytes of the specified objects. **How are multiple objects serialized?**

Raises NotAuthorized The supplied identity does not have READ permission on the object.

Raises NotFound The object specified by GUID does not exist.

checkDataQuality (*GUID*)

TODO: need details on what this operation is supposed to be doing.

Parameter *GUID* – object to check

create (*token, dataPackage*)

Adds a new data package to the Member Node. This mechanism is used by the Coordinating Nodes to indicate a Member Node should add a copy of the data package to itself.

The actual transfer of content should occur asynchronously, so this method should return immediately after the operation has been scheduled.

Parameters

- *token* – The authentication token returned by `login()`
- *dataPackage* – The information that is to be added to the Member Node. This will contain GUIDs that at least identify the data and metadata that are to be added. It is up to the MN to determine when the actual retrieval should occur.

Return type An identifier that can be used to check on the status of the operation. For example, this could be a URL that points to a status document on the MN.

Raises NotAuthorized

Raises UnsupportedType The MN can not deal with the content specified in the data package.

Raises InsufficientResources The MN is unable to execute the transfer because it does not have sufficient storage space for example.

create(*token, workflowObject*)

Note: Why is this method different from a regular create() operation?

getChecksum(*GUID*)
Returns a checksum for the specified object using the system wide hashing algorithm. The result is used to determine if two instances referenced by a GUID are identical, hence it is important that the same algorithm is used throughout

Parameter *GUID* – The identifier of the object the operation is being performed on.

Return type The checksum value originally computed for the specified object.

Note: for an MN, this should probably be a computed value. For CNs, it should be the value originally calculated for the object (and retrieved from system metadata)

getLogRecords(*token, fromDate, toDate*)
Retrieve log information from the MN for the specified date range (inclusive).

log(*create, creationDate, ID*)

register(*memberToken, ID, metadata*)

Parameter *token* – The authentication token returned by `login()`

5.1.3 Replication API

cancelReplication(*sess, transID*)

listAllObjects()

listRecords()

replicationSucceeds(*transID*)

startReplica(*ID, sessionID, transID, srcMN*)

sysOp(*token*)

listChangedRecords(*token*)
List the records that have changed since the last synchronization event.

Parameter *token* – A token with authority to retrieve a list of changed records.

5.1.4 Capabilities API

serviceRequest(*token*)

getCapabilities()
Returns a document describing the capabilities of the Member Node.

Return type Capabilities metadata document

5.1.5 State of Health API

sohQuery(*token, service*)

ping()
Low level "are you alive" operation. Response is simple ACK, but may be reasonable to overload with a couple of flags that could indicate availability of new data or change in capabilities.

Rtype pingresponse A simple chunk of data, perhaps XML, or plain text.

getStatus(*token*)
Returns the current status of the MN, perhaps with some accumulation and averaging over the last xxx minutes.

Parameter *token* – Identity with authority to retrieve the status information.

Return type A status document in an easily parsed syntax such as JSON or XML

5.2 Coordinating Node APIs

Todo

The APIs need to be edited for consistency of method signatures and naming. Update the sequence diagrams to reflect the edits to the API.

5.2.1 Authentication API

```
login(user, password, [auth_system]) -> token
requestToken ()
setAccess (token, GUID, accessLevel) -> boolean
setOwner(token, userid, objectList) -> objectList
newAccount(username, password, [auth_system]) -> identity_GUID
verify(token) -> boolean
```

exception InvalidCredentials

exception AuthenticationTimeout

exception InvalidToken

exception NotFound

exception IdentityAlreadyExists

login (*user, password, [auth_system]*)

Authenticates a user for interaction with the DataONE system. The returned token is trusted throughout the DataONE system as a reliable statement as to the identity of the authenticated user.

Parameters

- *user* – User name for authentication
- *password* – Secret used for verifying identity
- *auth_system* – Place holder for support of multiple authentication services for a single user.

Return type A composite of the user id and authentication token.

Raises InvalidCredentials The supplied credentials were invalid (password + username don't jive)

Raises AuthenticationTimeout A response from the authentication service was not received before the operation timed out.

Note: `login()` may delegate the actual login to an authentication service, and a given user may utilize several authentication services. If this is the case then an additional parameter identifying the authentication service to use will be required.

requestToken ()

This operation is used by CN applications to obtain a token for system level operations.

Note: This could be a normal `login()` operation using a system user and secret that is stored in a secure location.

setAccess (*token, GUID, accessLevel*)

Set the access permissions for an object identified by GUID.

Parameters

- *token* – The authentication token returned by `login()`. Used to verify that the calling user has authority to modify access control for the object.

- *GUID* – The object for which access control is being edited.
- *accessLevel* – The desired privileges to be assigned to the object.

Return type True if the operation succeeds, otherwise False.

Raises InvalidToken The supplied token is invalid.

Raises NotFound The object identified by GUID was not found.

Raises NotAauthorized The supplied identity does not have permission to alter access control rules for the object.

Note: Should permissions cascade? e.g. Should setting permission on a Data Package cascade to the contained objects as well (the experiment metadata and data)?

setOwner (*token, userid, objectList*)

Changes ownership of all objects in objectList to the identity specified by userid.

Parameters

- *token* – The authentication token returned by `login()`. Used to verify that the calling user has authority to take ownership of the object.
- *userid* – Identity of the user that will be taking ownership of the specified objects.
- *objectList* – a list of object identifiers (GUIDs) that are to be modified.

Return type List of objects for which the operation failed.

Raises InvalidToken The supplied token is invalid.

Raises NotAuthorized The supplied identity does not have authority to change ownership.

newAccount (*username, password, [auth_system]*)

Create a new identity in the DataONE system. Note that there should probably be a lot more metadata captred about the new user, and there should be a mechanism for specifying the default access control rules for the new account.

Parameters

- *username* – Requested username. Must be unique within the DataONE system.
- *password* – The secret associated with the account.
- *auth_system* – Place holder for support of multiple authentication services for a single user.

Return type GUID for the new identity, otherwise NULL.

Raises IdentityAlreadyExists The provided identity already exists in the system

verify (*token*)

Test if the provided token is valid.

Note that there is a potential security issue here as open access to this method enables a malicious user to potentially probe a large number of generated tokens to determine properties of the system, and perhaps even generate a token that matches a system token. Hence another level of restriction (perhaps network level blocking / limiting) is necessary.

Parameter *token* – The authentication token returned by `login()`

Return type True if the supplied token is valid, otherwise False

5.2.2 Query API

getAllCopies(*token, ownerID*)

getLinksAndCites(*GUID, fromDate, toDate*)

getLogRecords(*GUID, fromDate, toDate*)

getLogRecords(*nodeID, fromDate, toDate*)

query(*token, query*)

searchForData()

usageQuery(*token, GUID*)

5.2.3 CRUD API

assertRelation(*token, IDofSubject, relationship, IDofObject*)

batchWrite(*objectsList*)

deleteRecords(*token, listOfGUIDs*)

get(*token, GUID*)

register(*memberToken, ID, metadata*)

resolve(*GUID, token*)

storeRecords()

update()

5.2.4 Replication API

getDefaultReplicationPolicy(*sess*)

getMNReplicateList()

getReplicationPolicy(*sess, ID*)

isReplicationNeeded()

replicateMetadata()

setDefaultReplicationPolicy(*sess, policy*)

setReplicationPolicy(*sess, policy, ID*)

5.2.5 Synchronization API

pollForNewIDs()

synchronize()

verifyAllRecordsReceived()

scheduleSync(*nodeID*)

Schedules a synchronization operation on the specified Member Node.

Parameter *nodeID* – The member node that is being scheduled for synchronization.

Raises NotFound The specified Member Node ID is invalid.

5.2.6 Register API

addNodeCapabilities(*nodeID, capabilities*)
Adds the supplied capabilities to the system metadata describing the Member Node.

Parameters

- *nodeID* – A GUID that identifies the Member Node.
- *capabilities* – A capabilities document that describes the resources and supported services of the Member Node.

register(*token, capabilitiesURL*)
Used to register a new Member Node.

Parameters

- *token* – An authentication token with priviledges to rgister a new Member Node.
- *capabilitiesURL* – The URL from which capabilities will be retrieved.

Raises NotAuthorized The supplied identity does not have authority to register a node.

Raises NotFound The supplied capabilitiesURL does not resolve to a capabilities document

Return type True or False

registerForNotification(*token, dataOwnerID*)

verifyCapabilities(*nodeID, capabilities*)
Evaluates the supplied capabilities document. This may involve additional interaction with the MN as various aspects are evaluated.

Todo

The `verifyCapabilities()` will needto check functionality of the MN so need to define a few service hooks for checking these properties.

5.2.7 Server Authentication API

5.2.8 State of Health API

actOnPendingEvent(*event*)

reportPendingEvent(*event*)

Admin Application()

notify(*adminID, nodeID, service*)

generateReport(*token*)
Returns a status report for all of the registered Member Nodes. The informatio returned is used to give administrators information necessary to maintain the infrastructure.

Parameter *token* – Identity with authority to generate reports.

Return type Status report, ideally in XML + stylsheet or at least XHTML.

5.3 Investigator Toolkit APIs

Todo

Obviously need to flesh this out a bit.

Todo

Pull method signatures for client_app interaction with MNs and CNs

CHAPTER

SIX

TO DO LIST

List of things that need to be completed in these documents to reach a final draft stage.

Todo

The APIs need to be edited for consistency of method signatures and naming. Update the sequence diagrams to reflect the edits to the API.

(The original entry is located in CN_APIs.txt, line 5 and can be found *here*.)

Todo

The `verifyCapabilities()` will needto check functionality of the MN so need to define a few service hooks for checking these properties.

(The original entry is located in CN_APIs.txt, line 299 and can be found *here*.)

Todo

Obviously need to flesh this out a bit.

(The original entry is located in ITK_APIs.txt, line 6 and can be found *here*.)

Todo

Pull method signatures for client_app interaction with MNs and CNs

(The original entry is located in ITK_APIs.txt, line 10 and can be found *here*.)

Todo

The APIs need to be edited for consistency of method signatures and naming. Update the sequence diagrams to reflect the edits to the API.

(The original entry is located in MN_APIs.txt, line 7 and can be found *here*.)

Todo

Estimate some times for the unknowns here.

(The original entry is located in userrequirements.txt, line 213 and can be found *here*.)

Todo

The operational consequences of doing this do not appear to be well defined in the HTTP spec. Need to test against various browsers and libraries to evaluate behavior.

(The original entry is located in WhatIsData.txt, line 675 and can be found *here*.)

Todo

The URLs used here are place holders and may not represent the actual access URLs used in the DataONE system.

(The original entry is located in WhatIsData.txt, line 687 and can be found *here*.)

Todo

Byte stream equivalence of replicated experiment metadata would require that MNs record an exact copy of the metadata document received during replication operations in addition to the content that would be extracted and stored as part of the normal (existing) operations of a MN. Is this a reasonable requirement for MNs? Since MNs are required to store a copy of data, it seems reasonable to assume a copy of the metadata can be stored as well.

(The original entry is located in WhatIsData.txt, line 699 and can be found *here*.)

Todo

Is this reasonable? Likely that data owners may want to modify content after adding to DataONE. Perhaps allow a time period where data and metadata can be modified, after which it becomes more difficult (e.g. requiring administrator intervention or consensus from trusted authority)?

(The original entry is located in WhatIsData.txt, line 713 and can be found *here*.)

Todo

Define the procedures for content deletion - who is responsible, procedures for contacting authors, timeliness of response.

(The original entry is located in WhatIsData.txt, line 723 and can be found *here*.)

Todo

Describe how endianness is specified in various experiment metadata formats.

(The original entry is located in WhatIsData.txt, line 740 and can be found *here*.)

Todo

List and define the minimal set of fields with examples. A starting point would be the union of the required search properties and the information required for accurate citation.

(The original entry is located in WhatIsData.txt, line 772 and can be found *here*.)

CHAPTER

SEVEN

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*

MODULE INDEX

C

M

INDEX

R

S

U

V