LexEVS 5.0 Design and Architecture Guide

⚠️ The information and links on this page are no longer being updated and are provided for reference purposes only.

Unable to render {include} The included page could not be found.

Contents of this Page

> Unable to render {include}   The included page could not be found.

Overview

LexBIG software architecture and implementation is designed to facilitate flexibility and future expansion. The following diagrams are intended to aid the understanding of LexBIG service integration in context of the larger caBIG® universe and specific deployment scenarios:

This diagram depicts the LexBIG vision. Individual Cancer Centers will be able to use the existing set of caCORE EVS services. If desired, local instances of vocabularies can be installed.
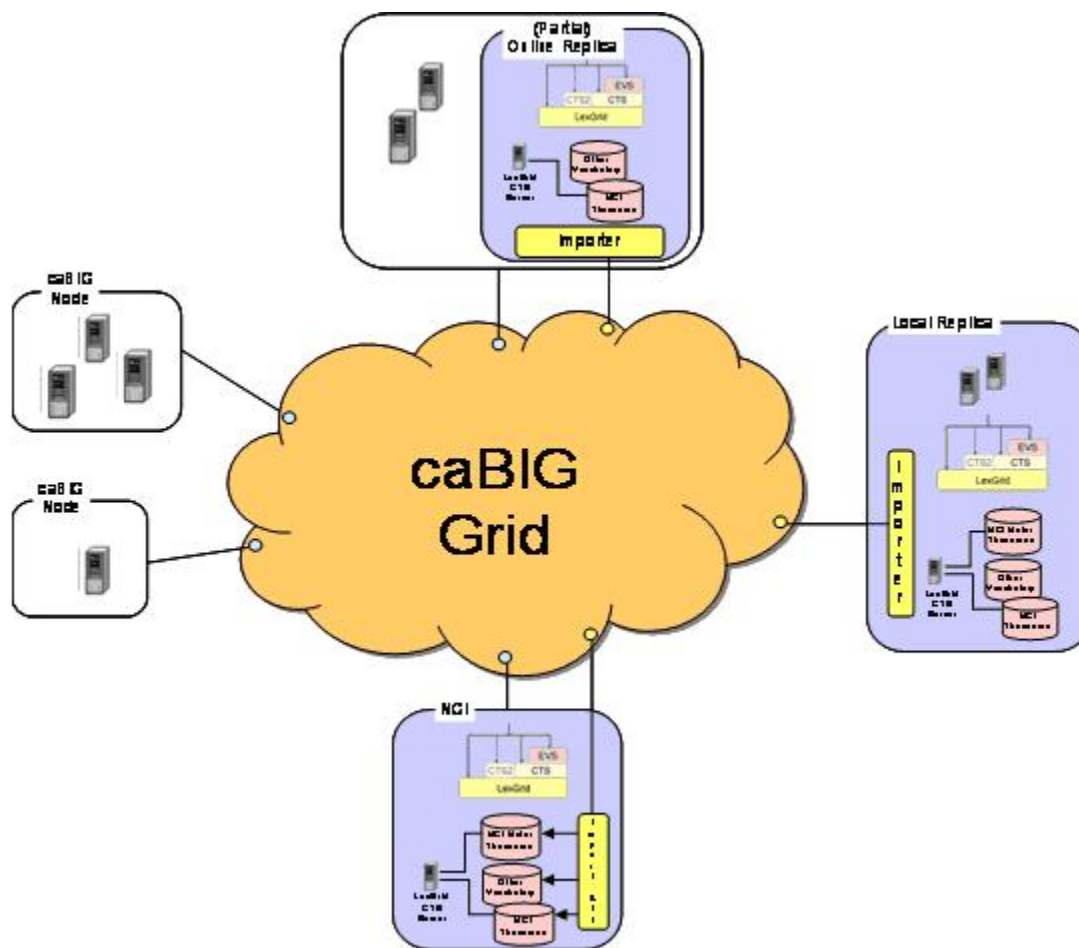
This diagram depicts direct Java-to-Java access to LexBIG functions. This is the primary deployment scenario for phase 1.

> ⓘ **Note**
>
> It is not required that the database be located on the same system as the program runtime.

LexBIG Runtime

Direct Programmatic Access
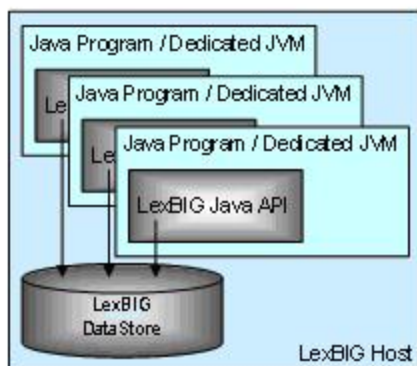
This diagram depicts access through caCORE Enterprise Vocabulary Services (EVS) to a LexBIG vocabulary engine.

LexBIG Runtime
Consolidated Host – EVS Access

caBIO Server / Java Runtime

LexBIG Java API — Mediator — EVS Service

LexBIG DataStore

EVS Clients — Editors — Browsers — Query Tools

LexBIG Host

2 March 2006   Draft
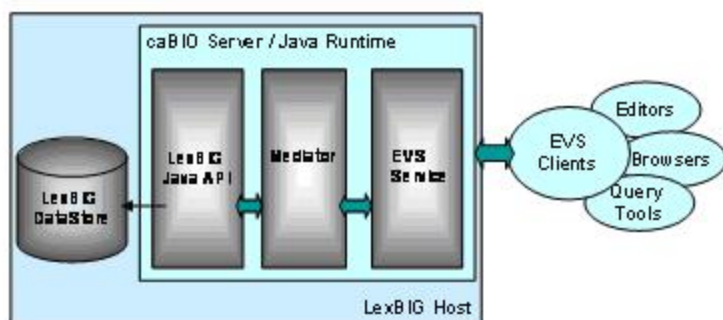
The primary goal is to provide a compatible experience for existing EVS browsers and client applications.

> ⓘ **Note**
>
> This diagram shows the possible inclusion of a mediation layer between EVS and the LexBIG runtime.

This would be done to facilitate alternate communications with the LexBIG server (e.g. through web services as described in the text that follows).

The LexBIG API is designed with web and grid-level enablment in mind. This diagram depicts deployments that wrap the current API to allow the runtime to be accessed through web or grid services.

LexBIG Runtime
Consolidated Host – Web Service Access

App Server / Java Runtime

LexBIG Java API — Server

LexBIG Data Store

Web or Grid Service Clients — Editors — Browsers — Query Tools

LexBIG Host

2 March 2006   Draft
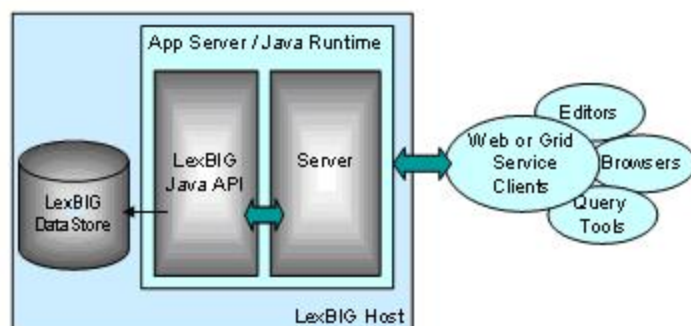
What is LexGrid?

LexGrid is an initiative of the Mayo Clinic Division of Biomedical Informatics that focuses on the representation, storage, and dissemination of vocabularies. This effort centers on, but is not limited to, the domain of medical vocabularies and nomenclatures. Focal points of the LexGrid project include the development and promotion of standards, tools, and content that:

- Provide flexibility to represent yesterday's, today's and tomorrow's terminological resources using a single information model.
- Provide the ability for these resources to be published online, cross-linked, and indexed.
- Provide standardized building blocks and tools that allow applications and users to take advantage of the content where and when it is needed.
- Provide consistency and standardization required to support large-scale terminology adoption and use.

Additional information for LexGrid is available at Mayo Clinic Informatics.

What is LexBIG?

LexBIG is a more specific project that applies LexGrid vision and technologies to requirements of the caBIG® community. The goal of the project is to build a vocabulary server accessed through a well-structured application programming interface (API) capable of accessing and distributing vocabularies as commodity resources. The server is to be built using standards-based and commodity technologies. Primary objectives for the project include:

- Provide a robust and scalable open source implementation of EVS-compliant vocabulary services. The API specification will be based on but not limited to fulfillment of the caCORE EVS API. The specification will be further refined to accommodate changes and requirements based on prioritized needs of the caBIG® community.
- Provide a flexible implementation for vocabulary storage and persistence, allowing for alternative mechanisms without impacting client applications or end users. Initial development will focus on delivery of open source freely available solutions, though this does not preclude the ability to introduce commercial solutions (e.g. Oracle).
- Provide standard tooling for load and distribution of vocabulary content. This includes but is not limited to support of standardized representations such as UMLS Rich Release Format (RRF), the OWL web ontology language, and Open Biomedical Ontologies (OBO).

The goal for the initial year of development was to achieve the Bronze level of compatibility with regard to the caBIG® requirements. Silver-level compatibility is being pursued.

LexGrid Model

The LexGrid Model is Mayo Clinic's proposal for standard storage of controlled vocabularies and ontologies. The LexGrid Model defines how vocabularies should be formatted and represented programmatically, and is intended to be flexible enough to accurately represent a wide variety of vocabularies and other lexically-based resources. The model also defines several different server storage mechanisms and an XML format. This model provides the core representation for all data managed and retrieved through the LexBIG system, and is now rich enough to represent vocabularies provided in numerous source formats such as OWL (NCI Thesaurus) and RRF (NCI MetaThesaurus).

Once the vocabulary information is represented in a standardized format, it becomes possible to build common repositories to store vocabulary content and common programming interfaces and tools to access and manipulate that content. The LexBIG API developed for caBIG® is one such interface, and is described in additional detail in LexBIG APIs.

Following are some of the higher-level objects incorporated into the model definition:

Code Systems

Each service defined to the LexGrid model can encapsulate the definition of one or more vocabularies. Each vocabulary is modeled as an individual code system, known as a *codingScheme*. Each scheme tracks information used to uniquely identify the code system, along with relevant metadata. The collection of all code systems defined to a service is encapsulated by a single *codingSchemes* container.

Concepts

A code system may define zero or more coded concepts, encapsulated within a single container. A concept represents a coded entity (identified in the model as a *concept*) within a particular domain of discourse. Each concept is unique within the code system that defines it. To be valid, a concept must be qualified by at least one designation, represented in the model as a *property*. Each property is an attribute, facet, or some other characteristic that may represent or help define the intended meaning of the encapsulating concept. A concept may be the source for and/or the target of zero or more relationships. Relationships are described in more detail in a following section.

Relations

Each code system may define one or more containers to encapsulate relationships between concepts. Each named relationship (e.g. "hasSubtype" or "hasPart") is represented as an *association* within the LexGrid model. Each relations container must define one or more association. The association definition may also further define the nature of the relationship in terms of transitivity, symmetry, reflexivity, forward and inverse names, etc. Multiple instances of each association can be defined, each of which provide a directed relationship between one source and one or more target concepts.

Source and target concepts may be contained in the same code system as the association or another if explicitly identified. By default, all source and target concepts are resolved from the code system defining the association. The code system can be overridden by each specific association, relation source (*associationInstance*), or relation target (*associationTarget*).

LexBIG Extensions

The LexBIG vocabulary model extends the LexGrid model to provide unique constructs or granularity required by caBIG® that are not present in the core model. While many extensions exist, this document will focus on some of direct relevance to the high-level architecture.

Concept Resolution

LexBIG allows the service runtime to provide managed resolution of code-based objects that are referenced through LexBIG-specific lists and iterators (mechanism that allow streaming of list content). These lists and iterators are typically returned when requesting sets or graphs of vocabulary terms through the LexBIG API (described in LexBIG APIs). Some model components involved in the resolution process include:

`ConceptReference` - A globally unique reference to a concept code.

`ResolvedConceptReference` - A concept reference for which additional information has been resolved, including description and relationship participation.

`AssociatedConcept` - A concept reference that contains full detail in participation as a source or target of an association, including indications of navigability and qualification.

> ⓘ **Note**
>
> Formal representation of the LexGrid and LexBIG models are discussed in VKC:Information Models.

Information Models

Scope of Information Provided

The information below is provided for introductory purposes. A full description of all available model components is also available in the javadoc distributed with the LexEVS installation package (see file breakdown in the LexEVS 5.x Installation Guide). Since the javadoc is automatically generated and synchronized during the build process, it is recommended as the primary reference for use by LexEVS developers.

LexGrid Model

The LexGrid model is mastered in XML schema. The LexBIG project currently builds on the 2008 version of the LexGrid schema. A formal representation, showing portions of this structure that are of primary interest to the LexBIG project, is presented below. A complete version of the model is available at Mayo Clinic Informatics.

CodingSchemes

The CodingSchemes branch of the model defines high level containers for concepts and relations. Each CodingScheme represents a unique code system or version in the LexBIG service. Components of interest include:

codingSchemes
A collection of one or more coding schemes.

codingScheme
A resource that makes assertions about a collection of terminological entities.

entities
A set of entity codes and their lexical descriptions

relations
A collection of relations that represent a particular point of view or community.

versions
A list of past versions of the coding scheme.

mappings
A list of all of the local identifiers and defining URI's that are used in the associated resource

properties
A collection of properties.

The following graphic shows coding schems.
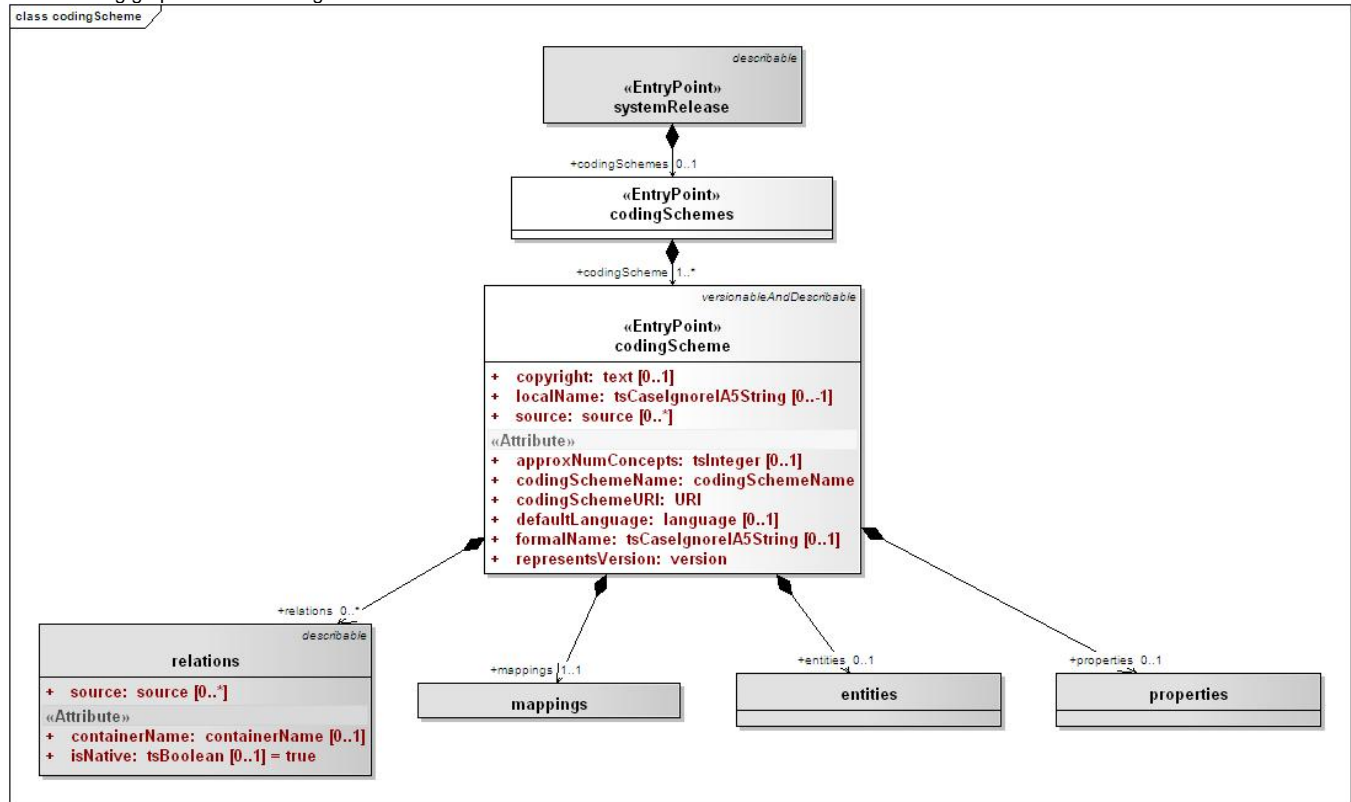
Concepts

Each concept represents a unique entity within the code system, which can be further described by properties and related to other concepts through relations.

conceptsAndInstances

codingScheme

A resource that makes assertions about a collection of terminological entities.

entities

A set of entity codes and their lexical descriptions

entity

A set of lexical assertions about the intended meaning of a particular entity code.

concept

An entity that represents a class or category. The entityType for the class concept must be "concept".

instance

An entity that represents an instance or an individual. The entityType for the class concept must be "instance".

relations

A collection of relations that represent a particular point of view or community.

association

A binary relation from a set of entities to a set of entities and/or data. The entityType for the class concept must be "association".

class conceptsAndInstances

«EntryPoint»
codingSchemes::codingScheme
versionableAndDescribable

+ copyright: text [0..1]
+ localName: tsCaseIgnoreIA5String [0..-1]
+ source: source [0..*]

«Attribute»
+ approxNumConcepts: tsInteger [0..1]
+ codingSchemeName: codingSchemeName
+ codingSchemeURI: URI
+ defaultLanguage: language [0..1]
+ formalName: tsCaseIgnoreIA5String [0..1]
+ representsVersion: version

+relations 0..*

relations::relations
describable

+ source: source [0..*]
«Attribute»
+ containerName: containerName [0..1]
+ isNative: tsBoolean [0..1] = true

+entities 0..1

entities

+entity 1..*

entity
versionableAndDescribable

+ entityType: entityType [0..*]
«Attribute»
+ entityCode: entityCode
+ entityCodeNamespace: namespaceName [0..1]
+ isAnonymous: tsBoolean [0..1]
+ isDefined: tsBoolean [0..1]

+association 1..*

relations::association

{count(entityType=concept) > 0} ₁) > 0}

instance

{count(entityType=association) > 0}
concept

The class labeled "instance" is intended to be equivalent to OWL:Individual. The label "instance" has been maintained for backwards compatibility. We apologize for any confusion that it may introduce.

Note: The class labeled "concept" represents "kinds" or "universal" entities. The OWL 1.1 equivalent would be OWL:Class. The "concept" label has been maintained for backwards compatibility. We apologize for any confusion that it may introduce.

entities

codingScheme

A resource that makes assertions about a collection of terminological entities.

entities

A set of entity codes and their lexical descriptions

entity

A set of lexical assertions about the intended meaning of a particular entity code.

concept

An entity that represents a class or category. The entityType for the class concept must be "concept".
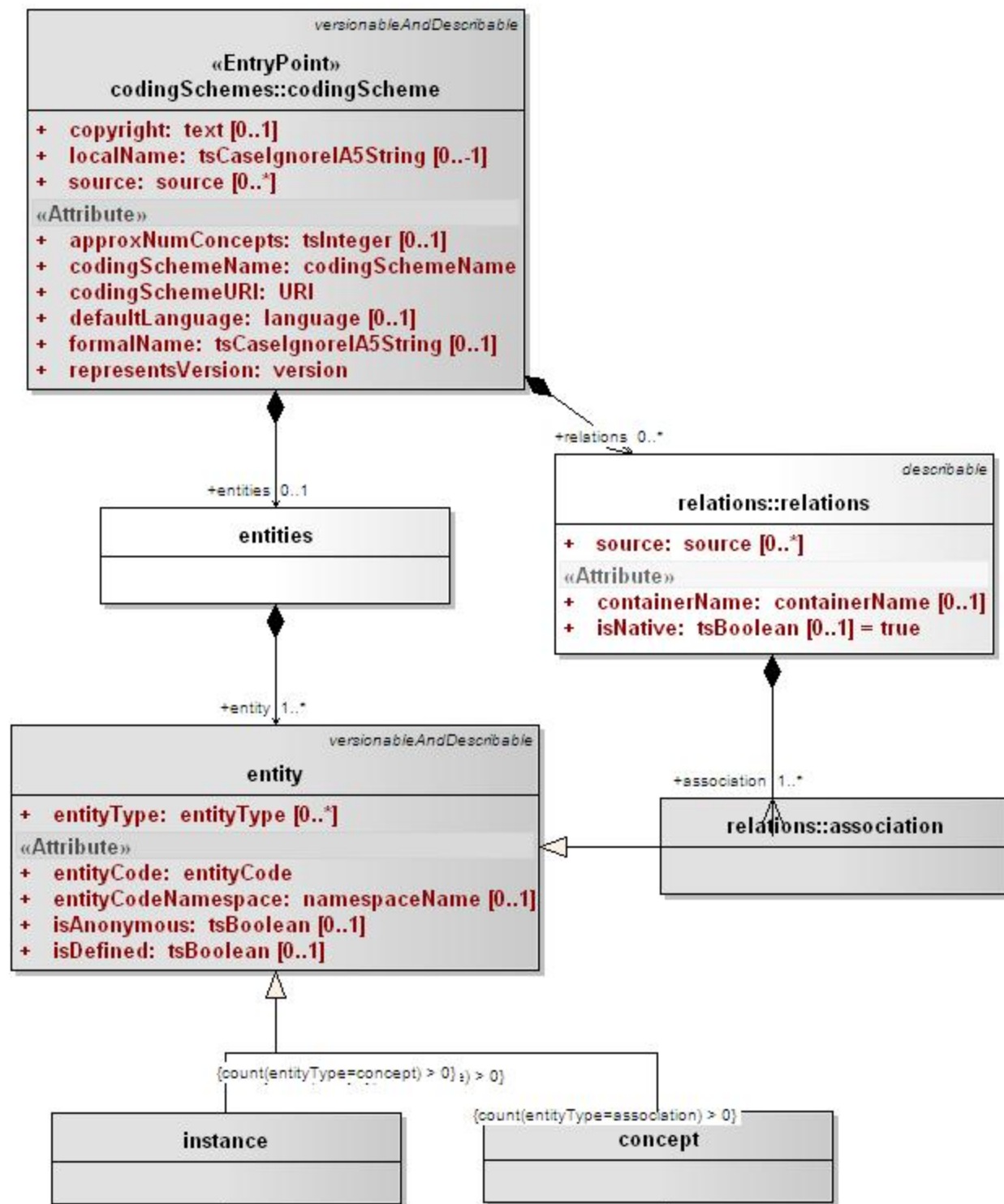
instance

An entity that represents an instance or an individual. The entityType for the class concept must be "instance".

association

A binary relation from a set of entities to a set of entities and/or data. The entityType for the class concept must be "association."
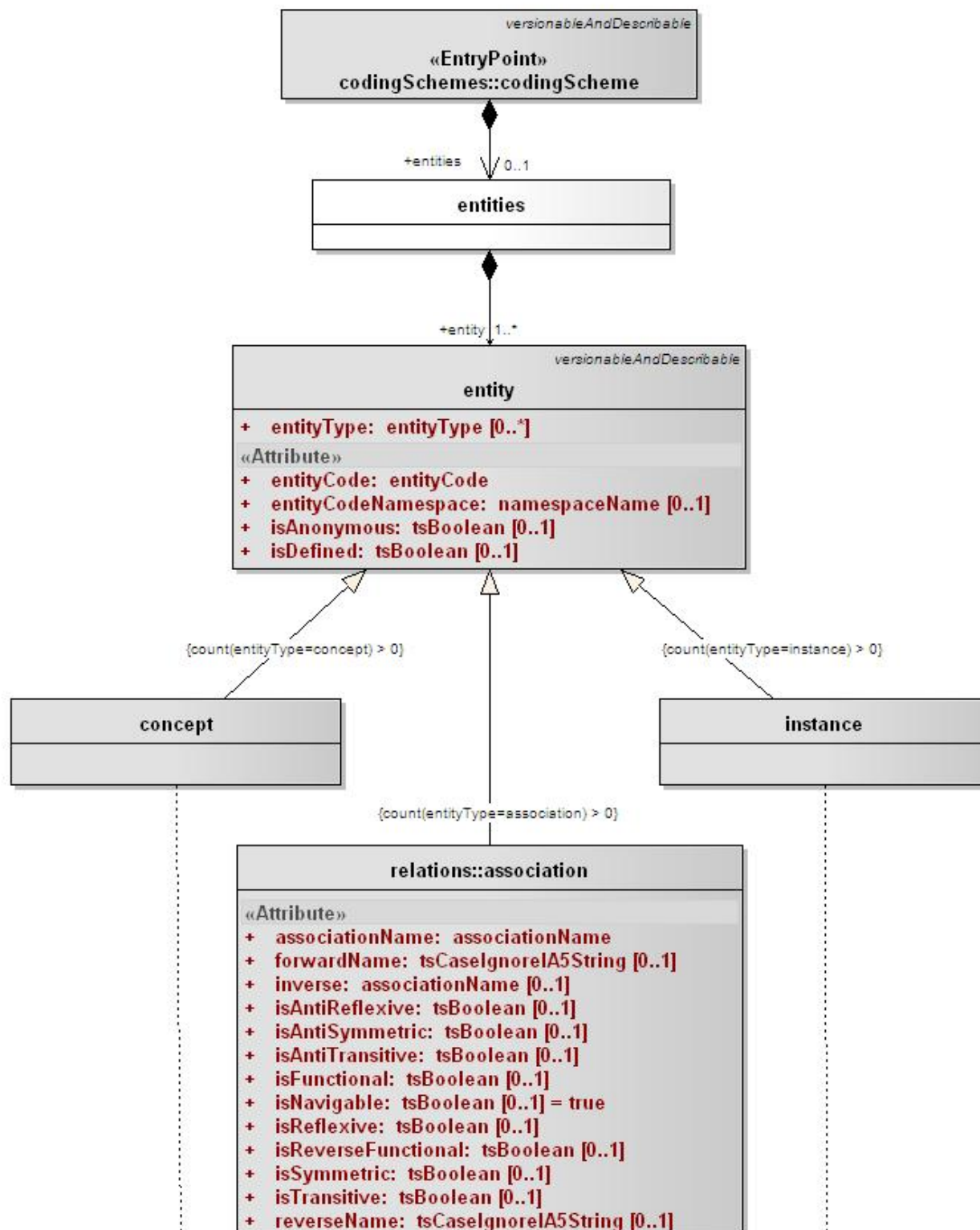
Note: The class labeled "concept" represents a "kind" or "universal" entity. The OWL 1.1 equivalent would be OWL:Class. The "concept" label has been maintained for backwards compatibility. We apologize for any confusion that it may introduce.

The class labeled "instance" is intended to be equivalent to OWL:Individual. The label "instance" has been maintained for backwards compatibility. We apologize for any confusion that it may introduce.

entity

entity

A set of lexical assertions about the intended meaning of a particular entity code.

comment

A property that is used as an annotation or other note about the state or usage of the entity. The propertyType of comment must be "comment"

definition

A property that defines the entity in a particular langage or context.. The propertyType of definition must be "definition"

presentation

A property ths represents or designates the meaning of the entityCode. The propertyType of presentation must be "presentation"

property

A description, definition, annotation or other attribute that serves to further define or identify an resource.

propertyLink

A link between two properties for an entity.. Examples include acronymFor, abbreviationOf, spellingVariantOf, etc. Must be in supportedPropertyLink.

The following graphic displays an entity.
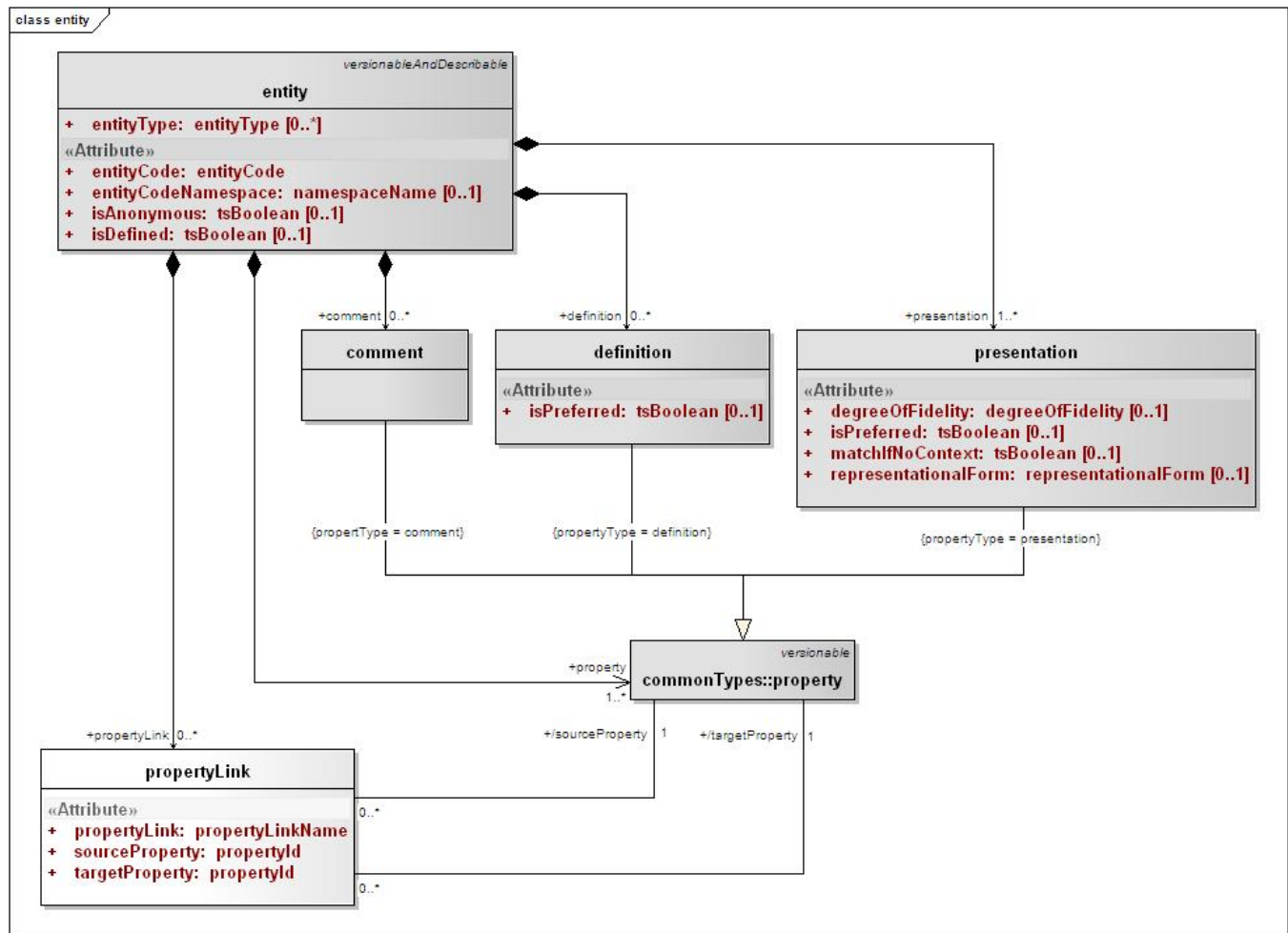
Relations

Relations are used to define and qualify associations between concepts.

association

codingScheme

A resource that makes assertions about a collection of terminological entities.

relations

A collection of relations that represent a particular point of view or community.

entity

A set of lexical assertions about the intended meaning of a particular entity code.

association

A binary relation from a set of entities to a set of entities and/or data. The entityType for the class concept must be "association".

associationSource

An entity that occurs in one or more instances of a relation on the "from" (or left hand) side of a particular relation.

The following figure shows an association.
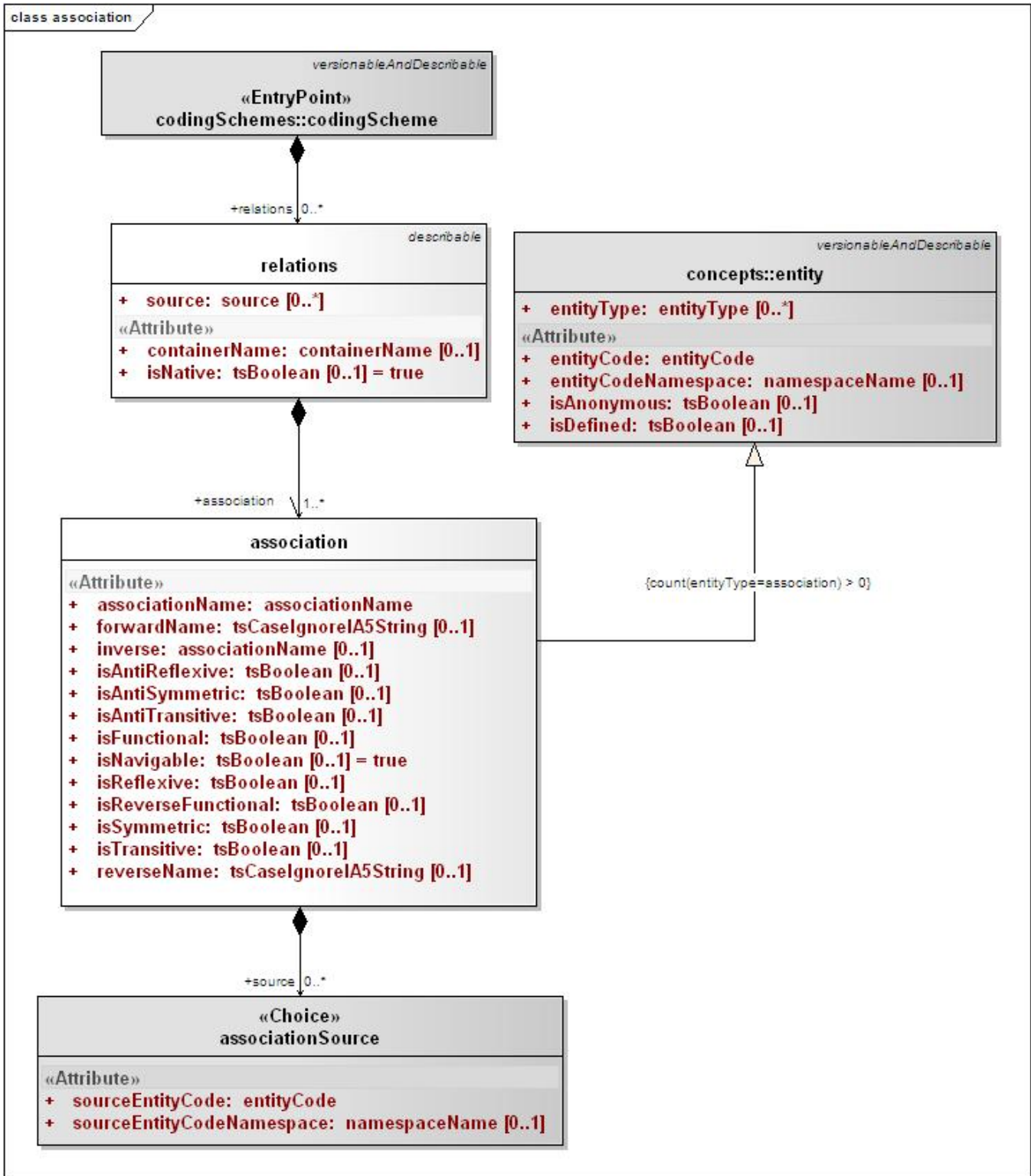
associationInstance

association

A binary relation from a set of entities to a set of entities and/or data. The entityType for the class concept must be "association".

associationSource

An entity that occurs in one or more instances of a relation on the "from" (or left hand) side of a particular relation.

associationTarget

An entity on the "to" (or right hand) side of a relation.

associationData

An instance of a target or RHS data value of an association.

associatableElement

Information common to both the entity and data form of the "to" (or right hand) side of an association.

associationQualification

A modifier that further qualifies the particular association instance.

The following figure shows associationinstance.

association *entity*

«Attribute»
+ associationName: associationName
+ forwardName: tsCaseIgnoreIA5String [0..1]
+ inverse: associationName [0..1]
+ isAntiReflexive: tsBoolean [0..1]
+ isAntiSymmetric: tsBoolean [0..1]
+ isAntiTransitive: tsBoolean [0..1]
+ isFunctional: tsBoolean [0..1]
+ isNavigable: tsBoolean [0..1] = true
+ isReflexive: tsBoolean [0..1]
+ isReverseFunctional: tsBoolean [0..1]
+ isSymmetric: tsBoolean [0..1]
+ isTransitive: tsBoolean [0..1]
+ reverseName: tsCaseIgnoreIA5String [0..1]

+source [0..*]

«Choice»
associationSource

«Attribute»
+ sourceEntityCode: entityCode
+ sourceEntityCodeNamespace: namespaceName [0..1]

+target [0..*]

associationTarget

«Attribute»
+ targetEntityCode: entityCode
+ targetEntityCodeNamespace: namespaceName [0..1]

+targetData [0..*]

associationData

+ associationDataText: text [0..1]

associatableElement *versionable*

+ usageContext: context [0..*]

«Attribute»
+ associationInstanceId: nodeId [0..1]
+ isDefining: tsBoolean [0..1]
+ isInferred: tsBoolean [0..1]

+associationQualification [0..*]

associationQualification

+ qualifierText: text [0..1]

«Attribute»
+   associationQualifier: associationQualifierName

Naming

These elements are primarily used to define metadata for a coding scheme, mapping locally used names to global references.

URIMap

A local identifier that is used in a specific context (e.g. language, property name, data type, etc) and an optional URI that can be used to find the exact definition and meaning of the local id. Note: the string portion of this entry can be used to provide additional documentation or information, especially when a URI is not supplied.

supportedAssociation

An associationName and the URI of the defining resource.

supportedAssociationQualifier

An associationQualifier and the URI of the defining resource

supportedCodingScheme

A codingSchemeName and the URI of the defining resource

supportedStatus

An entryStatus and the URI of the defining resource

supportedEntityType

An entityType and the URI of the defining resource

supportedContext

A context and the URI of the defining resource

supportedContainerName

A containerName and the URI of the defining resource

supportedDegreeOfFidelity

A degreeOfFidelity and the URI of the defining resource

supportedLanguage

A language and the URI of the defining resource

supportedProperty

A propertyName and the URI of the defining resource

supportedSortOrder

The local identifier and the URI of the defining resource

supportedHierarchy

A list of associations that can be browsed hierarchically.

supportedNamespace

A namespaceName and the corresponding URI

supportedPropertyType

A propertyType and the URI of the defining resource

supportedPropertyQualifier

A propertyQualifierName the URI of the defining resource

supportedPropertyQualifierType

A propertyQualifierType the URI of the defining resource

supportedPropertyLink

A propertyLinkName and ththe URI of the defining resource

supportedRepresentationalForm

A representationalForm and the URI of the defining resource

supportedSource

A source and the URI of the defining resource. Source references can also carry an additional compositional rule section that describes how to combine a subpart such as a page number, section name, etc. with the core URI in order to form a meaningful URL. An optional role can also be specified.

supportedSourceRole

A source role and athe URI of the defining resource

The following figure shows naming.
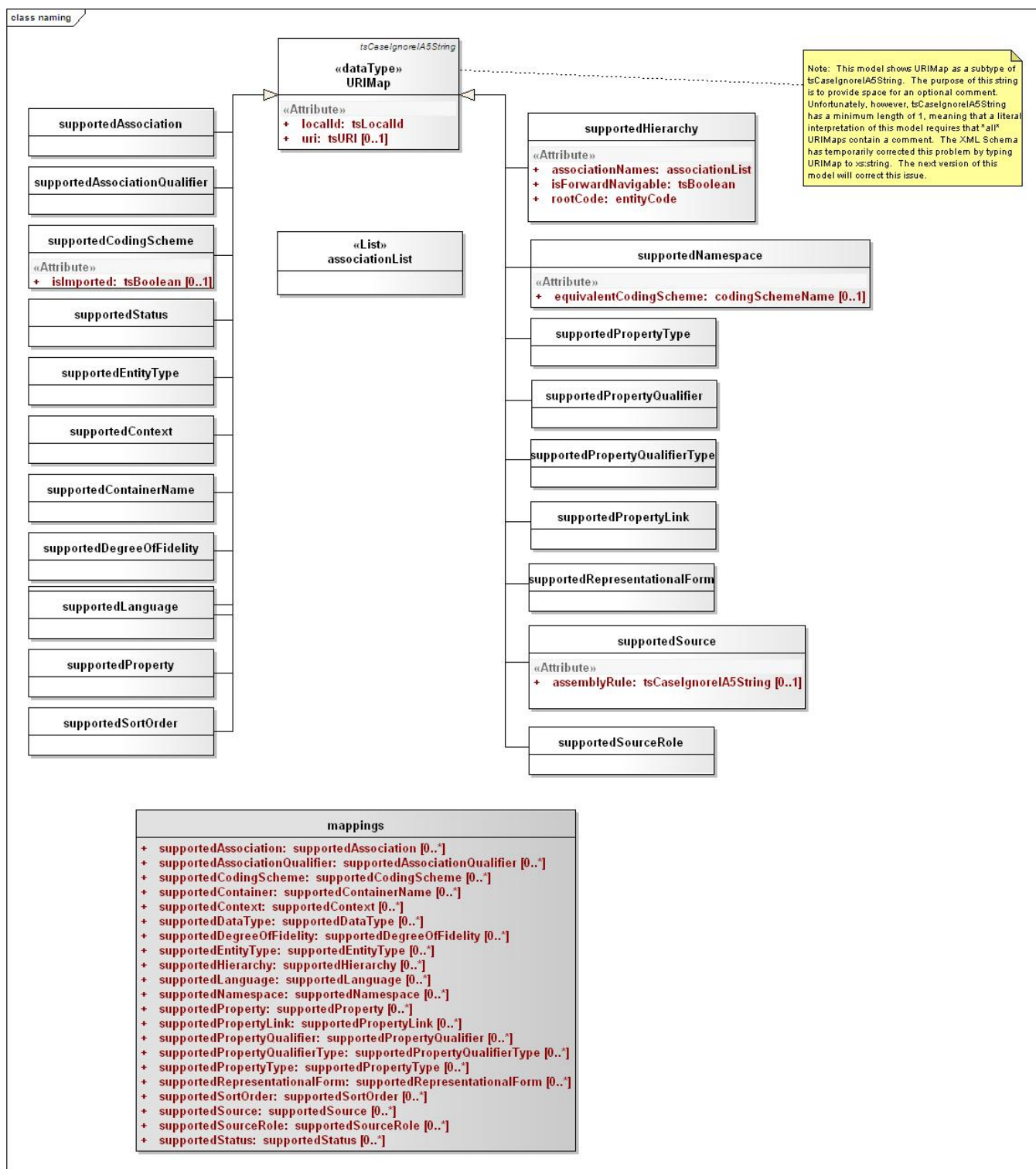
LexBIG Model

The following extensions to the LexGrid model were introduced in support of caBIG® requirements. As with the LexGrid model, this document provides a summary of the most significant elements for consideration by LexBIG programmers. The complete and current version of the model is available online at [Mayo Clinic Informatics](#).

Core

LexBIG core elements provide enhanced referencing and controlled resolution of LexGrid model objects.

The following figure shows the LexBIG core elements.
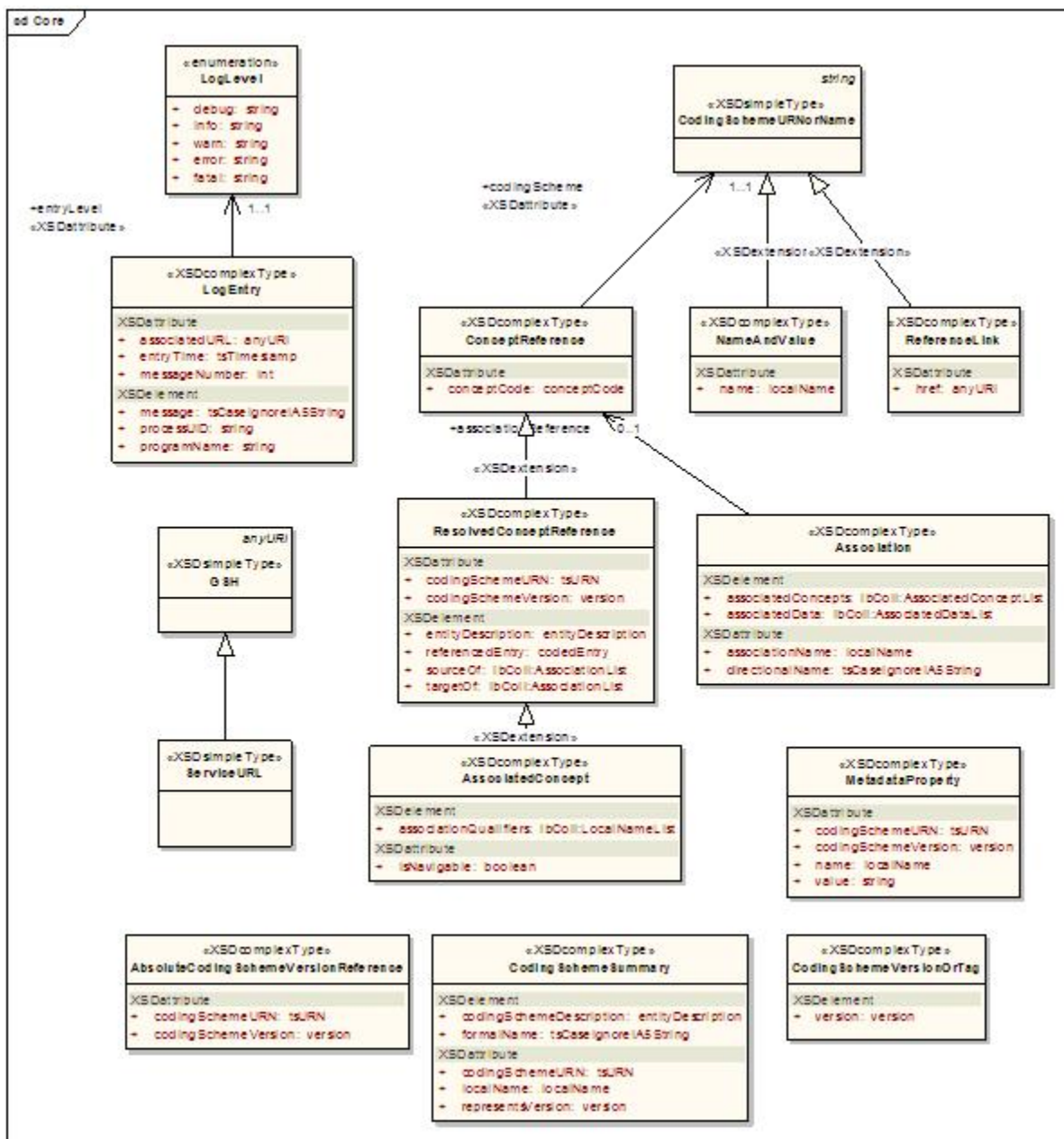
Components of interest include:

AbsoluteCodingSchemeVersionReference

An absolute reference to a coding scheme. This form of reference is service independent, as it doesn't depend on local coding schemes names or virtual tags.

AssociatedConcept

A concept reference that is the source or target of an association.

Association

The representation of a particular association as it appears in a CodedNode.

CodingSchemeSummary

Abbreviated list of information about a coding scheme.

CodingSchemeURNorName

Either a local name or the URN of a coding scheme. These two are differentiated syntactically - if the entity includes a colon ":" or a hash "#" it is assumed to be a URN. Otherwise it is assumed to be a local name.

CodingSchemeVersionOrTag

A named coding scheme version or a virtual tag (e.g. latest, production, etc). Note that the tagged form of identifier is only applicable in the context of a given service, as one service may identify the scheme as "production" and another as "staging".

ConceptReference

A reference to a coding scheme and a concept code.

LogEntry

A single recorded log entry.

LogLevel

Indicates severity of the log entry.

MetadataProperty

Reference to a property name and value stored in the coding scheme metadata.

NameAndValue

A simple name/value pair.

ReferenceLink

Any reference to another document element. Used by the REST architecture to embed links.

ResolvedConceptReference

A resolvable concept reference.

ServiceURL

References a service in the Globus environment, this will be a global service handle (GSH).

InterfaceElements

Defines metadata related to model objects required by the runtime.

The following figure shows the interfaceelements.
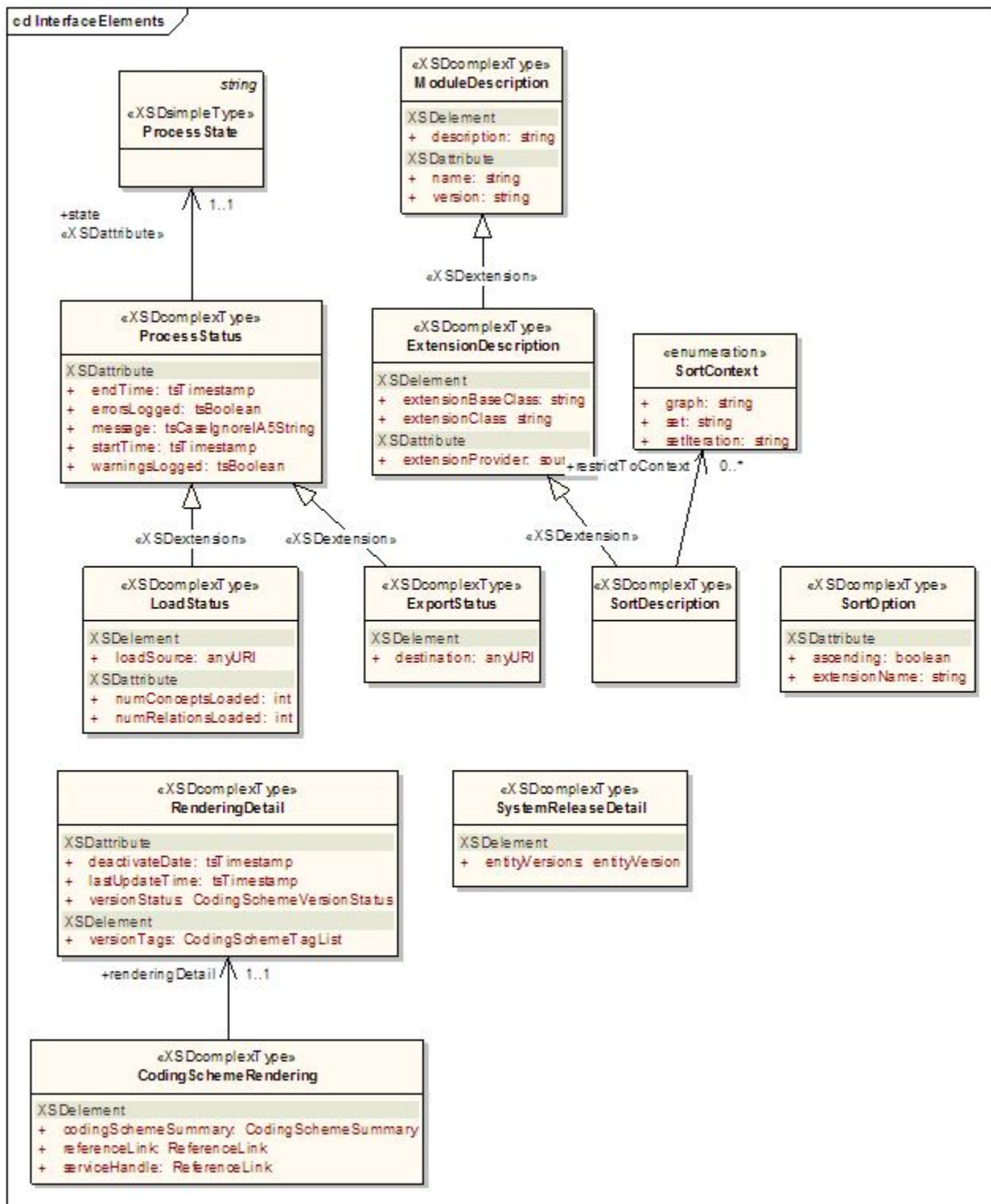
Components of interest include:

CodingSchemeRendering

Information about a coding scheme as it appears in a particular service.

ExportStatus

Reports the state of LexBIG export operations.

ExtensionDescription

Describes an add-on module registered to the LexBIG environment.

LoadStatus

Reports the state of LexBIG load operations.

ModuleDescription

Describes a LexBIG integrated software module.

ProcessState

Enumerates possible status reported for LexBIG runtime operations.

ProcessStatus

Reports the state of LexBIG runtime operations.

RenderingDetail

The details of how a coding scheme is rendered in a given service.

SortContext

Describes a LexBIG sort module.

SortDescription

A description of a LexBIG extension module.

SortOption

Represents a pairing of sort algorithm and order.

SystemReleaseDetail

The combination of a system release and all of the entityVersions that accompanied that release.

NCIHistory

Maintains a record of modifications made to a code system.

The following figure shows the NCIHistory.

Components of interest include:

changeType

Atomic modification actions. Currently populated from a combination of Concordia, SNOMED-CT list and NCI's action list.
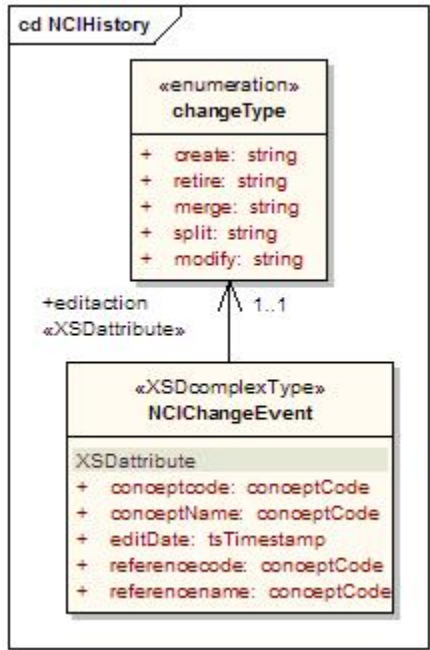
NCIChangeEvent

This link does not work, but I'm not sure it could without an ftp application.

```
Link provided for historical purposes ftp://ftp1.nci.nih.gov/pub/cacore/EVS/ReadMe_history.txt
```

. Note that date and time of the change event is recorded in the containing version. All change events for the same/date and time a recorded in the same version.

Architecture

LexBIG

LexBIG Services

This section describes architectural detail for services provided by the LexBIG system. These services are geared toward the administration, management, and serving of vocabularies defined to the LexGrid/LexBIG information model. A system overview is provided, followed by a description of key subsystems and components. Each subsystem is described in terms of its overall structure, formal model, and specification of key public interfaces.

The following figure shows the LexBIG service components.

- Service Metadata: Coding Schema Metadata and Licensing
- Query Service: Lexical Set Operations, Graph Operations, History
- Service Manager: Loaders and Indexers
- Extensions: Plug-ins
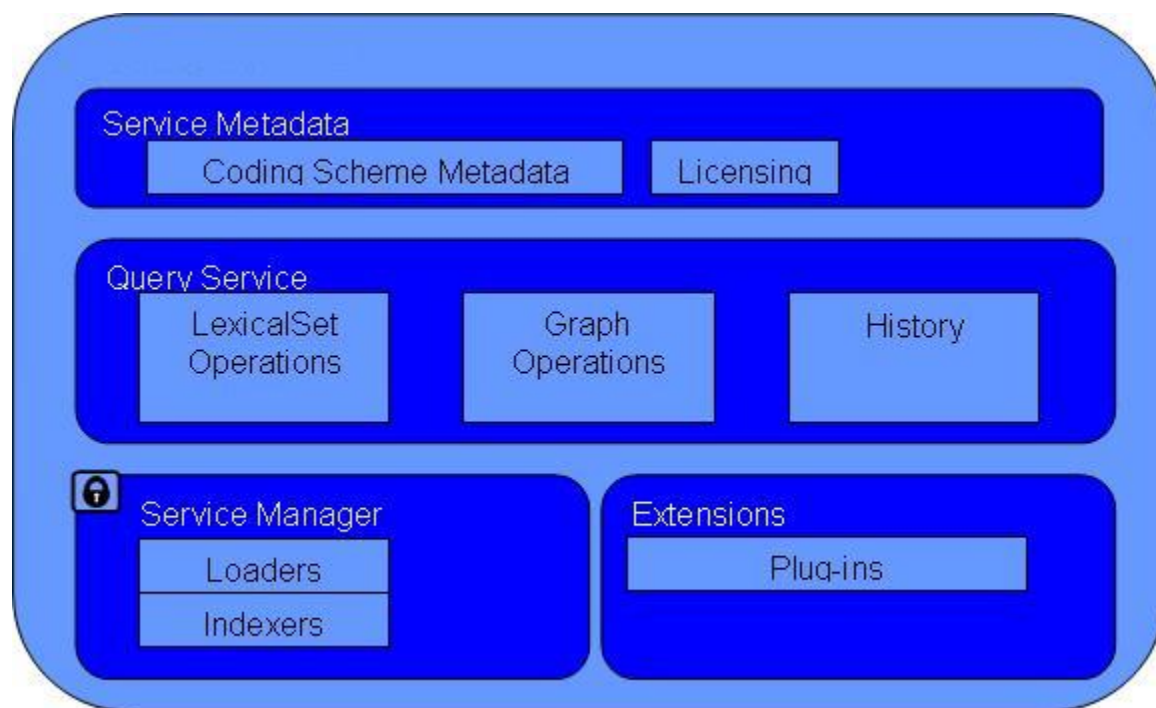
The LexBIG Service is designed to run standalone or as part of a larger network of services. It is comprised of four primary subsystems: Service Management, Service Metadata, Query Operations, and Extensions. The **Service Manager** provides administration control for loading a vocabulary and activating a service. The **Service Metadata** provides external clients with information about the vocabulary content (e.g. NCI Thesaurus) and appropriate licensing information. The **Query Operations** provide numerous functions for querying and traversing vocabulary content. Finally, the **Extensions** component provides a mechanism to extend the specific service functions, such as Loaders, or re-wrap specific query operations into convenience methods.

Primary points of interaction for programming include the following classes:

`LexBIGService` - This interface provides centralized access to all LexBIG services.

`LexBIGServiceManager` - The service manager provides a centralized access point for administrative functions, including write and update access for a service's content. For example, the service manager allows new coding schemes to be validated and loaded, existing coding schemes to be retired and removed, and the status of various coding schemes to be updated and changed.

caGRID Hosting

The following figure shows how the caGrid Hosting Environment comprises the LexBIG Service. The LexBIG Service comprises Service Metadata, Query Service, Service Manager, and Extensions. Service Metadata has a link outside the Hosting environment to a Service Discovery.
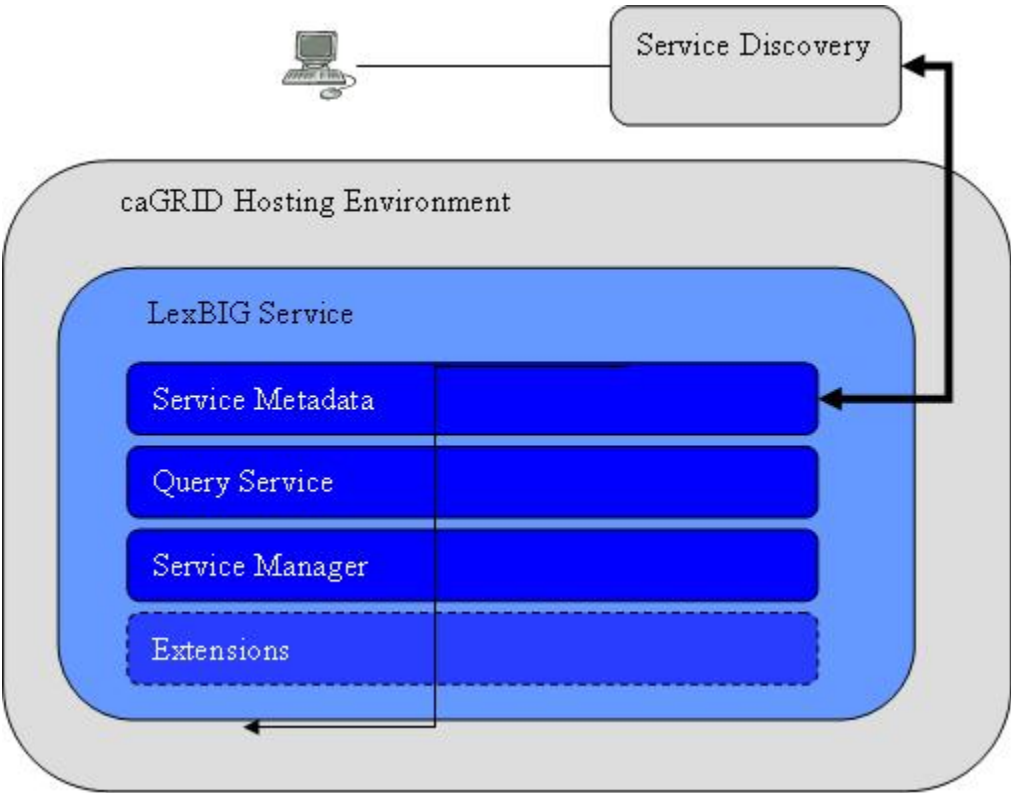
The LexBIG architecture provides the underpinnings LexBIG services to be made accessible through the caGRID environment in the future, where LexBIG services might optionally be deployed in a caGRID Globus container. caGrid provides a Globus service for service registration and discovery. LexBIG services deployed to the grid would be registered in the NCICB registry and be searchable through the NCICB index service.

> (i) **Specification**
>
> Additional specifications related to the registration and discovery of LexBIG services in the caGRID environment will be included later phases of work in concordance with caGRID 1.0. This is will be coordinated with caBIG® Architecture workspace designees.

Service Management Subsystem

The following figure shows the components of the Service Manager: Indexers, Loaders, Loader Administration, and Service Administration. The Loaders are extendable to LexGrid XML, OWL, RRF, OBO, and Other.
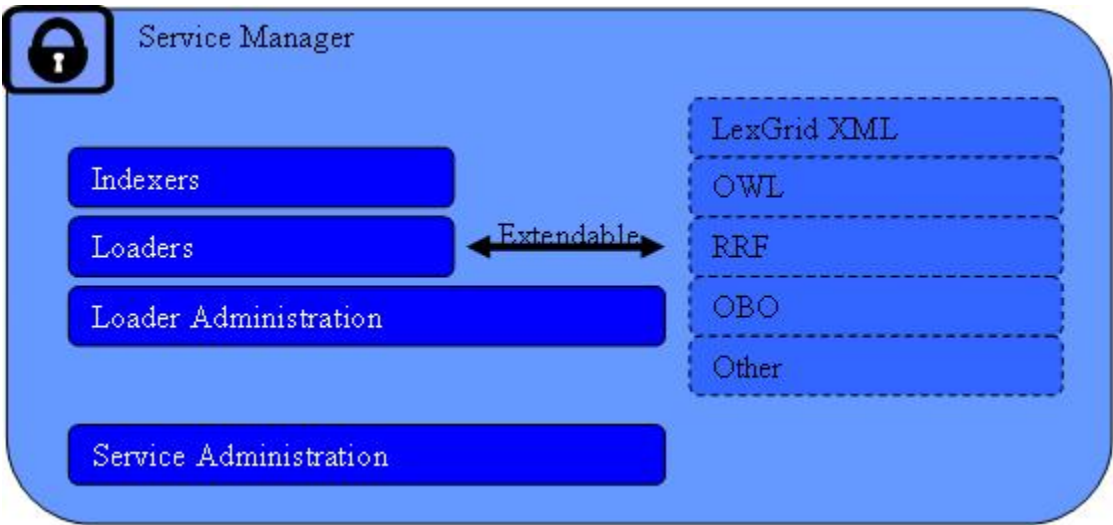
This subsystem provides administrative access to functions related to management and publication of LexBIG vocabularies. These functions are generally considered to be reserved for LexBIG administrators, with detailed instructions on how to secure and carry out related tasks described by the *LexBIG Administrator's Guide*.

This subsystem is further broken down into the following components:

- **Indexers**
  Vocabularies may be indexed to provide enhanced performance or query capabilities. Types of indexes incorporated into the LexBIG system include but are not limited to the following:
    - Lexical Match - for example, "begins-with" and "contains"
    - Phonetic - allows for the ability to query based on "sounds-like" entry of search criteria.
    - Stemming - allows for the ability to find lexical variations of search terms.
      Index creation is typically bundled into the load process. Architecturally speaking, however, this capability is decoupled and extensible.
- **Loaders**
  Vocabularies may be imported to the system from a variety of accepted formats, including but not limited to:
    - LexGrid XML (LexBIG canonical format)
    - NCI Thesaurus, provided in Web Ontology Language format (OWL)
    - UMLS Rich Release format (RRF)
    - Open Biomedical Ontologies format (OBO)
      As with indexers, the load mechanism is designed to be extensible from an architectural standpoint. Additional loaders can be supported by the introduction of pluggable modules. Each module is implemented in the Java programming language according to a LexBIG-provided interface, and registered to the loader runtime environment.

Metadata and Discovery Subsystem

The following graphic shows that the Metadata Service comprises Coding Scheme Metadata, Licensing, and Discover and Index Service.

This subsystem provides information about accessible vocabularies, related licensing/copyright information, and registration/discovery of LexBIG services.

The ability to locate and resolve vocabulary metadata is fulfilled through the LexBIGService class. Metadata defined by the LexGrid information model is resolved with each CodingScheme instance. Available metadata on each resolved scheme includes, but is not necessarily limited to, the following:

- License or copyright information
- Supported values (e.g. supported concept status, language, property names, etc)
- Mappings from names used locally to globally unique URNs

In addition, each LexBIGService provides a centralized metadata index that allows registration and query of code system metadata without requiring resolution of individual CodingSchemes. This metadata index is optionally populated, typically during the vocabulary load process. The metadata index allows for the metadata of multiple code systems to be cross-indexed and searched as part of the query subsystem.

Finally, the LexBIG architecture provides the underpinnings for LexBIG services to be made accessible through the caGRID environment in the future, where vocabulary services might be deployed and discovered within a caGRID Globus container. However, this portion of the API is preliminary and awaits coordination with caBIG® Architecture WS designees to determine exact recommendations and nature of LexBIG services on the grid.

Query Subsystem

The following figure shows the Query Subsystem which comprises Lexical Set Operations, Graph Operations, and History. Graph Operations sends data to LexBIG API and LexBIG Data Model (Loaders and Indexers) and LexGRID Vocabulary Model and Repository.
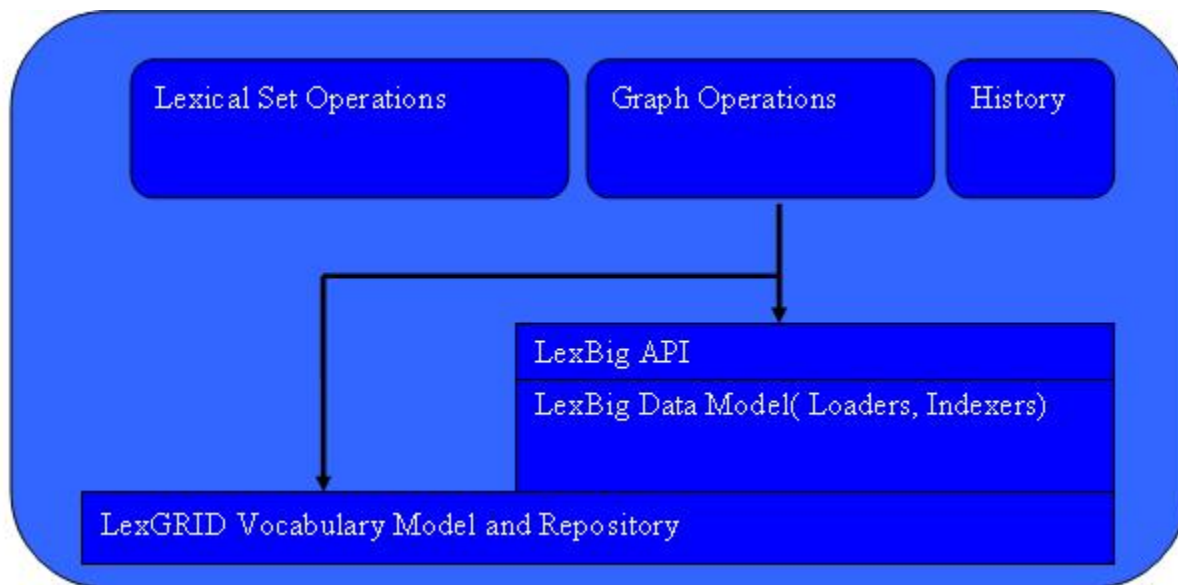
This subsystem provides the functionality required to fulfill caCORE/EVS and other vocabulary requests. The Query Service is comprised of Lexical Operations, Graph Operations, Metadata, and History Operations.

Lexical Set Operations

Lexical Set Operations provides methods to return a lists or iterators of coded entries. Supported query criteria include the application of match/filter algorithms, sorting algorithms, and property restrictions. Support is also provided to resolve the union, intersection or difference of two node sets.

Graph Set Operations

Graph Operations support the subsetting of concepts according to relationship and distance, identification of relation source and target concepts, and graph traversal. Additional operations include enumeration and traversal of concepts by relation, walking of directed acyclic graphs (DAGs), enumeration of source and target concepts for a relation, and enumeration of relations for a concept.

Metadata Operations

Metadata Operations allows for the query and resolution of registered code system metadata according to specified coding scheme references, property names, or values.

History Operations

History provides vocabulary-specific information about concept insertions, modifications, splits, merges, and retirements when supplied by the content provider.

LexEVS API/Grid Service Interaction

(DESIGN DOC IMPORT START)

Revision History

Content changes to this document from the previous to the current level are indicated by revision bars (|) unless a complete rewrite is indicated.

Date	Version	Description	Author
07/29/2008	1.0	Initial document	Kevin Peterson
8/30/2008	1.1	Revised for Security and Exception Handling	Kevin Peterson

> ⓘ **Note**
>
> If this document has been inspected, please indicate the inspection date that each version is based on in the "Change Description and Explanation" area. Entries in this log must be maintained for at least 3 years.

Document Purpose

This document provides the detailed design and implementation of LexBIG Enterprise Vocabulary Service (LexEVS) caGrid Service. It should be noted that the LexEVS Grid Service is no longer part of the caGrid 1.1 infrastructure and will be deployed as a separate unit. This is a change from the previous release of the LexEVS Grid Service.

The LexEVS caGrid service will allow programs to utilize the caGrid 1.2 infrastructure to access LexEVS information that is currently being produced by NCICB.

Implementation Overview

Team Members

The following table lists the team members for the implementation.

Role	Name
Development Lead	Kevin Peterson
Documentation Lead	Kevin Peterson
Project Manager	Tom Johnson

Description

The LexEVS grid service will be used to obtain data accessible via the LexEVS service, specifically, the Distributed LexEVS services. Please refer to the LexEVS 5.0 Programmer's Guide for more information.

For more Documentation, Build/Deployment instructions and examples, visit LexEVS 5.0 Documentation and Training.

Scope

The LexEVS Grid service will provide programmatic access to the LexBIG domain objects that are available via the LexBIG information model.

The LexEVS grid service will be registered in Cancer Data Standards Repository (caDSR) under the following category:

Context	caBIG
Classification Scheme	LexBIG
Version	LexBIG_v2_3_rv1

Architecture

The LexEVS Grid Service is implemented to expose the API and Model of LexBIG 2.3. For more information on LexBIG, see Mayo Clinic Informatics.

LexEVS Grid Service is deployed in a JBoss Application Server, inside of a Globus Web Application installation. LexEVS Grid Service depends on LexBIG APIs, which is also deployed to a JBoss container. For more information on the deployment of EVSAPI, see:

```
http://gforge.nci.nih.gov/docman/index.php?group_id=366&amp;selected_doc_group_id=1914&amp;language_id=1]
```

LexEVS API itself depends on an installation of LexBIG).

The diagram below shows the various components of the LexEVS Grid Service System and how they interact.

Client accesses LexEVS Grid Node Container via SOAP/HTTP. The LexEVS Grid Node accesses EVSAPI and then LexBIG via RMI. LexBIG accesses Database Servers containing vocabulary data. The database servers and loaded vocabulary configuration is a sample only.

deployment Deployment Model

Client

SOAP/HTTP

LexEVS Grid Node Container
- Application Server = JBoss 4.0.4
- caGrid Version = 1.2
- Globus Version = 4.0.3
- Java Version = 1.5
- LexEVS Grid Service Version = 4.2
- Requires Grid Security = False

RMI

RMI

EVSAPI (Distributed LexBIG) Container
- Application Server = JBoss 4.0.4
- EVSAPI Version = 4.2
- Java Version = 1.5

LexBIG
- Java Version = 1.5
- LexBIG Version = 2.3

JDBC

JDBC

JDBC

Database Server 1
- Vocabulary = Gene Ontology
- Vocabulary = HL7
- Vocabulary = LOINC
- Vocabulary = NCI MetaThesaurus
- Vocabulary = NCI Thesaurus
- Vocabulary = UMLS SemNet
- Vocabulary = Zebrafish

Database Server 2
- Vocabulary = MedDRA
- Vocabulary = MGED
- Vocabulary = SNOMED
- Vocabulary = VANDFRT

Database Server N
- Vocabulary = Other Vocabulary

NOTE: Database Servers and loaded vocabulary configuration is a sample only.

LexEVS Grid Service and EVSAPI need not be deployed to physically separate servers, but it is recommended that if they are co-located on the same server, they should be deployed to separate JBoss containers.

Below is the LexEVS Grid Service Architecture, viewed from inside of the Web Service Container. For more information on how Service Contexts and Resources are used, see the #Service Contexts and State section below.

In the following figure, there is an LexEVS Grid Service Client (Operations) contacting the Web Service Container, which communicates with the Globus /Axis WebApp. Within the Globus/Axis WebApp, a Grid Service communicates with the Service Context that distributes data to LexBIG. The Globus/Axis WebApp comprises Grid Service (LexEVS Grid Service which performs Grid Service Context Calls) and the Service Context. The Service Context comprises:

- caGRID Service Context
  - Resource Creator, Resource Home, and Resource Locator.
  - Service Context which performs Service Context Grid Calls.
- Resource Instances
  - Each of the three Resource Instances comprise Operations and Resource Properties.
  - Each Resource Instance forwards data to LexBIG.
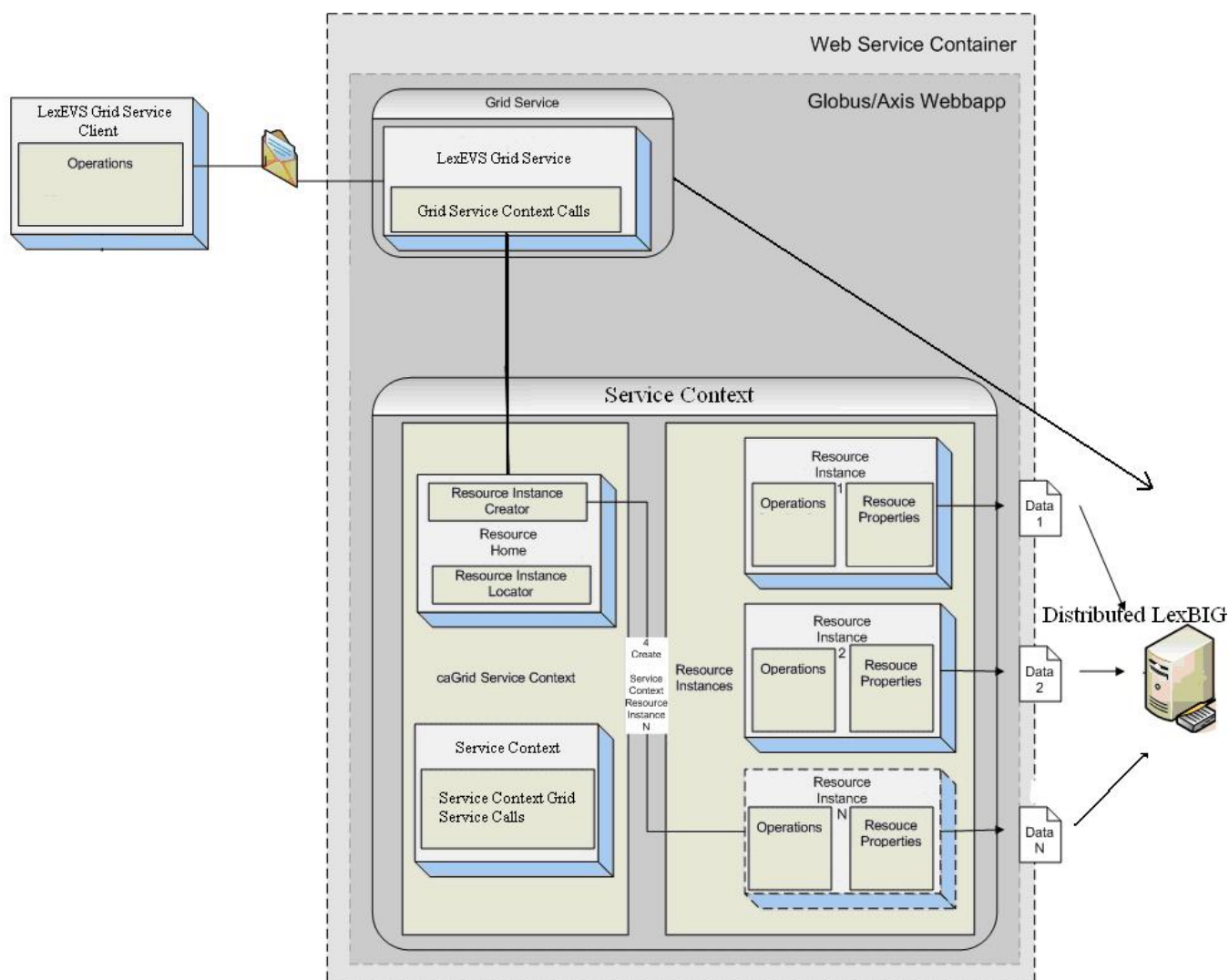
LexEVS Grid Service Class Diagram

The LexEVS Grid Service is built on the LexGrid/LexBIG model and implementation. For more information about this model, visit:

```
Historical link
https://gforge.nci.nih.gov/plugins/scmsvn/viewcvs.php/LexBIG_Core_Services/LexBIG-2.3/lexbig/lbModel/?
root=lexevs
```

```
Historical link
LexGrid|https://gforge.nci.nih.gov/plugins/scmsvn/viewcvs.php/LexBIG_Core_Services/LexBIG-2.3/lgModel/?
root=lexevs
```

Also, visit Mayo Clinic Informatics for background information as well as Class Diagrams, examples, and other information.

For information specific to the LexEVS Grid Service, visit:

```
Historical link
https://gforge.nci.nih.gov/plugins/scmsvn/viewcvs.php/LexBIG_Core_Services/LexBIG-2.3/lexbig/lbModel.cagrid/?
root=lexevs
```

This link contains Class Diagrams and descriptions for input/output parameters, as well as other information concerning the Silver Level Compliance submission package.

LexEVS Grid Service Sequence Diagram

The sequence diagram for the operation "getSupportedCodingSchemes" is described in the text that follows:
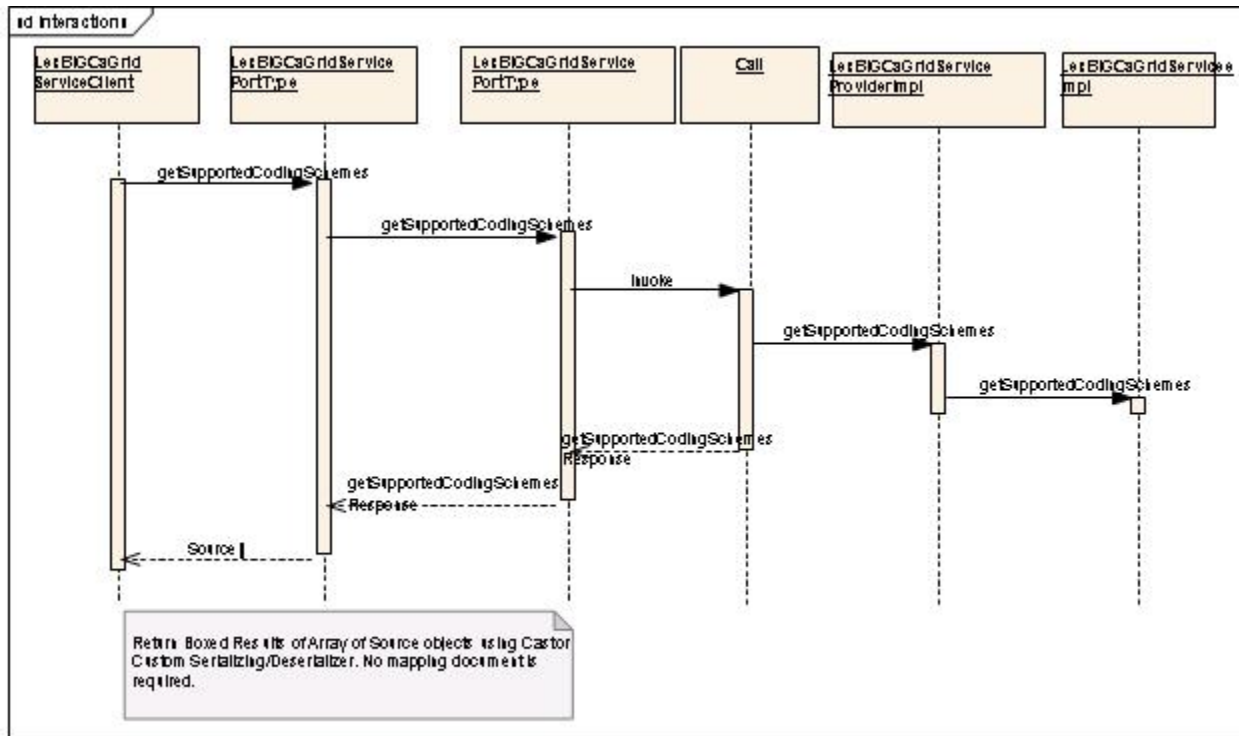
General Call Sequence Example

The client calls caGrid, 'getSupportedCodingSchemes()', caGrid Services calls Distributed LexBIG, 'getSupportedCodingSchemes()', LexBIG returns result of call to caGrid Service, and the results are returned to the client with all the appropriate caGrid security mechanisms.
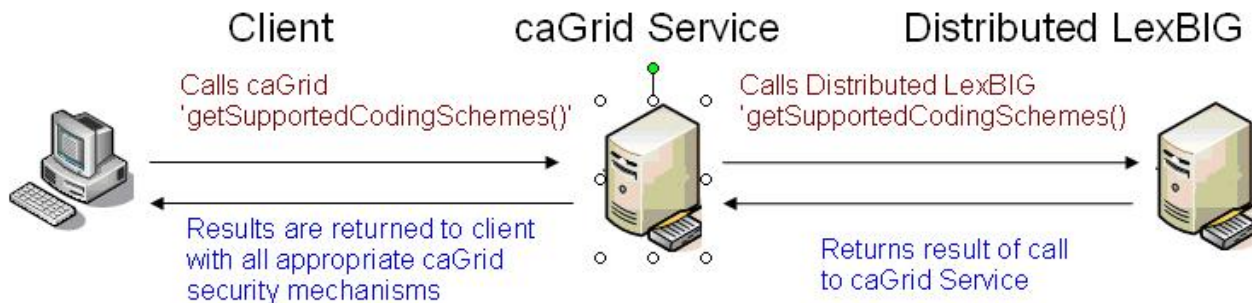
Assumptions

- The LexEVS service will be based on the latest LexEVS 5.0 release.
- The LexEVS Grid Service will not have any method level security. All security requirements will be handled by the actual deployment of the underlying LexEVS 5.0 service. Please see the "Security" section below for more information on how the LexEVS Grid Service utilizes this security.
- The LexEVS Grid Service will not be deployed as a "core" service by caGrid at NCICB as was previously done, but rather will now be deployed as a standalone service.
- The LexEVS Grid Service release schedule will no longer be coupled to the caGrid deployment schedule as previously done.
- Multiple version of LexEVS Grid Service may be active at the same instance in time depending solely on the availability of the underlining EVSAPI service.

Dependencies

- LexEVS 5.0 service needs to be available and running correctly.
- The LexEVS service and operations will use the Introduce toolkit to generate the appropriate structure for registering the service into caDSR.

Third Party Tools

- Introduce Toolkit
- Globus Toolkit (4.0.3) or appropriate version supported by caGrid 1.2
- caGrid 1.2 core infrastructure

Server

The LexEVS Grid Service will be deployed as a "stand alone" grid service at NCICB.

APIs

The main Service API exposed by the LexEVS Grid service will be the LexBIGServiceGrid Interface. All other APIs will not be directly exposed, but will be made available through Service Contexts.

In general, API calls follow this sequence:

- The client invokes a caGrid Service.
- caGrid Node Hosting Service uses Distributed LexBIG to implement a call.
- Distributed LexBIG returns the requested information to caGrid Service.
- caGrid Service returns a response to the client.
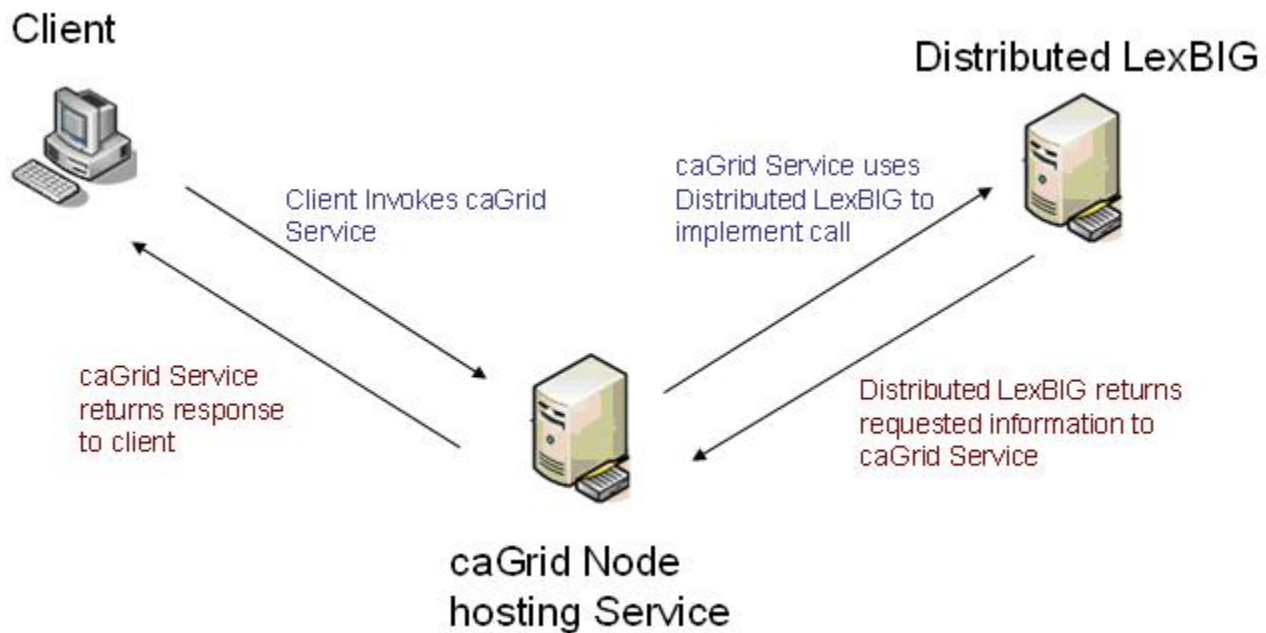
API Examples

Example clients, service calls, and SOAP messages (sample code on the LexEVS documentation GForge archive page)

Example API usage:

Searching for concepts in NCI Thesaurus containing the string "Gene"

Java Code Snippet

```java
    //Create a Connection to the Grid Service
    LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(gridServiceURL);

    //Set up the CodingSchemeIdentification object to define the Coding Scheme</font>
    CodingSchemeIdentification csid = new CodingSchemeIdentification();
    csid.setName("NCI Thesaurus");

    //Get the CodedNodeSet for that CodingScheme (This returns a CodedNodeSet Service Context)
    CodedNodeSetGrid cnsg = lbs.getCodingSchemeConcepts(csid, null);
    //getCodingSchemeConcepts is a Grid Service Call

    //Set the text to match
    MatchCriteria matchText = new MatchCriteria();
    matchText.setText("Gene");
    //Define a SearchDesignationOption, if any
    SearchDesignationOption searchOption = new SearchDesignationOption();

    //Choose an algorithm to do the matching
    ExtensionIdentification matchAlgorithm = new ExtensionIdentification();
    matchAlgorithm.setLexBIGExtensionName("contains");

    //Chose a language
    LanguageIdentification language = new LanguageIdentification();
    language.setIdentifier("en");

    //Restrict the CodedNodeSet
    cnsg.restrictToMatchingDesignations(matchText, searchOption, matchAlgorithm, language);
    //restrictToMatchingDesignations is a Grid Service Call

    //Create a SetResolutionPolicy to handle the details of Resolving the CodedNodeSet
    //Here, we will set the Maximum number of Concepts returned to 10.
    SetResolutionPolicy resolvePolicy = new SetResolutionPolicy();
    resolvePolicy.setMaximumToReturn(10);

    //Do the resolve
    ResolvedConceptReferenceList rcrlist = cnsg.resolveToList(resolvePolicy);
    //resolveToList is a Grid Service Call

    //Use the returned ResolvedConceptReferenceList to print some details about the concepts found
    ResolvedConceptReference[] rcref = rcrlist.getResolvedConceptReference();
    for (int i = 0; i < rcref.length; i++) {
        System.out.println(rcref[i].getConceptCode());
        System.out.println(rcref[i].getReferencedEntry().
                getPresentation()[0].getText().getContent());

    }
```

Service Contexts and State

Along with the Main Service as described, the Server will also host the following Service Contexts. These Service Contexts are not meant to be called directly as Grid Services. The main function of these Service Contexts is to provide additional functionality to the Main Service. The following figure shows how the Service Context comprises a CodedNodeSet that have properties, such as ResolvedConceptReferenceList, Destroy, SetTermination Time, and other tasks.

Service Context Operations Example in Introduce

> (i) **Important**
>
> Service Contexts are only meant to be called through the Main Service - not directly. Through the Main Service, References to these Service Contexts can be obtained. Calls are made to the Service Contexts through these References.

Obtaining a Service Context Reference

In the figure below, two LexEVS Grid Service Calls are highlighted, 'getCodingSchemeConcepts' and 'getNodeGraph'. These two Grid Service Calls have been selected because they return to the user a "Reference" to a Service Context. For 'getCodingSchemeConcepts', the return type is CodedNodeSetReference (which references the CodedNodeSet Service Context). For 'getNodeGraph', the return type is CodedNodeGraphReference (which references the CodedNodeGraph Service Context).

class Impl

«Main Service»
LexEVSGridService

+ getCodingSchemeConcepts(CodingSchemeIdentification, CodingSchemeVersionOrTag) : CodedNodeSetReference
+ getFilter(ExtensionIdentification) : FilterReference
+ getFilterExtensions() : ExtensionDescriptionList
+ getGenericExtension(ExtensionIdentification) : LexBIGServiceConvenienceMethodsReference
+ getGenericExtensions() : ExtensionDescriptionList
+ getHistoryService(CodingSchemeIdentification) : HistoryServiceReference
+ getLastUpdateTime() : Date
+ getMatchAlgorithms() : ModuleDescriptionList
+ getNodeGraph(CodingSchemeIdentification, CodingSchemeVersionOrTag, RelationContainerIdentification) : CodedNodeGraphReference
+ getServiceMetadata() : LexBIGServiceMetadataReference
+ getSortAlgorithm(ExtensionIdentification) : SortReference
+ getSortAlgorithms(SortContext) : SortDescriptionList
+ getSupportedCodingSchemes() : CodingSchemeRenderingList
+ resolveCodingScheme(CodingSchemeIdentification, CodingSchemeVersionOrTag) : CodingScheme
+ resolveCodingSchemeCopyright(CodingSchemeIdentification, CodingSchemeVersionOrTag) : CodingSchemeCopyRight

Cloneable

«Service Context»
CodedNodeSet

+ difference(CodedNodeSet) : CodedNodeSet
+ intersect(CodedNodeSet) : CodedNodeSet
+ isCodeInSet(ConceptReference) : CodeExistence
+ resolve(SetResolutionPolicy) : ResolvedConceptReferencesIterator
+ resolveToList(SetResolutionPolicy) : ResolvedConceptReferenceList
+ restrictToCodes(ConceptReferenceList) : CodedNodeSet
+ restrictToMatchingDesignations(MatchCriteria, SearchDesignationOption, ExtensionIdentification, LanguageIdentification) : CodedNodeSet
+ restrictToMatchingProperties(LocalNameList, PropertyType[], LocalNameList, LocalNameList, NameAndValueList, MatchCriteria, ExtensionIdentification, LanguageIdentification) : CodedNodeSet
+ restrictToProperties(LocalNameList, PropertyType[], LocalNameList, LocalNameList, NameAndValueList) : CodedNodeSet
+ restrictToStatus(ActiveOption, Status) : CodedNodeSet
+ union(CodedNodeSet) : CodedNodeSet

Cloneable

«Service Context»
CodedNodeGraph

+ areCodesRelated(RelationshipTypeBasedPolicy, NameAndValue) : CodeRelationship
+ intersect(CodedNodeGraph) : CodedNodeGraph
+ isCodeInGraph(ConceptReference) : CodeExistence
+ listCodeRelationships(RelationshipTypeBasedPolicy) : ConceptReferenceList
+ listCodeRelationships(RelationshipDistanceBasedPolicy) : ConceptReferenceList
+ resolveAsList(GraphResolutionPolicy) : ResolvedConceptReferenceList
+ restrictToAssociations(NameAndValueList, NameAndValueList) : CodedNodeGraph
+ restrictToCodes(CodedNodeSet) : CodedNodeGraph
+ restrictToCodeSystem(codingScheme) : CodedNodeGraph
+ restrictToDirectionalNames(NameAndValueList, NameAndValueList) : CodedNodeGraph
+ restrictToSourceCodes(CodedNodeSet) : CodedNodeGraph
+ restrictToSourceCodeSystem(CodingSchemeIdentification) : CodedNodeGraph
+ restrictToTargetCodes(CodedNodeSet) : CodedNodeGraph
+ restrictToTargetCodeSystem(CodingSchemeIdentification) : CodedNodeGraph
+ toNodeList(NodeListPolicy) : CodedNodeSet
+ union(CodedNodeGraph) : CodedNodeGraph

Resources

LexEVS Grid Services use the WS-Resource Framework (WSRF) to allow for stateful calls to the server. When a client requests a Service Context, the client is not only issued a Reference to the Service Context that was requested, but to a unique stateful Resource on the server as well. This Resource is used in the LexEVS Grid Services as a way of statefully holding objects for further use by the client.

Refer to the following sources for more information:

* How caGrid uses the WS-Resource Framework (WSRF)

```
Link provided for historical purposes http://www.cagrid.org/wiki/Metadata:WSRF
```

* How Resources are implemented in the LexEVS Grid Service

Service Context Sequence

Service Contexts API calls follow this general process:

1. The client requests a Service Context, such as CodedNodeSet.
2. The Grid Node receives the call and forwards the call to EVSAPI.
3. EVSAPI calls getCodingSchemeConcept on the local LexBIG installation.
4. The Grid Node creates a resource for the CodedNodeSetImpl object for future use.
5. The client is returned the Service Context Client for the CodedNodeSet.

1. The Client Requests a Service Context, such as a CodedNodeSet.
   CodedNodeSet cns = lbs.
   getCodingSchemeConcepts("Test Ontology", csvt);

5. The Client then is returned a Service Context Client for the CodedNodeSet. This client is associated to the Resource that was created.

2. The Grid Node recieves the call and forwards the call to EVSAPI

Resource containing a
CodedNodeSetImpl Object

4. The Grid Node then creates a Resource to store this CodedNodeSetImpl object for future use by the Client

3. EVSAPI calls getCodingSchemeConcepts on the local LexBIG installation and returns a CodedNodeSetImpl object to the Grid Node

Service Context and Resource Assignment

> ⓘ **Note**
>
> By default, these services are destroyed 5 minutes after creation.

Below is a listing of the supported Service Contexts:

1. CodedNodeSet

CodedNodeSetGrid Interface

To construct a CodedNodeSet, the user calls getCodingSchemeConcepts as described. When the user creates a CodedNodeSet through the API call getCodingSchemeConcepts, the server creates and stores the CodedNodeSet server-side as a Resource. This Resource is associated with the client and will be accessible only by the client that created it.

CodedNodeSet Call Sequence:

1. The user requests a CodedNodeSet using getCodingSchemeConcepts.

Java Code Snippet

```
LexBIGService lbs  = (LexBIGService)ApplicationServiceProvider.getApplicationServiceFromUrl(serviceUrl,
"EvsServiceInfo");
CodedNodeSet cns = lbs.getCodingSchemeConcepts(
        String codingScheme,
        org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag);
```

2. The server calls the Distributed LexBIG getCodingSchemeConcepts method, returning to the server an org.LexGrid.LexBIG.Impl. CodedNodeSetImpl (the implementation of org.LexGrid.LexBIG.LexBIGService.CodedNodeSet) object.
3. The server then creates an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.CodedNodeSet.service.globus.resource.CodedNodeSetResource. This Resource will be used to hold the instance of org.LexGrid.LexBIG.Impl.CodedNodeSetImpl, the implementation of org.LexGrid.LexBIG. LexBIGService.CodedNodeSet that was created above.
4. The server returns an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.CodedNodeSet.stubs.types.CodedNodeSetReference object to the client. This is the reference to the CodedNodeSet Service Context. This object has a direct reference to the Resource created above. The user now uses this client to make transparent Grid calls through the Service Context.
5. The client may continue to make statefull calls to the CodedNodeSetClient and the assigned Resource.
6. These restrictions are separate calls but statefully maintained on the server via the Resource.

2. CodedNodeGraph

CodedNodeGraphGrid Interface

To construct a CodedNodeGraph, the user calls getNodeGraph as described. When the user creates a CodedNodeGraph through the API call getNodeGraph, the server creates and stores the CodedNodeGraph server-side as a Resource. This Resource is associated with the client and will be accessible only by the client that created it.

CodedNodeGraph Call Sequence:

1. The user requests a CodedNodeGraph using getCodingSchemeConcepts.

Java Code Snippet

```
LexBIGService lbs  = (LexBIGService)ApplicationServiceProvider.getApplicationServiceFromUrl(serviceUrl,
"EvsServiceInfo");
CodedNodeGraph cng = lbs.getNodeGraph(
                            String codingScheme,
                            org.LexGrid.LexBIG.DataModel.Core.CodingSchemeVersionOrTag,
                            String relationsContainerName (Optional);
```

2. The server calls the Distributed LexBIG getNodeGraph method, returning to the server an org.LexGrid.LexBIG.Impl.CodedNodeGraphImpl (the implementation of org.LexGrid.LexBIG.LexBIGService.CodedNodeGraph) object.
3. The server then creates an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.CodedNodeGraph.service.globus.resource. CodedNodeGraphResource. This Resource will be used to hold the instance of org.LexGrid.LexBIG.Impl.CodedNodeGraphImpl, the implementation of org.LexGrid.LexBIG.LexBIGService.CodedNodeGraph that was created above.
4. The server returns an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.CodedNodeGraph.stubs.types.CodedNodeGraphReference object to the client. This is the reference to the CodedNodeGraph Service Context. This object has a direct reference to the Resource created above. The user now uses this client to make transparent Grid calls through the Service Context.
5. The client may continue to make statefull calls to the CodedNodeGraphClient and the assigned Resource. For example, the client may add Restrictions to the CodedNodeGraph before a Resolve:

Java Code Snippet

```
    cng.restrictToCodeSystem(org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeIdentification);
```

6. These restrictions are separate calls but statefully maintained on the server via the Resource.

3. LexBIGServiceConvenienceMethods

LexBIGServiceConvenienceMethodsGrid Interface

To construct a LexBIGServiceConvenienceMethods, the user calls getGenericExtensions as described. When the user creates a LexBIGServiceConvenienceMethods through the API call getGenericExtensions, the server creates and stores the LexBIGServiceConvenienceMethods server-side as a Resource. This Resource is associated with the client and will be accessible only by the client that created it.

LexBIGServiceConvenienceMethods Call Sequence:

1. The user requests a LexBIGServiceConvenienceMethods using getGenericExtensions.

> **Java Code Snippet**
>
> ```
>     LexBIGServiceConvenienceMethodsGrid lbscm = lbs.getGenericExtensions(org.LexGrid.LexBIG.DataModel.
> cagrid.ExtensionIdentification);
> ```

2. The server calls the Distributed LexBIG getGenericExtensions method, returning to the server an org.LexGrid.LexBIG.Impl.Extensions. GenericExtensions.LexBIGServiceConvenienceMethodsImpl (the implementation of org.LexGrid.LexBIG.Extensions.Generic. LexBIGServiceConvenienceMethods) object.
3. The server then creates an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.LexBIGServiceConvenienceMethods.service.globus.resource. LexBIGServiceConvenienceMethodsResource. This Resource will be used to hold the instance of org.LexGrid.LexBIG.Impl.Extensions. GenericExtensions.LexBIGServiceConvenienceMethodsImpl, the implementation of org.LexGrid.LexBIG.Extensions.Generic. LexBIGServiceConvenienceMethods that was created above.
4. The server returns an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServicesLexBIGServiceConvenienceMethods.stubs.types. LexBIGServiceConvenienceMethodsReference object to the client. This is the reference to the LexBIGServiceConvenienceMethods Service Context. This object has a direct reference to the Resource created above. This LexBIGServiceConvenienceMethodsClient implements org. LexGrid.LexBIG.Extensions.Generic.LexBIGServiceConvenienceMethods. The user now uses this client to make transparent Grid calls through the Service Context. Because this LexBIGServiceConvenienceMethods implements org.LexGrid.LexBIG.Extensions.Generic. LexBIGServiceConvenienceMethods, API calls will look to the user as being identical to direct LexBIG API calls.
5. The client may continue to make statefull calls to the LexBIGServiceConvenienceMethods Client and the assigned Resource.
6. These API calls are separate calls but statefully maintained on the server via the Resource.

4. LexBIGServiceMetadata

[LexBIGServiceMetadataGrid Interface](#)

To construct a LexBIGServiceMetadata, the user calls getServiceMetadata as described. When the user creates a LexBIGServiceMetadata through the API call getServiceMetadata , the server creates and stores the LexBIGServiceMetadata server-side as a Resource. This Resource is associated with the client and will be accessible only by the client that created it.

LexBIGServiceMetadata Call Sequence:

1. The user requests a LexBIGServiceMetadata using getServiceMetadata.

> **Java Code Snippet**
>
> ```
>     LexBIGServiceMetadataGrid metadata = lbs.getServiceMetadata();
> ```

2. The server calls the Distributed LexBIG getServiceMetadata method, returning to the server an implementation of org.LexGrid.LexBIG. LexBIGService.LexBIGServiceMetadata object.
3. The server then creates an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.LexBIGServiceMetadata.service.globus.resource. LexBIGServiceMetadataResource. This Resource will be used to hold the instance of an implementation of org.LexGrid.LexBIG.LexBIGService. LexBIGServiceMetadata.
4. org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.LexBIGServiceMetadata.stubs.types.LexBIGServiceMetadata object to the client. This is the reference to the LexBIGServiceMetadata Service Context. This object has a direct reference to the Resource created above. The user now uses this client to make transparent Grid calls through the Service Context.
5. The client may continue to make statefull calls to the LexBIGServiceMetadata and the assigned Resource.
6. These API calls are separate calls but statefully maintained on the server via the Resource.

5. HistoryService

[HistoryServiceGrid Interface](#)

To construct a HistoryService, the user calls getHistoryService as described. When the user creates a HistoryService through the API call getHistoryService, the server creates and stores the HistoryService server-side as a Resource. This Resource is associated with the client and will be accessible only by the client that created it.

HistoryService Call Sequence:

1. The user requests a HistoryService using getHistoryService .

> **Java Code Snippet**
>
> ```
>     HistoryServiceGrid history = lbs.getHistoryService(org.LexGrid.LexBIG.DataModel.cagrid.
> CodingSchemeIdentification);
> ```

2. The server calls the Distributed LexBIG getHistoryService method, returning to the server an implementation of org.LexGrid.LexBIG.History. HistoryService object.

3. The server then creates an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.HistoryService.service.globus.resource.HistoryServiceResource. This Resource will be used to hold the instance of an implementation of org.LexGrid.LexBIG.History.HistoryService.
4. The server returns an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.LexBIGServiceMetadata.stubs.types.LexBIGServiceMetadata object to the client. This is the reference to the HistoryService Service Context. This object has a direct reference to the Resource created above. The user now uses this client to make transparent Grid calls through the Service Context.
5. The client may continue to make statefull calls to the HistoryServiceClient and the assigned Resource. For example, the client may call any method in org.LexGrid.LexBIG.History.HistoryService
Example: history.getLatestBaseline();
6. These API calls are separate calls but statefully maintained on the server via the Resource.

6. Sort

Sort Interface

To construct a Sort, the user calls getSortAlgorithm as described. When the user creates a Sort through the API call getSortAlgorithm, the server creates and stores the Sort server-side as a Resource. This Resource is associated with the client and will be accessible only by the client that created it.

Sort Call Sequence:

1. The user requests a Sort using getSortAlgorithm .

   Java Code Snippet

   ```
   Sort sort = lbs.getSortAlgorithm(org.LexGrid.LexBIG.DataModel.cagrid.ExtensionIdentification);
   ```

2. The server calls the Distributed LexBIG getSortAlgorithm method, returning to the server an implementation of org.LexGrid.LexBIG.Extensions.Query.Sort) object.
3. The server then creates an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.Sort .service.globus.resource.Sort Resource. This Resource will be used to hold the instance of an implementation of org.LexGrid.LexBIG.Extensions.Query.Sort.
4. The server returns an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.service.SortClient object to the client. This is the client to the Sort Service Context. This object has a direct reference to the Resource created above. This SortClient implements org.LexGrid.LexBIG.Extensions.Query.Sort. The user now uses this client to make transparent Grid calls through the Service Context. Because this Sort implements org.LexGrid.LexBIG.Extensions.Query.Sort, API calls will look to the user as being identical to direct LexBIG API calls.
5. The client may continue to make statefull calls to the SortClient and the assigned Resource. For example, the client may call any method in

   Java Code Snippet

   ```
   sort.compare(codedNodeReference1, codedNodeReference2);
   ```

6. These API calls are separate calls but statefully maintained on the server via the Resource.

7. Filter

Filter Interface

To construct a Filter, the user calls getFilter as described. When the user creates a Filter through the API call getFilter, the server creates and stores the Sort server-side as a Resource. This Resource is associated with the client and will be accessible only by the client that created it.

Filter Call Sequence:

1. The user requests a Filter using getFilter

   Java Code Snippet

   ```
   Filter filter = lbs.getFilter(org.LexGrid.LexBIG.DataModel.cagrid.ExtensionIdentification);
   ```

2. The server calls the Distributed LexBIG getFilter method, returning to the server an implementation of org.LexGrid.LexBIG.Extensions.Query.Filter) object.
3. The server then creates an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.Filter.service.globus.resource.FilterResource. This Resource will be used to hold the instance of an implementation of org.LexGrid.LexBIG.Extensions.Query.Filter.
4. The server returns an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.service.FilterClient object to the client. This is the client to the Filter Service Context. This object has a direct reference to the Resource created above. This FilterClient implements org.LexGrid.LexBIG.Extensions.Query.Filter. The user now uses this client to make transparent Grid calls through the Service Context. Because this Filter implements org.LexGrid.LexBIG.Extensions.Query.Filter, API calls will look to the user as being identical to direct LexBIG API calls.
5. The client may continue to make statefull calls to the FilterClient and the assigned Resource. For example, the client may call any method in org.LexGrid.LexBIG.Extensions.Query.Filter

   Java Code Snippet

```
        filter.match(resolvedConceptReference);
```

6. These API calls are separate calls but statefully maintained on the server via the Resource.

8. ResolvedConceptReferencesIterator

ResolvedConceptReferencesIterator Interface

A ResolvedConceptReferencesIterator is created when a CodedNodeSet or CodedNodeGraph is resolved. It allows results to be returned from the server incrementally instead of all at once. When the user creates a ResolvedConceptReferencesIterator, the server creates and stores the ResolvedConceptReferencesIterator server-side as a Resource. This Resource is associated with the client and will be accessible only by the client that created it.

ResolvedConceptReferencesIterator Call Sequence:

1. The user gets a ResolvedConceptReferencesIterator from a Resolve.
2. The server calls the Distributed LexBIG resolve method on the CodedNodeSet, returning to the server an implementation of org.LexGrid.LexBIG. Utility.Iterators.ResolvedConceptReferencesIterator object.
3. The server then creates an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.ResolvedConceptReferencesIterator.service.globus.resource. ResolvedConceptReferencesIteratorResource. This Resource will be used to hold the instance of an implementation of org.LexGrid.LexBIG.Utility. Iterators.ResolvedConceptReferencesIterator.
4. The server returns an org.LexGrid.LexBIG.cagrid.LexBIGCaGridServices.service.ResolvedConceptReferencesIteratorClient object to the client. This is the client to the ResolvedConceptReferencesIterator Service Context. This object has a direct reference to the Resource created above. This ResolvedConceptReferencesIteratorClient implements org.LexGrid.LexBIG.Utility.Iterators.ResolvedConceptReferencesIterator. The user now uses this client to make transparent Grid calls through the Service Context. Because this ResolvedConceptReferencesIterator implements org.LexGrid.LexBIG.Utility.Iterators.ResolvedConceptReferencesIterator, API calls will look to the user as being identical to direct LexEVS API calls.
5. The client may continue to make statefull calls to the ResolvedConceptReferencesIteratorClient and the assigned Resource. For example, the client may call any method in org.LexGrid.LexBIG.Utility.Iterators.ResolvedConceptReferencesIterator

Java Code Snippet

```
    while(itr.hasNext){
        ResolvedConceptReference ref = itr.next();
    }
```

6. These API calls are separate calls but statefully maintained on the server via the Resource.

Error Handling

Error Connecting to LexEVS Grid Service

When connecting through the Java Client, java.net.ConnectException and org.apache.axis.types.URI.MalformedURIException may be thrown upon an unsuccessful attempt to connect.

A MalformedURIException is thrown in the case if a poorly-formed URL string. In this case, the exception is thrown before an attempt to connect is even made.

If the URL is well-formed, proper connection is tested. If the connection attempt fails, a ConnectException is thrown containing the reason for the failure.

Java Code

```
    try{
        LexBIGServiceGridAdapter lbsg = new LexBIGServiceGridAdapter
            ("http://localhost:8080/wsrf/services/cagrid/LexEVSGridService");
    } catch(java.net.ConnectException e){
        //Error Connecting
        e.printStackTrace();
    } catch(org.apache.axis.types.URI.MalformedURIException e){
        //URL Syntax Error
        e.printStackTrace();
    }
```

This example shows a typical connection to the LexEVS Grid Service, with the two potential Exceptions being caught and handled as necessary.

LexBIG Errors

LexBIG errors will be forwarded through the Distributed LexEVS layer and then on to the Grid layer. Input parameters, along with any other LexBIG (or Distributed LexBIG) errors will be detected on the server, not the client, and forwarded. All Generic LexEVS (or Distributed LexEVS) errors will be forwarded via a RemoteException, with the cause of the error and underlying LexEVS error message included.

Invalid Service Context Access

Service Context Services are not meant to be called directly. If the client attempts to do so, an `org.LexGrid.LexBIG.cagrid.LexEVSGridService.CodedNodeSet.stubs.types.InvalidServiceContextAccess Exception` will be thrown. This indicates a call was made to a Service Context without obtaining a Service Context Reference via the Main Service (see the above section Service Contexts and State for more information).

Client

The Introduce toolkit generates a "client" class that will be provided to the users.

Security Issues

Security in the LexEVS Grid Service is implemented in the Distributed LexBIG layer. The information in this section explains how the LexEVS Grid Services utilize this security implementation. For more information, see Distributed LexBIG Security Implementation: LexBIG Access to Licensed Vocabulary Implementation, attached to the [EVS API GForge documents archive](#).

LexEVS Grid Service Security

Certain vocabulary content accessible through the LexEVS Grid Service may require extra authorization to access. Each client is required to supply its own access credentials via Security Tokens. These Security Tokens are implemented by a SecurityToken object:
Name: SecurityToken
Namespace: gme://caCORE.caCORE/3.2/gov.nih.nci.evs.security
Package: gov.nih.nci.evs.security

Accessing Secure Content

A client establishes access to a secured vocabulary via the following Grid Service Calls:

1. *Step 1:*Connect to the LexBIG caGrid Service

    ```
    LexBIGServiceGrid lbs = new LexBIGServiceGridAdapter(url);
    ```

2. *Step 3:*Build an org.LexGrid.LexBIG.DataModel.cagrid.CodingSchemeIdentification to hold the Coding Scheme name.

    ```
    CodingSchemeIdentification codingScheme = new CodingSchemeIdentification();
    codingScheme.setName("codingScheme");
    ```

3. *Step 4:*Build an gov.nih.nci.evs.security.SecurityToken containing the security information for the desired Coding Scheme.

    ```
    SecurityToken token = new SecurityToken();
    token.setAccessToken("securityToken");
    ```

4. *Step 5:*Invoke the LexBIG caGrid service as follows: This will return a reference to a new "LexBIGServiceGrid" instance that is associated with the security properties that were passed in.

    ```
    LexBIGServiceGrid lbsg = lbs.setSecurityToken(codingScheme, token);
    ```

    It is important to note that the Grid Service "setSecurityToken" returns an *org.LexGrid.LexBIG.cagrid.LexEVSGridService.stubs.types .LexEVSGridServiceReference.LexEVSGridServiceReference*
    object. This reference must be used to access the secured vocabularies.

Implementation

Each call to "setSecurityToken" sets up a secured connection to Distributed LexBIG with the access privileges included in the SecurityToken parameter. The *LexEVSGridServiceReference* that is returned to the client contains a unique key identifier to the secure connection that has been created on the server. All subsequent calls the client makes through this *LexEVSGridServiceReference* will be made securely. If additional SecurityTokens are passed in through the "setSecurityToken" Grid Service, the additional security will be added and maintained.

The "setSecurityToken" Grid Service is a stateful service. This means that after the client sets a SecurityToken, any subsequent call will be applied to that SecurityToken.

Secure connections are not maintained on the server indefinitely, but are based on load conditions. The server will allow 30 unique secure connections to be set up for clients without any time limitations. As additional requests for secure connections are received by the server, connections will be released by the server on an 'oldest first' basis. No connection, however, may be released prior to 5 minutes after its creation.

ⓘ

If no SecurityTokens are passed in by the client, a non-secure Distributed LexBIG connection will be used. The server maintains one (and only one) un-secured Distributed LexBIG connection that is shared by any client not requesting security.

> ⓘ **Note**
>
> All non-secured information accessed by the LexEVS Grid Service is publicly available from NCICB and users are expected to follow the licensing requirements currently in place for accessing and using NCI EVS information.

Performance

The LexEVS service will take advantage of all improvements made to the LexEVS API services with the exception of lazy loading. LexEVS grid service, being in nature a web service is currently not taking advantage of lazy loading since objects are transferred as fully populated objects. However, future releases of LexEVS Grid Service may refractor the interface in such as way as to take advantage of some of the benefits brought about by the inclusion of lazy loading in to LexEVS API service.

LexEVS Grid Services utilize the performance enhancements of the LexBIG API.
For more information about LexBIG performance (which LexEVS Grid Services are dependent on), see Mayo Clinic Informatics.

Installation and Packaging

The service will be installed and deployed as a "stand alone" service at NCICB.

Migration

Both the current version of LexEVS grid service may be "in service" simultaneously if the corresponding underlying EVSAPI service is also "in service" to manage migration of clients.

System Testing

LexEVS Grid Service Testing Documentation (LexEVS Grid Service System Testing in the Project Documents, Development Documents section)

DOCUMENT APPROVAL

Approvers List

The individuals listed in this section constitute the approvers list for the Integration Test Plan document. Formal approval must be received from all approvers prior to the initiation of the next steps in the process.

TITLE	NAME
Project Manager	Name
Development Manager	Name

Reviewers List

The individuals listed in this section constitute the reviewers list for the Master Test Plan document. Formal approval is not required from the reviewers, however, it is desirable to have all reviewers review and comment on the document. Reviewers may choose to concentrate on reviewing only those sections that are in their area of responsibility, rather than the entire document.

TITLE	NAME
Technical Writer	Name

LexEVS Loader Source Mapping

This section is now moved to separate pages to provide details on how different formats are loaded into the LexEVS model.

- For LexEVS v5.0, see LexEVS 5.0 Loader Source Mapping.
- For LexEVS v5.1, which includes enhancements to the loader framework, see LexEVS 5.1 Loader Source Mapping.