LexEVS 6.x Architecture

Contents of this Page

LexEVS 6.x Design Links to Include

- Design Guide Main Page
  - LexEVS Information Models
  - LexEVS Architecture
- LexEVS 6.0 Main Page
- LexEVS Current Release

LexEVS Architecture Overview

LexEVS software architecture and implementation is designed to facilitate flexibility and future expansion in the research community. The purpose of LexEVS is to enable individual Cancer Centers and others to use the provided EVS services and if desired, install local instances of vocabularies.

History and Definition

What is LexGrid?

LexGrid is the model used to store terminologies. 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 (e.g., relational database, LDAP) 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 including:

- Open Biomedical Ontologies (OBO)
- Web Ontology Language (OWL), e.g., NCI Thesaurus
- Unified Medical Language System (UMLS) Rich Release Format (RRF), e.g., NCI Metathesaurus

This common model is a critical component of the LexGrid project. Once disparate vocabulary information can be represented in a standardized model, it becomes possible to build common repositories to store vocabulary content and common programming interfaces and tools to access and manipulate that content. The HL7/OMG Common Terminology Services (CTS2) and LexBIG API are two examples of APIs able to query information stored in the LexGrid Model.

What is LexBIG?

LexBIG is the set of services that EVS adapters use to store/retrieve terminology metadata. LexBIG is a more specific project that applied LexGrid vision and technologies to requirements of the research 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 NCI 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).

What is LexEVS?

LexEVS combines LexBIG and the EVS adapters into one set of services. LexEVS is a collection of programmable interfaces that provides developers with the ability to access any installation of the LexEVS terminology server. The controlled terminologies hosted by the NCI EVS Project are published via the Open-Source LexEVS Terminology Server. It is a caCORE Software Development Kit (SDK) generated system. The caCORE SDK is a set of tools that can be used by an intermediate Java developer to create a caCORE-like system.

Such systems are constructed using certain design principles:

- UML Modeling
- N-tier architecture with open API's
- Controlled vocabularies
- Registered metadata

LexEVS has a number of API mechanisms for use with various technologies. In addition, LexEVS provides developers GUIs for administration and testing of the terminology server. These GUIs are intended only for developers.

LexEVS 6.x Architecture Overview

The LexEVS 6.x infrastructure exhibits an n-tiered architecture with client interfaces, server components, domain objects, data sources, and back-end systems. This n-tiered system divides tasks or requests among different servers and data stores. This isolates the client from the details of where and how data is retrieved from different data stores.

The system also performs common tasks such as logging and provides a level of security for protected content. Clients (browsers, applications) receive information through designated application programming interfaces (APIs). Java applications communicate with back-end objects via domain objects packaged within the client.jar. Non-Java applications can communicate via SOAP (Simple Object Access Protocol) or REST (Representational State Transfer) services.

Most of the LexEVS API infrastructure is written in the Java programming language and leverages reusable, third-party components. The service infrastructure is composed of the following layers:

- **Application Service Layer** - accepts incoming requests from all public interfaces and translates them, as required, to Java calls in terms of the native LexEVS API. Queries are invoked against the Distributed LexEVS API, which acts as proxy to invoke the equivalent function against the LexEVS core Java API.

- **Core API Layer** - Local Java API implementation that underpins all LexEVS API requests. Search of pre-populated Lucene index files is used to evaluate query results before incurring cost of database access. Access to the LexGrid database is performed as required to populate returned objects using pooled connections.

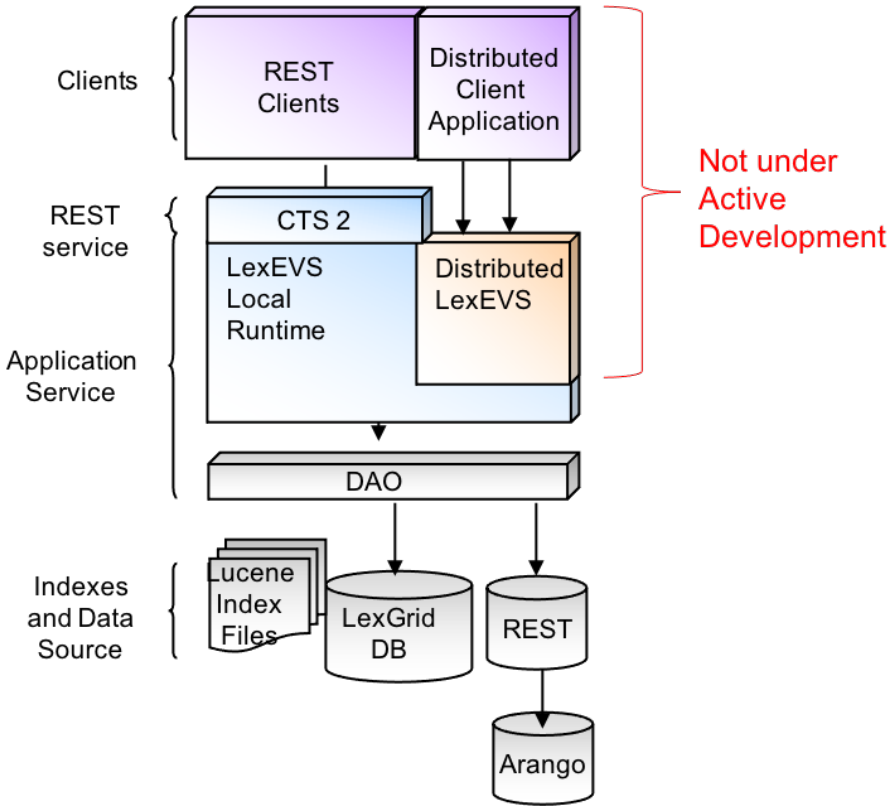
- **Data Source Layer** - is responsible for storage and access to all data required to represent the objects returned through API invocation.

LexEVS 6.x High-level Design Diagram

The figure that follows shows the following components of the Architecture Diagram:

- Clients: Local, Distributed, and caGrid clients, and SOAP, REST, QBE Clients.
- Application Service: CTS2 (LexEVS Local Runtime), Distributed LesEVS, LexEVS caCORE API's.
- Core API: LesEVS Model Objects Extended for CTS 2 = > DAO
- Indexes and Data Source: Lucene Index Files and LexGrid DB
- Optional Arangodb support for high speed graph resolution of terminology hierarchy and relationship graphs

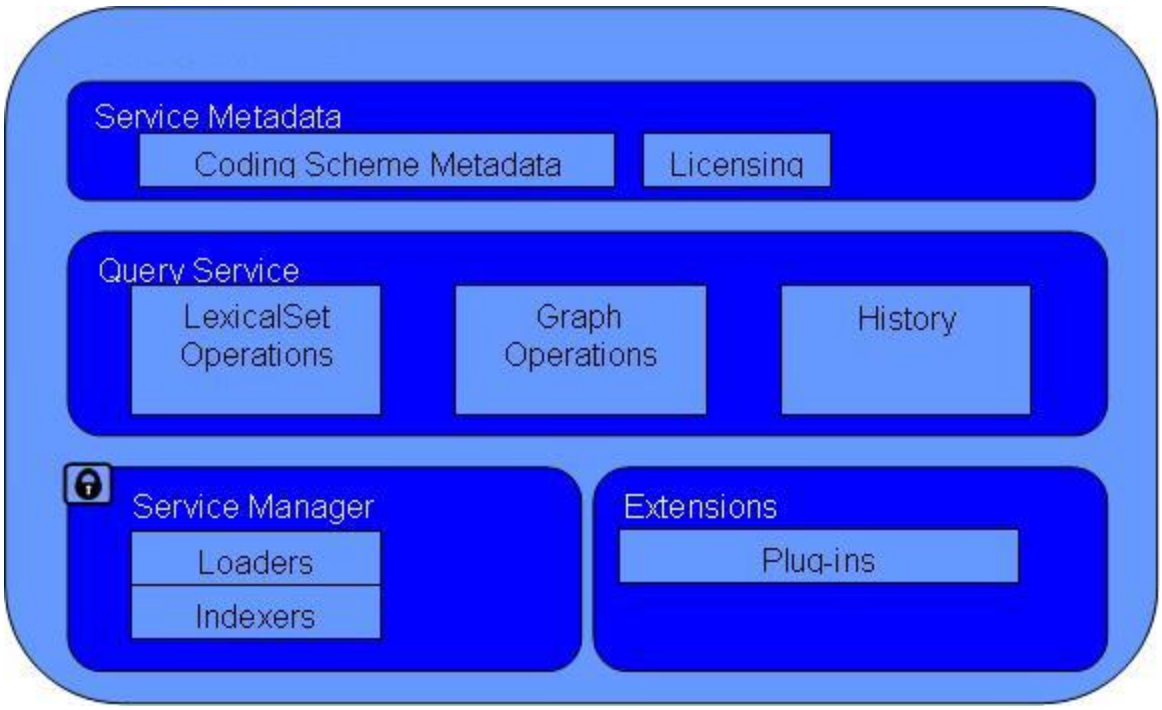
LexBIG Architecture



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 Services diagram.



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.

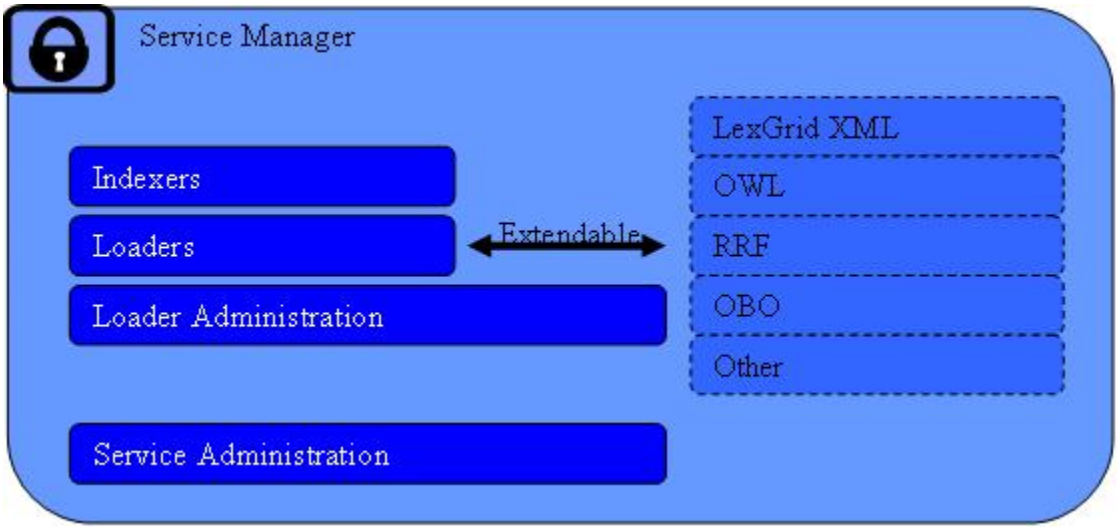
- The **Service Manager** provides administration control for loading a vocabulary and activating a service. The Service Manager comprises the Indexers and Loaders.
- The **Service Metadata** provides external clients with information about the vocabulary content (e.g. NCI Thesaurus) and appropriate licensing information. The Service Metadata comprises the Coding Scheme Metadata and Licensing.
- The **Query Operations** provide numerous functions for querying and traversing vocabulary content. The Query Service comprises the LexicalSet Operations, Graph Operations, and History.
- 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. The Extensions comprises of Plug-ins. 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.

Service Management Subsystem

The following figure shows a diagram Service Management Subsystem.



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.
  - Asserted Value Set Indexing – Indexes value sets separately from the source terminology
- **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.
  - Asserted Value Set Definition Loads – References a database load of an Asserted Value Set Source terminology to interpret and load value set definitions to the data model based LexGrid schema.

Metadata and Discovery Subsystem

The following figure shows the Metadata and Discovery Subsystem diagram.

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

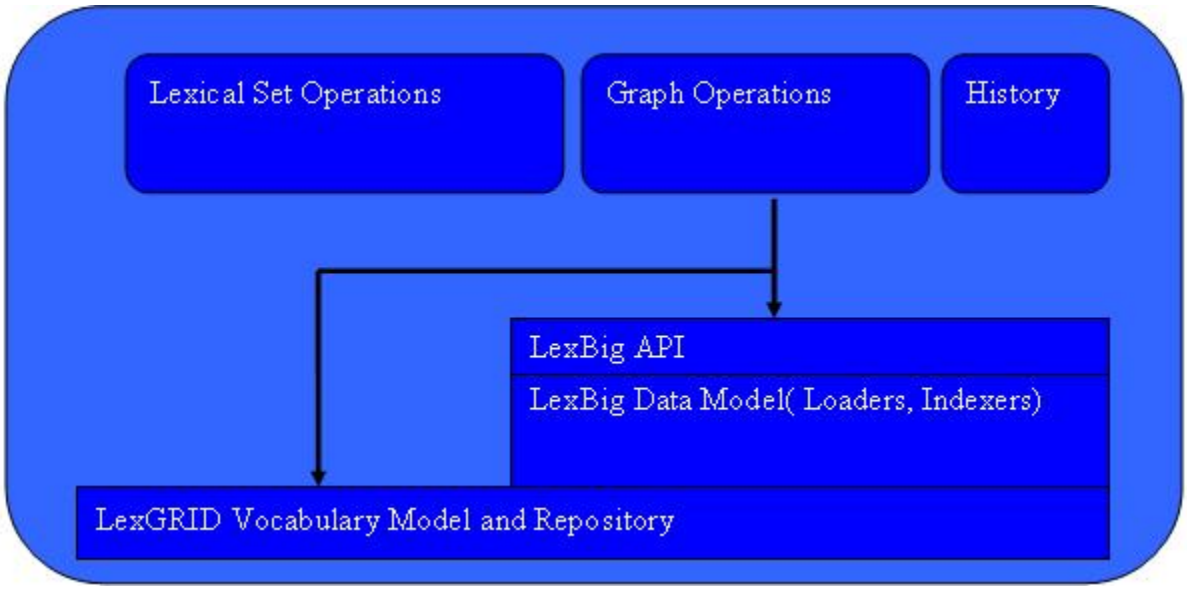
The ability to locate and resolve vocabulary metadata is fulfilled through the LexBIGService class. Metadata defined by the LexGrid information model is resolved with each CodingScheme instance. Available metadata on each resolved scheme includes, but is not necessarily limited to, the following:

- 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.

Query Subsystem

The following figure shows the Query Subsystem.



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 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 related 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 CTS 2 Services

Overview

The CTS2 standard is actually a collection of relatively small standards and a set of assembly rules.

The goal of the CTS2 standard is distribution and federation:

-Distribution means that different organizations can publish different aspects
-Federation means that organizations can share

Key is that implementations can be shared – No one organization has to offer all services

Standards

XML Schema

- used for REST, SOAP and JSON formatting
- Includes item / directory / list

Functions

- Read
- Query
- Update
- Import and Export content

Component Standards

- Code System
- Code System Version
- Entity Description
- Association
- Value Set
- Value Set Definition
- Resolved Value Set
- Map
- Map Version
- Map Entry
- Concept Domain
- Concept Domain Binding
- Statement

Assembly Rules

- Each component can be implemented by itself or in conjunction with other components
- Components reference other components via:
- URI's – if the referenced component is not known to the service
- HREF's (optional) – if the referenced component IS known to the service

LexEVS CTS2 Documentation

LexEVS 6.x REST CTS2 Service