LexEVS 5.x Design and Architecture Guide 5 - LexEVS Architecture

Contents of this Page

Introduction

This document is a section of the Design and Architecture Guide.

LexEVS Architecture Overview

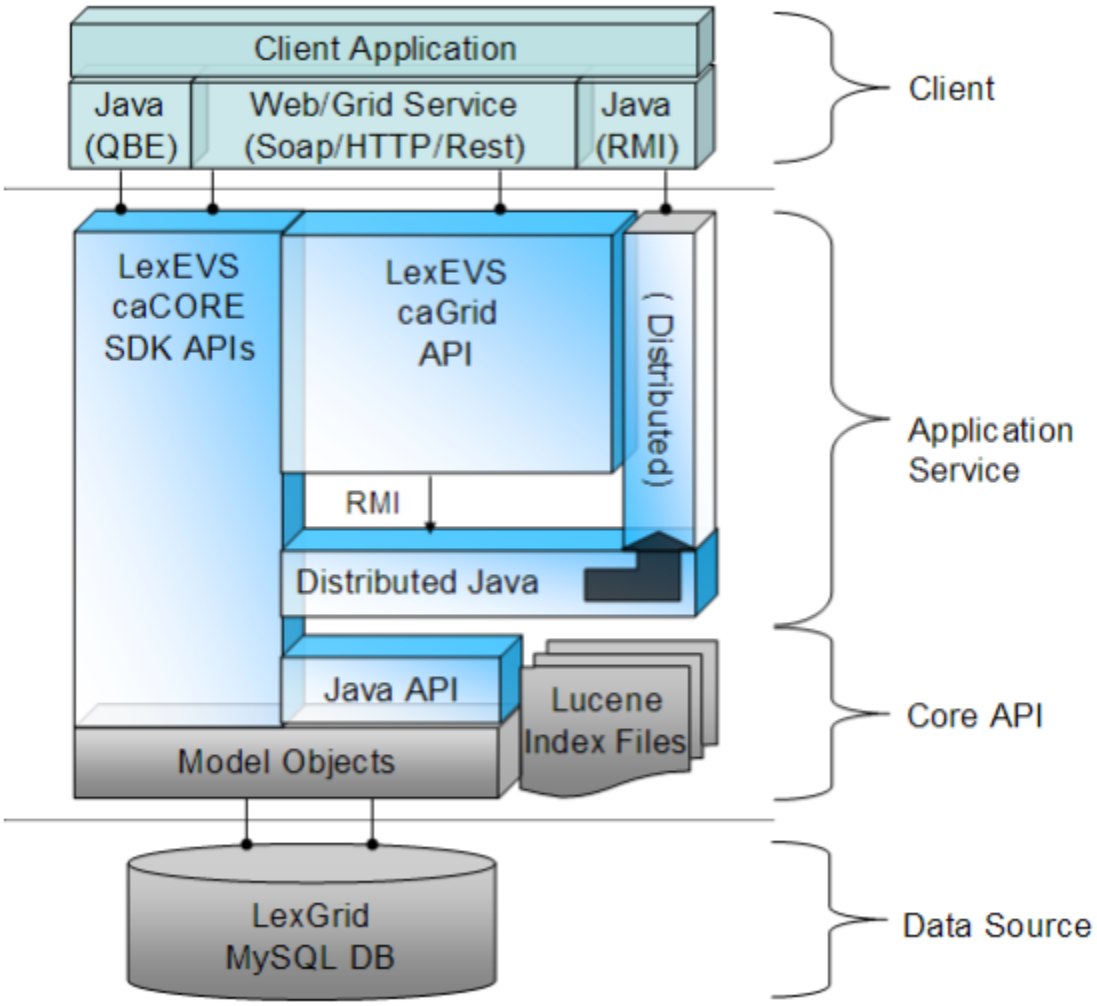
The LexEVS v5.1 infrastructure exhibits an n-tiered architecture with client interfaces, server components, domain objects, data sources, and back-end systems (illustrated below). This n-tiered architecture divides tasks or requests among different servers and data stores; it isolates the client from the details of where and how data is retrieved from different data stores.

LexEVS 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. Non-SDK queries are invoked against the Distributed LexEVS API, which handles client authentication and acts as proxy to invoke the equivalent function against the LexEVS core Java API. The caGrid and SDK-generated services are optionally run in an application server separate from the Distributed LexEVS API.
  The LexEVS caCORE SDK services work directly against the database, via Hibernate bindings, to resolve stored objects without intermediate translation of calls in terms of the LexEVS API. However, the LexEVS SDK services do still require access to metadata and security information stored by the Distributed and Core LexEVS API environment to resolve the specific database location for requested objects and to verify access to protected resources, respectively.
  From the client perspective, the LexEVS services function as "ports" accessible through the caGrid 1.3 service architectural model. LexEVS services follow the caGrid architecture for analytical and data services. See the caGrid 1.3 documentation for architectural details: https://cabig.nci.nih.gov/workspaces/Architecture/caGrid/
- **Core API layer** 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.

High-level Design Diagram



Cross-product Dependencies

No new dependencies since LexEVS v5.0. See the Core Product Dependency Matrix.

Changes in Technology

No changes in the technology stack since LexEVS v5.0.

Assumptions

None.

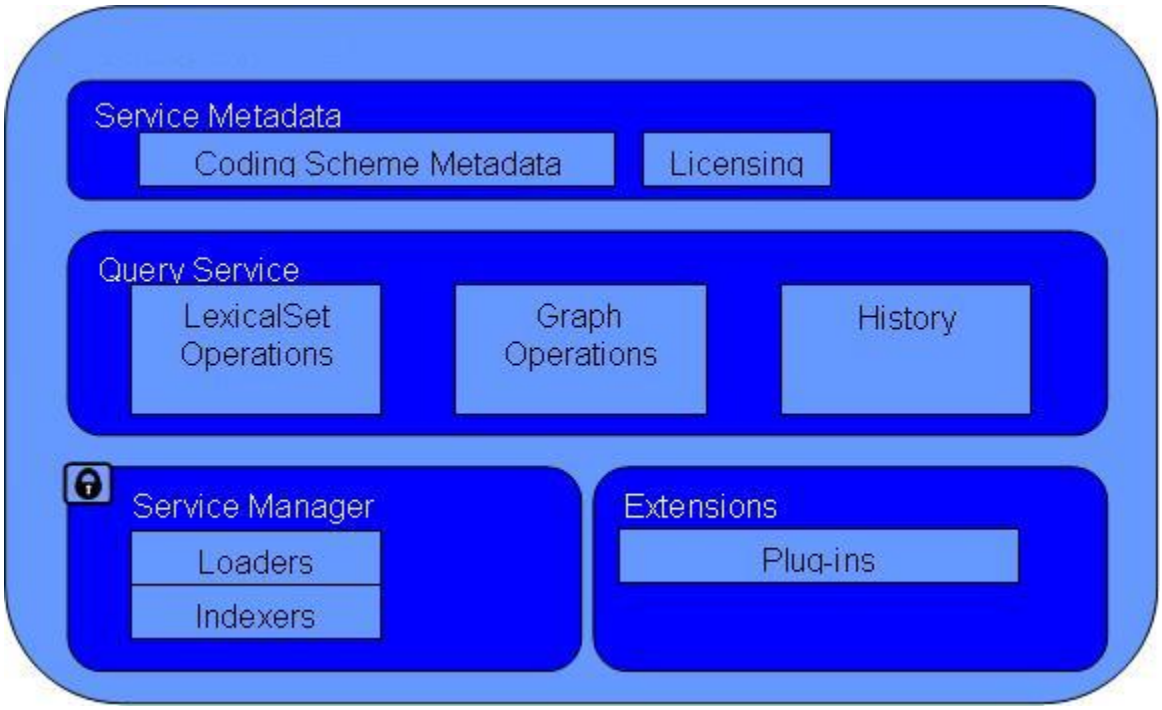
Risks

Due to the additional content to load from RRF files in LexEVS v5.1, there is a risk of increased loading times and increased storage requirements. However, the new loader framework should mitigate the increased loading times: it provides a faster load while increasing the capacity for content to be loaded.

LexBIG

LexBIG Services

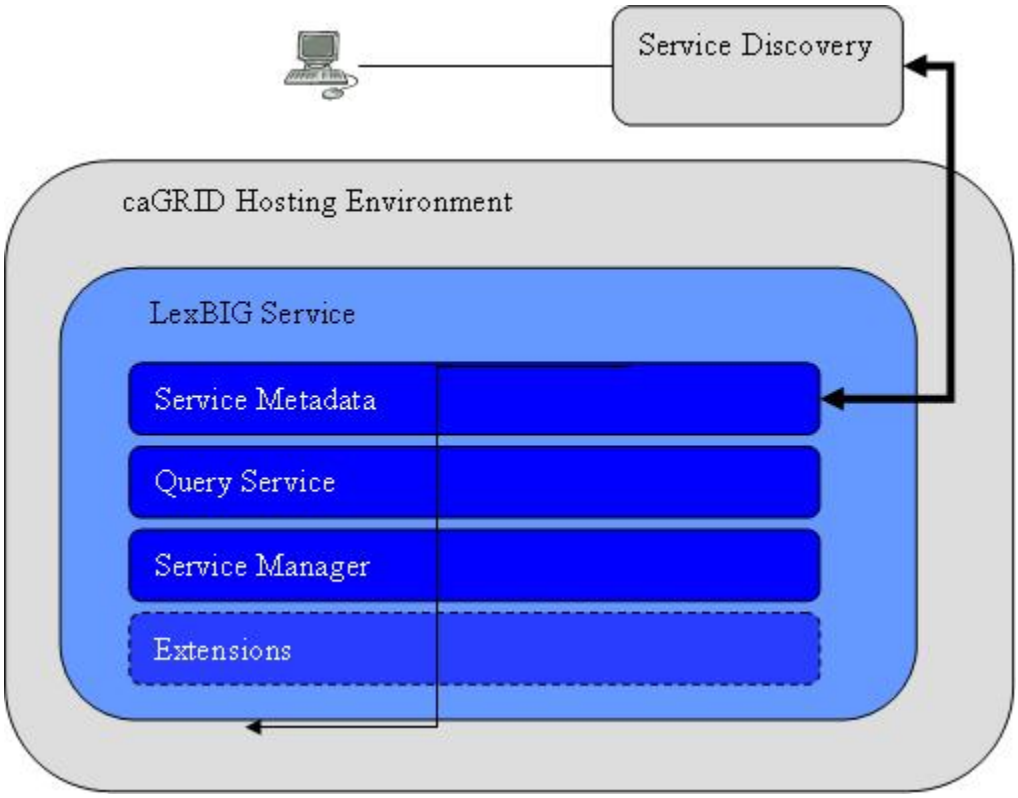
This section describes architectural detail for services provided by the LexBIG system. These services are geared toward the administration, management, and serving of vocabularies defined to the LexGrid/LexBIG information model. A system overview is provided, followed by a description of key subsystems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



The 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.

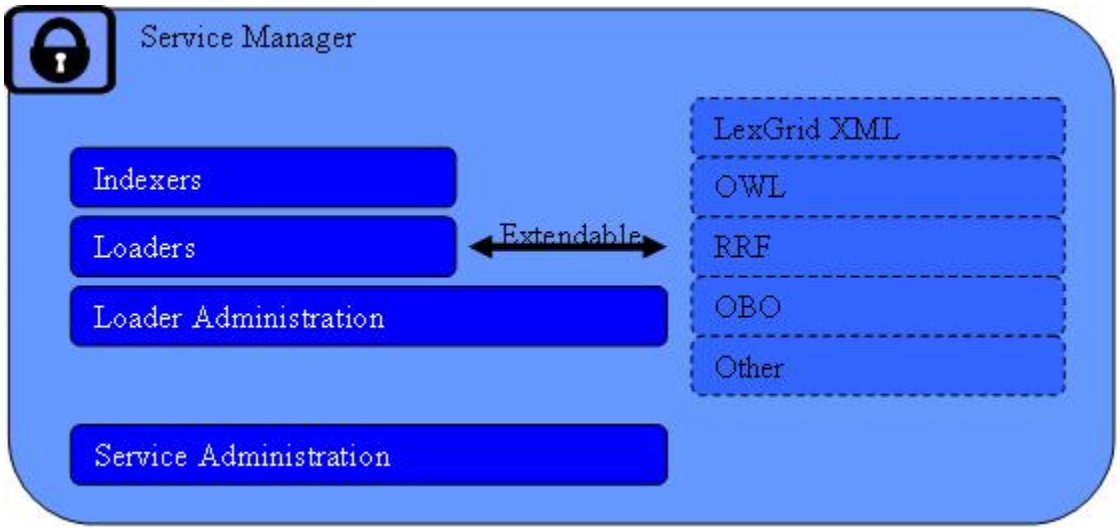
caGRID Hosting



The 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.

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



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

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

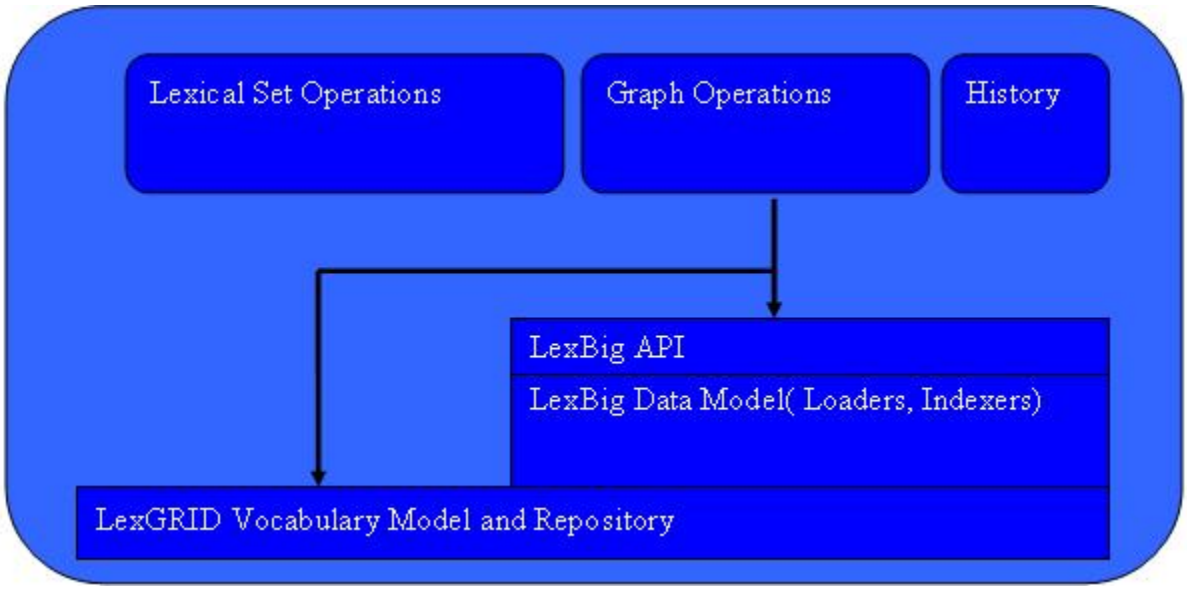
The ability to locate and resolve vocabulary metadata is fulfilled through the LexBIGService class. Metadata defined by the LexGrid information model is resolved with each CodingScheme instance. Available metadata on each resolved scheme includes, but is not necessarily limited to, the following:

- 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



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

See LexEVS 4.2 Grid Service Design and Implementation.