LexEVS 5.1 Loader Feasibility Doc - Temp

Contents of this Page

Feasibility Report and Recommendation

Persistence Layer Feasibility

The Persistence Layer enables LexEVS to have a single access point to the underlying database. This has several advantages:

1. The DAO is implemented as an Interface, not a concrete class. We can implement this interface with Hibernate, JDBC, Ibatis, or any other Persistence tool or framework.
2. All loaders can now share a single entry point to the database, and are not limited by memory constraints as some of the EMF persistence was.
3. Connection Pooling and management is abstracted from the code and pluggable. Data source implementations may be switched and Connection Pooling may be configured without recompiling code.
4. Transactions may be defined programmatically via AOP interceptors.

As LexEVS moves forward, the Persistence Layer is also flexible enough to play a part in the runtime Query API. With this, the runtime and loader code would be able to share a common Data Access Layer - we would then have a true DAO Layer.

Loader Framework Feasibiity

The Loader Framework has been implemented for two loaders, the UMLS single ontology loader and the NCI Metathesaurus loader. These loaders that implement the Loader Framework simple must define the READ and TRANSFORMATION mechanisms for the load, as well as load order and flow. All common details of Loading to LexEVS will be dealt with by the Loader Framework and will not have to be implemented. Tools exist for:

1. Lucene Indexing
2. Registering CodingSchemes
3. Changing CodingScheme status (to ACTIVE, INACTIVE, etc)
4. Building the Transitivity Closure table
5. Adding Supported Attributes
6. Detecting Database type
7. Staging temporary data to the database
8. Restarting failed loads
9. Integrating with LexEVS logging
10. Detecting and handling Root Nodes
11. ...and more common LexEVS load related tasks

Also, to aid in Transformation, basic building blocks have been created that users may extend, such as:

1. Processors for all of LexEVS Model Objects
2. Various List Processors
3. Grouping Processors
4. Auto-Supported Attribute adding Processors
5. Several basic Resolvers to extract LexEVS Specific data from the source
6. ... various other Processors for specialized tasks

Several Utilities are also available for Reading and Writing, such as:

1. Group Readers
2. Group Writers
3. Writers configurable to skip certain records
4. Partitionable readers to break up large source files
5. Error checking Readers and Writers
6. A Validating framework for inspecting content before it is inserted into the database.
7. etc.