The UMLS Knowledge Source Server: An Object Model For Delivering UMLS Data

Anantha Bangalore, Karen E. Thorn, Carolyn Tilley, Lee Peters
US National Library of Medicine, Bethesda, Maryland

The Unified Medical Language System® (UMLS ®), a project of the National Library of Medicine (NLM), regularly distributes a set of knowledge sources to the research community. These data are made available over the Internet through the UMLS Knowledge Source Server (UMLSKS). The new version of the UMLSKS is a complete redesign of the original system using Java and the Extensible Markup Language (XML) technologies to implement a fast, reliable, flexible, and extensible UMLS data retrieval system that includes an Application Programmer's Interface (API) and an Object Model of each of the Knowledge Sources: the UMLS Metathesaurus, the Semantic Network, and the SPECIALIST Lexicon. In this paper we present the design of the new system, outline each of the system design goals, the UMLS Object Model, and statistics showing the usage of the new UMLSKS and associated data. We conclude with implications for future work.

INTRODUCTION

The Unified Medical Language System® (UMLS®) approach involves the development of a set of widely distributed Knowledge Sources (Metathesaurus®, Semantic Network, and SPECIALIST Lexicon). These Knowledge Sources can be used by a variety of computerized applications to compensate for differences in the way concepts are expressed in a variety of biomedical vocabularies [1]. Currently, over 1900 individuals and institutions have signed the UMLS License Agreement, enabling them to receive the UMLS data either on CD-ROM or through the UMLS Knowledge Source Server (UMLSKS). A smaller number of licensees (approximately 1200) have registered for access to the UMLSKS.

The UMLS is large and complex and presents significant challenges in retrieving information in a comprehensive way. The centrally managed UMLSKS provides system developers with UMLS information remotely and on demand. The advantage of such an approach is that it makes the Knowledge Sources readily available and perhaps more importantly, developers do not need to invest time and effort in understanding the structure of the data files and other details to use the UMLS data in their applications. In 1995, the UMLS data were made available for the first time through the Internet-based UMLSKS [2]. Since then there have been significant improvements to the software and hardware components of the UMLSKS resulting in enhanced performance, increased flexibility, extensibility, and scalability, and better software developer access to UMLS data.

Functionally, the UMLSKS is similar to previous versions in facilitating remote site users, individuals as well as computer programs, to send requests to a server at the National Library of Medicine (NLM) through multiple channels. The similarity ends there. The old system ran as a single server using a flag-based command line Application Programmer's Interface (API) that was written in the "C" programming language. The new Java-based system was designed with the following tenets in mind:

- Extensibility for ease of new feature integration
- Flexibility by providing a rich API set to allow system developers access to all UMLS data elements
- Access to data through multiple channels (web, XML/socket API, and Java API)
- Provision of a unified data model for the Knowledge Sources for use by application developers
- Scalability in handling ever increasing user loads and increasing numbers of UMLS source vocabularies
- Performance enhancement to provide faster access to UMLS data
- Ease of administration by NLM staff and contractors

The UMLSKS Object Model for each of the Knowledge Sources allows users to ingest XML documents produced by the UMLSKS and to manipulate those data in an object-oriented fashion within their own programs. The load on the new system is spread across multiple machines to achieve load balance and fault tolerance.

UMLSKS API

The API provides a number of functions for querying UMLS Knowledge Source information from the UMLSKS. Two programming interfaces are available

to developers wishing to use the UMLSKS to retrieve UMLS data content – a Java Remote Method Invocation (RMI)-based mechanism and a TCP/IP socket-based mechanism. The first scheme utilizes the Java RMI package to establish a connection to the UMLSKS that allows client applications to make method calls from directly within their Java programs. The underlying communications mechanism is hidden and frees the user from needing to directly manage the communications with the UMLSKS server. The second scheme is a lower level mechanism that can be used with any programming language. The socket-based scheme includes a TCP/IP server running on the UMLSKS server that accepts socket connections from remote clients. Clients establish a connection to this server socket, compose a UMLSKS API request in XML format to send over this connection, and then await receipt of the XML response from the server. Client programs may be written in any language that supports TCP/IP socket communication. Java programmers can take further advantage of the API by using the Object Model to interpret the returned XML.

The API is built on the premise that all of the Metathesaurus may not be required by every developer. Many applications require only a fraction of the information available. With this in mind, the API was developed to slice the Metathesaurus into subsets of data. This results in a reduction of the total amount of information traveling between the UMLSKS and client applications and also provides applications with fine-grained control over the data they wish to receive. These modifications to the software yield significant performance improvements over the previous version.

The API exclusively uses XML for describing data for each of the Knowledge Sources. As an industry standard means of structuring information, XML provides a platform-independent form for representing hierarchical data like those of the Knowledge Sources. XML is basically ASCII text that is self-describing through use of descriptive data tags. Many tools exist for manipulating and displaying XML that make the developer's job easier by releasing them from this responsibility and allowing them to focus on the application details. The use of XML gives the system its extensibility and flexibility as proprietary formats are dropped in favor of a more-widely available and accepted form and XML is inherently forward compatible.

UMLS OBJECT MODEL

Previously, the onus has been on application developers to create their own usable data model for the Knowledge Sources. Each developer needed to understand the relational data representation delivered by the UMLS development group in order to abstract the Knowledge Source contents into application level components. Competing UMLS object models existed but without a consistent object model, complementary development was more difficult. As the user base continued to grow, the desire for a single, reusable data model evolved. By providing a single object-oriented data model, developers may focus more on developing application code instead of data modeling code. In addition, an object-oriented model lends well to extension of functionality thus allowing developer communities to develop and share software extensions of the model. Each of the Knowledge Sources has its own set of classes that form an object-oriented view of the data contained therein.

The Metathesaurus was founded on the principle of clustering synonyms to form a common meaning, an entity that has been labeled a concept. Lexical variants, including case, spelling, and inflectional variants of each member of this group of synonymous forms are clustered into an entity that has been labeled a UMLS term. Each of these lexical variants has been labeled as a UMLS string. Concepts contain terms and terms contain strings. There are concept attributes, term attributes, and string attributes. Concepts are related to other concepts via a number of different types of relationships. The Metathesaurus Object Model makes explicit this structure in a clear and understandable, manipulatable, navigable and extensible way. The model is composed of approximately sixty classes representing the various Metathesaurus components. A subset of these classes can interpret the XML returned by the UMLSKS API methods and instances of each class may be queried for the details as returned from the method calls.

The Semantic Network serves as an authority for the semantic types that are assigned to concepts in the Metathesaurus. It defines these types, both with textual descriptions and by means of the information inherent in its hierarchies. The semantic types are the nodes in the Network, and the relationships between them are the links. A similar argument for the need for a unified data model can be made for the Semantic Network [3]. Classes within the model abstract the nodes within the network and provide methods to extract details about each node. Classes also represent the network traversal allowing the network structure to be queried. The thirteen classes that comprise the Object Model for the Semantic Network provide XML interpretation capabilities and containers for the details of semantic nodes.

The SPECIALIST Lexicon has been developed to provide the lexical information needed for the

SPECIALIST Natural Language Processing System. The lexicon entry for each word or term records syntactic, morphological, and orthographic information. This lexicon is intended to be a general English lexicon that includes many biomedical terms. The UMLSKS provides an Object Model for the SPECIALIST Lexicon whose classes can interpret the XML generated by the associated UMLSKS API accessing SPECIALIST Lexicon data.

A USAGE SCENARIO

Here we will describe an example to elucidate the UMLSKS API and the associated Object Model (OM). The idea of a concept in the Metathesaurus has been translated into a class in the OM called the Concept class. The `Concept` class holds a number of other classes that represent the semantic type(s) for the concept (`SemTypeVector` class), the terms and their variants for the concept (`TermVector` class), the source's hierarchical context for the concept (`ContextVector` class), and other classes that constitute the complete definition of a concept in the Metathesaurus. Figure 1 shows a subset of the relationships between the Concept class and other OM classes. Once the desired API method has been executed and the XML result has been returned, the `Concept` instance can be constructed from the XML data. Example of the XML can be found in the UMLSKS documentation at http://umlsks.nlm.nih.gov. `Concept` instances can subsequently be queried for the semantic type(s), the definition, and its terms and term variants.

In our example, we wish to query the UMLSKS for information about the concept **Brain** that in our example will have the concept unique identifier (CUI) of C0000001. Specifically, we'd like to obtain the semantic type, definition, and term variations between source vocabularies for this concept. The UMLSKS API provides a method, **getBasicConcept** that uses the CUI to query the Metathesaurus and returns the desired information in XML form. The logical steps to obtaining these details are to 1) establish a connection to the UMLSKS, 2) request the data using the appropriate API method, and 3) interpret the results. Figure 2 shows a segment of Java code that performs this series of steps.

```
// 1) Establish connection to the
// UMLSKS server
KSSRetrieverV3_0 retriever =
     (KSSRetrieverV3_0)
     java.rmi.Naming.lookup(                    "KSSRetr

// 2) Request basic concept
// for CUI 'C0000001'
char[] result =
retriever.getBasicConcept(
     "2001", "C0000001", null, "ENG");

// 3) Interpret the XML result
// into the Concept instance
Concept myConcept = new
     Concept(new String(result));

// 4) Print out the concept's name
System.out.println(       "Concept
Name='" +
          myConcept.getCN() + "'");
```

Figure 2 - Basic Concept Retrieval Code Snippet

SYSTEM ARCHITECTURE

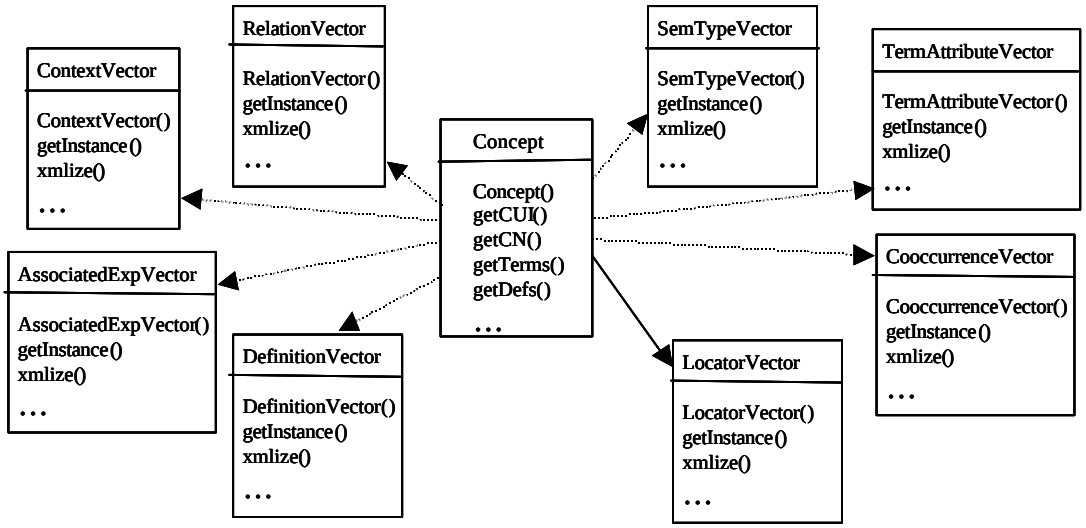


Figure 1 – Concept Class Diagram

The high-level logical system architecture for the Knowledge Source Server is shown in Figure 3. The services of the UMLSKS can be accessed three different ways: through a web client using a standard browser (Netscape or Internet Explorer), through a program written to use the UMLSKS Java API or through a TCP/IP based socket server. Data are returned to the user in XML. A set of classes that provide a data-centric representation of the UMLS Metathesaurus is provided with the API and is capable of reading the XML generated by the API methods.

The web interface issues HTTP requests through the Internet to a web server. The web server then issues a request using Java's Remote Method Invocation (RMI) methodology to execute a particular function on behalf of the user. The RMI Server receives the request, executes the request against the database, and formulates and returns that result to the web interface in XML. The web application, which is implemented as a collection of Java servlets, applies XSLT stylesheets to the returned XML and converts it into HTML. Open source software from Apache was used for the development of all aspects of the web application. The web server software, as well as the socket interface and Java API, connects through the Internet to a backend RMI server. This server processes all requests for Knowledge Source data, accessing an Oracle® database to obtain relevant information, and forwards those data sets through the Internet to the requestor. The UMLSKS is designed to simultaneously support multiple releases of the UMLS Knowledge Sources.

The API issues RMI or HTTP requests through the Internet directly to the RMI server using the RMI protocol. The RMI Server receives the request, executes the request against the database, and formulates and returns the result to the client API in XML form. The client program may subsequently use the Object Model classes to interpret the returned data as a set of data-centric objects. The new API is entirely written in Java with the goal of providing platform independence. In all, approximately sixty API methods have been defined allowing access to all details for each of the Knowledge Sources.

The socket server provides an interface to allow clients to send XML requests through a socket to the UMLSKS server, which in turn executes the request and returns the result of the query in its XML form. The XML API requests to the socket server resemble the Java API methods and thus provide a mechanism for both Java and Non-Java programs to interface to the UMLS through a standard TCP/IP socket. The flexibility provided by both APIs enables the UMLSKS to support system developers whose platforms range from PCs to high-end Unix machines.

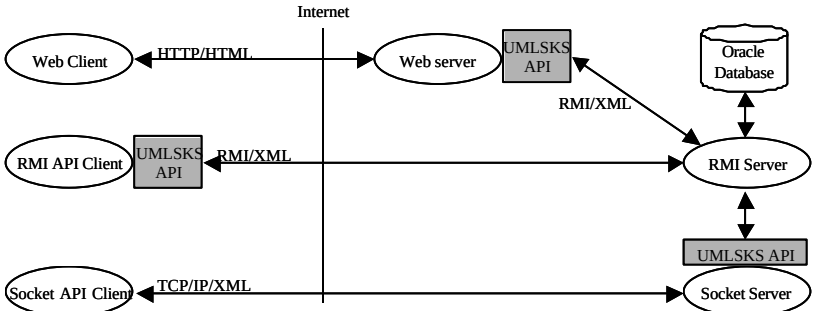
TERMINOLOGY SERVER: AN APPLICATION USING THE UMLSKS

The Terminology Server project uses the UMLSKS Java API to obtain UMLS information to provide synonymy data to other applications such as ClinicalTrials.gov at NLM. Through the Terminology Server, applications can define additional concepts and term variants, called local data, in addition to the UMLS data to form a synonymy data set suitable for the application's domain. The Terminology Server contains functions to tailor the UMLS data with concept merging, term variant deletion and additions, and other filtering operations.

The Terminology Server acts as a client of the UMLSKS, obtaining the concept and term variants via the API and using the Object Model to interpret the returned data. Once this is done, the local data is added, and any tailoring of the UMLS data is performed. The resulting data is then sent to the application. By keeping the local data separate from the UMLS data, updates to the UMLS are more easily managed. The Terminology Server has utility functions to analyze the local data with respect to new UMLS versions.

UMLSKS USAGE

Over 2000 total UMLS licenses have been signed with interested individuals and institutions in the years since the data have been made available. However, over time a few licenses (321) have ceased. Currently, there are over 1900 total UMLS active licensees (see table below). These licensees are either registered with NLM to receive UMLS updates on CD-ROM discs or they download the data from the UMLSKS, search the server using their web browser, or use programs that interact with the server's XML or Java API sets. Figure 4 shows the distribution of current UMLSKS users from around the world while the distribution of UMLSKS users in the United States by their Internet domain appears as Figure 5.



Active UMLS Licensees Through February, 2003		
U.S.	1327	68%
Non-U.S.	629	32%
Total	**1956**	100%

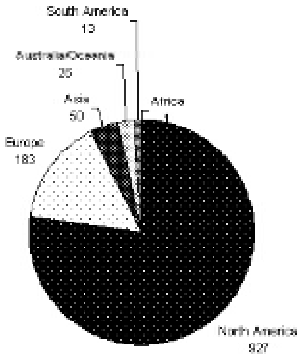


Figure 4 – Active UMLSKS Users By World Continent Through February 2003

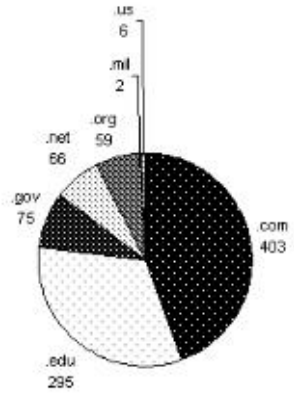


Figure 5 - UMLSKS Users By Internet Domain Through February 2003

CONCLUSIONS

The redesigned UMLSKS software has been operational for approximately one year and has successfully achieved all of the original design goals. The new architecture permits quicker incorporation and availability of new UMLS data set releases, generally on the order of one to two weeks. The balancing of server loading across multiple machines helps ensure reliable and fast access to multiple releases of the UMLS data simultaneously. The rich set of API methods that provide access to all UMLS data can be incorporated easily into programs written in any language for both PC and UNIX platforms and the Object Model makes explicit the UMLS structure in a clear and understandable, manipulatable, navigable and extensible way. For future work, we are looking at extending the UMLS object model, to provide a richer representation of the data.

REFERENCES

1. Lindberg DA, Humphreys BL, McCray AT, The Unified Medical Language System project: A distributed experiment in improving access to biomedical information, Methods Inf Med. 1993 Aug;32(4):281-91, PMID: 8412823.
2. McCray AT, Razi AM, Bangalore AK, Browne AC, Stavri PZ, The UMLS Knowledge Source Server: A Versatile Internet-Based Research Tool", Proc AMIA Annu Fall Symp. 1996;:164-8, PMID: 8947649.
3. Bodenreider O, An object-oriented model for representing semantic locality in the UMLS, Medinfo 2001;10(Pt 1):161-5, PMID: 11604725.
4. UMLS Knowledge Sources (11th ed). Bethesda (MD): National Library Of Medicine, 2001
5. Gu H, Perl Y, Geller J, Halper M, Liu LM, Cimino JJ, Representing the UMLS as an object-oriented database: modeling issues and advantages, J Am Med Inform Assoc. 2000 Jan-Feb;7(1):66-80, PMID: 10641964.
6. Mclaughlin B, Let your DOM do the walking: A Look at the DOM Traversal Module. *IBM Developerworks August 2001*
7. Rector A, Solomon W, Nowlan W, et. al. A terminology server for medical language and medical information systems. *Methods of information in medicine* 1995; 34(1-2): 147-157
8. Brissi P, Rousseau R, IREC: An object-oriented abstract representation to handle software components in a persistent framework. In *Object-Oriented Technology for Database and Software Systems*, pages 6-21. World Scientific Publishing Co Pte Ltd, Singapore, 1995.
9. UMLSKS Version 3.0 System Documentation, http://umlsks.nlm.nih.gov/.

ACKOWLEDGEMENT

We would like to acknowledge the following people: Alexa McCray, Guy Divita, Jan Willis, Mindy Nguyen, Chris Lu, Amir Razi.