The UMLS Knowledge Source Server: An Experience in Web 2.0 Technologies

Karen E. Thorn, MS, Anantha K. Bangalore, MS, Allen C. Browne, MS
National Library of Medicine, Bethesda, MD

Abstract
The UMLS Knowledge Source Server (UMLSKS), developed at the National Library of Medicine (NLM), makes the knowledge sources of the Unified Medical Language System® (UMLS®) available to the research community over the Internet. In 2003, the UMLSKS was redesigned utilizing state-of-the-art technologies available at that time. That design offered a significant improvement over the prior version but presented a set of technology-dependent issues that limited its functionality and usability. Four areas of desired improvement were identified: software interfaces, web interface content, system maintenance/deployment, and user authentication. By employing next generation web technologies, newer authentication paradigms and further refinements in modular design methods, these areas could be addressed and corrected to meet the ever increasing needs of UMLSKS developers. In this paper we detail the issues present with the existing system and describe the new system’s design using new technologies considered entrants in the Web 2.0 development era.

Introduction
The UMLSKS in production today utilizes a number of technological advancements available in 2003, including the use of Java™, the incorporation of eXtensible Markup Language as a means of data exchange, and employment of the eXtensible Stylesheet Language (XSL) to handle transformations of XML data into a form suitable for viewing in an HTML browser. The capabilities afforded by that design represented a significant improvement over the previous release yet these technologies imposed a set of limitations on the system’s capabilities. As internet and software technologies matured, the limitations that burdened the current UMLSKS design could be addressed.

Identification of those areas in the UMLSKS software where improvement was desired resulted in categorization of the enhancements into 4 main areas: web content, software interfaces, and system maintenance/deployment, and user authentication. Web 2.0 demarcated technologies such as web services and Asynchronous Javascript and XML (AJAX) enabled the first 3 areas of web content, software interfaces and system maintenance/deployment to be redesigned and improved. User authentication paradigms employing certificates addressed the fourth area. The redesign of UMLSKS software using these technologies effectively eliminated the limitations present in the current software release.

While the redesign allowed improvement in the areas discussed, those same technologies posed a different set of unique challenges. The advantages of using these technologies, however, outweighed the challenges involved. This paper serves to identify the set of challenges that exists with the UMLSKS software in current production and how the Web 2.0 era tools, authentication paradigms, and software design patterns have been utilized in the new design to improve the system’s functionality.

Technology Selection
Selection of the technologies to support the desired enhancements to the UMLSKS applied the following criteria during evaluation of each technology. Of primary concern was the ability to redistribute the component without charge and in the form of delivered source code. The second criterion was whether the software technology could easily be incorporated into the existing framework with minimal changes and implications imposed. Lastly, the learning curve of applying the new technology must enable quick employment of that technology in a useful manner. These criteria enabled selection of components and technologies that facilitated improvement in the identified four areas.

Software Interfaces
Client application software communicates with the UMLSKS through two software interfaces. The first is an interface defined for use by Java developers that utilizes Remote Method Invocation (RMI) to perform the equivalent of a remote procedure call. Accompanying this interface is an Object Model that represents the UMLS contents that may be exchanged through the interface. The second interface is designed for use in a non-Java environment and uses TCP/IP socket-based communication and XML to
transfer requests for UMLS data and the resulting UMLS content between the client program and the UMLSKS software. Both rely on socket communication to occur on specific ports, 1089 and 1099 for RMI communication, and 8042 for socket server communication. Communications messages are received by the UMLSKS server and rudimentary verification of the connected user’s identity is done by matching the internet address of the machine connecting to the server with the addresses kept within the UMLSKS user validation database. API users, therefore, would need to submit, a-priori, the set of all IP addresses of the machines from which they will potentially access UMLSKS resources.

Security concerns of today often lead system administrators to place firewalls between the internet and institutional software. In the strictest of security, only port 80 is open to the internet. The ports required for the RMI and socket server software are often, by default, not open for communication to occur which poses a problem for UMLSKS developers behind firewalls that wish to access the resources on the UMLSKS server. In addition, the use of routers and other internet addressing schemes (e.g. non-static IP addresses assigned by different Internet Service Providers) may alter the IP address of the incoming IP packet preventing the UMLSKS validation software from recognizing the message as a message from a valid user. In addition, it is feasible that a malicious user may attempt to spoof an IP address and thus surreptitiously gain access to the system. A viable solution must validate users and allow communication regardless of the location of the client software in internet space and with minimal constraints placed on the port over which that communication takes place. The solution selected for the redesign was based on web services and a new authentication paradigm.

Web services as defined by Sun Microsystems are “web-based enterprise applications that use XML-based standards and transport protocols to exchange data with calling clients.” Employing a web service framework for UMLSKS software enables replacement of the two distinct, non-standard interfaces currently in use (RMI and socket server) with a single standard interface. Deploying the web container to run on port 80 inside a web server addresses the second concern involving the RMI and socket server TCP/IP ports. The non-standard ports are no longer used and the number of firewall issues is significantly reduced.

Apache Tomcat was selected as the web server for the UMLSKS redesign, providing an environment in which Java code can run in cooperation with a web server. Tomcat has, as part of its suite of tools, an HTTP server configured at port 80 that handles the HTTP request/response circuit. Apache Axis is an open-source, XML-based web service framework that implements the Simple Object Access Protocol (SOAP) to exchange XML messages between software components. UMLSKS components are developed and deployed to Tomcat with Axis performing the SOAP messaging between the client and web service implementation.

The remaining issue raised with the current interfaces is that of client connection validation. With a web service implementation, access to the IP address for validation of a client software connection is not feasible and a new paradigm for validation was required. A certificate scheme was introduced to solve the problem. Upon creation of an account with the UMLSKS, a user is given a certificate used to generate single-use tickets that are placed in subsequent calls to the web service. The ticket is validated using the UMLSKS user authorization software and the call is permitted to proceed on verification success. The validation/verification scheme is described in greater detail in the section titled User Authentication.

Web Interface Content

The current production version of the web interface is implemented as a collection of Java servlets. These servlets make queries for UMLS data using the RMI API. The query results, in XML form, are returned to the servlets which then applies XSLT stylesheets to transform those results into HTML for display purposes. Despite the fact that this architecture is quite flexible in allowing us to implement new views of the data by writing new stylesheets, it is constrained by the fact that it can only provide a single, static view of the UMLS data. As the UMLS continues to grow in size and complexity, it is reasonable to expect that a single view of the data may not be sufficient to meet the requirements of a majority of the users. Ideally these views of the data should be user driven wherein a modular framework of reusable software components allows users to build dynamic, custom views. Web 2.0 technologies such as portals and AJAX presented an ideal approach to building a more modular, flexible and interactive UMLSKS interface.

A portal is a web interface that can be customized by the end-user both in look and feel and in available content using applications provided by the portal. The portal functions as an aggregator of portlets which
can be thought of as miniature Web applications that run inside a portal page alongside any number of similar entities. A portlet is a reusable web component managed by a portal container that can process requests and generate dynamic content. Each of the portlets generates fragments of mark-up, which the portal container ultimately pieces together to create a complete page. A portlet registry, similar to the web services registry, contains a list of available portlets. The Java Portlet Specification (JSR168) enables interoperability for portlets between different web portals. This specification defines a set of APIs for interaction between the portlet container and the portlet addressing the areas of personalization, presentation and security.

AJAX is a web development technique that allows the creation of highly interactive web applications that feel more responsive to users. This technique allows the exchange of small amounts of data with the server behind the scenes, eliminating the need for the entire web page to be reloaded each time the user requests a change. This increases the interactivity speed and usability of web pages. For example, in the new interface the output for a given query is displayed as an AJAX tree that allows users to expand and collapse individual nodes as desired. Figure 1 shows the AJAX tree for the semantic types. When a user clicks on a node in the tree with the intent to expand the node, only data associated with that node is retrieved from the backend and displayed. If the display were not AJAX-based, the entire page would have to be updated resulting in potentially slow response time and visually unappealing behavior for large trees of data.

![Figure 1. Portion of the browse tree for Semantic Types](attachment:figure1.png)

The new UMLSKS web interface is designed as a set of portlets, each of which displays a portion of the UMLS data. The outputs in each of these portlets are displayed as AJAX trees. Some of the portlets that have been implemented are Basic Concept Information portlet, Context portlet, Relations portlet, and Co-occurrence Information portlet.

An open source, freely available portal framework called uPortal was selected for the development of our portal-based interface. The software is an open-standard effort using Java, XML, Java Server Pages (JSP) and Java 2 Enterprise Edition (J2EE) technologies. The uPortal framework allows users to create any number of individual pages upon which a user can add any number of applicable portlets. These pages are in a sense a representation of the user’s view of the data. The portlets are described in a portlet registry that is searchable by name and description. Portets within an application communicate with each other using session variables.

We have developed a default view for each of the three Knowledge Sources: Metathesaurus®, Semantic Network, and the SPECIALIST Lexicon. These default views are static and are made available to all users of the web interface. The default views are a representation of the UMLSKS view of the data. Users can use this as a guide in developing their own views, incorporating new portlets into their data views as they become available. Future development efforts will focus on the development of individual portlets that meet the needs of the UMLS community at large. Potentially users may contribute their own portlets for inclusion in the baseline set available to all users.

**System Maintenance and Deployment**

The software in production today is built as a set of standalone Java applications that are run from the command-line. Components are tightly coupled and process memory and disk space requirements expand as the number of UMLS releases supported and additional functionality are introduced. This tight coupling also requires that the system software be shut down and restarted for new builds to take effect. This type of deployment model causes gaps in availability of the entire system, including the web interface, while the software is updated. Given the significant increase in the number of users expected over the next few years, this type of update model will not be acceptable as users will demand accessibility and availability approaching 24/7 operations.
The software redesign involves a significant decoupling of components and results in multiple web services implemented to perform the same functions as are available in the current release. A single web service is deployed to handle requests from users. This web service accesses internal services that access each UMLS release’s data, as well as services providing authorization, registry and logging functions. As new UMLS releases are made available from the National Library of Medicine, a new service is deployed to permit access to the data contained within that release. The use of Tomcat as the deployment platform will enable this deployment without a restart of the entire system. New functionality may be added to the system without impacting users currently connected to the services. In addition, each service is designed as a small set of coherent, related classes that provide a small, focused set of functions thus giving the software a small footprint and allowing it to execute at a higher level of performance. Figure 2 shows the architecture of the redesigned UMLSKS.

**User Authentication**

One of the primary goals of the redesign was to have a fast, secure, scalable authentication mechanism that can be used for both the web interface and API access. With a Web services and portal based architecture in the redesign, it was important that the authentication system also support a single-sign-on capability. The redesign allows us the flexibility of hosting individual web services and portlets on multiple machines across multiple domains each with potentially different authentication schemes. In this context, single-sign-on, permitting users to sign on to the system once and access all the services irrespective of where those services are hosted, was imperative. The Central Authentication Server (CAS) provided this desired capability.

The CAS is an authentication system originally created by Yale University to provide a trusted way for an application to authenticate a user. The goal of the CAS was to facilitate single-sign-on across multiple applications and also to allow services offered by organizations to authenticate users without having access to their passwords.

The CAS is designed as a standalone web application implemented as a set of Java servlets that run on a secure web server. Typically applications using the CAS for authentication will send a username, password and the name of a service for which access is required to the CAS to validate against an appropriate backend authentication mechanism. If authentication is successful, the CAS creates a long, random number, which is called a ticket. It then associates this ticket with the user who successfully authenticated and the service to which the user was trying to authenticate. That is, if username U is passed from service S, the CAS creates ticket T which allows U to access service S. This ticket is intended as a one-time-use-only credential; it is useful only for U, only for service S, and only once. The ticket expires as soon as it is used. Since a ticket is intended as a one-time-use-only credential for security reasons, the CAS can also provide a ticket-granting authority (TGA) to client applications that may access multiple services from within their applications. The applications can then use the TGA to generate valid single-use tickets for the various services. The TGA expires after a specific time interval. The CAS architecture is illustrated in Figure 3.
On successful authentication via the UMLSKS web interface, the CAS sends a ticket-granting cookie (TGC) as an in memory cookie to the browser. This TGC is then used subsequently to generate tickets for the various services accessed by the web interface. Each of the services on receiving the ticket will validate this ticket with the CAS. The TGC is destroyed when the browser is closed or expires after a specific time interval. Similarly, applications using the UMLSKS API obtain a TGA by sending in a valid username and password to the CAS web service. The application then uses this TGA to generate tickets for various services. The initial username and password request is sent over a secure HTTPS connection. Once the TGA is obtained subsequent connections to various services are made over the non-secure http connection.

Lessons Learned

The main challenges encountered during the development process centered around the lack of good documentation for some of these technologies. This is not surprising as these technologies are new and collective experience within the developer community has been insufficient to allow for definitive and comprehensive documentation to be produced.

Another challenge revolved around interoperability between the different available web service platforms. Initial prototyping was done using Sun’s Java Webservices Development Pack but was abandoned in favor of Apache Axis for its compatibility with the uPortal and authentication software. In theory frameworks for supporting web services implement the same standards; however, in practice, it is difficult to port applications written on one framework to another without making substantial changes. Users developing client applications that interact with the UMLSKS may be forced to use Axis as their development platform, as other web services platforms may not be able to communicate properly with the UMLSKS implementation.

The last challenge encountered focused on system performance. Implementations of the SOAP messaging protocol (e.g. Axis) impose additional overhead during message transit. During prototype testing, no appreciable delays were seen. This may be due to the modular redesign and the new user interface paradigm.

Conclusion

The new system meets the following goals of the redesign

- Enhance speed, usability, and maintenance of web interface
- Enable user tailoring of web interface
- Simplify user validation and verification
- Address firewall issues
- Employ dynamic deployment model of new UMLS releases
- Simplify user validation for API users
- Facilitate machine and API programming language freedom
- Provide software framework for easier integration of new features

This system is scheduled to replace the current production system in late 2007 or early 2008. For the future we plan to actively involve the user community in making enhancements and modifications to the system. We plan to set up a UMLSKS WIKI to facilitate discussion among the user community and to solicit comments. We plan to encourage users to submit web services and portlets provided they confirm to accepted standards and follow a few basic rules. A downloadable version of the new system is also in the planning stages.

References