A Component-Based Problem List Subsystem for The HOLON Testbed

Vincent Law, Howard S. Goldberg, MD, Peter Jones, Charles Safran, MD, MS
Center for Clinical Computing, Beth Israel Deaconess Medical Center, Harvard Medical School, Boston, Massachusetts, USA

One of the deliverables of the HOLON (Health Object Library Online) project is the specification of a reference architecture for clinical information systems that facilitates the development of a variety of discrete, reusable software components. One of the challenges facing the HOLON consortium is determining what kinds of components can be made available in a library for developers of clinical information systems. To further explore the use of component architectures in the development of reusable clinical subsystems, we have incorporated ongoing work in the development of enterprise terminology services into a Problem List subsystem for the HOLON testbed. We have successfully implemented a set of components using CORBA (Common Object Request Broker Architecture) and Java distributed object technologies that provide a functional problem list application and UMLS-based “Problem Picker.” Through this development, we have overcome a variety of obstacles characteristic of rapidly emerging technologies, and have identified architectural issues necessary to scale these components for use and reuse within an enterprise clinical information system.

INTRODUCTION

The HOLON (Health Object Library Online) architecture espouses a component-based implementation for clinical information systems to facilitate both the rapid deployment and addition of value-added functionality to such systems. Because HOLON is envisioned as a library, one of the challenges facing the consortium is determining exactly what entities can be made available in the library for developers of clinical information systems. Patrons of the library may desire entire clinical applications or packaged sets of functionality with which to build custom applications. They may want to implement a HOLON-compliant data repository in which to persistently store data or they may want to integrate existing data stores with new components. Therefore, the Health Object Library should make available components with a varying degree of granularity: applications encompassing data store through user interface, health care objects exposing reference data structures and behavior, large-scale back-end services such as master-patient index, vocabulary, decision support, etc., and abstractions which facilitate integration into existing systems.

In separate work, we have collaborated with Lexical Technology Inc. (LTI), Mayo Clinic, and others in the development of UMLS-based terminology services—the Metaphrase toolkit. In our own environment at Beth Israel Deaconess Medical Center, we have several applications which should share enterprise terminology services—the OMR (Online Medical Record system) used in the ambulatory faculty practices, Medicaologic™ used in affiliate practices, CareVue™ and MDAssist™ used in intensive care units, etc. Each of these systems implements its own version of a problem list or assessment and planning tool that ideally should be standardized across the enterprise. It is likely these disparate systems could reuse components from a common problem list subsystem, they can do so to a varying degree given the diversity of implementations.

In this report, we describe ongoing work on the development of a component-based problem list subsystem designed to be compliant with the architectural principles recommended by the HOLON consortium. The goal of this work is to develop a series of modular, reusable components for deployment within the HOLON testbed as well as within the myriad of systems described above. This work also provides an opportunity to report first-hand experience with a series of new technologies designed to promote modularity, distribution of systems over distance, and software reuse.

MATERIALS AND METHODS

Five major components were identified as integral parts of the Problem List subsystem: a
Terminology service—the Metaphrase™ Server, a Problem List and “Problem Picker” user interface, and Problem List and Problem List “Factory” health care objects (Figure 1).

**Terminology Service—Metaphrase™**
Metaphrase™ is a general terminology service developed by LTI in collaboration with several informatics laboratories, including our own. Metaphrase™ is layered over an implementation of the UMLS Knowledge Sources and provides general terminology services over HTTP (HyperText Transport Protocol) in its initial version. On top of the general terminology services, we have added specialty services specific to the domain of clinical problems, e.g., filtering of non-problem terms that are returned from a search operation.

**Problem List And Problem Picker Interfaces**
The Problem List interface reimplements the problem list model used in OMR. An entry in the problem list is composed of a problem label, comment, activation date, and inactivation date. In the initial version, entry of problem labels is facilitated by the separate Problem Picker interface.

The Problem Picker interface mediates an interaction between a health care provider and the terminology server. The interface is designed to build a neighborhood of relevant problem labels based on an input string provided by the user. The current interface uses a directory tree metaphor to provide access to problem labels matching the given input string as well as to other semantically related labels. The Problem Picker also provides users with access to definitions to clinical terms if they are available within UMLS.

**Problem List And Problem List Factory Health Care Objects**
The Problem List object is a “middleware” object that implements the required data structure and “business logic” for problem lists. The initial version of the Problem List object exposes very core functionality: the ability to persist and restore its state, the ability to add new problem entries, and the ability to stamp a problem as inactive. The Problem List Factory is an object factory that finds or creates Problem List objects when presented with a reference to a patient.

---

**Figure 1. Problem List Implementation**

[Diagram showing the flow of interaction between holon, problem list, problem picker, CORBA, database, Metaphrase server, and UMLS.]
RESULTS

Implementation
The Problem List and Problem List Factory objects are implemented as CORBA (Common Object Request Broker Architecture) 2.0 compliant objects. Visigenic’s Visbroker for Java is used as the ORB (Object Request Broker) implementation. The interfaces of the health care objects are exposed to clients via IDL (Interface Definition Language). The objects themselves are implemented in Java. The state of Problem Lists are persisted and restored from an Oracle-based repository using JDBC (Java Data Base Connectivity).

Problem List objects exist as fine-grained objects in that an object is instantiated for every patient. The Problem List Factory is a coarse-grained object so that one factory object currently serves the needs of the demonstration system. The factory object is a persistent CORBA object—this object is well known to the rest of the system and has an immutable reference when the system is turned off. In contrast, the data objects for problem lists are transient and do not exist between invocations of the ORB, although the data for each object is persisted in a database. In the current system design, the factory object is responsible for managing the data objects (e.g., avoiding duplicate instantiation of a problem list) and not the ORB.

Metaphrase services are provided over HTTP. While the consortium recommends CORBA compliance, HTTP-based services have proven useful for prototyping. Time did not permit development of a CORBA-compliant interface for the terminology service during the last demonstration cycle, and this has been added as a task for a future cycle.

The Problem List and Problem Picker interfaces are implemented as Java applets that work in conjunction with a World Wide Web browser. The Problem List interface is a client of a Problem List object with which it communicates via IIOP (Internet Inter-ORB Protocol). The Problem Picker interface is a client of Metaphrase and employs a set of class libraries that encapsulate HTTP-based communication with the server.

Security
HOLON user interfaces are currently web-based which imposes significant functional restrictions on what actions are allowable by an applet. For security purposes, applets downloaded from the Internet run inside the “sandbox” of the browser. This prevents applets from damaging the client computer, by reading and writing to system files, or making connections to untrusted machines on the network other than the machine from which the applet was downloaded. Because the Metaphrase-based terminology server exists on a machine separate from the machine storing the Problem Picker core, the Problem Picker is subject to these restrictions which would essentially defeat the architecture.

To overcome these limitations, we have employed the Netscape® Capability model. In the Capability model, an applet can be “signed” using public/private key encryption, and, once verified by the browser, can request and be granted access to fine-grained services called “capabilities”. In this manner, instead of giving an application full access to system resources, the application is granted a limited degree of access which can be fully enumerated before the application begins to run. With respect to the Problem Picker, the applet is granted only the ability to make a network connection to another machine. While extra steps are required initially to obtain a certificate to verify the identity of an applet and configure the browser, this model permits robust applets to run via a browser while minimizing security risks.

DISCUSSION

Several achievements are notable in this implementation of the Problem List subsystem for the HOLON testbed. We

- Implemented a series of health-provider-related data objects, components, and services using distributed object technology;
- Operated a terminology service without regard to geographic location (server residing in California);
- Added “signed” applets to list of HOLON security features;
- Identified series of architectural issues and “next steps” necessary to scale the subsystem in order to facilitate large numbers of users and objects;

413
In developing the prototype, we gained a greater appreciation for “componentizing” the system to facilitate reuse. For example, the M-based OMR could reuse the services of the terminology server in its problem list application; however, reuse of the user interfaces is not possible as the current system is character-based, and reuse of the middleware objects would be difficult without an “M” mapping to CORBA. Other systems might be able to incorporate all vertical components of the new subsystem. Other options would include reusing the new middleware objects and developing applications within the existing application framework, and replacing the persistence implementation in order to store data in an alternate repository. By modularizing the functionality in this way using object-oriented techniques, a wide variety of reuse becomes possible.

Early adoption of new technologies is not without its share of trials and tribulations. By employing Java, we hope to eventually benefit from the ability to “write once, run everywhere”. Clinical applications implemented in Java can potentially be deployed on low-cost hardware as envisioned in the network computer—a significant consideration when aggregate hardware costs are calculated across a regional health care delivery network. However, at this stage of Java development, we were forced to work around incompatibilities in different Java implementations in order to successfully complete the prototype.

As of this writing, not every Internet browser supported the latest version of JDK (Java Development Kit). For instance, prior to version 3 of either Internet Explorer or Netscape, AWT (Abstract Window Toolkit) 1.1 event model handling—the central event handler for interacting with GUI and other components—was not supported. Currently, our clinical components are written in JDK 1.1.5 with AWT 1.1 event model handling, and employ the Netscape Capability model as described above. Therefore, client-side code requires customization to work appropriately across browsers. Additionally, we discovered that Netscape does not currently support the downloading of multiple JAR—Java Archive—files which contain class and other files supporting an application. Because the subsystem requires the download of both CORBA support code as well as the application code, it was necessary to build a single JAR file to work around this deficiency.

While the Problem List subsystem works well as a prototype, further architectural consideration is necessary in order to scale the system to production quality in order to handle large numbers of users and data objects. In the abstract, patient data comprising a clinical information system is can be represented by graphs of persistent, fine-grained objects including problem lists, medication lists, observations, notes, etc. In a production clinical information system at a major medical center, there are likely to be millions of fine-grained clinical objects of which thousands may be active at any one time. To implement such a system, a distributed framework will need to provide 1) a persistence model in which an immutable object identifier is associated with persisted state data; 2) a relationship model through which persistent associations between clinical objects could be created and navigated.

In practice, CORBA-based systems have emphasized coarse-grained objects such as our Problem List Factory which internally manages the lifecycle of large numbers of fine-grained objects without exposing these objects. The disadvantage to this approach is that lifecycle management code must be duplicated for every clinical object “server”. In theory, the Object Request Broker should be able to provide this functionality and provide value-added services such as load-balancing and fault-tolerance. Additionally, while a CORBA Relationship Service has been specified, major vendors have not currently provided implementations, again perhaps because production systems to date emphasize server objects as the unit of distribution. The latest version of the CORBA specification better addresses lifecycles for persistent objects so that the next generation of CORBA products may be better able to support this notion of graphs of clinical objects. A challenge to the HOLON consortium in the upcoming year is to better define the practical scope of clinical data objects given the state of existing technology as well as define a roadmap as planned refinements for Object Request Brokers are implemented.
CONCLUSION

We have described work on a series of modular, reusable components comprising a problem list subsystem for the HOLON testbed. We have successfully implemented this series of components using distributed, object-oriented technologies. In the process, we have overcome a variety of obstacles characteristic of rapidly emerging technologies, and have identified architectural issues necessary to scale these components for use within a production clinical information system. This initial implementation has clarified issues in designing shareable clinical components for use across a health care enterprise, and has aided in defining what entities should populate a “Health Object Library”.

ACKNOWLEDGMENTS

Supported in part by a cooperative agreement award from the Agency for Health Care Policy and Research and the National Library of Medicine (HS08749), and a contract from the National Institute of Standards and Technology (1995-10-0067N).

References

7. Delaney DP, Zibrak JD, Samore M, Peterson M. Use of Relational Database Management System by Clinicians to Create Automated MICU Progress Note from Existent Data Sources. JAMIA 1997;Suppl:7-11.