

A Client-Server Framework for Deploying a Decision-Support System in a Resource-Constrained Environment

Martin J. O'Connor, M.Sc.¹, Ravi D. Shankar, M.S.¹, Samson W. Tu, M.S.¹,
Aneel Advani, M.D.^{1,2}, Mary K. Goldstein, M.D.^{2,1}, Robert W. Coleman, M.S.²,
Mark A. Musen, M.D., Ph.D.¹

1. Stanford Medical Informatics, Stanford University School of Medicine,
Stanford, CA 94305; 2. VA Palo Alto Health Care System, Palo Alto, CA 94304

There is a conflict between the desire for increasingly complex and robust decision-support applications and the reality of deploying them in legacy systems in existing clinical settings. The hardware and software environments at a clinical site can often conspire to complicate this deployment process. These difficulties can be increased when deploying research-based decision-support systems because of the limited administrative and staffing resources typically available to such systems. We addressed these problems by designing a client-server framework for rapid deployment of a decision-support system in a resource-constrained environment. This framework minimizes the resources required of client machines and also simplifies administrating the decision-support applications on those machines. We deployed a decision-support system developed using this framework at a hospital within the Department of Veterans Affairs.

Introduction

Elaborate decision-support tools are useless if they cannot solve problems in real clinical settings. Unfortunately, the amount of work required to deploy such tools can often approach the amount of work that went into their initial development. Several factors generate difficulties in the deployment process:

- (1) Typical health care environments have a motley collection of clinic workstations, many with aging CPUs and severely limited system memory. Modern software applications are increasingly resource-intensive and do not usually run well on such constrained machines.
- (2) Clinic workstations are rarely installed system-wide, but rather *ad hoc* in phases. Consequently, they often may not have the same versions of system software. This situation complicates the deployment of client systems.
- (3) This configuration diversity can also complicate software maintenance and updates. In a diverse system, these tasks can be as labor-intensive as writing the client application.
- (4) Clinic workstations are usually installed throughout a health care site. Installing and administrating client systems that have been deployed on physically separate machines can increase the system development effort.

(5) Complex and resource-intensive systems are often necessary to ensure a consistently high quality of advice. In addition, physician acceptance of a decision-support system is usually predicated on the quality of the system's advice. Given the time-critical nature of modern medical care, a slow system is as useless as no system.

Deploying research-based decision-support systems can be even more complex because they are less stable than commercial systems and they evolve continuously. Client and server code and knowledge base content are updated frequently, and the updates need to be reflected on client machines. Adding to this complexity, sufficient administrative and staffing support is seldom available. The resources to constantly update and maintain the guideline application on client machines are seldom present either.

This paper presents a framework for designing a decision-support system that addresses these issues. The primary goal of this framework is to allow rapid deployment of complex decision-support systems in a resource-constrained environment with limited administrative support.

Background

Our laboratory has twenty years of experience in developing decision-support systems for patient care. We have developed the EON architecture—a set of reusable middleware components that developers use to for building guideline-based decision-support systems that can be integrated into host information systems [1]. One of EON's problem-solving components is a guideline interpreter called *Padda* [2], which interprets an explicit model of clinical guideline and patient data to generate recommendations. An explanation engine called *WOZ* is used to explain the reasoning of the guideline interpreter to end-users [3]. EON has a database-mediator component called *Chronus II* [4], which handles all the patient-data requirements of the problem-solving components. *Chronus II* uses a temporal abstraction system called *RASTA* [5], which generates high-level temporal concepts from patient data. A domain-model component, *Protégé-2000* [6], provides relevant medical terms and relations that are used by all other components. The EON Server module

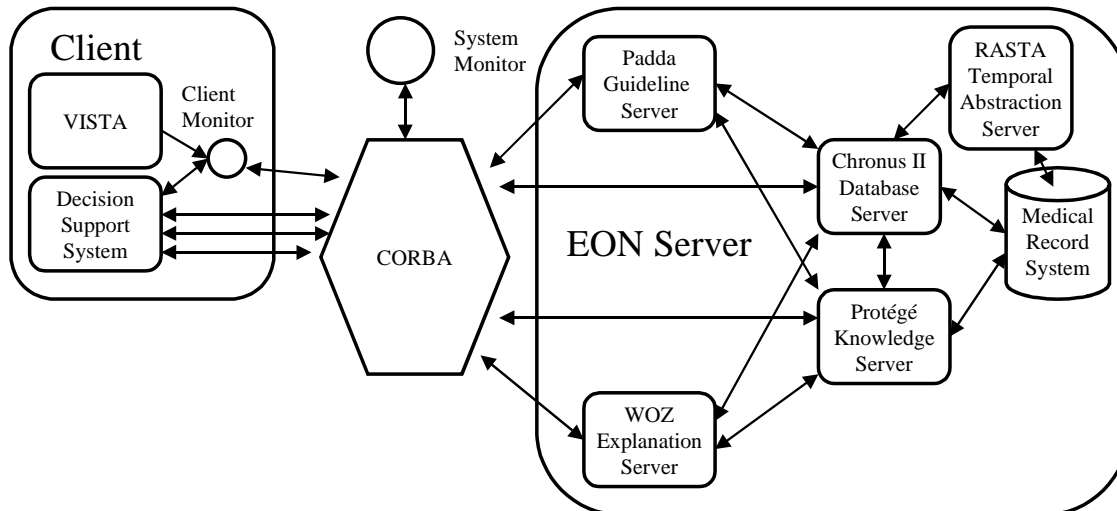


Figure 1: The ATHENA Decision Support System. The multiple processes in the server are shown together with a single client node. The decision support client application runs on a physician workstation and communicates with the guideline server and database server using CORBA. A daemon on each node monitors its associated client application; an overall system daemon coordinates with each client daemon to monitor all decision support clients on a network.

in Figure 1 shows interactions between various components in a typical EON system.

We used EON components to build an application called *ATHENA DSS*, which has been deployed at the General Medical Clinic and Hypertension Clinic at the VA Palo Alto Health Care System [7]. We used the EON guideline model, to create a computer interpretable representation of the hypertension guideline in 1) the *Sixth Report of the Joint National Committee on Prevention, Detection, Evaluation and Treatment of High Blood Pressure* [8] and 2) closely related VA hypertension guidelines. *ATHENA DSS* applies relevant patient data to the guideline model, and determines 1) a patient's eligibility to be treated under the guideline, 2) his target blood pressure, 3) if his blood pressure is under control, 4) his risk group, and 5) drug recommendations. It also provides general management messages. *ATHENA DSS* displays patient-specific treatment information at the point of care, and allows a clinician to modify patient data and fetch updated advisories, or to request an explanation for any recommendation.

Our deployment of this system was carried out in two phases. In the initial phase, we deployed a simple system that generated rudimentary prescription advisories based on VA clinical practice guidelines. In that phase, we dealt mainly with the difficult issue of database integration [9], and we used only a small subset of EON's capabilities. The second phase of our system was considerably more complex and was deployed in Palo Alto and will be deployed at additional VA clinics in San Francisco and North Carolina.

We anticipate limited staff resources at the two remote sites, and expect that administrating client applications at these sites needs to be performed centrally from the VA site in Palo Alto. The application that we were to deploy was a research system, so frequent changes in the client and server code and the knowledge bases were expected. Hence, software at the remote sites would need to be updated regularly. Complicating this deployment was our inability to fully integrate with the VA medical record client application at each site. A loose coupling between our application and the VA's client was anticipated. Given our goal to evaluate a state-of-the-art guideline-based advisory system, the deployment of a simple scaled-down client was not an option.

Developing a useful system that works within these constraints is difficult. To structure the development process, we designed a framework to serve as the basis for the second phase of our deployment.

System Framework

To meet our design goals, the system had to satisfy three main requirements: (1) the client must use a minimal amount of system memory and CPU cycles; (2) it must be easy to install; and (3) it must be easy to monitor and update.

Clients

The client application must be easy to install in order to perform the process remotely and without human intervention at the target machine. Our goal for our system was to install our client application on each

target machine in a single simple step that was identical for all machines. There were two main ways to meet this requirement: (1) keep local client configuration information to a minimum; and (2) store client code on the server. Local client configuration information can be minimized by storing it in server processes. A client needs only a single piece of information indicating the server that holds its configuration details. Similarly, user configuration information should be stored centrally. For example, in some cases, advisories may be customized for different types of medical practitioners. Information indicating this type of customization should be stored in the central guideline interpreter.

To meet the goal of remote client update, all client code should be stored on a server machine. Clients can access this code through the mapping of a network drive. When a network drive is mapped, the client can treat the drive as though it were local. We thus do not need to directly distribute any code to client machines. In a production system, client updates would be infrequent and the necessary administrative support would most likely be in place to perform manual client software updates. In the absence of this support, the client mapping of a server drive can provide a useful short term software distribution mechanism.

Servers

To minimize client resource use, the system must offload as much work as possible to server processes. A guideline interpreter running on a server should be responsible for computing patient advisories. It can get this data from a database server, which should also be executed on a server machine. The client requests a guideline advisory and data for a patient and then displays them. If a user requests an updated advisory, the request is forwarded to the server for processing. A client application should simply be a display module and a conduit for user requests.

To meet the rapid response requirement, the server may precompute patient advisories. The server can use encounter information to decide which patients to precompute. Thus, when a client application requests an advisory, the system presents the precomputed advisory. In addition to simple precomputation, the server must also be able to rapidly update any advisories that it has given if a physician modifies any relevant patient information during an encounter. For example, a physician may change the drug dosage information for a patient and request an updated advisory that incorporates this revised knowledge. The regeneration of this advisory should be instantaneous. In order to meet this performance goal, the guideline interpreter must store the intermediate state information generated by the advisory computation process. Thus, when a new

piece of patient information arrives, the server can update the advisory state and generate a new advisory quickly, a process that will be considerably quicker than starting the entire process from the beginning.

Monitors

Monitor processes are useful for remotely managing clients. A single simple process to monitor each client, together with a single system monitor, can provide the infrastructure for these capabilities. For example, a client monitor can ensure that its client remains running by pinging it regularly and restarting it if it fails. The client monitor can also work with a system monitor to allow remote updates of client software. When an update is required, the system monitor can tell all client monitors to shut down the client application. It can then update the code on the server and tell the client monitors to restart their applications with the updated code. This process can also be used to inform clients of server moves or server downtimes.

While the framework outlined here is mostly operating system independent, certain activities do require system-dependent tasks. For example, a network drive cannot usually be mapped in a system-independent way. Similarly, the communication mechanism between a legacy application and our software will probably be operating system- and client-dependent. The portability of the client and server code is also an issue, though it is easily addressed with portable languages and libraries.

Deployment

This framework is intended to act as general guide for the rapid deployment of a complex guideline system in a resource-constrained environment with limited administrative support. Many details will differ in the deployment of individual systems. To illustrate how this framework works in practice, we describe the deployment of the second phase of the ATHENA decision-support system.

System Configuration

Figure 1 shows the configuration of our deployed system. For simplicity, the figure only shows a single client. We installed a central guideline interpreter at the VA Palo Alto Health Care System and deployed a network of client applications at this site. Note that server processes can also be distributed on multiple machines. All software was written in Java.

The entire system was executed within the VA intranet, so our software did not need to address any additional security issues [10]. Note that security would be an issue in other environments, but that the basic approach would be unchanged.

Client Application

The VA uses a medical record application called *CPRS*. *CPRS* is a Windows-based graphical interface that lets clinicians access medical data for VA patients. The system can provide some simple reminders, but it generally does not provide high-level advisories.

Our hypertension decision-support system had to work with *CPRS*. Unfortunately, we did not have direct access to the system's source code and *CPRS* does not provide a mechanism to integrate new software modules, so we could not tightly integrate our client application with it. Fortunately, *CPRS* uses a simple Windows messaging system to inform other applications on a machine when a patient has been selected. When a selection is made, *CPRS* broadcasts a message containing a patient record number. We wrote a client monitor to intercept this message and forward it to the guideline client on the same machine. On receiving this message, the client can request any advisories for that patient from the guideline interpreter. If advisories are present, it displays a window containing them. This window contains a detailed description of each advisory, and a user can query the system for an explanation of its reasoning with EON's WOZ explanation tool [3].

The client program does a minimal amount of processing, as all requests for information are presented to the guideline and database servers. If a user requests an updated advisory, the system forwards the request to the guideline interpreter for processing. The only real function of the client is to display the guideline advisories in a user-friendly fashion, and to forward update requests to the server.

The system uses CORBA for client server communication. We chose CORBA because it requires minimal client resources and does not require establishing local configuration information on the client machine. Its only real requirement is that the client machine supports the TCP/IP protocol.

Each client connects to the database server via a CORBA connection. Although ODBC may seem to be a more obvious choice, it requires configuration information on the client machine. If the database server machine is moved, for example, all client ODBC configuration information must be updated. Because CORBA is the connection mechanism, the only information the client needs is the address of the new server.

As a result of these factors, the client is quite small. It requires minimal memory and very few CPU cycles. In addition, no client configuration is necessary — the client only needs to know the location of its servers to function properly. This characteristic also dramatically simplifies the client installation process.

CORBA also adds a degree of robustness, because it can ensure that broken client-to-server connections are reestablished automatically. To ensure a higher level of monitoring, we used the client monitor on a machine to ping the client frequently. If no response was received, the client was restarted automatically.

Remote updates of the client software are also possible. When a software update is required, the administrator can tell all client monitors to shut down the client application. It can then update the code on the server and tells the client monitors to restart their applications with the updated code.

Precomputed Advisories

The guideline server is similarly resource-intensive, requiring large amounts of system memory to operate effectively. Generating an advisory for a single patient is not usually possible in real time, even on fast modern workstations. To ensure that clients receive a rapid response, our system precomputes patient advisories whenever possible. The system determines which patient advisories are appropriate for precomputation by examining encounter information stored in the medical record system and selecting individuals with upcoming appointments. When a physician selects one of those patients at the appointment, the client application requests the precomputed advisory from the server and displays it.

The guideline interpreter saves the final advisory given to a physician and the intermediate state information generated by the advisory computation process. When a new piece of patient information arrives, the system can update the advisory without starting the entire process from scratch. Therefore, when a physician changes some patient information and requests a new advisory, the guideline interpreter can generate a new advisory quickly and pass it back to the client application.

Installation

We bootstrapped each client by manually installing a single file that acted as a link to an initialization file on the server. Thus, no code or configuration information was stored on the client machine. The server script informed the client application of the location of the guideline and database servers and mapped the drive on the server containing the application client code. The only other step was to install the client monitor as a system service on Windows. Installing the client monitor as a Windows system service ensured that the operating system kept it running. The monitor, in turn, ensured that the client application continues to run. In this way, we produced a fully automated mechanism for keeping the decision-support client running on a machine without further human intervention at the client

machine. Ideally, we would have used some sort of software distribution tool to perform our bootstrapping process remotely. However, such a tool was not enabled at the Palo Alto site when we performed our installation.

Our system has been running successfully for several months at one VA hospital and we anticipate that it will be running by April, 2001, in two additional facilities.

Discussion

Many problems addressed in this paper are similar to problems that must be addressed when developing Web-based client applications [11]. Such clients must also meet the constraints of rapid response time, minimal footprint, and possibly limited resources on the client machine.

Many Web-based clients are developed as Java applets, which are downloaded and executed by client Web browsers. Since our entire system was developed using Java, we initially considered deploying our client as a Java applet, which would have dramatically simplified some of the installation, configuration and maintenance issues. Unfortunately, the startup time of Java applets is considerable, particularly on low-end machines. Even simple applets can take ten seconds or more to start. This initial delay is not acceptable for a decision-support system requiring an almost instantaneous response time. As Java VMs improve their performance, the applet approach may become a viable means of client deployment, thus simplifying the process somewhat. Most Java applications can be converted to run as applets relatively easily. In any case, the overall deployment methodology would be similar.

Acknowledgements

This work has been supported, in part, by grant LM05708 from the National Library of Medicine, and by a grant from FastTrack Systems, Inc., and by VA grants HSR&D CPG-97-006 and RCD-96-301.

We thank Valerie Natale for her valuable editorial comments.

References

1. Musen MA, Tu SW, Das AK, and Shahar Y. *EON: A Component-Based Approach to Automation of Protocol-Directed Therapy*. Journal of the American Medical Informatics Association, 1996; 3(6): 367-388.
2. Tu, SW and Musen, MA. *From Guideline Modeling to Guideline Execution: Defining Guideline-Based Decision-Support Services*. Proceedings of the AMIA Annual Symposium. Los Angeles, CA, November, 2000; 863-867.
3. Shankar, RD, Martins, SB, Tu, SW, Goldstein, MK, Musen, MA. *Building an Explanation Function for a Hypertension Decision-Support System*. Medinfo 2001, London, UK, submitted.
4. O'Connor, MJ, Tu, SW, and Musen, MA. *Representation of Temporal Indeterminacy in Clinical Databases*. Proceedings of the AMIA Annual Symposium. Los Angeles, CA, November, 2000, 615-619.
5. O'Connor, MJ, Grosso, WE, Tu, SW, and Musen, MA. *RASTA: A Distributed Temporal Abstraction System to Facilitate Knowledge-Driven Monitoring of Clinical Databases*. MedInfo 2001, London, UK, 2001, submitted.
6. Musen MA, Gennari JH, Eriksson H, Tu SW, and Puerta AR. *PROTEGE II: Computer Support for Development of Intelligent Systems from Libraries of Components*. MedInfo 1995, Vancouver, B.C., Canada, 1995; 766-770.
7. Goldstein MK, Hoffman BB, Coleman RW, Musen MA, Tu SW, Advani A, Shankar R, and O'Connor M. *Implementing Clinical Practice Guidelines while taking account of Changing Evidence: ATHENA DSS, an Easily Modifiable Decision Support System for Managing Hypertension in Primary Care*. Proceedings of the AMIA Annual Symposium. Los Angeles, CA, November 2000; 300-304.
8. National High Blood Pressure Education Program. *The Sixth Report of the Joint National Committee on Prevention, Detection, Evaluation and Treatment of High Blood Pressure*. 1997, National Institutes of Health, Washington, D.C.
9. Advani, A, Tu, SW, O'Connor, MJ, Coleman, R, Goldstein, MK, and Musen, MA. *Integrating a Modern Knowledge-Based System Architecture with a legacy VA Database: The ATHENA and EON Projects at Stanford*. Proceedings of the AMIA Annual Symposium. Washington, DC, November, 1999; 653-657.
10. Masys, DR and Baker, DB. *Protecting Clinical Data on Web Client Computers: the PCASSO Approach*. Proceedings of the AMIA Annual Symposium. Los Angeles, CA, November, 1997, 340-343.
11. Hripcsak G, Cimino JJ, and Sengupta S. *WebCIS: Large Scale Deployment of a Web-based Clinical Information System*. Proceedings of the AMIA Annual Symposium. Los Angeles, CA, November, 1999, 804-808.