Using Software Agents to Maintain Autonomous Patient Registries for Clinical Research

Shawn N. Murphy MD, Ph.D., Usman H. Rabbani, and G. Octo Barnett MD
Laboratory of Computer Science
Massachusetts General Hospital, Boston, MA.

A software agent is an application that can function in an autonomous and intelligent fashion. We have used mobile software agents to maintain clinicians' patient research databases (patient registries). Agents were used to acquire data from the clinician and place it into the registries, copy data from hospital databases into the registries, and report data from the registries. The agents were programmed with the intelligence to navigate through complex network security, interact with legacy systems, and protect themselves from various forms of failure at multiple levels. To maximize the separation between our system and the hospital information infrastructure we often used Java, a platform-independent language, to program and distribute our software agents. By using mobile agents, we were able to distribute the computing time required by these applications to underutilized host machines upon which the registries could be maintained.

INTRODUCTION

Clinicians are attracted to teaching hospitals because large and varied patient populations offer important opportunities for clinical research. However, in the present health care environment of severe economic pressures, the opportunity to study and learn from this unique population is often lost in the hustle of everyday clinical practice. An important task for those clinicians who wish to pursue clinical research is to maintain a database (also called a registry) on a segment of the patient population. When clinical research projects are immature, or funding is tight, these registries must be personally maintained by the clinician. The registries often consist of clinical data elements collected with each patient encounter. Additional data may need to be collected and related to each patient encounter, often from laboratory data, demographic data, or other specialized hospital data. Ordinarily, the volume of clinical data to be collected for each encounter is not high, and much of the additional data is conceivably available through networked resources.

Although conceivably much of the demographic and laboratory data for the patient registries may be available through networked resources, it is difficult for the average clinician to exploit this avenue to information. Foremost are the intentional barriers placed by hospital administrators to prevent unrestricted data gathering from hospital databases. Additionally, the technical expertise necessary to construct computerized methods that can extract data from the proprietary laboratory and administrative databases is significant. Finally, many clinicians have such uncertain futures at their teaching institutions that investing time to link their personal registries to networked hospital resources is not practical.

In order to assist the research-oriented clinician in maintaining a registry, an institution could take various approaches. The institution could maintain a very large public registry and allow clinicians to define an unlimited number of personal fields within this public registry. This would allow links from these fields to some hospital data structures to exist. However, simple relational links to our legacy data structures do not exist. Furthermore, making the registry continuously available for data entry and query while ensuring proper isolation of each clinician's data would be a difficult task. Additional concerns regarding portability and ownership of their private fields may discourage the clinicians from using the public registry.

Another approach would be to help research clinicians maintain their personal registries as they exist now, registries located on the clinician's own computer in a wide variety of database formats. The clinician could query and view the database using familiar methods. The registries would not be integrated into a central registry, but rather would be serviced by programs from our laboratory. A similar concept has been presented for maintaining a departmental research database using the ARIS system[1]. However, our system differs in its use of agents to achieve a distributed architecture that can scale up more easily, and the fact that our system does not involve bringing the data from the personal registries into our possession.

1091—8280/97/$5.00 © 1997 AMIA, Inc.

AGENT ARCHITECTURE

Over the past year, we have been working with a software agent architecture for maintaining clinical research databases. An agent refers to intelligent software programs that are designed to act autonomously so as to realize a set of goals[2]. In our architecture, the goals are centered about maintaining a patient registry loosely coupled to the Electronic Medical Record (EMR). In our present trials we used patient registries that consisted of single Microsoft Access databases. Registries were maintained in the clinical workflow of Massachusetts General Hospital (MGH) physicians. Data entry from the point of care was managed with a standalone, non-mobile, non-Java software agent application which differed fundamentally from the mobile, Java software agents that also serviced the registries. Mobile Java software agents would travel to the computer upon which the registries were maintained, and perform a given function upon the registries at scheduled intervals. Data input into the registries was not amenable to this paradigm of scheduled servicing, and a direct path for data from the agent to the registry needed to be provided.

Several options existed to allow the clinician to input data into the registries. Data could be entered directly into the database from the desktop as with any Microsoft Access database. Data could also be entered over the Intranet from any location in the hospital using the MEDPhrase agent application. The MEDPhrase agent application has been previously described[3] and allows data entry into an autonomous database to be to incorporated into the workflow of clinical note-taking (Fig. 1). The MEDPhrase agent application allows enumerated data elements to be entered into the fields of the registries using a commercial World Wide Web (WWW) server and Common Gateway Interface (CGI). A third option for entering data into the registries was to use a custom WWW form with a WWW Browser such as Netscape. Both MEDPhrase and the WWW forms require a WWW server and CGI program to be purchased and maintained at the site of the patient registry.

During data entry one wants to enter an absolute minimum amount of data. For example, it is convenient during data entry to just enter a patient's medical-record number to identify the patient in the registry. However, when querying the patient registry, one often wishes to query by the name of the patient, since this is what one usually remembers. To not require a clinician to go through the extra step of entering the patient name, the function of a software agent might be to add the name of the patient to a predetermined field of the database based upon the medical-record number.

The goals of the Java software agents are database-centric, that is, they are coordinated solely by their orientation around a common database. Generally, they function independently, and thus the failure of one does not affect the operation of another. The agents are dispatched from a central computer to run on the host computer that actually contains the databases. The agent architecture is heavily based upon the Aglet system developed by International Business Machines (IBM)[4]. The Aglet system is written completely in Java. Agents are written as Java applications and distributed using Remote Method Invocation (RMI). RMI is an infrastructure supported in Java to serialize code and data classes and thus send applications from one computer to another. The application is run when it arrives at the host computer. Messaging and handshaking between agents and servers occurs using the Agent Transfer Protocol (ATP), also developed at IBM. This mobility allowed us to run the Java agents on the host computers at times when the host computers were known to be underutilized.

Once the Java software agent reaches its destination, it begins to run on the host computer. The Java software agents access data in the Microsoft Access registries with the Open DataBase Connectivity (ODBC) tools available from Microsoft, using bridges from ODBC to Java DataBase Connectivity (JBDC). With the use of these database drivers, the software agents can actually communicate with any ODBC compliant database on the host machine, as long as it is set up to use ODBC drivers. These databases can include Microsoft Excel or Oracle databases.

CLINICAL DATA INPUT INTO THE PATIENT REGISTRIES

A data entry agent provided the means of getting data from the point of care into the Microsoft Access patient registry. The MEDPhrase application was created as an agent that is used alongside the EMR. It aids the clinician with data entry by storing phrases and sentences that are typically used by the clinician. The physician chooses a phrase such as "The patient had a mild-moderate tremor in the right hand," and this phrase is written into the clinical note being actively created by the clinician. However, if this phrase happens to describe a physical finding that is being maintained in one of the clinicians patient registries, a WWW POST message is sent over the intranet to the web-server on the computer where the patient registry

resides (see fig. 1). Cold Fusion™ (Allair Corporation) is used as the CGI that places the data into the Microsoft Access patient registry. Some of the name-value pairs act as the primary key(s) and identify the records in the registry which should be updated or inserted. The other name-value pairs identify values that should be placed in named fields. For example, when the above sentence is written to the EMR, the value "2" would be placed in the field "RightHandTremor."

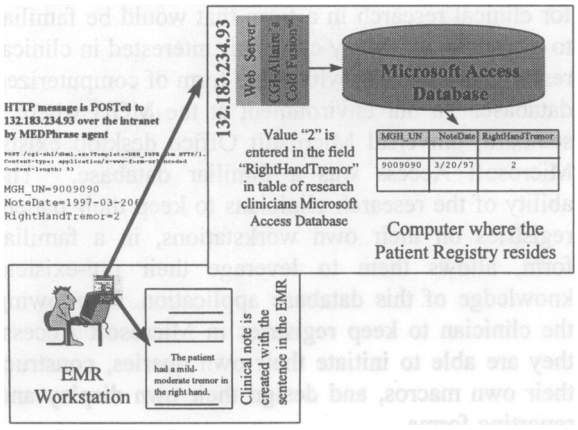

Figure 1. The physician uses a data entry agent at the EMR workstation to enter commonly used sentences into the text of the clinical note. When a sentence is used that corresponds to a data value that is to be entered into the clinician's patient registry, a message is sent over the Intranet to direct a value to be entered into the patient registry .

All sentences that are of interest to the patient registry need to be pre-associated with enumerated values, values that will be written to fields of the patient registry when the corresponding sentence is written to the EMR. This approach only works when the data that is to be collected for a patient registry has been pre-enumerated.

IMPLEMENTATION OF JAVA AGENTS

Two mobile Java agents were built and deployed to assist the research clinician with the maintenance of the patient registries. The Java software agents were distributed to run on the host machines periodically, usually every 24 hours. In order for the agents to know what data to expect in the registries, certain kinds of data needed to always be stored under specific field names within specific tables. The agents knew to always expect the patient's medical-record number in the field marked "MGH_UN." The date of the patient encounter was always present in a field labeled "NoteDate."

An agent was built that would query the MGH_UN field of each record in the registry, and then fill in the patient's last name, first name, middle initial, date of birth, and sex in other registry fields designated to receive this data. In order to perform this function, the agent is dispatched from the central computer to the host computer. The agent begins to run on the host computer and establishes links through JDBC-ODBC to the Microsoft Access patient registry. If a medical-record number is read from the MGH_UN field with a blank LastName field in the record, a socket connection is opened to a hospital server which responds with the patient's demographic data wrapped in a Health Level 7 (HL7) message. The message is then unwrapped and the fields of the patient registry are filled with the demographic information.

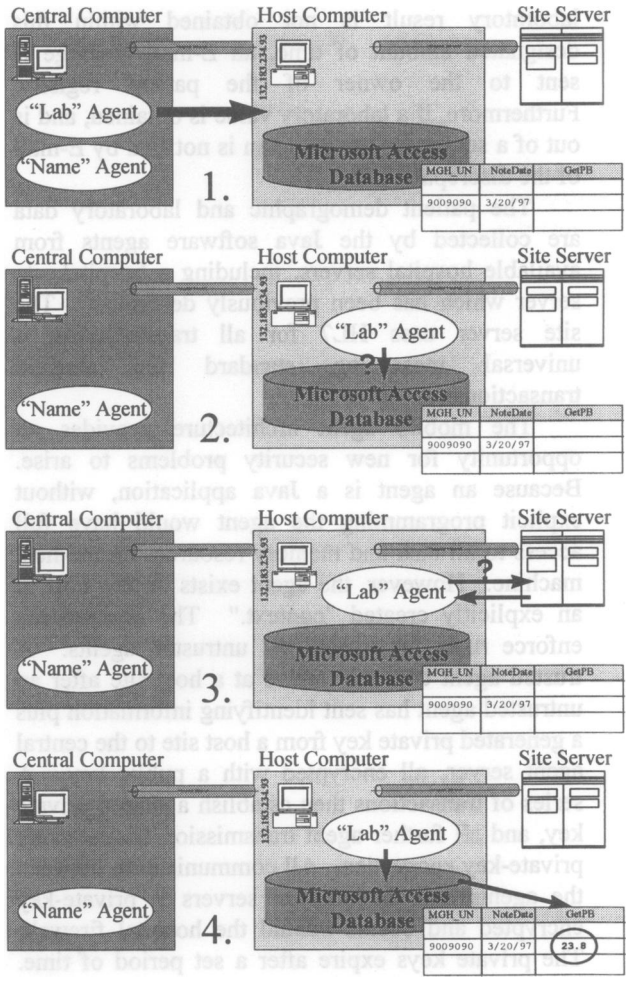

Figure 2. Life cycle of a mobile Java software agent that enters laboratory data into patient registries. (See text for explanation)

A second agent was built to check for laboratory data that was desired by the patient registry (see fig 2). First, the agent is dispatched from the central computer to the host computer (fig 2, step 1). The agent then queries the MGH_UN fields of the Microsoft Access patient registry to find the medical record number of patients in the database, and then queries the fields which are designated to receive laboratory values (fig 2, step 2). For example, the field name "GetPB" is used for fields that expect a phenobarbital level. The agent will then go to a hospital site server once every 24 hours and query for a phenobarbital level on this patient until one is obtained (fig 2, step 3). When the phenobarbital level is obtained, it is placed in the "GetPB" field of the proper patient record (fig 2, step 4). The agent stops looking for a phenobarbital level after a certain amount of time has elapsed from the date in the NoteDate field, usually one week. If an expected laboratory result is not obtained within this designated amount of time, an E-mail message is sent to the owner of the patient registry. Furthermore, if a laboratory value is obtained, and is out of a set range, the clinician is notified by E-mail of the discrepant data.

The patient demographic and laboratory data are collected by the Java software agents from available hospital servers, including a hospital site server which has been previously described[5]. The site server uses HL7 for all transmissions, a universal messaging standard for medical transactions.

The mobile agent architecture provides an opportunity for new security problems to arise. Because an agent is a Java application, without explicit programming the agent would have full access to all disk and memory resources on the host machine. However, the agent exists on the host in an explicitly created "context." The context can enforce rules for trusted vs. untrusted agents. A trusted agent can only arrive at a host site after an untrusted agent has sent identifying information plus a generated private key from a host site to the central agent server, all encrypted with a public key. A series of transactions then establish a shared private key, and all further agent transmission occurs using private-key encryption. All communication between the agents and the hospital servers is private-key encrypted and occurs behind the hospital firewall. The private keys expire after a set period of time. To ensure that only authorized clinicians have access to patient data through the agents, the agents are set up using the same accounts used by the hospital to access patient data. Automatic delivery of data to the patient registry is only initiated through human negotiation. Several database attributes are checked to verify the authenticity of the database with every transaction. Finally, the transaction is compared to past transactions to alert us with an inconsistency within a request, which stops the transaction until we have a chance to review it.

DISCUSSION

Our objective was to maintain patient registries for clinical research in a form that would be familiar to the clinician. Many clinicians interested in clinical research are familiar with some form of computerized database. In our environment at the MGH where a standard, universal Microsoft Office desktop exists, Microsoft Access was a familiar database. The ability of the research clinicians to keep their patient registries on their own workstations, in a familiar form, allows them to leverage their pre-existent knowledge of this database application. In allowing the clinician to keep registries in Microsoft Access, they are able to initiate their own queries, construct their own macros, and design their own display and reporting forms.

The implementation of software agents provided a way to maintain patient registries that were loosely coupled to the MGH patient-care databases. A data entry agent provided the means of getting data from the point of care into the patient registries, and mobile Java agents provided laboratory and demographic data for the registries.

The mobile Java agents provided an avenue for us to transfer the complex network transaction technology that our laboratory has developed to desktops throughout the hospital. While doing so, we retained control over this technology since our mobile agents are destroyed when they complete their transactions. The agents have the ability to handle complex security negotiations with the hospital's information servers, a strategy that is unavailable with simple relational database links. The agents are able to ensure patient data privacy and security at many levels.

The key to enabling software agents is the existence of a universal infrastructure to support them. Clearly the idea of distributing and running applications on several independent machines is not new. Theory of distributed software architecture has been developed for over 15 years[6]. Furthermore, distributed architectures have been applied in many computing environments, including our own Forte environment[7]. The advantages of distributed architectures are 1) the ability to optimize computational load distribution, and 2) the ability to

handle the failure of one computer with relatively little impact to the system's functioning as a whole. The disadvantages of distributed environments stem mostly from the high cost of development of such systems.

The Java language has been developed by SUN Microsystems for (distributed) computing within an Internet browser. In our implementation, we use the Java Virtual Machine to run our agents as platform independent applications. The shouldering of the development costs of this distributed environment by the major software manufacturers has been the key to the success of the distributed Java programs.

Our mobile agent system allows the computational burdens of Java and intelligent agents to be distributed to many host computers. Future operating systems are being designed around the Java language. It is likely that computer hardware will be available in the near future to support the Java language.

The Java language suffers from several disadvantages over other well know third generation languages. It is fairly slow in comparison to a program compiled with a Microsoft Visual C++ compiler. The user interfaces available in Java are not as sophisticated as those interfaces that may be constructed using the full set of application program interface calls available in native systems. Development tools have been slow to achieve the sophistication available with applications written in other languages.

We have applied our agent architecture within a very small sphere. Similar approaches are being explored to maintain databases for clinical guidelines, clinical alerts, and operation improvement. Agents by definition are easily extensible and customizable. Most of the hard work involves setting up the secure infrastructure within which they may operate freely. With a secure infrastructure, agents can evolve gradually and seamlessly into an age where they could carry knowledge bases and guideline logic. Finally, with the development of standards for the context's and messaging of the agents (such as HL7), agents could be used to transport knowledge throughout the healthcare system.

FUTURE DIRECTIONS

Agents to be written within the next few months include evaluation agents, the task of which will be to report the use and state of the registries (unused for several weeks, extensive manual changes, or extensive empty elements). This may help provide a measure of the usefulness of the agents we have described.

Acknowledgments

The authors thank Greg Estey and Henry Chueh for enlightening discussions and ongoing advice. This work was supported in part by Training Grant NLM LM 07092 and in part by NLM research grant LM05854 and AHCPR grant HS06575. An equipment grant was provided by Hewlett Packard.

References

1. Timmers, T., Pierik, F., Steenberger, M. et al. ARIS: Integrating Multi-source Data for Research in Andrology. In Gardner RM, ed. *Proceedings of the Nineteenth Annual Symposium on Computer Applications in Medical Care*, pp. 445-8, 1995.
2. Maes, P. ed. Designing Autonomous Agents, Cambridge, MA: MIT Press. 1990.
3. Murphy, S.N. and G. O. Barnett (1996). Achieving Automated Narrative Text Interpretation Using Phrases in the Electronic Medical Record. In Cimino J.J, ed., JAMIA, Symp. Sup, pp. 532-6, 1996.
4. Lange, D. and Chang, D. IBM Aglets Workbench, Programming Mobile Agents in Java, 1996. http://www.ibm.co.jp/aglets/whitepaper.htm.
5. Wingerde, F.J. van, J. Schindler, P. Kilbridge et. Al. Using HL7 and the World Wide Web for Unifying Patient Data from Remote Databases. In Cimino J.J, ed., JAMIA, Symp. Sup, pp. 643-7, 1996.
6. Carriero, N. and Gelernter, D. Coordination languages and their significance. Communications of the ACM, Vol. 35, No. 2, pp. 97-107, Feb. 1992.
7. Chueh, H.C., Raila, W.F., Pappas, J.J. et al. A Component-Based, Distributed Object Services Architecture for a Clinical Workstation. In Cimino J.J, ed., JAMIA, Symp. Sup, pp. 638-2, 1996.