Title: Integrating Multiple Clinical Information Systems using the Java Message Service Framework

Authors:

Wyatt M. Tellis
University of California San Francisco

Katherine P. Andriole, Ph.D.
University of California San Francisco

Primary Author's Contact Information:

Address:

Wyatt Tellis
Laboratory for Radiological Informatics
University of California San Francisco
530 Parnassus Ave., Room CL-158
San Francisco, CA 94143-0628

Phone: (415) 502-0213
Fax: (415) 502-3217
Email: wyatt.tellis@radiology.ucsf.edu

Paper Type: Technical note

Note: Full paper for SCAR 2003 abstract #29

ABSTRACT:
Title: Integrating Multiple Clinical Information Systems using the Java Message Service Framework

Purpose: This paper describes an application for capturing, delivering and tracking urgent radiology exam results. Urgent exam findings are entered using a web form embedded within the PACS display station. The findings are then accessible via softcopy using either the PACS display stations, HIS terminals, or wireless enabled PDAs or via hardcopy printouts that are generated automatically or on demand. Additionally quality control is performed on those findings entered by radiology residents or fellows, the results of which are used for both performance tracking and educational activities.

Methods: The application was developed using Sun Microsystems' Java programming language. The Java Message Service (JMS) was used to manage the delivery of findings. JMS provides a robust, flexible framework for exchanging messages between disparate applications.

Results: The application is now used for all urgent exams; completely replacing the original paper based system. The use of JMS provides the necessary level of reliability needed by this application.

KEYWORDS:
1) Integration
2) Java Message Service (JMS)
3) Urgent exams
4) PACS
5) Clinical Information Systems
6) RIS
7) HIS
8) PDA

BACKGROUND:

Integration of clinical information systems can facilitate radiology and urgent care interdepartmental communications, and contribute to improved quality of patient care in the emergency department by delivering medical information at the point-of-care. It was decided that to enable the rapid delivery of urgent exam results at the authors' institution, the old paper based "wet-read" process would be replaced with an electronic system. In the old process the radiologist wrote the findings on an exam requisition form and faxed it to the emergency department (ED). This process was fraught with problems, including illegible handwriting and lost faxes. The new application is directly integrated with the PACS display stations where the radiologists use an embedded web form to type in their findings. To minimize the need to reengineer ED workflow during the initial phase of the application's deployment, the wet-reads are printed out using networked printers located in the ED. During the second phase, these printouts will be supplemented with networked PDAs (personal digital assistants), which can receive the exam results over a WLAN (wireless local area network). Additionally the wet-readings were passed on to the HIS (hospital information system) where they are available from any HIS terminal. As well as electronically capturing wet-reads, the new application contains an educational component for radiology attendings to review and comment on wet-reads entered by radiology residents or fellows. These comments are automatically emailed to the resident or fellow and any discrepancies between the wet-read and final report are recorded and presented at the monthly resident Q/A conference.

METHODS:

The application was developed entirely in the Java programming language from Sun Microsystems (Menlo Park, CA). Java was used because of its built-in support for databases through the Java Database Connectivity (JDBC) framework, its support for web-based applications through the Java servlet and Java Server Pages APIs (Application Programming Interfaces), and its object oriented nature, which greatly facilitates application design. Development was done using the freely available NetBeans (http://www.netbeans.org/) IDE (Integrated Development Environment). The application consists of three major components: a database server, a web server, and a Java Message Service (JMS) provider. An overview of the application's architecture is presented in Figure 1. The individual components are described in detail below. It should be noted that an IRB Exempt Certification was obtained before the system was placed into clinical operation.

The database server used for this project was PostgreSQL version 7.2 (http://www.postgresql.org/) running on Sun Solaris 8. PostgreSQL is an open-source ANSI (American National Standards Institute) SQL92 compliant relational database that runs on a variety of UNIX based platforms (including Linux). PostgreSQL was chosen for this project because it was the open-source database with the largest installed base and developer community that supported the advanced database features necessary for ensuring the logical integrity of the data being stored; a mandatory requirement imposed by the need to manage clinical data. Such features include triggers, which permit the execution of database functions when data is inserted, updated or deleted from the database, foreign key constraints, which manage related data across multiple tables within the database, and ACID (atomicity, consistency, isolation, and durability) transactions, which allow for the grouping of data inserts, updates or deletes.[5]

The web based component of this application used the servlet and JSP (JavaServer Pages) technologies from Sun Microsystems. A servlet is a lightweight Java application that runs within a web server. Servlets are usually used to generate dynamic HTML pages, though they can generate any type of content including images and XML (extensible markup language) documents. Servlets have full access to the entire Java API, making them a very powerful tool for creating web-based applications. JSP is an extension of the Java servlet technology that allows a web application developer to embed Java code directly within HTML to create a JSP page. Just before the JSP page is sent to a browser, the web server executes the Java code embedded within it and combines the result with the static HTML to produce the final document. JSP pages are used in place of servlets because they make it easier to write dynamic HTML content by allowing the developer to directly embed Java code within the HTML code of the page. Additionally JSP pages can use a “tag library” in place of Java code. A tag library is a collection of HTML-like tags that encapsulate the Java code that generates the content for the page. These tags can perform functions such as generating an HTML table from the results of a database query without requiring any Java code to be included within the JSP page itself. Since JSP tags are reusable, the logic they contain can be included on multiple JSP pages, thereby eliminating duplicate Java code. Both the servlet and JSP APIs are defined by a set of specifications issued by Sun Microsystems which can be found at: http://java.sun.com/products/servlet/ and http://java.sun.com/products/jsp/ respectively.

In order to function, all servlets and JSP pages must be hosted within a “servlet container” that is responsible for providing the environment needed for the servlet or JSP page to run. This includes handling all network communication between the client browsers and the servlet or JSP page, parsing incoming requests from client browsers and directing them to the appropriate servlet or JSP page, as well as managing client sessions. The container used for this project was Tomcat version 4.1.12 (http://jakarta.apache.org/tomcat/) running on Windows 2000 Server Edition (Microsoft, Redmond, WA). Tomcat is an open-source, Java servlet and JSP container developed under the auspices of the Jakarta Project of The Apache Software Foundation.

Input of findings and reviews takes place from the PACS display stations using web-based forms embedded within the display stations' GUI (graphical user interface). The forms are generated by a set of Java servlets and JSP pages using data from the PostgreSQL database. The PACS display stations used at this institution are IMPAX R4.1 displays (AGFA Medical Imaging, Ridgefield Park, NJ) running on Windows 2000 Professional Edition. Integration with the PACS display makes use of two, vendor specific APIs: the "context_server" and the "script button" scripting language. The context_server is a hidden Windows program that runs on each PACS display. The context_server provides functionality similar to HL7's CCOW (Clinical Context Object Workgroup) standard (http://www.hl7.org/special/committees/visual/visual.cfm). It acts as a broker between the IMPAX display software and other clinical desktop applications and enables these applications to share the same clinical context. For example the context_server allows notification of a third party desktop application of when a user logs into IMPAX or when a study is displayed. Communication between IMPAX, the context_server, and any third party applications takes place through COM (Component Object Model), a Microsoft API for integrating Windows applications.[3] The script button scripting language enables a third party developer to add customized buttons to the display station's GUI. The scripting language is specific to the IMPAX software and allows the developer to obtain information about the current clinical context (i.e. login name, patient name and medical record number associated with the study being displayed, etc.) from the context_server as well as launch applications such as a web browser. For this application several script buttons were created to capture the name of the current user along with the UID (unique identifier) of the study being displayed and launch a web browser (Internet Explorer version 5.5, Microsoft, Redmond, WA) using a URL (uniform resource locator) containing this information. The URL points to a JSP page, which uses the information contained in the URL to dynamically generate an HTML page to send back to the browser.

The Java Message Service (JMS) provider is the final component of this project. JMS is a vendor agnostic API from Sun Microsystems for enterprise messaging. "An enterprise message system, or Message-Oriented-Middleware (MOM), allows two or more applications to exchange information in the form of messages."[2] In JMS nomenclature a MOM is considered a "provider" and applications that send and/or receive messages are referred to as "clients". Since JMS is independent of the MOM vendor, a developer can replace one provider with another without having to completely rewrite their application. A key feature of any provider is support for asynchronous message delivery, where a message sender does not have to wait for the message to be received but instead can continue processing immediately after sending a message. Instead the sender relies on the provider to ensure the message will be delivered to the appropriate recipients. The provider is responsible for handling any software or hardware failures that may occur while attempting to deliver the message. In the event of a failure, the provider is expected to retain the message in a "persistent store" (either a database or file) for redelivery. The provider will attempt to redeliver the message until either it is successfully sent or a message specific timeout condition is triggered. Consequently each message must be a self-contained package with enough state information and data for the recipient to process it. The

JMS specification places no restrictions on the structure of a message. A message can contain text, a mixture of text and other data types such as numbers or dates, or it can consist of an opaque array of bytes.[1, 2]

The JMS API defines two messaging architectures: the publish-and-subscribe (pub/sub) model and the point-to-point (PTP) model. In the pub/sub model a single client will publish to a messaging channel called a "topic" at which point the provider will deliver the message to all recipients subscribed to that topic. In the PTP model a single client sends a message to a "queue" which the provider then sends to a single receiver listening on that queue. In addition to filtering by topic or queue, a message receiver can use "selectors" to further filter incoming messages based upon each message's properties and payload.[1, 2]

Initially OpenJMS version 0.7.4 (http://openjms.sourceforge.net/), a freely available, open-source JMS provider, was used, however, subsequent versions of the application rely on Softwired's (Zurich, Switzerland) iBus//MessageServer version 4.5.2. iBus//MessageServer was used in place of OpenJMS because it is closely integrated with iBus//Mobile Gateway (Softwired, Zurich, Switzerland), which acts as a proxy between the JMS provider and mobile JMS clients. iBus//Mobile Gateway provides an extra layer of robustness for handling the volatile nature of mobile device connections as well as the ability to connect to a variety of mobile devices including PDAs, pagers and cell phones using a multitude of protocols such as TCP/IP, SMS (short message service), or WAP (wireless application protocol). Both the iBus//MessageServer and iBus//Mobile Gateway were deployed under Windows 2000 Server Edition.

The PDA used for initial development is the Sharp (Mahwah, NJ) Zaurus 5500 outfitted with an 802.11b ("Wi-Fi") CompactFlash card, model DCF-650W (D-Link Systems Inc., Irvine, CA). The Zaurus runs a version of the Linux kernel designed for embedded and mobile devices. In addition the PDA ships with the Insigna (Fremont, CA) Jeode JVM (Java Virtual Machine). The Jeode JVM conforms to the PersonalJava API version 1.2, a subset of the standard Java API optimized for PDAs (http://java.sun.com/products/personaljava/).

RESULTS:

Input of a wet-reading from a PACS display station makes use of the web-based form pictured in Figure 2. The form is accessed using a script button embedded within the PACS display station's GUI. The form automatically contains the current clinical context such as the name and medical record number of the patient being displayed, the exam's accession number and description, as well as the name of the current user. The form allows the radiologist to record any additional clinical history as well as the findings and with whom they discussed the findings. Once a wet-reading has been entered using this form, the web server converts it into a JMS message and sends it to a JMS topic on the JMS provider (see Figure 1). The provider then distributes the same message to any clients subscribed to the topic. In the case of a new wet-reading submission, the JMS message is sent to the printing client, the HL7 client, and the iBus//Mobile Gateway. To ensure the wet-reading message is fully self-contained all the clinical and demographic data associated with the wet-reading is stored within the message. Each client is responsible for interacting with a particular external system and therefore processes the same message differently. In the case of the printing client, the message is used to generate the printout pictured in Figure 3. The printouts are sent to networked printers in the emergency department. The HL7 client converts the message into an HL7 message, which is used to notify the RIS of the availability of the wet-reading. The iBus//Mobile Gateway distributes the message to JMS clients running on the networked PDAs, which then present an alert to the user (see Figure 4). The PDA JMS clients use JMS selectors to filter out those messages that are not relevant to the user. The selectors allow only those messages whose requesting physician name matches that of the PDA owner to be accepted by the JMS client and to trigger an alert.

Confidential Q/A reviews of wet-readings are also entered from the PACS display stations using the form pictured in Figure 5. The reviews are performed by radiology attendings and are done on those wet-reads entered by radiology fellows or residents. The review tracks whether a questionable, minor, or major discrepancy between the wet-reading and final report exists as well as whether the discrepancy affects short or long term patient management. If a discrepancy is found or if the attending has any comments an email is automatically sent to the resident or fellow. If an email is to be sent, the web server creates a JMS message, which it sent to the JMS provider and delivered to the email JMS client. When the email JMS client receives this message, it attempts to use the radiology department's email server to relay the wet-read review on to the email recipient. In addition the reviews can be printed out using the same JMS mechanism as the wet-read printouts.

DISCUSSION:

The built-in error recovery mechanisms of JMS have allowed the application to gracefully deal with a variety of error conditions, including printer and network failures, and email and RIS downtimes. As of this writing the printer, email and HL7 clients of the application have been active for approximately six months. The printer and HL7 clients have been processing approximately 80 messages per day, with the email client processing approximately 5 messages per day. During this time no messages were lost, despite multiple occurrences of each failure condition mentioned above. The PDA piece is still under active development, so no performance and reliability statistics are available at this time, though development testing, such as simulated JMS provider, gateway and network outages, has shown that the iBus//Mobile Gateway is able to handle the types of network failures and bandwidth fluctuations usually associated with Wi-Fi connections. Having this level of robustness was a requirement for acceptance by the radiology faculty and integration into the interpretation workflow. Given that the application has completely replaced the original fax based wet-read notification process, it now being used for other non-ED, urgent cases, such as those from the "pre-op" and screening clinics.

The vendor independent nature of the JMS API also proved important during development of this application. During the first stages of development the OpenJMS provider was used. The subsequent migration to the iBus//MessageServer provider was seamless, and only required changes to the JMS clients' configuration instead of modification of each client's Java code. Development time was also reduced by the application of JMS selectors and the use of the pub/sub architecture. For example, a single topic was used to handle all wet-read JMS messages. Each JMS client subscribed to the topic (the printer client, HL7 client, and PDA clients) used selectors to decide whether to process the message. The web server, which generates the JMS messages, need only set flags on the messages indicating whether it wants them to be printed, sent to the RIS, or distributed to the PDAs. All the machinery required to route these messages is contained within the JMS provider, allowing the programmer to focus development effort on the application's business logic and GUI design.

CONCLUSION:

This application has completely replaced the use of the fax based wet-read notification process. The ability for the JMS infrastructure to recover from network and server failures, has allowed the application to process approximately 15,000 wet-reads over a span of six months without a single loss. Additionally the switch from one JMS provider, OpenJMS, to another, iBus, without requiring a complete redesign of the application is evidence of the JMS framework's vendor agnostic design and platform independence. This project has shown that JMS is one potential solution for integrating clinical systems.

ACKNOWLEDGMENTS:

This work is supported by the SCAR 2002 research grant in imaging informatics.

REFERENCES:

1. Hapner M, Burridge R, Sharma R, Fialli J, Hasse K (2002) Java Message Service API Tutorial and Reference: Messaging for the J2EE Platform, Addison-Wesley, San Francisco, California

2. Monson-Haefel R, Chappell DA (2001) Java Message Service, O'Reilly & Associates Inc., California

3. Shepard G, King B (1999) Inside ATL, Microsoft Press, United States of America

4. Tellis WM, Andriole KP, Jovais CS, Avrin DE (2002) RIS Minus PACS Equals Film, J Digit Imaging, 15: 20-26

5. Ullman JD, Widom J (1997) A First Course in Database Systems, Prentice Hall, New Jersey

LEGENDS:

Figure 1 – Overview of system architecture

Figure 2 – Web form used for entering wet-reading from PACS display station

Figure 3 – Sample printout sent to emergency department

Figure 4 – PDA GUI screens

Figure 5 – Web form used for entering Q/A review of wet-reading

FIGURE 1

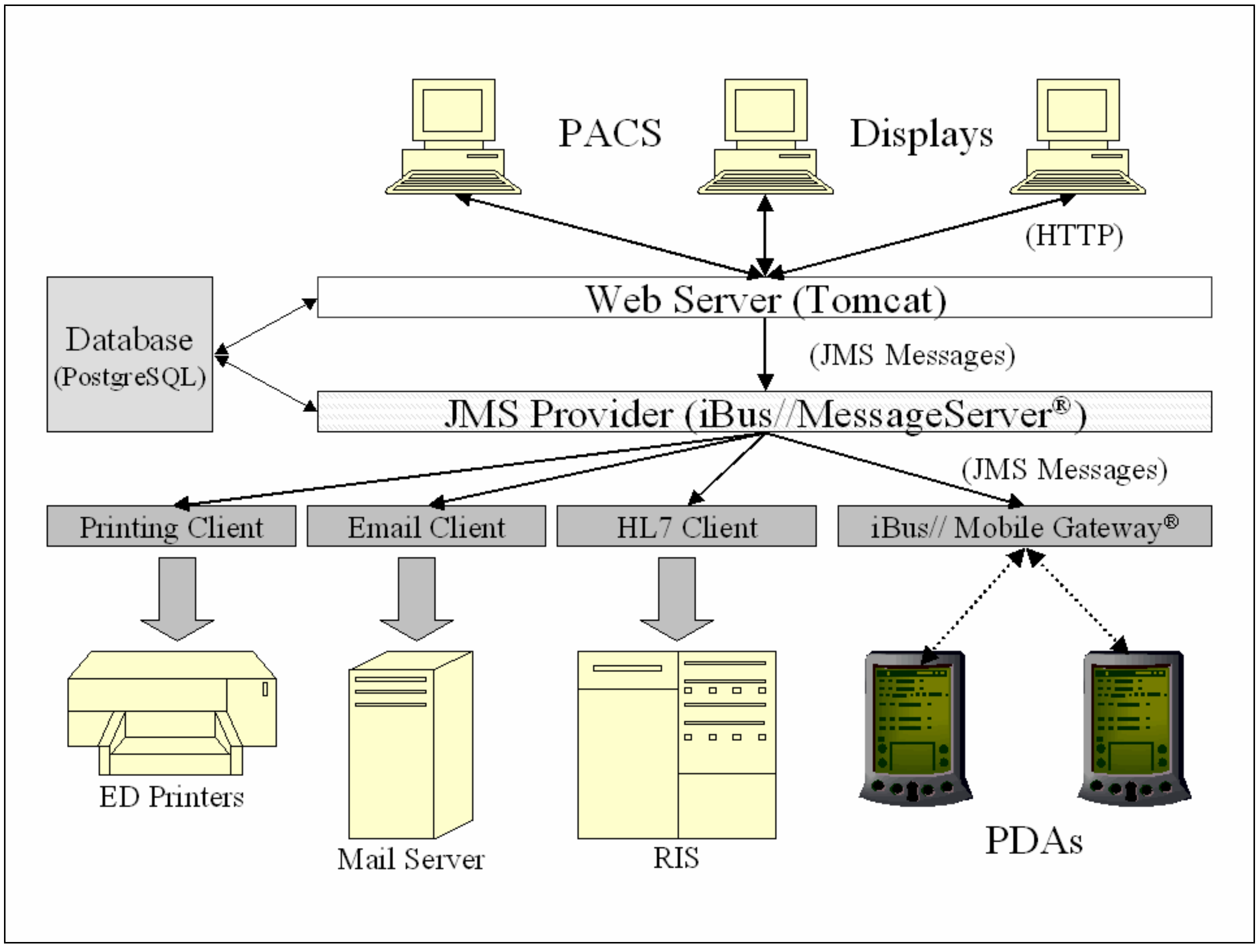
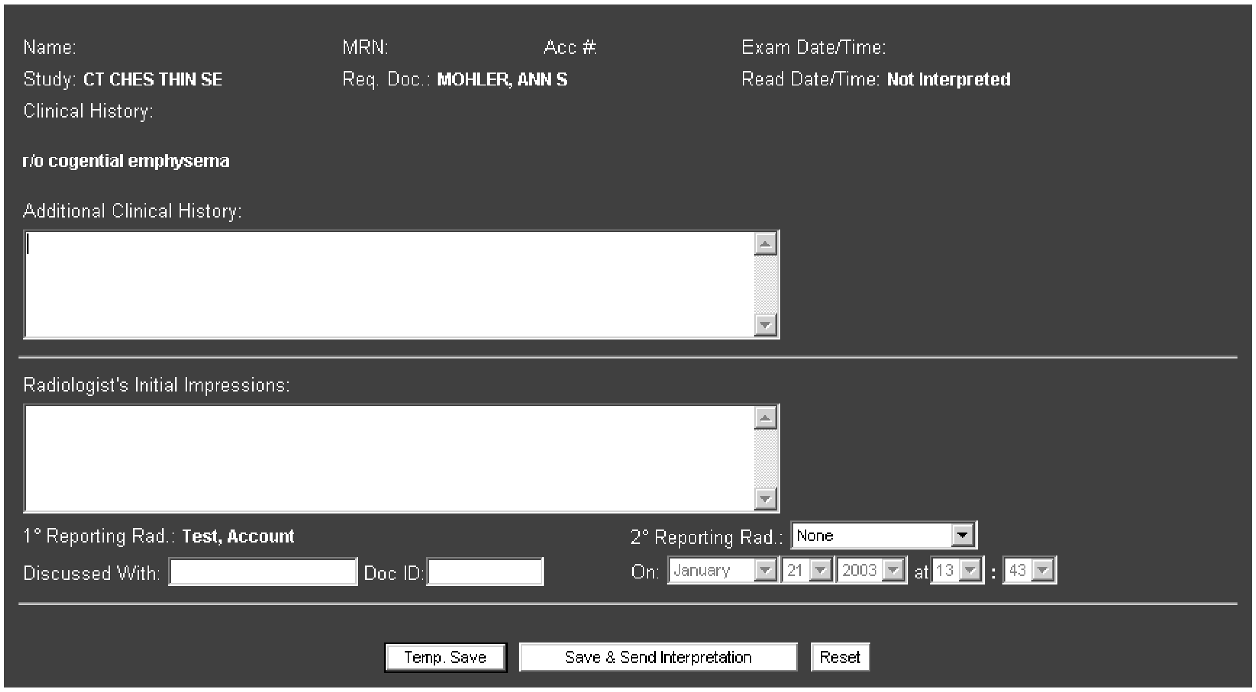

PACS
Displays
(HTTP)
Database
(PostgreSQL)
Web Server (Tomcat)
(JMS Messages)
JMS Provider (iBus//MessageServer®)
(JMS Messages)
Printing Client
Email Client
HL7 Client
iBus// Mobile Gateway®
ED Printers
Mail Server
RIS
PDAs

FIGURE 2


Name: MRN: Acc #: Exam Date/Time:

Study: **CT CHES THIN SE** Req. Doc.: **MOHLER, ANN S** Read Date/Time: **Not Interpreted**

Clinical History:

r/o cogential emphysema

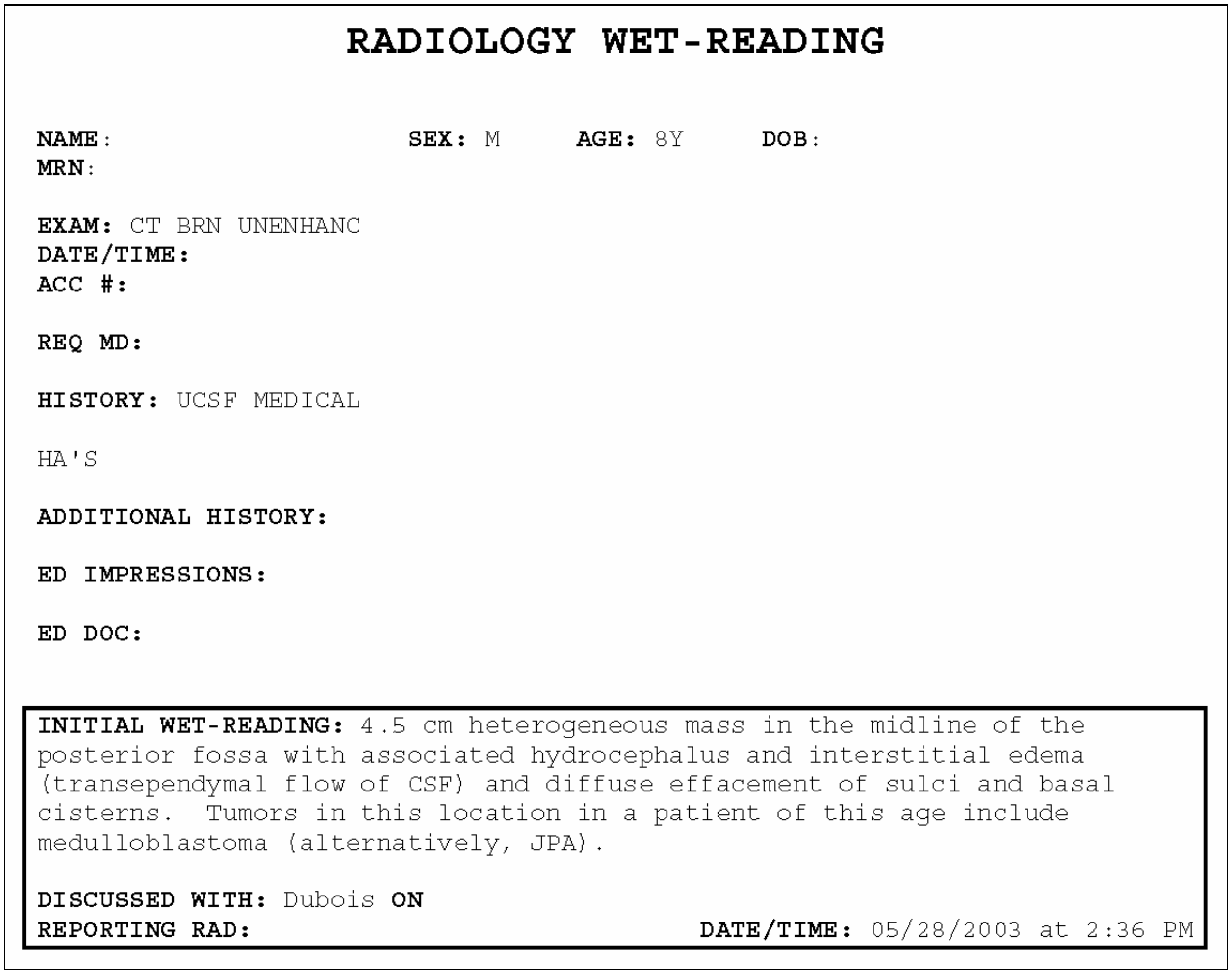
Additional Clinical History:

Radiologist's Initial Impressions:

1° Reporting Rad.: **Test, Account** 2° Reporting Rad.: None

Discussed With: Doc ID: On: January 21 2003 at 13 : 43

FIGURE 3



RADIOLOGY WET-READING

NAME: **SEX:** M **AGE:** 8Y **DOB**:
MRN:

EXAM: CT BRN UNENHANC
DATE/TIME:
ACC #:

REQ MD:

HISTORY: UCSF MEDICAL

HA'S

ADDITIONAL HISTORY:

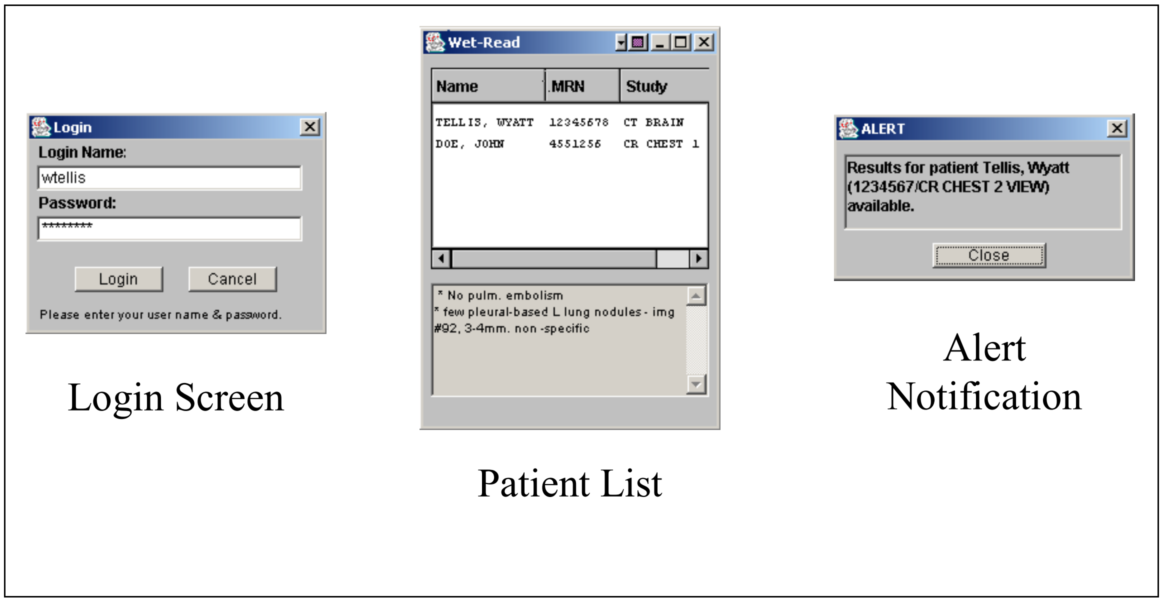
ED IMPRESSIONS:

ED DOC:

INITIAL WET-READING: 4.5 cm heterogeneous mass in the midline of the posterior fossa with associated hydrocephalus and interstitial edema (transependymal flow of CSF) and diffuse effacement of sulci and basal cisterns. Tumors in this location in a patient of this age include medulloblastoma (alternatively, JPA).

DISCUSSED WITH: Dubois **ON**
REPORTING RAD: **DATE/TIME:** 05/28/2003 at 2:36 PM

FIGURE 4



Please enter your user name & password.

Login Screen

Name	MRN	Study
TELLIS, WYATT	12345678	CT BRAIN
DOE, JOHN	4551256	CR CHEST 1

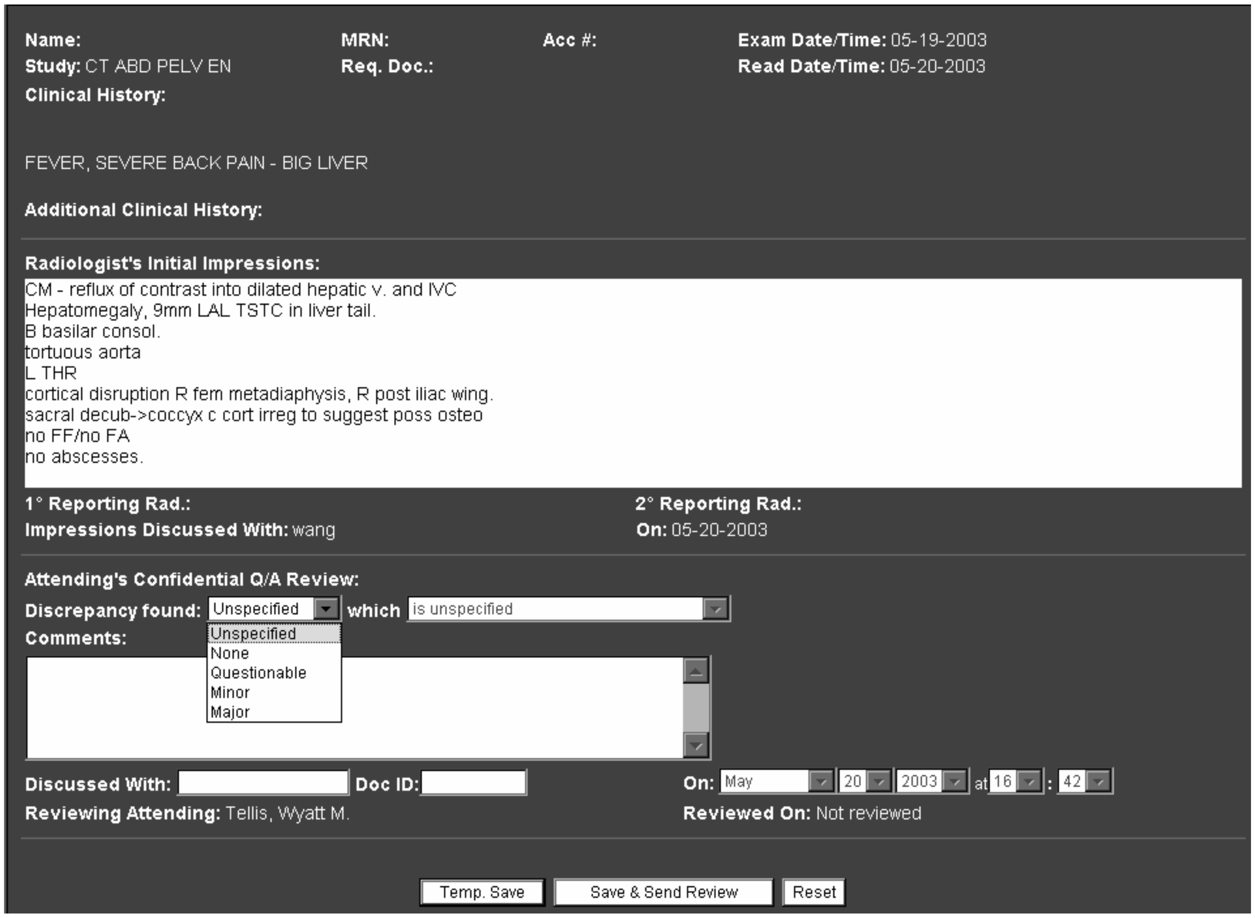
* No pulm. embolism
* few pleural-based L lung nodules - img #92, 3-4mm. non -specific

Patient List

Results for patient Tellis, Wyatt (1234567/CR CHEST 2 VIEW) available.

Alert Notification

FIGURE 5



Name: **MRN:** **Acc #:** **Exam Date/Time:** 05-19-2003
Study: CT ABD PELV EN **Req. Doc.:** **Read Date/Time:** 05-20-2003
Clinical History:

FEVER, SEVERE BACK PAIN - BIG LIVER

Additional Clinical History:

Radiologist's Initial Impressions:
CM - reflux of contrast into dilated hepatic v. and IVC
Hepatomegaly, 9mm LAL TSTC in liver tail.
B basilar consol.
tortuous aorta
L THR
cortical disruption R fem metadiaphysis, R post iliac wing.
sacral decub->coccyx c cort irreg to suggest poss osteo
no FF/no FA
no abscesses.

1° Reporting Rad.: **2° Reporting Rad.:**
Impressions Discussed With: wang **On:** 05-20-2003

Attending's Confidential Q/A Review:
Discrepancy found: Unspecified **which** is unspecified
Comments:

Discussed With: **Doc ID:** **On:** May 20 2003 at 16 : 42
Reviewing Attending: Tellis, Wyatt M. **Reviewed On:** Not reviewed