Benefits of Distributed HIS/RIS-PACS Integration and a Proposed Architecture

Robb G. Keayes and Len Grenier

The tight integration of the Hospital Information System/Radiology Information System (HIS/RIS) and the Picture Archive Communication System (PACS) has become a priority in modern healthcare delivery systems. Traditional paper-based systems are being replaced by gateway style interface engines. Gateways provide a tightly integrated link between the HIS/RIS and the PACS, increasing productivity by automating many mundane clerical tasks associated with paper-based systems. A centralized gateway, however, represents a processing bottleneck and single point of failure. A self-monitoring distributed gateway architecture that replicates essential services increases the fault tolerance and the overall availability of the gateway, while providing reduced, consistent transaction times. A possible distributed architecture is proposed as a means to realize the advantages of a distributed architecture.

Copyright © 1997 by W.B. Saunders Company

KEY WORDS: HIS/RIS, HL7, PACS, DICOM, distributed systems, interface engine.

UNTIL RECENTLY, there has been little support for interfacing HIS/RIS and PACS systems. Instead, paper-based systems or ad hoc point-to-point interfaces were used to aid workflow and manage information. There is growing consensus that tight Hospital Information System (HIS)/Radiology Information System (RIS)-Picture Archive Communication System (PACS), HIS/RIS-PACS, integration can provide substantial cost and productivity benefits to the delivery of diagnostic imaging services.^{1,2} Integration provides a more complete patient record, increased consistency, accuracy, and availability of information, automation of mundane and error prone activities, reduced data entry errors, workflow enhancements and improved productivity, lower costs, increased quality of care, and better outcomes.³⁻⁵ It also allows automation of value-added functions including data store and forward, database synchronization, record linking, worklists and scheduling, image pre-fetching and routing, quality control, clinical event tracking, vocabulary services, electronic patient record (EPR) access, and outcomes analysis.⁶⁻⁸

There have been many recent HIS/RIS-PACS integration efforts, and most have focused on a gateway or interface engine model that provides a

single point of transaction across the information system-imaging system domain boundary.⁹⁻¹² A gateway can provide all of the benefits of integration, but has a number of inherent limitations. The gateway's centralized processing model presents a significant bottleneck, scales poorly, may increase network traffic, and in the event of failure, can incapacitate the imaging system.¹³ These factors can reduce overall reliability, which limits practitioner acceptance and reduces productivity.^{4,14,15}

Experiences while implementing an HIS/RIS-PACS gateway have underscored the limitations of centralized processing. Furthermore, experiences while developing and installing a commercial PACS in 60 sites worldwide have proven the benefits of a distributed, fault-tolerant architecture. A distributed gateway model provides all of the benefits of integration, while minimizing the drawbacks.¹⁶ It increases throughput and ensures the availability, fault-tolerance, and consistency of the interface at all times through the use of redundant servers, data replication, load balancing and network monitoring.^{12,17-19} As with any distributed processing model, data consistency and the synchronization of services must be maintained. The distributed gateway architecture proposed in this report provides the advantages of a distributed processing environment and addresses the potential dangers.

DISTRIBUTED GATEWAY MODEL

Any distributed architecture must satisfy the requirements outlined in Table 1.^{12,20}

The mechanisms that implement these requirements are shown in Table 2.

OVERVIEW

At the conceptual level, the gateway is implemented using a client-mediator-server architecture.²¹⁻²³ Clients receive unsolicited messages from

From A.L.I. Technologies Inc, Richmond, B.C., Canada.

Address reprint requests to Robb Keayes A.L.I. Technologies Inc, 95-10551 Shellbridge Way, Richmond, BC, Canada V6X 2W9.

Copyright © 1997 by W.B. Saunders Company
0897-1889/97/1003-1033\$3.00/0

Table 1. Distributed Architecture Requirements

Availability:	Services must be available at all times
Consistency:	System information must be accurate and consistent
Timeliness:	Transactions must be processed in a timely manner and latency times have low variability
Expandability:	The distributed system must be expandable to an arbitrary size
Transparency:	System's distributed nature, including access, failure and recovery, must be transparent

external applications, which are normalized into an internal format and routed to the mediator. The mediator provides the business intelligence, routing the messages to their appropriate destination servers. Servers translate the messages into protocol-dependent formats and sends them to external systems. Responses are then routed back from the servers to the originating client. This is shown in Fig 1.

This is conceptually identical to the centralized model. A distributed implementation of the model allows multiple instances of each class of client, server, and mediator to run simultaneously, but requires that the duplicates share state data. Should one instance fail, another can then assume responsibility for the failed instance's tasks without loss, corruption, or inconsistency of data. The distributed architecture presented here provides this.

Components are the functional modules that collectively implement the distributed gateway. *Architectural* components provide essential infrastructure services and include the Integrated Ser-

vices Layer (ISL) and system monitoring servers.^{21,24,25} The ISL provides intercomponent communication, data synchronization and replication, registration and lookup, and load-balancing services to components. *Functional* components include the client and server *filters*, the *mediator*, and supporting servers. Redundant component instances share state data and coordinate activities via the ISL, ensuring a continuously available gateway. These concepts are illustrated in Fig 2 by a gateway implementing two filters and a mediator, each with two instances.

Data Normalization

The gateway uses a protocol-independent *normalized* format to represent the data content of messages. Normalization eliminates the need for point-to-point mappings between protocols. An inbound message is normalized once, and then mapped to outbound message formats. Where one-to-one mappings exist between inbound and outbound messages, this technique doubles of the number of total translations performed. In the more common case where a given inbound message is routed to n destinations, normalization results in only a slight overhead. With normalization, routing intelligence is centralized in the mediator, not dispersed in clients and servers, providing a more consistent and maintainable routing policy. It also allows the gateway's external interfaces to be changed more easily—a single normalization step instead of n point-to-point translations.

Messages and Transactions

The term *message* is used to refer to an encoding of an event and its data content. Messages received from and sent to external entities have encoding and information content that depend on the protocol used. These messages are said to be protocol-dependent. Messages sent between components of the gateway are normalized and are said to be protocol-independent. These internal messages have a *message context* defined by source and destination descriptors, a transaction identifier and data content profile. In most cases, a sequence of n messages in protocol₁ maps to a sequence of m messages in protocol₂. The sequence, its inputs (dependencies), transitions, and outputs (messages) and are modeled as a *transaction*. The transaction service is hooked into the transaction processor and must support conditional operations, compositions,

Table 2. Enabling Mechanisms

Resource sharing:	Consistent views of persistent data objects are available to all components
Registration:	System components and data objects are registered to provide transparency
Replication:	Data is shared between system components to ensure transparency and consistency
Load balancing:	Transparently routes requests to under-utilized system components
Fault tolerance:	Independent failure modes allow components to be transparently stopped and restarted without affecting the consistency or availability of system data or services
Monitoring:	Intelligent agents monitor hardware and software components

Fig 1. Conceptual processing model.

explosions, translations, and other operations. Normalized messages are usually organized into transactions as *action* (ie, notification or query request) and *response* (ie, acknowledgment or query response) pairs—a particular action can expect a particular response.

External Entity (EE) Profiles

An EE is a logical representation of an application, database, or other entity that communicates with the gateway filters. A profile of each EE, stored in the ISL's persistent data store, describes the EE's communications scheme, including session establishment rules, number of allowed connections, session style (one or many messages per session), and the message acknowledgment style (immediate, deferred, none). Each EE is also characterized as being a Producer and/or Consumer for one or more normalized messages. Each consumed message may have a *satisfaction criterion* that determines whether a given message satisfies the data and context (ie, source) requirements of the EE.

ARCHITECTURAL COMPONENTS

Integrated Services Layer (ISL)

The ISL consists of a set of servers, each managing a set of components defined in the

server's configuration. Each ISL server is responsible for starting, stopping and monitoring its components, and for providing essential communications and data management services to these components and to peer ISL servers. A component managed by an ISL server typically executes on the same physical hardware as the ISL server, although is not required to do so, and uses the managing ISL server as its access point to the system. The ISL services provides a layer through which all system components can intercommunicate. It further ensures that any instance of a class of component may be accessed with identical results, and uses this to optimize resource utilization. Thus, an instance of component X may access *any* instance of component Y, with identical results. The ISL provides the services described in Table 3.

Monitor

Each ISL server has a corresponding monitor that monitors the status of the server's components and supporting hardware and software. The monitor polls components for their status using an API, but can use any other means to gather information (eg, system calls, SNMP).²¹ The monitor notifies its ISL server when a status change is detected. The ISL server is then responsible for initiating corrective action. The monitor also watches the ISL

Fig 2. Distributed architecture overview with example.

Table 3. ISL Servers

Routing:	Messages from components can be routed to a specific component instance or broadcast to all instances of a class of component. The ISL can load balance by choosing the least-loaded of several available components, returning a destination handle to the original sender.
Configuration:	The set of managed components is maintained by the server as part of the data store.
Component management:	Managed components are started and stopped as required, such as during initialization or after a failure, or when a component has become unresponsive.
Statistics:	Component status and load information is gathered from the monitors and other ISL servers. It is used to assist the component management and load-balancing services.
Registration:	Each component registers itself with the server when it becomes active and before it shuts down. Components may register their data objects and notification requirements at any time.
Load-balancing:	Uses statistics to drive load-balancing when assigning components to requests for service.
Notification:	Provides facilities for notifying components or ISL servers of changes to objects in the data store. The registration service provides the data that drives the notification service.
Replication:	Ensures that the data store is replicated to other ISL servers. This includes configuration, registration, statistical, and notification data is consistent and available.
Data store:	Manages data used by the ISL and its components, including the configuration, statistics, registration, and notification services and data registered by components. Components can query for objects in the data store, or register to be notified of changes. Data objects tagged as persistent are mirrored to other ISL servers, to ensure that the server can recover from failures. Components store local copies of required data for performance reasons, but should treat the data store as having the most up-to-date version. Components should also store the location of an alternate ISL server in case of local ISL server failure.

server itself, restarting it should it fail. This provides a higher degree of fault-tolerance to the system.

FUNCTIONAL (GATEWAY) COMPONENTS

Filters

Filters communicate directly with External Entities (EEs) and handle protocol dependencies at the transport through application layers. Filters may be *passive* (accept unsolicited connections), *active* (establish connections), or *hybrid* (both). As part of the configuration, filters indicate the EE profiles they service. Filters listen for messages inbound from EEs and outbound from the Mediator. Inbound messages are parsed, normalized, and sent to the mediator. Outbound messages are mapped to protocol-dependent messages and sent to destination EEs. Each message received may trigger the creation of a new transaction or may alter the state of one or more transactions already in progress. For example, the receipt of a 'order created' message will result in a transaction that expects an 'order acknowledged' response before it is completed. *Action* messages usually initiate transactions, while *response* messages usually complete them.

Mediator

The mediator provides the gateway's intelligence. It accepts messages from filters and makes routing decisions. Decisions are made by extracting the *message context* from each message and applying it to the consumer lists of each filter. Where there is a match, a new transaction record is created for the message, indicating where the message came from, where it was sent, and what reply is expected. The sender can provide hints about the handling of responses, such as 'send to n receivers, but return only the first response', and so on. The mediator ensures that inbound messages are sent to their intended destinations, and the consistent handling of failures when this is not possible. The transaction list is stored as part of the ISLs persistent data store, to ensure recovery from failures.

Value Added Services

These are services that automate system integration functions. A distinction is made between those functions *enabling* interoperability and those that are *enabled* by it. *Enabling* services are plug-in components added to the ISL configuration and made available to other components. Examples are lexicon query and record linking services that

filters use to map vocabularies and identifiers between systems. *Enabled* services are implemented as separate filters that implement or communicate with a value-added service such as a prefetcher or image router.

DISCUSSION

The proposed architecture overcomes the limitations of a centralized gateway. It provides the essential features of a distributed system through a set of cooperating services, as outlined in Table 4.

Further, the benefits of integration are realized by the seamless incorporation of value added services and simplified development of inter-system interfaces. Acceptance of computerization is enhanced by decreased costs, greater productivity and above all, the reliability of the interfaces.

FUTURE DIRECTIONS

The architecture proposed here forms the basis for development of an interoperable, distributed HIS/RIS-PACS interface. Deployment of a centralized gateway has provided the impetus to upgrade the architecture to a distributed model. Deployment of the distributed implementation in a large scale clinical setting will provide further refinements to the model, and will provide confirmation of the benefits of a distributed approach.

Development of a generalized model for interoperability, incorporating elements of business process and workflow modeling, domain meta-modeling, and generalized data representation and translation methods will move the field of interface development in healthcare into the realm of infor-

Table 4. Implementation of Core Distributed System Features

Availability:	The component manager and system monitor ensure components are always running. Registration and replication ensure the ISL is aware of all instances of a component and can provide transparent access to any instance. Replication of transaction records ensures that any filter or mediator can service a given message. ISL failure is recorded by the monitor, and components know the location of at least one alternate ISL instance.
Consistency:	Replication and notification ensure that data is consistent between component instances. The data stores insures against data loss.
Timeliness:	Load balancing ensures that response times are constant. System monitoring and component management ensure that servers failures are caught and recovered.
Expandability:	Use of ISL servers ensures that the ISL can be expanded, and configuration allows components to be added as needed. Only network bandwidth limits expandability.
Transparency:	Automatic component message routing and load-balancing provide transparent access to server components. Data store replication ensures that all component class instances have equivalent state. Monitors and component management ensure transparent failure recovery.

mation engineering where it belongs. This will, in turn, enable healthcare providers to focus on the delivery of quality healthcare rather than the technology that supports it.

REFERENCES

1. Soehlke K, Fisher P: Evaluation of a Generic RIS-PACS Interface. SPIE Medical Imaging VI 1654:455-466, 1992

2. Hayes E: Systems Integration: Information needs soar in managed health care. Diagnostic Imaging Sept. 24-26, 1995

3. Langer S, Wang J: User and system interface issues in the purchase of imaging and information systems. J Digit Imaging 9:113-118, 1996

4. Smith DV, Smith S, Bender GN: Evaluation of the MDIS system based on 2 years of clinical experience. J Digit Imaging 8:75-87, 1995

5. Siegal EL, Diaconis JN, Pomerantz S, et al: Making filmless radiology work. J Digit Imaging 8:151-155, 1995

6. Honeyman JC, Frost MM, Huda W, et al: Performance improvements in PACS, in Kilcoyne RF, Lear JL, Rowberg AH (eds): S/CAR Proceedings. Encinitas, CA, Symposia Foundation, 1996, pp 377-382

7. Piqueras J, Carreno JC, Ovelleiro M, et al: Worklists, preloading and archiving strategies: 3 years of clinical experience with the Barcelona PACS. Medical Informatics 19:123-128, 1994

8. de Silva M, van den Berg W, Brown D: Design and Implementation of the First Children's Hospital, in Kilcoyne RF, Lear JL, Rowberg AH (eds): S/CAR Proceedings. Encinitas, CA, Symposia Foundation, 1996, pp 40-145

9. Mosser H, Urban M, Hruby W: Filmless Digital Radiology—feasibility and 20 month experience in clinical routine. Medical Informatics 19:149-159, 1994

10. Kuzmak PM, Norton GS, Dayhoff RE: Using Experience with Bidirectional HL7—ACR-NEMA Interfaces Between the Federal Government HIS/RIS and Commercial PACS to Plan for DICOM. SPIE Medical Imaging 1995

11. Sawyer RW: Interface engine software: The evaluation and selection process. HIMSS Proceedings 3:253-265, 1995

12. Breant CM, Taira RK, Tashima GH, et al: Issues and solutions for interfacing a PACS database with a RIS. SPIE Medical Imaging VI 1654:255-263, 1992

13. Kayser A, Reijns GL: A Distributed database for a PACS. *SPIE Medical Imaging VI* 1654:566-572, 1992
14. Piqueras J, Carreno JC: Data interface between a RIS and a CR system using a PC and standalone software. *AJR* 161:1313-1315, 1993
15. Cork RD, Friedman CP, Detmer WM: Academic Radiologists and Computers: Use and Attitudes, in Kilcoyne RF, Lear JL, Rowberg AH (eds): *S/CAR Proceedings*. Encinitas, CA, Symposia Foundation, 1996, pp 105-112
16. Sauter K: Trends of strategic importance for the future information landscape in the hospital, in Greenes RA (ed): *MEDINFO Proceedings*. 8:454-457, 1995
17. Stewart BK, Taira RK, Dwyer SJ: Development and implementation of a PACS network and resource manager. *SPIE Medical Imaging VI* 1654:530-535, 1992
18. Huang HK, Andriole K, Bazzill T, et al: Design and implementation of a PACS: The second time. *J Digit Imaging* 9:47-59, 1996
19. Ratib O, Ligier Y, Funk M, et al: Distributed architecture for image archival in a hospital-wide PACS. *SPIE Medical Imaging VI* 1654:222-228, 1992
20. Couloris GF, Dollimore J: *Distributed Systems: Concepts and Design*. Workingham, England, Addison-Wesley Publishing Company, 1988
21. Orfali O, Harkey D, Edwards J: *The Essential Client/Server Survival Guide*, 2E. New York, NY, Wiley Computer Publishing, 1996
22. Ottes FP, Bakker AR, Schulz CA, et al: Prototyping a PACS-HIS/RIS interface in Europe. *SPIE Medical Imaging VI* 1654:551-558, 1992
23. Taira RK, Breant CM, Chan HM, et al: Architectural design and tools to support the transparent access to HIS, RIS and PACS. *J Digit Imaging* 9:1-10, 1996
24. Grimson J, Murphy J: The Jupiter Approach to Interoperability with Healthcare Legacy Systems, in Greenes RA (ed): *MEDINFO Proceedings* 8:367-371, 1995
25. Engelmann U, Jean FC, Meinzer HP, et al: Integrated Image Processing in Clinical Applications: The HELOIS Approach, in Greenes RA (ed): *MEDINFO Proceedings* 8:411-415, 1995