Symposium ■

IAIMS Architecture

GEORGE HRIPCSAK, MD

Abstract An information system architecture defines the components of a system and the interfaces among the components. A good architecture is essential for creating an Integrated Advanced Information Management System (IAIMS) that works as an integrated whole yet is flexible enough to accommodate many users and roles, multiple applications, changing vendors, evolving user needs, and advancing technology. Modularity and layering promote flexibility by reducing the complexity of a system and by restricting the ways in which components may interact. Enterprise-wide mediation promotes integration by providing message routing, support for standards, dictionary-based code translation, a centralized conceptual data schema, business rule implementation, and consistent access to databases. Several IAIMS sites have adopted a client-server architecture, and some have adopted a three-tiered approach, separating user interface functions, application logic, and repositories.

■ **J Am Med Inform Assoc.** 1997;4: March–April Supplement:S20–S30.

An Integrated Advanced Information Management System (IAIMS)[1,2] is an information system that provides health care users with the information they need, when and where they need it.[3] Because it is not possible to supply all the needs of a health care enterprise—administrative, clinical, scholarly, and research—with a single computer application, any comprehensive system will contain many components. There are many users, and each user may play several roles. Furthermore, users are engaged in using many applications that have undergone various forms of automation, ranging from word processors to completely paperless systems. The result is a multitude of systems with redundant and conflicting data. The objective of IAIMS is to integrate these disparate systems into a logical whole—to take something heterogeneous and make it function as a single coordinated system.

Affiliation of the author: Columbia–Presbyterian Medical Center, New York, NY.

Grant support: This work was funded by National Library of Medicine grants LM05627 and LM04419.

Presented at the IAIMS Consortium Symposium, Vanderbilt University, Nashville, TN, September 27, 1996.

Correspondence and reprints: George Hripcsak, MD, Department of Medical Informatics, Columbia–Presbyterian Medical Center, 161 Fort Washington Avenue, DAP-1310, New York, NY 10032. e-mail: ⟨hripcsak@columbia.edu⟩.

Architecture

To accomplish this difficult task, the IAIMS developer relies on an architecture.[4] An information system architecture can be defined as a logical construct for defining and controlling the interfaces and integration of all of the components of a system.[5] Put another way, an architecture is a definition of the components of a system, the boundaries of the components, and the communications among them.[6] There is no single architectural view. Instead, the view will differ depending on the context[5]: an analysis of the network connections among several machines will be very different from an analysis of the data flow among the applications.

Without an architecture, one is tempted to buy or build whatever applications are needed and connect them together in the most expedient way possible. While this approach appears fast and cheap, it is temporary at best. Soon the system will need to be modified, and even the smallest change will require extensive effort. The cost of modifying such a system may exceed the initial cost of putting it together.

An architecture primarily provides flexibility. It allows a system to accommodate the variety of users, roles, and processes. In a large enterprise, no single vendor will supply all that is needed—electronic mail, bibliographic searching, molecular modeling, electronic courseware, clinical information systems, etc. An architecture allows one to incorporate these

disparate components yet still maintain their boundaries so that components can be swapped in and out as vendors turn over, as users' needs evolve, and as technology advances.

Imagine a system where one has to replace a campus' word processors because its network is modified. It seems ludicrous today, but in 1987, Columbia–Presbyterian Medical Center (CPMC) faced this prospect. The hospital's centralized word processing was tied to IBM's System Network Architecture network protocol, and switching to an alternative protocol such as Transmission Control Protocol/Internet Protocol (TCP/IP) would have required replacing the word processors and retraining the users. A good architecture avoids such dependencies, making the system flexible enough to handle change.

An architecture promotes the reuse of components for multiple purposes. If a new method to look up patient names based on inexact matches is being implemented for an admitting system, then all the clinical applications should be able to benefit from it. A properly designed architecture allows each new project to push the entire enterprise forward a little bit at a time. The result is the evolution of the system with changing needs and new technology.[6]

Integrating a health care information system is a challenge. There are many sources of data that are often redundant and conflicting, and there are many applications with diverging user interfaces. A good architecture provides the tools to integrate—a dictionary for translating codes, business rules for reconciling conflicts in data, etc.—and methods to avoid conflicts in the first place. Merely mapping out the system in a clear and logical way (for example, a diagram of data flows) may point out ways to avoid data conflicts.

Basic Concepts

Modularity

Modularity is the architect's primary tool. The information system is divided into parts based on some property: function, structure, location, purpose, etc. The choice of property depends on the context: physical location is important for a network architecture, but not so important for an analysis of data flow. Communication between modules occurs through an interface, which generally defines what a module does, how to ask for it, and how the answer is returned.

Separating a system into modules reduces its overall complexity. What was a mass of interconnections and dependencies becomes more orderly and comprehensible. Changes to a particular function are constrained to a single module. Other modules that use the function are isolated from the details of the change. Thus, modules can be replaced with minimal effect on the other modules. Problems can often be isolated to single modules, facilitating maintenance and recovery.

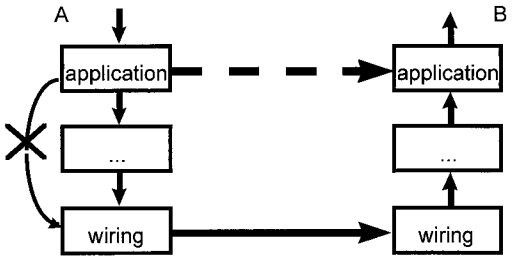

Figure 1 Layering. Application A wishes to communicate with application B. It does so by requesting services from the layer immediately below it, which in turn requests services from the layer below it, and so on. The bottom layer represents the physical wiring between the computers that run the applications. The message is transferred to the layers below application B, and the message percolates up. The two applications feel like they are talking to each other (dashed line), but the complex details of communication are hidden from them. Application A is not allowed to skip layers (curved line) because this breaks the integrity of the system.

Layering

Layering is an important form of modularity. The information system is divided into modules that provide similar functions or "services." The modules are arranged in a series of horizontal layers (Figure 1), and the interactions between pairs of modules are strictly controlled. A module may request services from modules in the layer immediately below it, and it may be called on to provide services for modules in the layer immediately above it. To communicate with other modules in the same layer, a module generally uses services from the layer below it; this hides the details of the communication from the module.

A concrete analogy is a telephone call. A person (first layer) picks up a telephone (second layer) and makes a telephone call. The call is sent through a complex switching network (third layer), but the details are hidden from the caller, who needs only to dial a symbolic telephone number to accomplish the complex routing task. The caller talks to the recipient of the call (also first layer) as if he or she were talking directly, but the interaction is mediated by the telephone system.

Not all aspects of an information system can be ex-

pressed in this way, but for those that can, layering is a very powerful tool for reducing complexity and isolating components. Layering—with its horizontal approach—offers a cross-enterprise view of the architecture that is not available in other forms of modularity. For example, dividing the information system into modules by department (registration, radiology, pharmacy, bibliographic searching, etc.) does not give the architect a good view of the database needs of the enterprise or how messages are passed across the enterprise. By dividing the departmental systems themselves into functional modules and then organizing them all into enterprise-wide layers, one can get a better view of the overall database needs and message passing.

The benefit, however, is realized only when one adheres strictly to the layered scheme. It is often tempting to circumvent the layering to achieve an immediate goal. For example, if a user asks to print from a word processor, it is quickest to attach a printer to the personal computer, install printer drivers, and let the word processor control the printer directly. This is equivalent to skipping important architecture layers, such as the printer queues and network connectivity. As a result, when the user asks to print from a mainframe application, there will be no easy way to accommodate the request. Instead, the printer will have to be reinstalled with the proper layered support.

Network

The network is the foundation for the IAIMS, because it provides the basic communication necessary to integrate disparate systems. The network must route information reliably, quickly, and securely. The network must be flexible to accommodate the constant turnover of computers and the constant appearance of new technology, and there must be an efficient means to manage the network.

Fortunately, installing a network is a largely straightforward task. Given sufficient resources, knowledge, and time, such an effort is likely to be successful. Incredible options are available today, from billion-bit-per-second networks that support advanced multimedia applications and large campuses to wide-area wireless networks for mobile health care workers. There is also an incredible ability to connect network components and software of many different types. Decisions that were important ten years ago—Ethernet or Token Ring, TCP/IP or Novell's IPX—have been rendered moot by the advancement of network standards. These standards have allowed institutions to incorporate networks built many years ago and to accommodate renegade departments that insist on installing their own networks independent of the rest of the institution. New technology is revolutionizing networks. The replacement of traditional networks with network-switch technology is improving speed and reliability (in many cases without changing wiring or computer network adapter cards) and making it possible to define virtual local-area networks based not on physical location but on logical organization.

All this is possible only with a solid network architecture. The primary organizational tool for networks is the International Standards Organization Open Systems Interconnection (OSI) network layers, shown in Table 1. This famous example of layering isolates network components and provides well-defined interfaces between levels.

Table 1 ■

International Standards Organization Open Systems Interconnection Network Layers

Network Layer	Example
7. Application	File transfer protocol
6. Presentation	EBCDIC to ASCII conversion
5. Session	Communications channel
4. Transport	End-to-end communications
3. Network	Routing
2. Link	Adapter cards, frames of bits
1. Physical	Wiring

An alternative view of the architecture, which is orthogonal to the layers, is based on modules and submodules. The enterprise is divided into campuses, then buildings, vertical risers, lobes on floors, and nodes. High-speed networks are provided at the bottlenecks (for example, the campus backbone, which links a campus' buildings). The result is a more manageable network and the ability to isolate network problems to limited physical locations.

The main challenge in installing a network is planning for the future, both for expansion within the enterprise and for advancement of newer, faster protocols and equipment.[7] Keeping clear divisions among the OSI layers is the key tool planners have. Whereas the closets and conduits in a building are expected to last for the life of the building, the network wiring (physical layer in Table 1) may last five to ten years, and the network electronics (link and network layers) may last three to five years. By keeping the layers independent of each other, the network wiring may survive several generations of network electronics.

Application Integration

A network provides basic connectivity, but it is only the foundation for a much more complex system. True communication—exchange of information between high-level entities—requires additional infrastructure (servers, standards) and applications. It is this higher level that holds the greatest IAIMS challenge: integration.[8–10] This includes the ability to access any application from any location at any time, a consistent user interface, and a coordinated conceptual database. Modularity and layering are standard approaches to promote flexibility, but they do not ensure integration. Several approaches to application integration are shown graphically in Figure 2.

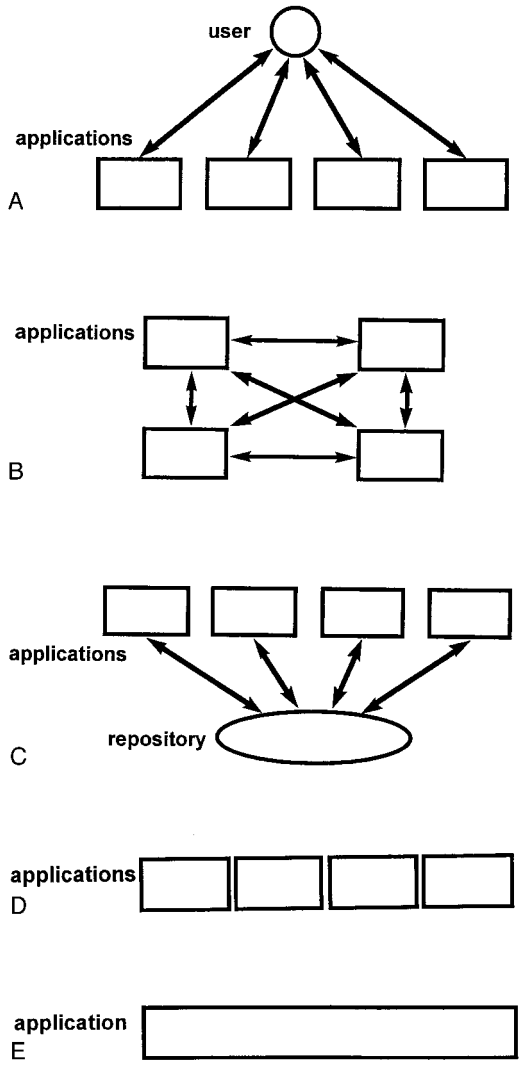

Figure 2 Application integration. There are several approaches to application integration (*top to bottom*): A, terminal emulation; B, interfacing; C, interfaced with a central repository; D, merged; and E, homogeneous. (See text for definitions.)

Terminal Emulation

Getting many applications to work from a single desktop computer used to be a challenge, but the advancement of standards and terminal emulation software have made the task much easier. Applications that use the World Wide Web, X-Windows, or terminal interfaces (such as Digital Equipment's vt100 or IBM's 3270) can be accessed from any platform. For example, one can run a mainframe registration system, UNIX-based electronic mail, and a Web-based clinical information system from the same desktop computer. Only those applications that are tied to a particular desktop platform are problematic, and even some of those can be run on other platforms through emulation (for example, Microsoft Windows on UNIX).

In this approach, the integration really occurs in the user's head. An electronic mail message with patient information has no link to the clinical information system. Duplicate data must be entered into many applications, and there may be contradictions among the applications. Nevertheless, this form of integration is essential because it will never be possible to tie every application together, and it may not be necessary if the logical interaction is minimal.

Interfaced System

In an interfaced system, pairs of applications share data. Applications run essentially unchanged—usually with terminal emulation—but either in real time or in batches, the applications send information to each other. This approach requires a small or moderate amount of work. One must agree on what will be transferred, a data-transfer protocol, coding schemes, and how the transfer will occur.

The benefit is reduced redundant data entry as applications share data. But because there is no centralized control, the possibility of conflicting data still exists. For example, an application may share a patient address with other applications, obviating the need to reenter the data. But if one of the recipient applications already has a different address, there is no guarantee how it will treat the new one: save it, ignore it, or keep both.

An additional disadvantage is the difficulty of creating an application such as result review or an automated decision support that relies on data derived from many different applications. It must be interfaced to all the source applications, and it must either query them in real time, reducing performance, or

keep a copy of all the relevant information locally. If there is more than one such review application, then each one must either query source applications or keep its own copy.

Interfaced System with a Central Repository

One can achieve better integration for a similar amount of work by interfacing applications to a central repository rather than to each other. A central repository does not necessarily imply a single physical database or even a single database management system. Instead, it is a conceptual entity. What makes it "central" is the presence of a unified data schema that ensures that there is a single conceptual place for every data element and there are business rules that map out what is to be done when there is a conflict. For example, what should be done if an application tries to store a patient address that differs from the existing one? The answer may depend on the source of the existing data, the reliability of the storing application, the difference between the addresses, etc. Enforcing the unified data schema and business rules is accomplished through "mediation," which is discussed further in the next section.

If the central repository does happen to be a single physical database, then queries for display and decision-support applications can be implemented very efficiently. The central repository need not hold everything. For example, a laboratory system need not share detailed specimen information if no other application in the system needs it.

For those systems that are not focused around data, it may make more sense to talk about a centrally-coordinated interface rather than a central repository. For example, one would not try to put overlapping bibliographic knowledge bases into a central repository, but one might use a common information-retrieval engine to coordinate a search.

Merged System

A merged system (also known as an "integrated" system) is one in which disparate applications have been modified so that they appear to be one system. For each pair of applications that communicate, this approach requires extensive changes to at least one of them. An example is building a front-end tool for an existing back-end transaction-processing application. The front-end tool may use automated terminal emulation ("screen scraping") to mimic the actions the user would perform in the back-end application. The user sees only the more sophisticated front-end tool, hiding a potentially clumsy application in the back end. While this method results in a well-integrated system, it is difficult and creates dependencies among the modules. It is therefore usually reserved for specific limited situations.

Homogeneous System

If an entire system can be built as a single complex application, then many of the integration issues will be eliminated. It is unlikely that the breadth of IAIMS can be achieved in a single application, however. Some groups have implemented portions of IAIMSs in homogeneous environments with good results,[11] but the efforts took many years. A good deal of vendor software does exist today, and a homogeneous architecture cannot exploit it without using one of the other methods above.

Assessment

Terminal emulation and interfacing are important tools, but they do not achieve full integration. Merged and homogeneous systems achieve integration, but they are not feasible across an enterprise. It appears that interfacing with a central repository, or at least a centrally-coordinated interface, is the best current approach to enterprise-wide integration.

Mediation

Client–Server

The client–server approach refers to the organization of a system into a set of information services that are provided by servers and a set of clients that use those services. Servers often act as clients to other servers. The division is a conceptual one, and it is possible to run a client–server application on one computer. Most often, however, the client refers to the user-interface application running on a local workstation, and the server refers to the database and perhaps application logic that run on one or more separate physical servers. Typical services include a clinical repository, transaction processing, knowledge resources, and security functions. The separation of client from server allows one to concentrate on one area of the system at a time. For example, the clinical workstation[12,13] can be designed separate from the servers that support it.

There are many ways to break an application into client and server modules. If the bulk of the application logic sits in the client, then this is referred to as a "thick" client. If the client contains only what is necessary to run the user interface and the application logic sits on the server, then this is referred to as a "thin" client.

Originally, dedicated terminals acted as very thin clients because the applications ran entirely on servers. As personal computers gained favor, there was a strong push to put as much application logic as possible on the client, making it a thick client. The goal, in part, was to exploit all the extra processing power contained in the personal computers. Practical experience revealed that deployment and maintenance of these thick client applications were very costly and cross-platform development was difficult, leading to a swing back toward thin clients.

The development of the World Wide Web[14] demonstrated the effectiveness of thin clients. The Web browser is a thin client that performs only user-interface tasks, and the bulk of the application sits on a Web server. Its success is due in part to its platform-independent network-wide nature, which reduces deployment and maintenance costs. The current Web browser provides excellent display capabilities, but its data-entry and application logic abilities are limited due to its Common Gateway Interface (CGI) protocol, which is stateless[15] and somewhat slow. New Web client programming languages such as JAVA represent an attempt to use thicker clients without the high deployment and maintenance costs.

Client–Mediator–Server

The client–server approach primarily encourages modularity and reuse of services, but a collection of clients and servers does not make an integrated system. Different servers may call the same data element different names, and they may hold contradictory data.

Cross-enterprise coordination can be achieved only through an entity that acts across the enterprise. Figure 3 shows a "mediator" layer that arbitrates all interactions between clients and servers. This mediator layer provides the tools to map individual data schemata and vocabularies to one conceptual enterprise-wide model, and it insulates clients and servers from changes in each other.[16] (The mediator layer can be seen as just another server in the client–server architecture, but Figure 3 emphasizes that it mediates all interactions.) The interactions between clients and servers are carried out by passing messages to each other, and the mediator can be seen as a message filter.

The mediator layer itself may be divided into layers (Table 2).[17] These layers may be arranged into an "onion skin" of concentric circles, which emphasizes that not all messages need to pass down through all the layers.[17] For example, for a particular message, the mediator may simply need to route the raw message to other servers without further translation or processing.

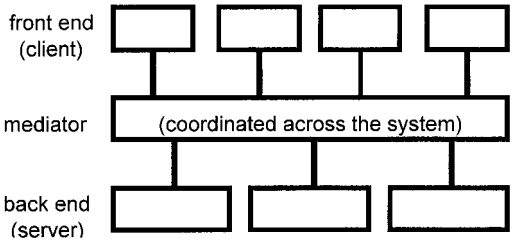

Figure 3 Client-mediator-server layering. An enterprise-wide mediator layer coordinates the interaction between clients and servers. It provides a common conceptual data schema, message and code translation, and monitoring.

Table 2 ■

Mediator Layers

Mediator Layer	Example
Message handling	Parse HL7
Translation	Translate codes
Routing	Send a copy to a research database
Monitoring	Run rules against the data
Access	Modules to store data

Standards and Message Handling

For the multiplicity of applications to communicate, there must be a common language for messages or a means for translation. The mediator facilitates both these methods.

Messaging standards[18] such as Health Level Seven (HL7)[19] define the syntax of the message: whether the data are character or binary, how fields are delineated, etc. For standards such as HL7 that are specific to health care, the definition and organization of the fields are also specified. For example, there is a specific location and format for the patient's name. No one standard covers all areas in a health care enterprise, so choosing a consistent set of standards is an important architecture task. Wherever possible, applications that comply with those standards are chosen.

Unfortunately, messaging standards are incomplete. One must still interpret what each field means, what codes to put in, and what fields to ignore. There is often more than one way to specify the same information. Therefore, even for those applications that use the same standard (and certainly for those that use different standards), it is necessary to perform some form of translation. The mediator parses each message

into fields, moves data among fields to accommodate differing interpretations of the standard, and translates among coding schemes (see below).

Dictionary

No one coding scheme satisfies all the requirements of an enterprise. In some cases, different departments use different coding schemes for the same information: many systems are purchased and have predefined codes, and many departments are unwilling to give up their traditional codes. For example, the laboratory system may refer to the patient's gender with different codes than a patient registration system.

The mediator translates among these coding schemes using a dictionary (vocabulary)[20,21] during message processing. The dictionary defines all the known codes in the enterprise and contains a means for mapping among the codes. The task is straightforward when there is a one-to-one correspondence between coding systems (for example, for gender), but there is often no simple translation (for example, for diagnoses), and the result can be ambiguous or distorted. In the latter case, data are often stored in the central repository according to the original codes, and when data are sent to applications that use different codes, the closest match possible is given.

The dictionary may be a simple set of translation tables or a more complex structure such as a semantic network.[21] By supporting hierarchical classification and semantic links, a semantic network can facilitate other mediator tasks such as routing, decision support, and dictionary maintenance. For example, if a pulmonary researcher wants to collect all the pulmonary test results for a set of patients, it may be necessary only to specify a class called ''Pulmonary Tests'' rather than to name each test separately.

Routing

The mediator routes messages among applications based on the origins and contents of the messages. For example, rather than have the laboratory system determine where its results should be sent, it merely indicates that the message comes from the laboratory. The mediator contains the tables and rules that determine where the message should be routed. Typical destinations for messages include a central repository, a research database, and ancillary systems. In this way, data sources are insulated from the destinations, and there is a single point of control for the routing of messages.

The mediator may forward messages to a clinical event monitor.[22] Based on the message and relevant data in the repository, the event monitor can generate its own messages and carry out actions. It may be used to remind or alert health care providers to clinically important situations, and it may be used to enforce an enterprise's business rules.

Data

The organization of the enterprise's codes, textual data, images, etc., in the central repository (be it real or virtual) is known as the data schema (also referred to as the data model).[23,24] The data schema gives an overall view of the enterprise's data, facilitating understanding and management. It provides a structure for adding new types of data and a means for uncovering duplicate and ambiguous values in the repository.

The data schema must have exactly one definition for each unique data element. If two patient address fields are found, for example, something must be done to resolve them. Both addresses may be kept only if they are defined as being different in some way: one may be designated ''primary'' or ''current,'' and the other ''secondary'' or ''expired.''

A data schema and repository cannot fix bad data. Systems need to be organized and deployed so that given data elements are collected once from the persons most likely to have accurate information.[6,25] The data must then be shown to all relevant parties, and there must be the ability to correct the information when mistakes are found.

Database Access

Queries and updates to actual physical databases may be mediated[16] through data-access modules,[26] which insulate the rest of the system from the details of database access. Data-access modules were essential at CPMC[26] when a variety of databases (relational DB2, hierarchical IMS, and indexed file VSAM) were used to store clinical data. Data-access modules provided a single application-programming interface. When database management systems were replaced and data had to be migrated to new databases, most of the changes occurred in data-access modules rather than in the applications. Application changes were not eliminated, however, because of some essential differences among the database management systems and because the migration was used as an opportunity to improve the interface or data schema.

If a single database type (for example, relational) or even a single database vendor is used, then one may rely on a standard such as Structured Query Language (SQL) to mediate access to the database. Un-

fortunately, SQL is not well standardized (embedded versus function call, extensions, etc.); it requires detailed knowledge of the organization of the database (that is, not just health concepts, but database table and view organization); and it is not suited to complex queries that can be embedded in data-access modules (for example, the ability to retrieve the last ten items from a group is difficult in SQL).

Object-oriented technologies and standards such as Common Object Request Broker Architecture* and the Andover Initiative[27] promise to provide mediator service within a consistent object framework.[28] The object-oriented approach ensures modularity through encapsulation and data abstraction. Use of classification and inheritance can facilitate maintenance. A consistent view of data as objects eliminates artificial distinctions introduced by implementation issues in other systems. (For example, rather than defining data fields and the content elements in different ways, both are considered objects.)

Implementation

A number of vendors provide interface engines (also called data hubs or message routers) that include message parsing, field translation, simple code translation, and message routing. They do not generally support complex dictionaries, good data schema tools, or decision support.

It has been recognized that moving from a monolithic system to a client–server architecture is generally associated with increased cost due to increased complexity. The comparison, however, may be similar to comparing paying for a telephone to running next door to talk to a neighbor. If all one wants to do is talk to the next-door neighbor, then a telephone will not be cost effective. The benefit is only seen when one realizes that calls can be made around the world. Similarly, a monolithic architecture may in fact be cheaper for a simple system, but a client–server architecture allows one to build a larger, more diverse system than is possible with a monolithic architecture.

Security and Recovery

Authenticating who a user is and authorizing the user to access a particular application are complex tasks in a large enterprise due to the numbers of systems and users and the frequent turnovers of both.[29] Ideally, each user should have a single identifier and password, but there is no single solution that covers the full range of operating systems, network protocols, and applications that can appear in an enterprise. Systems that have centralized security servers such as Kerberos[30] and several commercial products are expanding, and operating-systems vendors are including more security function within their products, so security tasks should become easier over time. Additional functions that need to be addressed are the encryption of information as it flows over the network and keeping track of software and information-resource licensing.

One of the most powerful patient-privacy tools is the audit log,[31–34] which records who looked at what when. Knowledge that actions are being audited appears to act as a deterrent against infractions of privacy. The log serves as a permanent record in case a patient ever complains of a breach of privacy, and the log can be surveyed automatically to detect such breaches.

Recovery from hardware and software failures is largely accomplished through redundancy at the network, server, and application levels. The more ways a backup system differs from the original, the more likely it will work when needed. For example, two identical result-display systems are likely to fail simultaneously and therefore offer little security against failure. On the other hand, using an ancillary laboratory system as a backup for a centralized result-display system offers security against many types of failures.

While the presence of duplicate data elements undermines the integrity of a system, replication of data does not. There must be a single conceptual location for each element, but the element may be physically stored (replicated) in many places. Replicated data can improve performance (caching), help isolate vendor systems (by allowing an application to maintain its own database), and protect against data loss. For replication to work without becoming duplication, each physically stored item must map to an element in the central conceptual data schema, there must be a means to synchronize copies, and when synchronization fails, there must be a procedure to resolve conflicts (for example, one copy may be designated the primary one).

IAIMS Architecture

Clinical Example

To demonstrate the concepts enumerated above, Figures 4 and 5 show the architecture of a hypothetical IAIMS with a strong clinical focus. While this architecture does not represent a particular institution, it

*http://www.omg.org/corbamed.

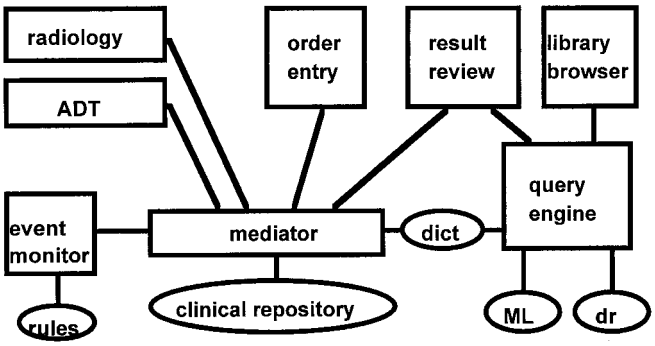

Figure 4 Modular view of a clinical IAIMS architecture. Rectangles represent system modules, ovals represent databases and knowledge bases, and lines represent the flow of messages. (See text for module definitions.)

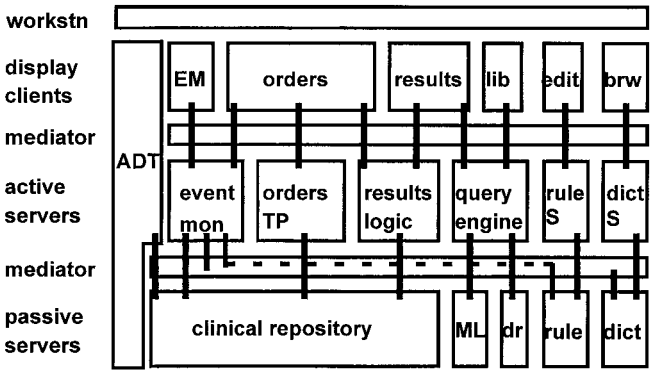

Figure 5 Layered view of a clinical IAIMS architecture. The same system as in Figure 4 is depicted here with a layered view. Rectangles represent modules and mediators. Heavy lines and the dotted line indicate the flow of messages. (See text for module definitions.)

was drawn from several institutions with similar architectures.[9,35,36]

Figure 4 presents a modular view of the system. There is a physical central repository, and all access to it is mediated; the mediator relies on a central dictionary (dict). Ancillary systems such as laboratory and admit–discharge–transfer (ADT) send data to the database and receive data from other systems through the mediator. When data are stored, the mediator triggers the event monitor, which makes decisions based on a set of rules. Providers enter orders through the order-entry system and review results through the result-review system. A library browser uses a query engine to retrieve information from a number of resources, including MEDLINE (ML) and a drug-information knowledge base (dr). The query engine also relies on the central dictionary (dict), and the result-review system uses the query engine to present relevant information resources with the clinical results.

Figure 5 shows a layered view of the same system. A "three-tiered" architecture is shown, consisting of the display clients that carry out user-interface functions, the active servers that carry out application logic, and the passive servers that hold data and knowledge bases. Between each two consecutive tiers is a layer that mediates interactions. The bottom mediator layer, in particular, relies on the central dictionary (dict).

At the top is the workstation (workstn), which provides the least common denominator for presenting applications to the user. A legacy ADT system does not follow the layers of the architecture because it includes the user interface, application logic, and the database as a monolithic whole. It is therefore shown as crossing all the layers except the workstation itself, which provides terminal emulation. Data from the ADT system are stored in the clinical repository through the mediator layer. (This is an example of interfacing with a central repository.)

When data are stored in the repository, the mediator triggers the event monitor (event mon), which makes decisions based on data in the repository and clinical rules (rule); the connection to the latter is shown as a dotted line. A user-interface application for the clinical event monitor administrator (EM) is also shown. Data are shown to providers through the results-display application, which has two parts: the results user interface (results) and the results application logic (results logic). Similarly, the order-entry application is divided into the user interface (orders) and the transaction processing (orders TP). When providers write orders, they review results through the order-entry application. To avoid redundant software, the order-entry application reuses the results application logic.

The dictionary and rule knowledge base are maintained through their corresponding servers (rule S and dict S) and authoring tools (edit and brw). The library browser (lib) uses a query engine to access MEDLINE (ML) and a drug-information knowledge base (dr). The results-display application also uses the query engine to link information resources to clinical results.

The two architectural views are complementary. The modular view stresses the functional requirements of the system as a whole, and it is relatively invariant to implementation. For example, systems built decades ago had similar architectural views.[11] The layered view stresses the implementation of the system, and it has changed significantly over time. Nevertheless, many groups have now adopted an approach similar to this one.[9,35,36]

IAIMS Survey

The publications of funded IAIMS sites were reviewed. Planning issues—which are of primary importance—received much greater emphasis than did architecture, and few sites have published details of their information systems architectures. Nevertheless, a number of conclusions can be drawn from the literature.

Of 12 sites that gave some mention of architecture,[9,35–45] seven had a stronger clinical care focus[9,35–37,39,42,44] and five had a stronger library focus,[38,40,41,43,45] including curriculum, continuing education, and information retrieval. (The assessment of clinical versus library focus was made for the publication that covered architecture. The sites themselves may have a dual focus,[39,46] and some sites have changed focus over time.[37,47])

All 12 sites stressed the importance of the network to the IAIMS, making it clear that this is an essential component. Seven sites specifically mentioned that they were using a client–server architecture.[9,35–37,39,41,43] With the deployment of the Web, it is likely that all are now using some form of client–server architecture. More interesting is the fact that four of the seven had a clinical focus and three had a library focus. It therefore appears that client–server architecture is important regardless of the IAIMS focus.

Three sites mentioned the use of a central repository, be it physical or conceptual.[9,35,36] All three had a clinical focus. Although the number of sites reviewed is small, it appears that the use of a central repository is more important for a clinically-focused IAIMS. The issue for an IAIMS focused on information resources is more likely to be coordinating information retrieval[40,41] rather than trying to put resources in one repository.

Conclusion

Building an integrated, flexible information system is a difficult undertaking that is achievable only with a good architecture. Use of modularity, layering, client–server separation, and mediation permits the swapping of components in a system that still acts as an integrated whole. IAIMS sites appear to be converging on similar methods, and the advancement of technology and standards is making the task easier.

The author thanks Soumitra Sengupta, Stephen Johnson, William Stead, and Paul Clayton for their advice and support.

References ■

1. Matheson NW, Cooper JA. Academic information in the academic health sciences center: roles for the library in information management. J Med Educ. 1982;57:1–93.
2. Lindberg DA, West RT, Corn M. IAIMS: an overview from the National Library of Medicine. Bull Med Library Assoc. 1992;80:244–6.
3. Fuller S, Braude RM, Florance V, Frisse ME. Managing information in the academic medical center: building an integrated information environment. Acad Med. 1995;70:887–91.
4. Gorry GA. Information technology and the academic medical center. Acad Med. 1992;67:18–21.
5. Zachman JA. A framework for information systems architecture. IBM Systems J. 1987;26:276–92.
6. Stead WW, Borden R, Bourne J, et al. The Vanderbilt University fast track to IAIMS: transition from planning to implementation. J Am Med Informat Assoc. 1996;3:308–17.
7. Stead WW, Borden R, McNulty P, Sittig DF. Building an information management infrastructure in the 90s: the Vanderbilt experiment. Proceedings of the Annual Symposium on Computer Applications in Medical Care. 1993; 534–8.
8. Bleich HL, Slack WV. Designing a hospital information system: a comparison of interfaced and integrated systems. MD Comput. 1992;9:293–6.
9. Clayton PD, Sideli RV, Sengupta S. Open architecture and integrated information at Columbia–Presbyterian Medical Center. MD Comput. 1992;9:297–303.
10. Stead WW, Bird WP, Califf RM, Elchlepp JG, Hammond WE, Kinney TR. The IAIMS at Duke University Medical Center: transition from model testing to implementation. MD Comput. 1993;10:225–30.
11. Pryor TA, Gardner RM, Clayton PD, Warner HR. The HELP system. In: Blum BI (ed). Information Systems for Patient Care. New York: Springer Verlag, 1984:109–28.
12. Ball MJ, Silva JS, Douglas JV, Degoulet P, Kaihara S. The health care professional workstation. Int J Bio-Med Comput. 1994;34:1–416.
13. Greenes RA, Collen M, Shannon RH. Functional requirements as an integral part of the design and development process: summary and recommendations. Int J Bio-Med Comput. 1994;34:59–76.
14. Lowe HJ, Lomax EC, Polonkey SE. The World Wide Web: a review of an emerging Internet-based technology for the distribution of biomedical information. J Am Med Informat Assoc. 1996;3:1–14.
15. Cimino JJ, Socratous SA, Clayton PD. Internet as clinical information system: application development using the World Wide Web. J Am Med Informat Assoc. 1995;2:273–84.
16. Wiederhold G. Modeling for software system maintenance. In: Papazoglou MP (ed). OOER'95: Object-Oriented and Entity Relationship Modeling. New York: Springer Verlag, 1995:1–20.
17. Johnson SB, Forman B, Cimino JJ, et al. A technological perspective on the computer-based patient record. In: Steen EB (ed). Proceedings of the First Annual Nicolas E. Davies CPR Recognition Symposium. Washington, DC: Computer-based Patient Record Institute, 1995:35–51.
18. Hammond WE. The role of standards in creating a health information infrastructure. Int J Bio-Med Comput. 1994;34:29–44.
19. HL7 Working Group. Health Level Seven. HL7 Working Group, 1990.
20. Tuttle MS, Nelson SJ. The role of the UMLS in 'storing' and

'sharing' across systems. Int J Bio-Med Comput. 1994;34: 207–37.
21. Cimino JJ, Clayton PD, Hripcsak G, Johnson SB. Knowledge-based approaches to the maintenance of a large controlled medical terminology. J Am Med Informat Assoc. 1994;1:35–50.
22. Hripcsak G, Clayton PD, Jenders RA, Cimino JJ, Johnson SB. Design of a clinical event monitor. Comput Biomed Res. 1996;29:194–221.
23. Johnson SB. Generic data modeling for clinical repositories. J Am Med Informat Assoc. 1996;3:328–39.
24. ANSI/HISPP MSDS JWG for a Common Data Model IEEE P1157 Medical Data Interchange Working Group. Trial-Use Standard for Healthcare Data Interchange—Information Model Methods. New York: American National Standards Institute, 1994.
25. Stead WW, Sittig DF. Building a data foundation for tomorrow's healthcare information management systems. Int J Bio-Med Comput. 1995;39:127–31.
26. Hripcsak G, Johnson SB, Sideli RV, Cimino JJ, Clayton PD. Using data access modules for legacy databases. In: Rindfleisch TC (ed). 1994 Spring Congress, Washington, DC: American Medical Informatics Association, 1994:107.
27. Anonymous. HP rallies industry giants to deliver plug-and-play computing for healthcare. Adv Healthcare. 1996;1–9.
28. Sengupta S, Clayton PD. Clinical workstations: an architectural prospective. In: van Bemmel JH, McCray AT (eds). Yearbook of Medical Informatics 1996. Stuttgart, Germany: Schattauer, 1996:59–64.
29. Shea S, Sengupta S, Crosswell A, Clayton PD. Network information security in a phase III Integrated Academic Information Management System (IAIMS). Proceedings of the Annual Symposium on Computer Applications in Medical Care, 1992;283–6.
30. Miller SP, Neuman BC, Shiller JI, Saltzer JH. Kerberos Authentication and Authorization System. Section E.2.1. Project Athena Plan. Cambridge, MA: Massachusetts Institute of Technology, 1988.
31. Barrows RC, Clayton PD. Privacy, confidentiality, and electronic medical records. J Am Med Informat Assoc. 1996;3: 139–48.
32. Safran C, Rind D, Citroen M, Bakker AR, Slack WV, Bleich HL. Protection of confidentiality in the computer-based patient record. MD Comput. 1995;12:187–92.
33. Hammond JE, Berger RG, Carey TS, et al. Report on the clinical workstation and clinical data repository utilization at UNC hospitals. Proceedings of the Annual Symposium on Computer Applications in Medical Care, 1994;276–80.
34. Hayam A. Security audit center—a suggested model for effective audit strategies in health care informatics. Int J Bio-Med Comput. 1994;35:Suppl:115–27.
35. Stead WW, Sittig DF. Integrated advanced information management systems: strategies and architectures for fast track development. In: Prokosch HU, Dudeck J (eds). Hospital Information Systems. Amsterdam, The Netherlands: Elsevier Science Publishers B.V. 1995:313–30.
36. Stead WW, Borden RB, Boyarsky MW, et al. A system's architecture which dissociates management of shared data and end-user function. Proceedings of the Annual Symposium on Computer Applications in Medical Care. 1991;475–80.
37. Buffone GJ, Petermann CA, Bobroff RB, et al. A proposed architecture for ambulatory systems development. Medinfo. 1995;8:Pt 1:363–6.
38. Schmidt D, Mitchell JA. The J. Otto Lottes Health Sciences Library and the Microcomputer Learning Laboratory. Comput Meth Programs Biomed. 1994;44:193–9.
39. Shifman MA, Clyman JI, Paton JA, Powsner SM, Roderer NK, Miller PL. NetMenu: experience in the implementation of an institutional menu of information sources. Proceedings of the Annual Symposium on Computer Applications in Medical Care. 1993;554–8.
40. Broering NC. Fulfilling the promise: implementing IAIMS at Georgetown University. Medical Progress through Technology. 1992;18:137–49.
41. Fuller S. Creating the integrated information infrastructure for the 21st century at the University of Washington Warren G. Magnuson Health Sciences Center. Proceedings of the Annual Symposium on Computer Applications in Medical Care. 1993;529–33.
42. Hendee WR, Urlakis MA, Blackwelder M. The Health Information Technology Center at the Medical College of Wisconsin. WI Med J. 1994;93:159–63.
43. Kruper JA, Jones TM. The Centennial Patient Care Program: binding patient, student, and clinician–teacher in a learning triad. Proceedings of the Annual Symposium on Computer Applications in Medical Care. 1993;752–6.
44. Beck JR, Krages KP, Ash J, Gorman PN. Outreach to Oregon physicians and hospitals: 5000 by 2000. Ann NY Acad Sci. 1992;670:91–7.
45. DeGeorges KM. Computing outreach initiatives for medical literature services: the IAIMS approach of the American College of Obstetricians and Gynecologists. J Med Sys 1993; 17:367–70.
46. Paton JA, Belanger A, Cheung KH, et al. Online bibliographic information: integration into an emerging IAIMS environment. Proceedings of the Annual Symposium on Computer Applications in Medical Care. 1992;605–9.
47. Fowler J, Barber S, Gilson H, Long KB, Gorry GA. The MEDLINE Retriever. Proceedings of the Annual Symposium on Computer Applications in Medical Care. 1992;473–7.