An Integrated Auditing Architecture For Internet And Information System Design Under A CORBA Environment

Fengyi Lin, (E-mail: fylin@public.iis.sinica.edu.tw) Chih Lee College of Business, Taipei, Taiwan
Deron Liang, (E-mail: drliang@iis.sinica.edu.tw) Academia Sinica Taipei, Taiwan
Soushan Wu, National Chiao Tung University, Taipei, Taiwan
Ray M Yang, Trade-Van Information Services Company, Taipei, Taiwan

Abstract

This article proposes Integrated Auditing architecture (IA architecture) on the Internet with three emerging information technologies, namely objected oriented technologies, distributed middleware and Internet security technologies. This IA architecture provides an infrastructure for software components interoperating in a heterogeneous environment. Under an IA architecture environment, it is possible for auditors to encapsulate "legacy" EDP system with a standard interface. Furthermore, we discuss to what extent this IA architecture may expand Computer-assisted auditing techniques (CAATs) on the Internet. For concept verification, a prototype system based on the proposed IA architecture is illustrated in this article.

1. Introduction

Organizations everywhere are under tremendous pressure to evolve their electronic data processing (EDP) systems so that they can better respond to marketplace demands and rapidly changing technologies. Since EDP systems have been widely used in business to manage their daily business transactions and strategic accounting records, auditors' roles have changed along with the complexity of the tasks they must perform [2,3,4]. These changes have created major challenges in performing the auditing and attestation function to evaluate auditees' EDP systems [29,40] (1) Organizations' accounting records are kept in electronic form, which is readable only to computers. In response to this, auditors must reevaluate traditional evidentiary paradigms as the audit trail moves from paper to magnetic media [28]. Furthermore, auditors are forced to identify and implement controls to ensure that transactions are processed accurately, completely, and with the proper approvals and audit trails [2,4,30,39]. (2) The advent of a variety of hardware and software platforms has allowed numerous organizations to develop their own proprietary software or accounting applications [1,18]. Under this heterogeneous and fast-paced environment, auditors must always face the problem of how to cope with auditees' endless new software releases, and stave off hardware and software obsolescence. EDP auditing has become difficult

Readers with comments or questions are encouraged to contact the authors via e-mail.

and time consuming. In response to these challenges, auditors have developed CAATs to deal with traditional audit functions and begun to use advanced technologies in support of auditing [6,11,15,19]. For example, generalized audit software (GAS) can process some files and file type e.g. dBase, DIF, ASCII, spreadsheets and various word processing packages [1,2]. GAS also enables auditors to access useful information on client master files that are not included in reports produced by the client. Other examples of CAATs are the use of continuous monitoring systems such as SCARF or concurrent embedded modules that target financially significant risk areas and subject transactions to reasonable testing [13,21]. Instead of using technologies to report on past events, the above technologies can be developed and implemented at the design stage of an EDP system in order to detect and report unusual or suspicious transactions at the time they are processed through the EDP systems [13,19,38].

Although there is a clear necessity to increase the scope of audit coverage in areas of high organizational risk, there are surprisingly few cost-effective alternatives available [35]. Most EDP systems currently in use are closed mainframe systems. Each system has its own proprietary operating systems, application programs, and data files. It is very difficult to exchange information from one system to another. As a result, it is nearly impossible to design one GAS to audit all EDP systems [2,5]. In addition, current computers are inter-connected within the heterogeneous network environment of LANs and WANs (such as Ethernet, FDDI, and ATM) where the different protocols and characteristics restrict auditing software reusability, portability and interoperability. Development teams of IS auditors have had to create the same functionality over and over again under different EDP systems, which wastes time and money. For other concurrent CAATs, early involvement of auditors at the time when the system is under development becomes necessary. Furthermore, this will increase the requirement for software specialists. A 1996 survey showed that a majority of internal auditors believed software technologies could enhance the role of internal audit departments in the monitoring, control, and assessment of risk [10,26,28]. We believe that the implementation these advanced auditing applications will be more cost-effective if new technologies, such as object-oriented technologies and distributed middleware standards, are effectively applied.

In order to perform reliance quality and reliability in EDP auditing under today's heterogeneous computing environment, auditing applications must work on a variety of hardware and software platforms and they must be able to interoperate legacy systems and make use of existing infrastructures [16,32,37]. Distributed middlewares provide opportunity to "glue" many heterogeneous systems with one common layer [7,12]. This common layer defines standard application interfaces as well as standard data structures. Therefore, it becomes possible for auditors to design one single self-designed audit software to access and analyze auditee's data via distributed middlewares. Object-oriented technology can make distributed system design easier and provide high level abstractions. Object-oriented technique is also the mechanism for defining the methods of "plug and play" (PnP). Auditors, with limited computer backgrounds, can develop and modify the audit modules independently from the auditee's EDP system design. This eliminates the need for early involvement in the clients' EDP systems and assistance from computer software specialists.

With object-oriented technology and distributed middlewares or, as we call it -- *distributed objects*, auditors can improve the interoperability, reusability, extendibility and portability in a heterogeneous environment. Consequently, we propose an Integrated Auditing architecture with the support of emerging technologies -- distributed object technologies such as CORBA ORB or DCOM to provide an infrastructure for software components inter-

operating in a heterogeneous environment.

This article is divided into five sections. In the next section, we provide an overview of IA architecture with emerging information technologies that could facilitate IA architecture on the *Internet*. Section 3 of the article provides a step by step demonstration of how this IA architecture can be facilitated based on CORBA middleware. Section 4 offers two examples to illustrate the advantage of IA. The last section will discuss the challenges and future research opportunities in IA architecture implementation.

2. Integrated Auditing Architecture

Figure 1 is a high-level graphical representation of the IA architecture that is comprised of three major advanced technologies: *distributed middleware, object-oriented technologies,* and *Internet security technologies*. The structure and composition of the architecture are described in more detail in the following sections.

2.1 Distributed Middleware And Object-Oriented Technologies

The primary goal of this IA architecture is to take advantage of *distributed object technologies* (a combination of *distributed middleware* and *object-oriented technologies*) to provide an infrastructure for software components interoperating in a heterogeneous environment [17,31].

Distributed middleware technology can support an open and programmable environment for flexible developing auditing applications. "Open" means that there are some public Application Programming Interfaces (APIs) which can be invoked by programming languages for the high-level control and management of the underlying resources. The EDP systems of most organizations have their own proprietary operating systems, application programs, and data files. In the past, auditors in the heterogeneous environment had limited ability to communicate (interact) with their auditees. Under the IA architecture, auditors can access different auditees' accounting information with the same API or standard interfaces. Therefore, auditors can look into computer data files and do data extractions and analysis by themselves without interference by auditee's EDP systems. In addition, auditors then can design one special CAATs to review all of its auditees' EDP systems.

Moreover, *distributed middleware* provides an infrastructure that enables invocations of operations on objects located anywhere on a network as if they were local to the application using them. The location transparency feature and programming transparency further provides auditors the ability to conduct an audit from a remote site and set checkpoints for important internal controls as well. The idea is to provide auditors with a transparent access to auditees' accounting information and audit evidence anywhere on the LAN or WAN from any desktop.

Object-oriented technology has been gaining popularity in recent years as a means for constructing large-scale software systems. Programmers can easily decompose a system into well-structured entities in distributed programs. Therefore, object-oriented technology can make distributed system design easier and provide high level abstractions. There are three magical properties that make them incredibly useful: inheritance, encapsulation, and polymorphism - which allow the creation of reusable and extendible object [8].

- Inheritance: Inheritance is the automatic passing of properties or characteristics from a parent or ancestor to a child. Child classes inherit their parent's methods and data structures. Reusable code is stored in a repository rather than expressed repeatedly. Due to the inheritance feature, the IA architecture can provide reusability and extendibility for the audit software. This mechanism can save a lot of time and costs for the designers and programmers.

- Encapsulation: The object does this by hiding its own resources with a public interface that defines how other objects or applications can interact with it. It also separates the user of an object from the author of the object. The IA architecture can "encapsulate legacy" EDP systems with standard interfaces. In other words, auditors can add or remove any element without effecting existing auditees' legacy systems.

- Polymorphism: Polymorphism is a highbrow way of saying that the same method can do different things, depending on the class that implements it. Under this mechanism, you can view two similar objects through a common interface but eliminate the need to differentiate between the two objects.

To simplify information access from the ever-growing WWW, several object-based distributed middlewares have emerged. Microsoft's DCOM [17], OMG's CORBA [14], and JavaSoft's RMI [33] are three of the most important industry standards.

2.2 Internet Security Technologies

The standard interface, which defined by, distributed middleware makes auditors easier to access auditees' EDP systems. However, this move unfortunately exposes EDP systems to uncontrolled risks on the Internet community unless sufficient security measures are installed. We therefore propose to include a robust security layer in the IA architecture as shown in Figure 1. The security layer provides variety functions to guard not only the safety of the data stored in the EDP systems but also the messages exchanged between two parties over the Internet against malicious third parties [34]. These functions include *authentication, confidentiality, integrity,* and *non-repudiation.*

Authentication

Authentication is a function that the receiver of a message (or the server of a request) is able to ascertain its origin, i.e., the sender of the message (or the client of the request). On the other hand, an intruder should not be able to masquerade as someone else. Authentication process can be achieved by security technologies such as *digital certificate* [34].

Confidentiality

Confidentiality means that only authorized people can see protected data, regardless of these data are in a secured place or in transit. This function is usually enforced by a number of cryptography technologies. For example, logging to EDP system is granted only by legal passwords. Retrieve sensitive information is subjective to proper access control. Various encryption algorithms such as public-key [34] protect information transmitted over the Internet/intranet.

Integrity

A message receiver should be able to verify that the message has not been modified in transit; an intruder should not be able to substitute a false message for a legitimate one. The function of integrity can be achieved via various security technologies such as *message digests* [34].

Non-Repudiation

Non-repudiation is a function that a sender must not falsely deny later that he/she sent a message. In the context of the IA architecture, non-repudiation implies that a document, such as financial statement which auditors retrieve from an EDP system via authorized interfaces over the Internet shall be considered as a legal document. This document can be seen as "send" out by organization's EDP system. Non-repudiation can be achieved by a security technology called *digital signature* [12,34]. In other words, legal documents retrieved from the EDP system via authorized interface are *signed electronically* with organizations' digital signatures.

The organizations can not "deny" later that they did not issue such documents.

The advantages of IA with the possible impact of IT on current auditing practices are as follows:

- Easy to use: It is possible for an auditor to design one copy of GAS for all his auditees since the information is at the auditor's finger tips. Furthermore, auditors can develop audit software at the auditor's side independently from the auditee's EDP system design. Traditionally, the audibility of a data processing system is dependent on the underlying system of controls that has been built into the system during its development [18,20]. Therefore, auditors must participate in the systems development process to ensure that the necessary audit and control features are built into auditee's EDP system. With IA architecture, auditors can independently design their own audit software through standard component technologies such as CORBA and actively monitor auditees' EDP systems.

- Reliability: Auditors can access live account data stored in various file formats that are machine-readable only. This function will place an increased level of reliance on the auditing records (while accounting operations are being performed) instead of evidence from related activities (e.g., preparedness audits). Auditors directly access to evidence can surely increase auditing reliability [22,42].

- Efficiency: Traditional auditing involves the examination of auditees' accounting records substantially after the event and emphasizes paper-based evidence [25,27]. IA architecture may facilitate on-line tests of an internal control with the next generation of information technologies, such as PnP servets and information interception through the *Internet* [9,18]. Internal control checkpoints can easily shift control monitoring to the auditor as previously programmed. Unlike concurrent audit modules such as an embedded module or SCARF [24,38,41] which require extensive programming and testing to impact on the auditee's EDP system with even a minor change. The concurrent technique designed under IA architecture can be adjusted in seconds by auditors, with limited technical background, through the programmed distributed objects and end-user computers.

- Interoperability: IA architecture with distributed objects is capable of interoperating across operating systems, networks, languages, applications, tools, and multivendor hardware among various auditees' EDP

IA architecture leading to a successful CAAT application must have a flexible software architecture so that the software can be organized into components. The followings summarize the IA architecture feature:

- Easy to use: one GAS for all auditees' EDP systems.
- Reliability: data can be accessed separately, which can increase evidence reliability.
- Efficiency: separately deployed and developed audit software can increase efficiency.

Interoperability: auditing applications can work on a variety of hardware and software platforms, and be able to interoperate legacy systems.

Table 1 Summarizes The IA Architecture Feature

systems. An object bus provides a unified system architecture that can be of tremendous help to system integrators of organization EDP systems.

3. CORBA implementation for IA architecture

Previous section has demonstrated the promising features of emerging technologies to support our IA architecture. This section presents CORBA implementation for IA architecture. Because *object-based distributed* processing has been highly valued, Object Management Group (OMG) is devoting itself to the standardization of distributed object exchange, and CORBA is exactly its first standard for distributed object exchange [14]. This application chooses CORBA as *distributed object technology* for IA architecture because CORBA is considered an industrial standard. Under the CORBA environment, communication between objects is supported by CORBA message passing technology. The overall structure of CORBA is illustrated in Figure 2.

CORBA is the specification of the functionality of the ORB, the crucial message bus that conveys operation invocation requests and their results to CORBA object residents anywhere, no matter where they are implemented. The CORBA specification provides certain interfaces to components of the ORB, but leaves the interfaces to other components up to the ORB implementers. By standardizing the interfaces between the underlying Object Request Broker (ORB) core, both the Client and Object Implementation to specify the manners of interactive object invocations. When a service is invoked, the Client needs only to hold a reference to the target object without knowing where the object resides. The object bus or ORB provides a unified system architecture that can be of tremendous help to system integrators of EDP systems. It should be equally easy to invoke an operation on an object residing on a remote machine, as it is to invoke a method on an object in the same address space. In addition, programming language transparency provides the freedom to implement the functionality encapsulated in an object using the most appropriate language, whether because of the skills of the programmers, the appropriateness of the language to the task, or the choice of a third-party developer who provided off-the-shelf component objects. The key to this freedom is to provide separation of interface and implementation.

There are two steps to facilitate IA architecture based on CORBA middleware. The first step is to gather background information on the OMG GL Facility [35]. IA architecture takes advantage of the proposed OMG GL Facility and further develops an algorithm for auditors to interoperate with different auditees' accounting application in order to perform better audit work. Therefore, the next step is to develop proposed audit objects so that IA architecture can be facilitated.

3.1 Introduction to OMG GL Facility

Broad recognition exists regarding the need to develop industry standards for accounting systems where differing protocols and applications abound [4]. The OMG (Object Management Group) General Ledger Facility (GL Facility) recently submits proposed interfaces, and their semantics, that are required to enable interoperability between General Ledger systems and accounting applications, as well as other distributed objects [35]. The OMG GL Facility is conformant with international accounting standards for double entry bookkeeping. The GL interfaces comprise a framework (in the object-oriented sense), that supports the implementation of accounting client applications, for example: Accounts Payable, Accounts Receivable, Payroll, and so forth. The architectural intention is to facilitate the convenient implementation of interoperable accounting applications, referred to as "clients" in this specification.

Figure 3 illustrates the GL Facility for auditee's EDP system and basic interface structure of IA architecture. Under this IA architecture, auditee's EDP system employed the OMG GL Facility to initiate its accounting life cycle and ensure a controlled development and implementation. The innermost boxes of Figure 3 are comprised of an OMG's proposed GL Facility, which contains 6 basic elements as shown in Table 2.

Interface	Purpose	Primary Client(s)
GLProfile	Client Session Establishment	All GL clients
GLBookKeeping	Data entry	Data entry clients
GLRetrieval	Data extraction	Reporting clients
GLAccountLifecycle	Account lifecycle management	GL administration clients
GLIntegrity	Data integrity checks	GL administration clients
GLFacilityLifecycle	GL lifecycle management	GL administration clients

Table 2: Synopsis of General Ledger Facility Interfaces
Source: SSI Ltd., et at. "General Ledger Facility", OMG DTC Document finance/98-07-02,1998 [35].

Through utilizing the interfaces above, the GL Facility can create the basic functions of maintaining an organization's accounting information system. The GL Facility specifies interfaces that encapsulate distributed object frameworks implementing accounting general ledgers. The most essential data structure/interface for general ledger is an "*Account*".

In order to illustrate more clearly on "*Account*", we shall use an abstracted form of C Language to define "*Account*" interfaces. The GL Facility maintains additional state values for GL accounts with their identifier and descriptive name. The basic GL *Account* interface is defined as follows:

```
struct Account {
        wstring         GLAcc_ref;              // GL Account reference
        wstring         GLAcc_name;     // account name
        wstring         GLreporting_code;       // grouping code
        Currency default_currency;
        Money           balance;
        Boolean         is_control;
        Money           mth_bal;
        Money           ytd_bal;
        wstring         con_acc_kind;
        wstring         con_acc_desc;
        wstringList     optional_fields;  };
```

Because *object-oriented technology* has the function of inherence, we therefore expand the previous GL *Account* Interface to specific accounts such as Cash, Accounts Receivables, Accounts Payables, Equity, Revenue and Expense in Appendix A. All these accounts are special designations of GL *Accounts*, that otherwise behave as "regular" accounts.

3.2 Proposed Audit Objects

After understanding and implementing the OMG GL Facility on auditees' EDP systems, the auditor should have gained an initial understanding of both the organization's accounting structure and how it affects the way in which the auditor can directly access the accounting information for all his auditees. In step 2, the auditor can build his own audit objects such as bank confirmation, accounts receivables confirmation, inventory counting sheet, bank reconciliation and so on to cope auditee's live data with auditor's needs. These audit objects allow the auditor to improve his focus and scope of the audit.

This section gives definitions of CORBA IDL interfaces as well as data structure that are not defined in the GL Facility of OMG and are purely for audit purposes. We use a confirmation letter as an example to illustrate proposed audit objects. The following is a CORBA interface needed for a confirmation letter.

```
struct ConfirmationLetter {
  wstring          company_name;
  wstring          address;
  Date             date;
  Money            balance;
  wstring          account_type;                  // (A/R, A/P, Cash)
  wstring          cash_dep_type;                 // deposit data items for Cash type
  wstring          cash_dep_acc_number;
  wstring          cash_dep_Interest_Rate;
  wstring          cash_dep_balance;
  wstring          cash_debt_acc_type;            // debt data items for Cash type
  wstring          cash_debt_balance;
  wstring          cash_debt_exp_date;
  wstring          cash_debt_interest_rate;
  wstring          cash_debt_thr_which_int_is_paid;
  wstring          cash_debt_collateral;
  Boolean          is_postive;                    // if the letter is positive or negative
```

4. The Advantage Of IA Architecture

This section presents two auditing tests, one is an internal control test and the other is a substantive test, to illustrate the advantage of IA architecture. A typical purchasing transaction is shown in Figure 4, This purchasing transaction illustrates the role of IA architecture elements and how the proposed architecture will potentially be affected by (or affect) outside trading parties. We assumed that all of the parties involve at this transaction are supported with the CORBA GL Facility. These examples will demonstrate how auditors may make use of IA architecture to perform both an internal control test and a substantive test. We also discuss how audit objectives can be better achieved in a more effective and efficient manner with IA architecture. The supporting information technologies that correspond to each step are included in the tables.

4.1 Perform Internal Control Test Under IA Environment

In performing any attestation function, auditors continue to be responsible for ensuring the stakeholders of the existence, adequacy and functionality of internal controls [28]. The scenario below shows how an auditor takes advantage of IA architecture to perform an internal control test. CPA L audits internal control of Company D under IA architecture environment.

With standard interface via a homogeneous object bus over the *Internet*, CPA L can directly access Company D's *Accounts Payable (A/P)* system anytime through proper authorization in order to enhance the reliability of audit evidence. This mechanism eliminates the needs for the auditor to develop different specialized audit programs to cope with the inconsistent file formats of each auditee. Therefore, the auditor can design one audit software program for all his auditees' EDP systems under IA architecture.

This example also shows that internal control checkpoints could either be created in the system or embedded in the system through use of a portable code. For example, CPA L can set an audit module with a PnP function on Company D's EDP system. If there is any A/P amount over $100,000, the audit module will shift A/P information back to CPA L's computer. Audit software will check whether the embedded portable code of the digital approval signature is complete or not. The PnP function can also provide the auditor with information about when to start the audi software in order to disclose exceptions and potential frauds with distributed object infrastructures and high speed networks. This mechanism accomplishes the same goals as other concurrent CAATs such as embedded modules and SCARF. However, embedding any audit modules into an auditee's EDP system at early stage are no longer required and modifying audit strategies can be done locally.

CPA L can easily to access Company D's accounting information anywhere on the LAN or WAN from any desktop. A single search will be able to access documents in any repository. Auditors can use the IA architecture to conduct significant portions of an audit from any location, including external offices, either through direct system access or approved dial-up access. The IA architecture also specifies multilevel security based to ensure the authentication of the users. The CORBA interfaces needed for this example are *Accounts Payable (A/P)*.

Business logic	Supporting Information Technology
1. CPA L loads the self-designed audit software to actively monitor Company D's internal control test of *Accounts Payable (A/P)* account via CORBA interface.	1. Audit software.
2. CPA L loads the PnP function of audit module to implant proper checkpoints for *A/P* balance over $100,000.	2. Confidentiality, integrity, non-repudiation, PnP, and CORBA.
3. The audit module will shift the *A/P* information back to auditor's office through CORBA.	3. PnP.
4. Upon receiving *A/P* information, audit software runs locally to check if proper approval performed.	4. Audit software.
5. The audit software prints an exception report and informs CPA L immediately if there is any *A/P* exceeds $100,000 without proper approval.	5. Audit software, confidentiality, and authentication.

4.2 Perform Substantive Test Under IA Environment

IA architecture can also support a substantive transaction testing. One of the examples is the confirmation process. The confirmation function more than likely will continue to be a useful form of audit evidence, as long as auditors require a substantive evidence of trading related balances. This example illustrates how CPA L can gather confirmation request data from Company D through CORBA standard interfaces and send out the confirmation letters to the vendors via prearranged secure channel directly. To reduce the manually intensive process, it may be beneficial to electronically confirm transaction through GL Facility/CORBA.

We designed an auditor's proprietary interfaces for a confirmation letter, as shown in the previous section. A confirmation request can be prepared using audit software based on the live data retrieved from the auditee's EDP system. Conceptually, the confirmation response would function like an application acknowledgment, except that the acknowledgment is sent to the auditor rather than to the trading partner. If no exceptions are noted, it could be automated. The CORBA interfaces needed for this example are *Accounts Payable* and *Confirmation Letter*.

Business Logic	Supporting Information Technology
1. Get all A/P information from Company D via its CORBA interfaces *(GLRetrieval)* through audit modules. For each vendor in the account *A/P*, performs Steps 2-8.	1. Authentication, confidentiality, PnP and CORBA.
2. Retrieve information from *A/P* account for a typical confirmation letter, that includes vendor's name, address, invoice number(s), account balance, due date, and transaction description, etc.	2. Confidentiality, integrity, non-repudiation, PnP and CORBA.
3. Get *year-to-date balance* of each vendor from *A/P* account via CORBA interfaces.	3. Confidentiality, integrity, non-repudiation, and CORBA.
4. Prepare the confirmation letters and get *Digital Signature* of Company D.	4. Non-repudiation.
5. Send the confirmation letters to the vendors through CORBA middleware.	5. E-mail, EDI, or other internet technology.
6. Repeat until all confirmation letters are sent out.	
7. Wait for replies from vendors.	
8. Examine the replied letters from Company D's vendors. Prepare an exception report if needed.	

IA architecture offers numerous potential benefits. Automated confirmation process or similar substantive tests can save time for auditors and increase the reliability of audit evidence. Electronically review for special transactions periodically avoids interruptions to normal workflow and stores important audit trails. Travel and research costs are reduced. Auditing efficiency is increased because CPAs and audit supervisors can review audit progress from remote locations whenever needed.

5. Conclusion

In this paper we identified three emerging information technologies to constitute software architecture to facilitate IA. The technologies include *object-oriented technologies, distributed middlewares, and Internet security technologies*. We also expanded an established OMG GL Facility from the field of accounting to auditing.

The architecture described in this paper elaborates the steps to facilitate integrated auditing concepts and design effective objects for auditing. For the purpose of concept verification, we presented a prototype EDP system based on CORBA standards, a well-known object-oriented distributed middlewares from OMG. This system emulates an internal control test and a substantive test where it supports OMG's *General Ledger Facility*.

By using IA architecture, auditing software is no longer constrained by the nature of the hardware platforms being used, the information processing applications being examined, or the types of examinations being performed. The use of IA architecture provides a convenient way for the auditor to implement audit procedures through the computer, not around the computer. With proper audit module (PnP) design, system alarms and reports can call the auditor's attention to any deterioration or anomalies in the auditee's EDP systems. Moreover, audit software can be designed and deployed at the auditor side independently from the auditees' EDP systems. Therefore, it does not suffer from drawbacks as concurrent CAATs.

6. Suggestions For Future Research

We have outlined IA architecture to integrate three different technologies for auditors. Future research can identify specific audit objects that will be useful in different industrial situations. Some questions of interest that may need research are (a) what are the benefits of integration through workflow management in IA architecture [24]? (b) Which technologies are most useful for promoting integration in auditing implementation? And (c) the issues of legal and regulatory evidentiary requirement, records retention and disaster recovery [28].

7. Reference

1. American Institute of Certified Public Accountant, Computer Assisted Audit Techniques, AICPA, New York (1979).
2. American Institute of Certified Public Accountant, Auditing with Computers, AICPA, New York (1994).
3. American Institute of Certified Public Accountant, Auditing in Common Computer Environments, AICPA, New York (1995).
4. American Institute of Certified Public Accountant, Information Technology Age: Evidential Matter in the Electronic Environment, AICPA, New York (1997).
5. American Institute of Certified Public Accountant, Analytical Procedures, AICPA, New York (1997).
6. American Institute of Certified Public Accountant, The IT Committees' Top 10 List, Journal of Accountancy 183 (2 (February)) (1998).
7. J. Birtcher, Auditing client/server information processing, EDPACS 22 (8 (February)) (1995) 1-4.
8. G. Booch, Object-oriented Design with Applications, The Benjamin Cumings Publishing Company, Inc. (1991).
9. B. Brown, Audit and Control of Electronic Funds Transfer, EDPACS 23 (10 (April)) (1996) 1-6.
10. Steven P. Casazza and Mark Magath, Continuous Monitoring of Application Systems, IS Audit & Control Volume 5 (1998) 21-25.
11. J. I. Cash, A. D. Bailey, Jr. and A. B. Whinston, A survey of techniques for auditing EDP-based accounting information systems, Accounting Review 3 (4 (October)) (1977) 813-832.
12. S. Chan, Managing and auditing EDI systems development, CMA 65 (November)

(1991) 12-15.
13. D. Coderre, Broadening the Audit Scope with Computer-Assisted Audit Tools and Techniques, EDPACS 14 (8 (February)) (1997) 1-15.
14. The Common Object Request Broker: Architecture and Specification, Revision 2.0, Jul. 1995, http://www.omg.org/corba/corbiiop.htm.
15. B. E. Cushing and M. B. Romney, Accounting Information Systems-A Comprehensive Approach, 5th ed. Addison-Wesley (1990).
16. D. R. Carmichael, J. H. Willingham, and C. A. Schaller, Auditing Concepts and Methods-A Guide to Current Theory and Practice, 6th ed., McGraw-Hill, (1996).
17. The Component Object Model Specification, http://www.microsoft.com/oledev/olecom/title/htm.
18. M. Donahue, Entering a New Era in IS Audit and Control, EDPACS 21 (12 (June)) (1994) 1-5.
19. C. Fritzmeyer and Carmichael, ITF: a promising computer audit technique, Journal of Accountancy, (February) (1973) 74-58.
20. M. Fischer, EFT: Controlling the Risk", Journal of Accountancy,(June,1988)130-131
21. R. J. N. Gascoyne, CAATs it if you can (auditing), CA Magazine, 110 (June) (1992) 38-40.
22. Glenn L. Helms, Top Technology Concerns for the Attest, Audit and Assurance Services Functions, IS Audit & Control Journal 2 (1999) 46-48.
23. J.V. Hansen, and N. C. Hill, Control and Audit of Electronic Data Interchange, MIS Quarterly (December 1989) 403-414.
24. Peter van Helden and Andre Lighthart, Workflow Management in SAP R/3, a New era in Automated Business Process Management? IS Audit & Control Journal 3 (1997) 46-50.
25. R. Hoffman, Micros in Accounting, Journal of Accountancy (September 1988) 146-150.
26. Barry Knaster, Trial Balancing Act, Accounting Technology (June 1998) 23-32.
27. R. B. Lanza, Take My Manual Audit, Please, Journal of Accountancy (June) (1998) 33-36.
28. R. F. Lucy, Electronic Document Management: an Internal Control Paradigm Shift, IS Audit & Control Journal 5 (1997) 31-35.
29. T. W. Lin, and D. C. H. Yang, The use of microcomputers in auditing- a survey, EDP Auditor Journal 4 (1990) 73-80.
30. B. McKinzie, Understanding and Using Audit Trails, EDPACS 21 (13 (July)) (1994) 6-15.
31. R. Powell, Auditing Electronic Funds Transfer, IS Audit & Control Journal 2 (1994) 48-51.
32. D. Prawitt, M. Romney, and S. Zarowin, The Software CPAs Use, Journal of Accountancy183 (2 (February)) (1997).
33. Java Remote Method Invocation Specification, Revision 1.42, JDK 1.2, Javasoft Ltd., Mountain View, CA USA. (October 1997).
34. Bruce Schneier, Applied Cryptography - Protocols, Algorithms, and Source Code in C, John Wiley and Sons Inc., New York (1996).
35. SSI Ltd., et at. General Ledger Facility, OMG DTC Document finance/98-07-02, (1998).
36. R. Watson, The Use of Microcomputers in the Audit Environment, The EDP Auditor Journal 1 (1988) 31-42.
37. Myles Walsh, Why Software Continues to Cost More—and what the is auditor can do, EDPACS (May) (1994) 8-15.
38. Ron Weber, "Information Systems Control and Audit", Prentice Hall, New Jersey, USA. (1999).
39. H. Will, The New CAATs: Shifting the Paradigm, EDPACS 22 (11 May,1995) 1-14
40. A. L. Williamson, The Implication of Electronic Evidence, Journal of Accountancy 183 (2 (February)) (1997).
41. D. A. Watne and P. B. B. Turney, Auditing EDP Systems, 2nd ed., (1990)
42. C. Zoladz, Auditing in an Integrated EDI Environment, IS Audit & Control Journal 2 (1994) 36-40.

Appendix A. Related CORBA General Ledger Interfaces

This section gives definitions of CORBA IDL interfaces as well as data structures that are basic accounting objects and used in the examples of internal control test and substantive test but not defined in the GL Facility of OMG[41]. These interfaces are designed for EDP systems commonly used in the manufacture industry.

```
interface GLProfile;                                    // establish client session
interface GLBooKeeping;                                 // information entry
interface GLRetrieval;                                  // information acquisition
interface GLAccountLifecycle                            // account lifecycle management
interface GLIntegrity;                                  // information integrity checks
interface GLFacilityLifecycle                           //GL lifecycle management

typedef Fdcurrency:             :Date     Date;
typedef Fdcurrency:             :Money    Money;
typedef wstring                 Currency;

enum AccountType{
        Cash,
        Accounts Receivable,
        Accounts Payable,
        Equity,
        Revenue,
        Expense,
};

typedef sequence,AccountType. AccountTypeList;

interface       GLRetrieval  {
                CashInfoList                            cash();
                AccountsReceivableInfoList      accounts_receivable();
                AccountsPayableInfoList                 accounts_payable();
                EquityInfoList                          equity();
                RevenueInfoList                         revenue();
                ExpenseInfoList                         expense();      };

interface GLProfile  {
      GLRetrieval  gl_retrieval();    };

struct Account {
        wstring                 GLAcc_ref;              // GL Account reference
        wstring                 GLAcc_name;             // account name
        wstring                 GLreporting_code;       // grouping code
        Currency                default_currency;
        Money                   balance;
        Boolean                 is_control;
        Money                   mth_bal;
        Money                   ytd_bal;
        wstring                 con_acc_kind;
        wstring                 con_acc_desc;
        wstringList             optional_fields;   };
```

```
struct Cash{
        Account                 basic_acc;          // basic account information
        Date                    date;
        wstring                 client_name;
        wstring                 bank_account_number;
        wstring                 bank_account_type;
        wstring                 bank_interest_rate;
        Money                   beginning_balance;
        Boolean                 is_debit_credit;
        wstringList  summary;                   // description about bank's names, etc.};

struct AccountsReceivable{
        Account                 basic_acc;          // basic account information
        Date                    date;
        wstring                 client_name;
        Money                   beginning_balance;
        Boolean                 is_debit_credit;
        wstringList  summary;                   // description of clients' information, etc.};

struct AccountsPayable
        Account                 basic_acc;          // basic account information
        Date                    date;
        wstring                 vendor_name;
        wstring                 address;
        wstring                 year_to_date_balance;
        Date                    payment_date;
        Money                   beginning_balance;
        Boolean                 is_debit_credit;
        wstringList  summary;                   // description about vendors' information,  etc. };

struct Equity{
        Account                 basic_acc;          // basic account information
        Date                    date;
        wstring                 stockholders'_name;
        Money                   beginning_balance;
        wstringList             summary;            // description about stockholders' information,
                                                       etc.};

struct Revenue{
        Account                 basic_acc;                  // basic account information
        Date                    date;
        wstring                 client_name;
        Boolean                 is_debit_credit;
        wstringList  summary;                   // description, revenue type, etc.};

struct Expense{
        Account                 basic_acc;                  // basic account information
        Date                    date;
        wstring                 client_name;
        Boolean                 is_debit_credit;
        wstringList  summary;                   // description, expense type, etc.};
```

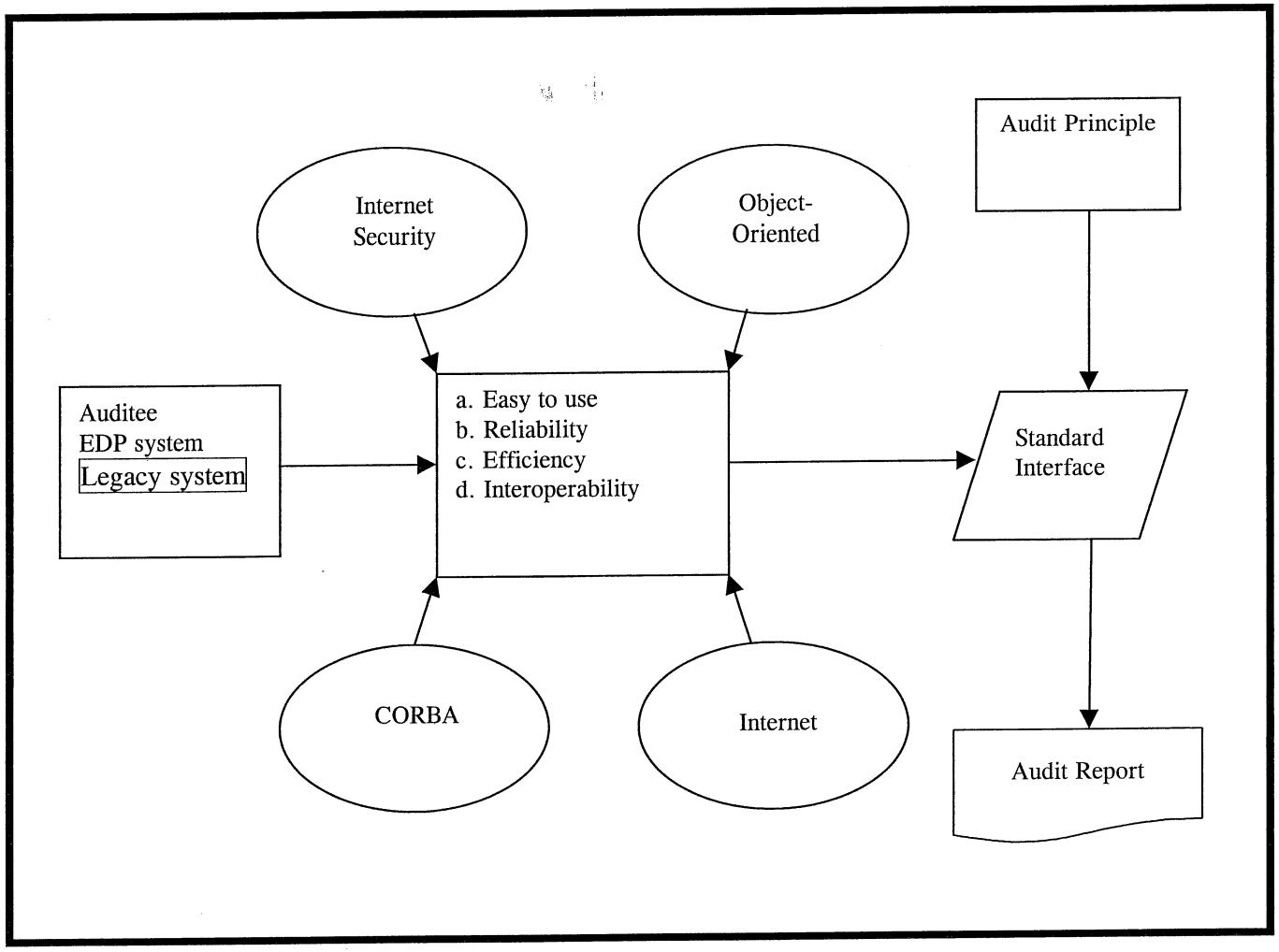

Figure 1: Integrated Auditing Architecture

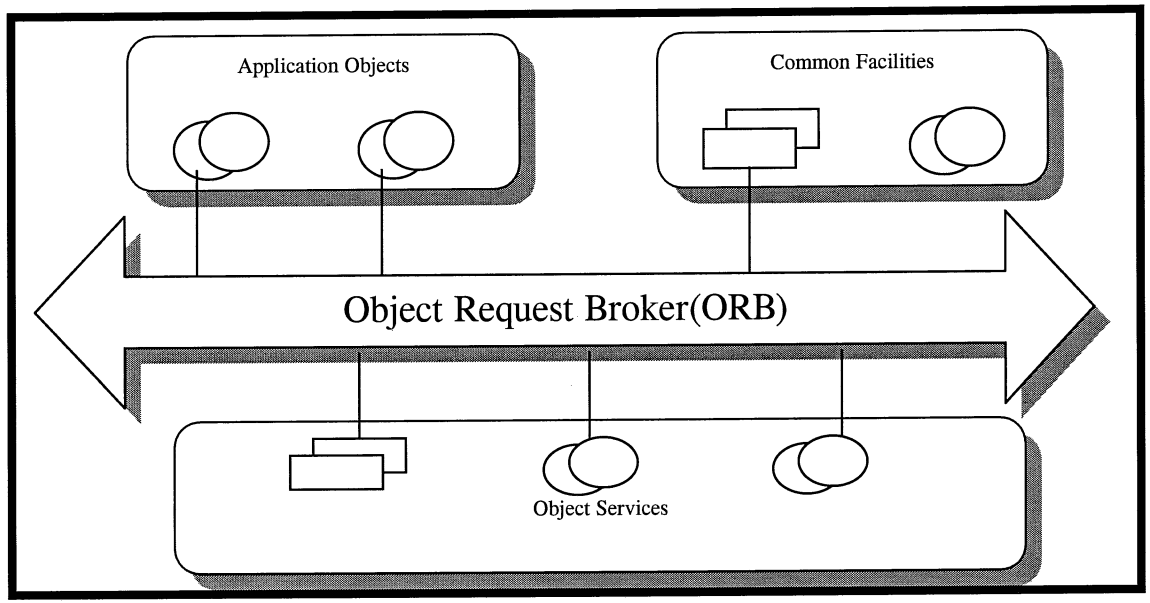

Figure 2: OMG's COBRA Structure

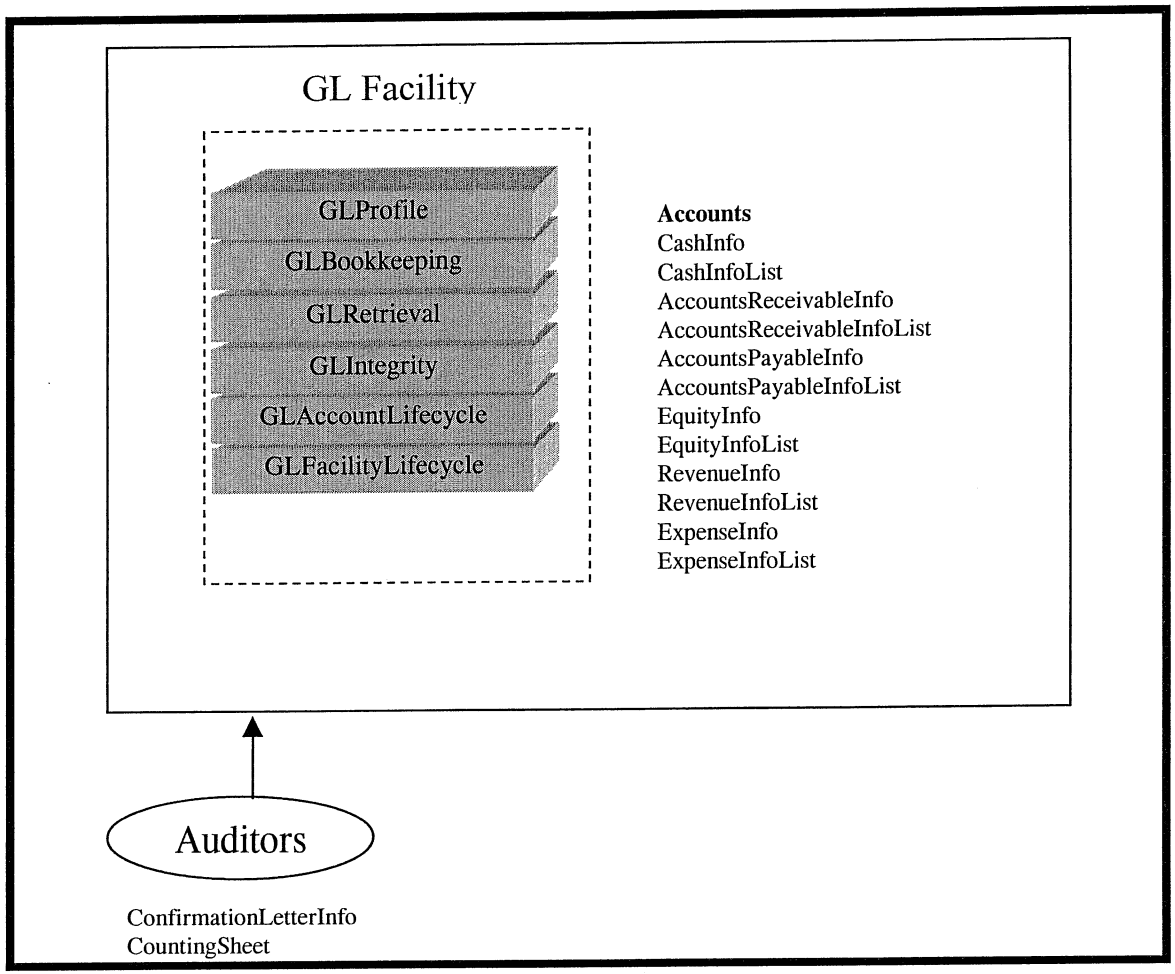

Figure 3: Structure Of OMG GL Facility

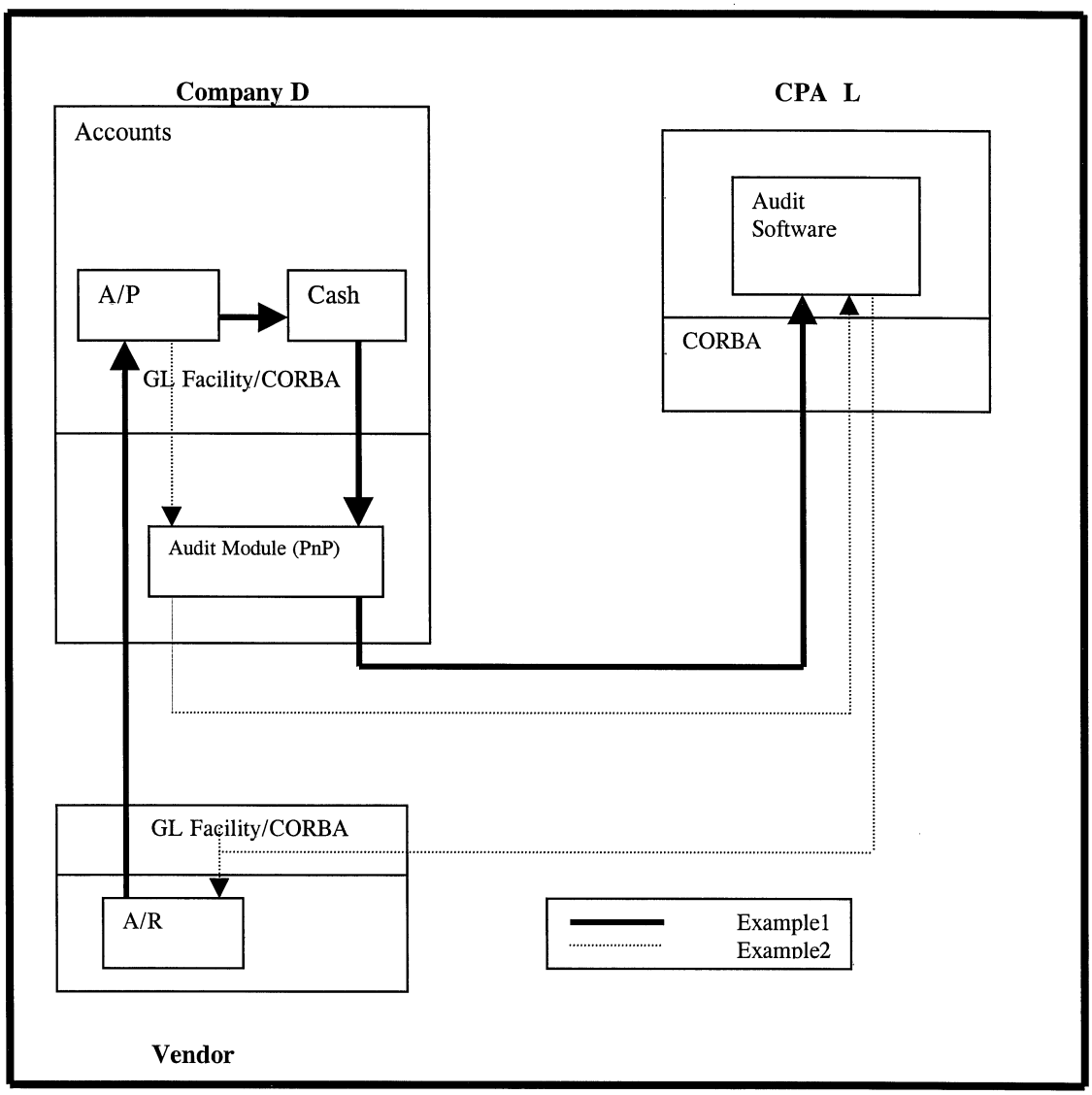

Figure 4: Implementation Of IA Architecture