A Method For Reverse Engineering Legacy Accounting Systems

Joseph H. Callaghan (E-mail: callagha@oakland.edu), Oakland University
Thomas W. Lauer (E-mail: lauer@oakland.edu), Oakland University
Eileen Peacock (E-mail: peacock@oakland.edu), Oakland University

Abstract

The growing emphasis on information for the development of products, services, and managing activities has increased the need for the integration of data which has been collected and used for various IS applications. Problems arise from the integration of information from such a variety of sources. One approach to this problem is reverse engineer these systems. Reverse engineering derives a data model from existing systems with the aim of redesigning it. The paper describes a data modeling approach that takes data from general journal and an archetypal specialized journal and translates it into an entity-relationship diagram. The article also discusses areas for future research.

Introduction

Over the last decade, organizations have recognized the escalating maintenance burden imposed by existing legacy systems. Information systems executives may be responsible for hundreds or thousands of relatively old legacy programs and data files. These systems are often mission critical to the firm and typically require heavy maintenance. Systems supporting accounting functions are disproportionately represented in this group. This is due largely to the fact that these systems were historically among the first to be developed on now obsolete hardware and software technology and at a time when software engineering was at its infancy. The IS executive would be very happy to replace such systems to improve their functionality and reduce the maintenance burden, but

must do so at the expense of new development projects that may already be in a queue with a two to three year backlog. One approach to this dilemma is to reengineer some systems that are critical to the ongoing operations of a firm. Reengineering consists of 'reverse engineering' to discover the program and data structure followed by 'forward engineering' to render the system more technically sound.

Choices about how to allocate resources toward maintenance should be based on criteria that include both the technical characteristics and the business functionality of the system to be maintained. Technical characteristics have a direct bearing on the ease with which a given piece of software can be maintained while the business functionality is the determinant of the value of expending the effort. Thus technical characteristics relate to the cost aspect of the cost benefit

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

analysis, while business functionality determines the benefits. Those systems with high business functionality and low maintainability due to technical weaknesses are the best targets for maintenance efforts.

Maintenance strategies for legacy systems include approaches that range from total replacement through rewriting or purchasing a new system on one extreme to strategies that utilize most or all of an existing system [e.g. Sprague and McNurlin 1993]. Systems that have low business functionality together with technical weaknesses that make them hard to maintain are often candidates for replacement. Systems that are hard to maintain but have high business functionality are candidates for reengineering. Typically, such systems require considerable effort to maintain because the underlying logic is difficult to follow and the data are not managed. This may have come about over the life of the system as it underwent numerous 'bug fixes' and enhancements to address errors in some of the system's programs and changes in the business environment. Some typical situations are: a) the logic and data of the system were maintained, but the documentation was not updated to reflect system changes, b) changes in the logic altered the original structure of the system, c) data input to the system may come from a variety of sources, and d) data output from the system may be sent to a variety of targets.

Reengineering an existing system consists of reverse engineering the system to extract its underlying business logic and data requirements and then forward engineering the business logic and data requirements to new platforms. McClure [1992] distinguishes between data reverse engineering and logic reverse engineering. Data reverse engineering entails uncovering the data structure required to support certain business logic. Some benefits of this approach are that: it enables the migration to new improved database management systems, it creates a logical data model (e.g. an entity relationship diagram) to support future maintenance of the system, it organizes the data into a database which

can be used to develop new systems, and it organizes the data to ensure its integrity. Data reverse engineering has as its objective the improvement of data accessibility and integrity.

However, reverse engineering accounting legacy systems is not without problems. Most of these systems were developed to mimic double entry bookkeeping. Various authors [McCarthy 1982, McCarthy 1979, and others] have advocated the use of data modeling techniques from database theory, namely the entity-relationship model [Chen 1976] for modeling accounting data *in lieu* of the traditional accounting model. McCarthy [1982] suggests that this approach will remedy some weaknesses in the conventional accounting model identified by others [e.g. AAA, 1969, 1971]: lack of non-monetary measurements, inappropriate classification schemes for data that may hide their nature from non-accountants, aggregation of data at too high a level, and restricted integration with data from other areas of the enterprise. Regardless of the merits of these arguments, movement away from traditional models has been slow [Weber 1986].

One reason for this relatively slow adoption is that companies have tremendous investments in "legacy" systems, the oldest still functioning computerized systems in an enterprise. Most often, these systems are concerned with accounting data and were developed in conformance with traditional accounting models. A significant problem attributable to the existence of these systems is the integration of accounting data with other corporate data. One significant approach to modeling accounting data can be found in the work of McCarthy and a group of colleagues who have developed the REA model and more recently the REAL model [Hollander, *et al.* 1995]. Whereas the REAL model is a top-down clean slate approach to data modeling, what we are proposing is akin to reverse engineering. Reverse engineering derives a data model from existing systems with the aim of re-designing it. Thus our approach is aimed at the problem of converting the large amounts of existing data. This paper begins with a brief dis-

cussion of the problem of data integration in the modern enterprise and the benefits of data modeling. The main body of the paper describes a data modeling approach to data from general journal and an archetypal specialized journal. Finally conclusions are offered.

Integration of Corporate Data.

One characteristic of the modern era is the growing emphasis on information for the development of products and services and managing enterprise activities. Products and services can be thought of as consisting of both physical material and information. For example products developed in the financial sector can be mostly informational in their make-up. These products can use information derived from transaction data or from data integrated from disparate corporate sources. The value of enterprise data is further enhanced because it may be re-used any number of times. In order to realize such benefit, the corporate data resource must be well managed and designed with the capability of integrating data from disparate sources.

Integrating data from "legacy" systems poses significant problems. Data from pre-database systems are stored in files tied to specific programs. In many companies, chains of programs were created where one program used the output of another for input. Maintenance changes to one program could cascade through several other programs producing violations of data integrity and redundant data, [Inmon 1989]. In many firms the burden of deriving useful information in such an environment has led to attempts to centralize data in databases. Two approaches to integrating data within corporate databases are data warehousing and reverse engineering. Data warehousing brings together and integrates large quantities of data from "legacy" systems and historical transactions. Reverse engineering derives a data model from existing systems [McClure 1992]. This enables the existing system to be re-engineered to remove inconsistencies in the data and improve the functioning of the system. Thus data integrity is im-

proved and data redundancy is eliminated for the re-engineered system. In order for data warehousing to be successful, something akin to reverse engineering must be carried out so that data entered into the data warehouse is not corrupted.

Benefits of Data Modeling.

Data modeling takes place at different levels of abstraction: conceptual, internal, external, and physical. In general, conceptual models are hardware and software independent, internal and external models are hardware independent and software dependent, and physical models are hardware and software dependent. Conceptual models, the most common of which is the entity-relationship diagram (ERD), model data as seen by the business. Three internal models are hierarchical, network, and relational, all of which can be derived from an ERD. The ERD is most easily translated into a relational model.

Data modeling serves important functions aimed at enabling a firm to realize benefit from their data assets. Modeling is the basis for creating databases that are as stable as possible. Modeling ensures that end users can access the information that they need. In addition, modeling helps ensure that different applications can work together. One particularly important function of an ERD is that it serves as a communications medium for facilitating interaction among database designers, programmers, and end users. This function is particularly important for bridging the gap between data from "legacy" systems and integrated corporate databases.

One implementation of a data model would be in a relational database framework. Such a framework entails adherence to certain system integrity rules, e.g. entity integrity, referential integrity, as well as database specific rules, e.g. through the use of appropriate domains. Further these systems support the use of *ad hoc* querying capabilities implemented with a language such as SQL (Structured Query Language). Moreover, relational systems allow the

integration of corporate data, through the use of relational operators, e.g. joins, projections, restricts, etc. Finally, such systems are more amenable to changes relative to traditional accounting systems and can, therefore, facilitate the reverse engineering approach advocated here for "legacy" accounting systems.

Integration of Accounting Systems and RDBMS

It can be shown that traditional accounting information systems (AIS) and relational database management information systems (RDBMS) face many of the same issues and problems. Both are fundamentally data processing models seeking efficient, effective, and flexible ways at processing and viewing data. When these data are business data, the linkage is even clearer. In fact, it will be shown that, with very little manipulation, apparently unrelated topics are very much related. Thus, it can be shown that the need for specialized journals and the notion of data normalization are connected. It can even be shown that the fundamentals like debits equaling credits and data normalization are equivalent problems. Thus, with only minor changes to the way accounting information systems are generally understood, a foundation can be established to learn data processing models generally and DBMS, especially relational ones, more specifically. Thus, the linkages could reinforce one and the other.

An essential aspect of RDBMS is that all data can be viewed as tables, consisting of attributes, rows and columns [Date 1995]. The two dimensionality of tables is enforced by the first normal form (1NF)¹, which requires that each row/column data item be a scalar value. Thus, no cell can contain more than one data element. Similarly, a general journal could be viewed as a table, consisting of the attributes: transaction number, date, account title, account number, \$debit or \$credit, logical post, etc. Moreover, the general ledger and the chart of accounts may be viewed as tables in the same manner, each having its own (not necessarily unique) attributes. In the case of the general journal, each row would not constitute a transaction, but rather an element of a transaction, perhaps even a compound one. It would be impossible to pre-specify the number of rows constituting a transaction, unless it were known in advance that the entry was simple, and therefore two-rowed. One problem immediately apparent is that each row would contain either a \$debit or \$credit leaving the other one zero or null, which is inefficient from a RDBMS normalization point of view.

General Journal as Normalized Relation

Table 1 depicts the traditional general journal. Tran_#, or transaction number, is taken to be a counter within each day. Date is transaction date, Title is account title, and Acct_# is

Table 1						
Traditional General Journal Entries						
Tran_#	Date	Title	Acct_#	Debit	Credit	Explain
1	1/3/96	Cash	11001	100.00		
		Receivable	11003	200.00		
		Sales	41001		300.00	
credit sale with down payment						
2	1/3/96	COGS	51001	150.00		
		Inventory	11004		150.00	
sale of inventory						

the account number or folio to which transaction amounts, debits and credits, are posted to the general ledger. Explain provides space for an open-ended textual explanation of the entire transaction.

The general journal contains both transaction level information and account level information. The latter in relational terms represents a repeating group. More specifically, the domain for account level information does not consist of scalar values. It is rather straightforward to represent the transaction information of the general journal in first normal form (1NF) by filling in the blank values to support the repeating group values. Table 2 depicts this form.

A requirement of a relation is that each tuple or row be uniquely identifiable. Any set of attributes that can accomplish this is a candidate key, whereas the actual set of attributes selected, in any given context, is the primary key (PK). The PK of Table 2 is the composite of the *tran_#*, *date*, and *acct_#* attributes. The meaning of any particular tuple is the transaction's effect on the general ledger balances.

Table 2 still poses problems related to database update (insert, delete, and update) operations. These problems are referred to as update anomalies, since the repeating information requires coordination for fulfilling integrity constraints. It is also inefficient in data entry terms.

A more desirable form to achieve is second normal form, where all nonkey attributes are functionally dependent on the PK. This can be accomplished by three projections of Table 2:

1. T (*Tran_#*, *Date*, *Explain*)
Primary Key (*Tran_#*, *Date*)
2. D (*Tran_#*, *Date*, *Acct_#*, *Debit*, *Credit*)
Primary Key (*Tran_#*, *Date*, *Acct_#*)
Foreign Key (*Tran_#*, *Date*) references T
Foreign Key (*Acct_#*) references C
3. C (*Acct_#*, *Title*)
Primary Key (*Acct_#*)

Since the nonkey attributes of T and C are mutually independent, i.e. they can be updated independently, these relations are also in 3NF. Relation T tracks the transaction level information. The traditional AIS analog to C is obviously the chart of accounts, with an embedded coding structure associated with the *Acct_#*. For instance, the first digit of *Acct_#* indicates classification (1 = > Asset, 2 = > Liability, ..., 5 = > Expense), and the second digit reserved for an account subclassification scheme, e.g. 11 = > current asset. This would reserve the last three digits for account identification, within classification and with subclassification. Although, these hierarchical relations could be made explicit to the RDBMS, for the purposes of this paper, this scheme is encapsulated. As far as the RDBMS is concerned, *Acct_#* is a scalar value. However, from an AIS point of view, the account number helps facilitate maintaining the underlying accounting data model, i.e. the fundamental accounting identity.

The Debit and Credit attributes of Relation D are not mutually independent in that an account in any particular transaction may be increased or decreased, both can not occur. Thus, if it is known that an account is being debited, then its credit value must be null or zero, and *vice-versa*. Relation D can be improved by the appropriate use of a signing convention to convert the debit and credit attributes into a single attribute, say Amount. This will eliminate the mutual dependence of debit and credit.

Establishing a Signing Convention

In a traditional AIS, all accounting classifications are either implicitly or explicitly positive. Therefore, a rule is necessary in order that the system will permit the equality of currency debits and credits, while maintaining the underlying fundamental accounting identity. The choice of where to put the increases and decreases is arbitrary. The important point is that the choice must result in a signing that is symmetric about the identity sign. The choice documented by Pacioli [Sprague 1972], resulted

Table 2 General Journal Entries: 1NF						
Tran_#	Date	Title	Acct_#	Debit	Credit	Explain
1	1/3/96	Cash	11001	100.00	0.00	credit sale
1	1/3/96	Receivable	11003	200.00	0.00	credit sale
1	1/3/96	Sales	41001	0.00	300.00	credit sale
2	1/3/96	COGS	51001	150.00	0.00	sale of inventory
2	1/3/96	Inventory	11004	0.00	150.00	sale of inventory

Table T				
General Journal Entries: 2NF, 3NF				
Tran_#	Date	Explain		
1	1/3/96	credit sale with down payment		
2	1/3/96	sale of inventory		
Table D				
General Journal Entries: 2NF				
Tran_#	Date	Acct_#	Debit	Credit
1	1/3/96	11001	100.00	0.00
1	1/3/96	11003	200.00	0.00
1	1/3/96	41001	0.00	300.00
2	1/3/96	51001	150.00	0.00
2	1/3/96	11004	0.00	150.00
Table C				
General Journal Entries: 2NF, 3NF				
Acct_#		Title		
11001		Cash		
11003		Receivable		
41001		Sales		
51001		COGS		
11004		Inventory		

in the following rule: for classifications to the left of the identity sign, the left hand side (DR) is positive and indicates the normal balance, while for classifications to the right of the identity sign, the right hand side (CR) is positive and indicates the normal balance. The negatives, of course, are inferred to be where the positives are not. The choice could have been made the other way, with the same control effect resulting, i.e. that \$debits equal \$credits. The result is the debit-credit equation:

$$\begin{array}{c} \Delta A_t + EX = \Delta L_t + \Delta SE_{t-1} + REV \\ \hline \begin{array}{cc|cc|cc|cc|cc} DR & CR & DR & CR & DR & CR & DR & CR & DR & CR \\ + & - & + & - & - & + & - & + & - & + \end{array} \end{array} \quad (1)$$

To circumvent this possibility would require that some sort of signing convention be implemented. Rather than arbitrarily assign \$debits as negatives and \$credits as positives or *vice-versa*, more insight can be developed by manipulating the debit-credit equation. Apart from showing their equivalence, this permits accountants to better understand both the traditional AIS and its RDBMS analogue. In addition, users can appreciate that the choice is one of either balance sheet or income statement orientation. Further, certain spreadsheet applications, like the standard accountant's worksheet could be column (attribute) reduced in an understandable fashion. Finally, understanding of the normalization process would be introduced in an accounting context, setting the stage for future development. Thus both accounting and RDBMS systems knowledge are acquired in an integrative manner.

One way to resolve this inefficiency requires that the Asset and Expense classifications of the debit-credit equation be moved to the right-hand side. The result would be:

$$\begin{array}{c} 0 = \Delta L_t + \Delta SE_{t-1} + REV - \Delta A_t - EX \\ \hline \begin{array}{cc|cc|cc|cc|cc} DR & CR & DR & CR & DR & CR & DR & CR & DR & CR \\ - & + & - & + & - & + & + & - & + & - \end{array} \end{array} \quad (2)$$

Classifications would zero sum, but the moved classifications would be negatively signed. In order to maintain positive classifications, the signs of the moved classifications would have to be reversed, resulting in:

$$\begin{array}{c} 0 = \Delta L_t + \Delta SE_{t-1} + REV + \Delta A_t + EX \\ \hline \begin{array}{cc|cc|cc|cc|cc} DR & CR & DR & CR & DR & CR & DR & CR & DR & CR \\ - & + & - & + & - & + & - & + & - & + \end{array} \end{array} \quad (3)$$

Thus under this income statement approach, revenues would be positively signed, while expenses would be negatively signed. Note that \$debits do not equal \$credits, but rather \$credits $\equiv$ - \$debits, or \$credits + \$debits $\equiv$ 0. However, the same control feature, so important to accounting systems is maintained. Conversely, if the right hand classifications of the debit-credit equation are moved to the left hand of the identity sign:

$$\begin{array}{c} \Delta A_t + EX - \Delta L_t - \Delta SE_{t-1} - REV = \\ \hline \begin{array}{cc|cc|cc|cc|cc} DR & CR & DR & CR & DR & CR & DR & CR & DR & CR \\ + & - & + & - & - & + & - & + & - & + \end{array} \end{array} \quad (4)$$

Making the classifications positive by reversing the increase and decrease assignments, results in:

$$\frac{\Delta A_t + EX}{\begin{array}{c|c} \text{DR} & \text{CR} \\ \hline + & - \end{array}} + \frac{\Delta L_t + \Delta SE_{t-1}}{\begin{array}{c|c} \text{DR} & \text{CR} \\ \hline + & - \end{array}} + \frac{REV}{\begin{array}{c|c} \text{DR} & \text{CR} \\ \hline + & - \end{array}} = 0 \quad (5)$$

This is the balance sheet approach, resulting in assets being positive, while asset claims, i.e. liabilities and equities, are negatively signed. It is noteworthy that both approaches are adopted by major software developers. For instance, ACCPACJ uses the balance sheet approach, while dBASEJ uses the income statement approach. Using the balance sheet approach, Relation D becomes:

Row Reduction and Special Journals

A reconstructed traditional general journal, ((T join D; on Tran_#, Date) join C; on Acct_#), in the RDBMS demonstrates the nonloss decomposition of the normalization process. Thus, in a relational system the traditional general journal can be presented to an accounting user as a view, while concurrently conforming to the relational model. However, although a general journal is a necessary default journal in an AIS, since all possible transactions and adjustments can not be pre-specified, certain high volume simple transactions can be modeled as single tuple events. This process is analogous to the use of specialized journals and subsidiary ledgers in a manual system. This would permit the relation T and D to be collapsed in a transaction-per-tuple relation or table. For example, credit sales transactions may be depicted as follows.

A derived attribute for amount could also be included for the extension of quantity and price. In a multi-product firm, the particular inventory item sold could also be included in this relation as a foreign key to an inventory relation, with the potential of a price lookup for the particular inventory item in yet another relation. Other aspects (attributes) of the credit sales transaction could be easily added in more "real world" contexts, e.g. invoice number. The point to notice is that the credit sales transaction, unlike its representation in the general journal, has been tuple-reduced to a single tuple. Relational operators and functions would permit both a column total "posting" of the derived attribute amount to accounts receivable (11003) and sales (41001), if a BOP L relation is employed or as an aggregate view, if all transaction relations are left open indefinitely. (See General Ledger discussion below.) Finally, although not modeled here, it would be straightforward to derive Customer Balances, i.e. a subsidiary ledger, at any point in time, by joining the TD relation with a Cash Receipts relation.

The general ledger in this relational system is a derived relation, consisting of aggregating the effects of all transactions by Acct_#. This means that the general ledger is a view of the modified D relation, and its relation to C; and not a base relation itself. In this sense, the AIS becomes "ledgerless." For practical reasons (mainly due to the cardinality (number of tuples) of the D relation), an alternative approach is to create a base, beginning-of-period (BOP) relation, say L, establishing the cumulative effect of D on account balances. This would be accomplished by a "closing process" that would take a snapshot of the derived relation for account balance. The tuples of relations T and D could first be archived and then deleted. The archived tuples would serve to provide a partial audit trail and support various historical, comparative reports.

Figures 1 and 2 depict the Entity Relationship Diagrams (ERD) for both the general and spe

Table D General Journal Entries: 3NF			
Tran_#	Date	Acct_#	Amount
1	1/3/96	11001	\$100.00
1	1/3/96	11003	\$200.00
1	1/3/96	41001	(\$300.00)
2	1/3/96	51001	\$150.00
2	1/3/96	11004	(\$150.00)

Table TD Sales Journal Entries: 3NF				
Tran_#	Date	Cust_#	Quantity	Price
1	1/4/96	31001	100	\$50.00
2	1/4/96	31005	200	\$50.00
1	1/5/96	31001	100	\$51.00
2	1/5/96	31005	300	\$51.00
3	1/5/96	31006	200	\$51.00

Table Cust Sales Journal Entries: 3NF				
Cust_#	CDate	Fname	Lastname	Address
31001	1/1/96	Joe	Smith	112 Mason
31002	1/1/96	Tom	Doe	435 Jackson
31003	1/1/96	Susan	Lauer	1512 Elm
31004	1/1/96	Mary	Peacock	555 Oak
31005	2/96	Eileen	Callaghan	335 Main

1. TD (Tran_#, Date, Cust_#, Quantity, Price)
Primary Key (Tran_#, Date, Cust_#)
Foreign Key (Cust_#) references Cust
2. Cust (Cust_#, CDate, Fname, Lastname, Address, etc.)
Primary Key (Cust_#)

Figure 1
Entity Relationship Diagram: General Journal

Figure 2
Entity Relationship Diagram: Special Journal

cialized transaction tables (journals) outlined above. These data models represent a first pass at reverse engineering the traditional AIS. Note that the implied classification and subclassification entity types are still imbedded in the account numbering scheme.

Conclusion and Suggestions for Future Research

This paper has outlined an approach to reverse engineer traditional Accounting Information Systems using a relational database approach. Such an approach would facilitate the integration of accounting and other enterprise-wide data, enhance the integrity of accounting data, enable the *ad hoc* querying capability of SQL, and facilitate further systems development.

Among the issues not addressed by this paper, but suitable for future research, are those related to recent advances in systems development. Examples of these systems advances are model-oriented tool-enhanced (MOTE) methods and the advent of objected-oriented databases. Using a MOTE approach entails developing both data and activity models at high levels of abstraction, while concurrently using software tools to generate code, data structures, and user interfaces. Thus, after having reverse engineered a legacy AIS, these methods and techniques could be utilized to re-engineer and implement an amended AIS. In a related manner, a reverse engineered data model could be redeployed as an object-oriented database rather than a relational one. These object databases not only permit non-traditional data types, e.g. pictures, video clips, and other objects, but also combine data and executable actions in a single repository. In either case, however, the reverse engineering techniques outlined herein could provide a first step to the use of these approaches, while retaining all the data elements of the legacy AIS.

In summary, AIS have been developed in a variety of technical environments utilizing different approaches to data management. Most often, the data and logic of legacy systems is

based on the double-entry method found in manual systems. In order to derive the benefits of the relational DBMS model, it is necessary to specify the means for translating double-entry data into third normal form.

AIS legacy systems are excellent candidates for reengineering. In many instances, they are technically deficient due to weaknesses in their data structure. They have a high level of business functionality because they support critical organizational activities. By re-structuring their underlying data models, the way is opened to increase their realized value while reducing the cost of their maintenance. The method proposed in this paper adds the necessary detail to accomplish the crucial step of reverse engineering the data.

Endnotes

1. A relation is in 1NF if and only if all underlying domains contain scalar values only; A relation is in 2NF if and only if it is in 1NF and every nonkey attribute is irreducibly dependent on the primary key; and A relation is in 3NF if and only if the nonkey attributes if any are (a) mutually independent, and (b) irreducibly dependent on the primary key [Date 1995].

References

1. American Accounting Association, "Report of Committee on Managerial Decision Models," *The Accounting Review*, Supplement, pp. 43-76. 1969.
2. American Accounting Association, "Report of the Committee on Non-Financial Measures of Effectiveness," *The Accounting Review*, Supplement pp. 164-211. 1971.
3. Chen, P.P. "The Entity-Relationship Model - Toward a Unified View of Data," *ACM Transactions On Database Systems* (March 1976), pp. 9-36.
4. Date, C. J., *An Introduction to Database Systems*, (Addison-Wesley, 1995).

5. Everest, G. C. and R. Weber "Accounting Systems in a Shared Data Environment," *The Accounting Review* (July 1982), pp. 554-578.
6. Inmon, W. H. *Information Engineering for the Practitioner. Putting Theory into Practice*. Yourdon Press. Prentice Hall. 1988
7. McClure, C. *The Three R's of Software Automation: Re-engineering, Repository, Reusability*, (Prentice Hall, 1992).
8. Sprague. C. *The Philosophy of Accounts*. (Scholars Book Co., 1907, 1908, 1972).
9. Sprague, R. H. Jr. and B. C. McNurlin, *Information Systems Management in Practice*, 3rd Edition. Prentice- Hall Inc.: Englewood Cliffs, NJ (1993).
10. Weber, R. "Data Models Research in Accounting: An Evaluation of Wholesale Distribution Software", *The Accounting Review*, (July 1986) pp. 498-518.